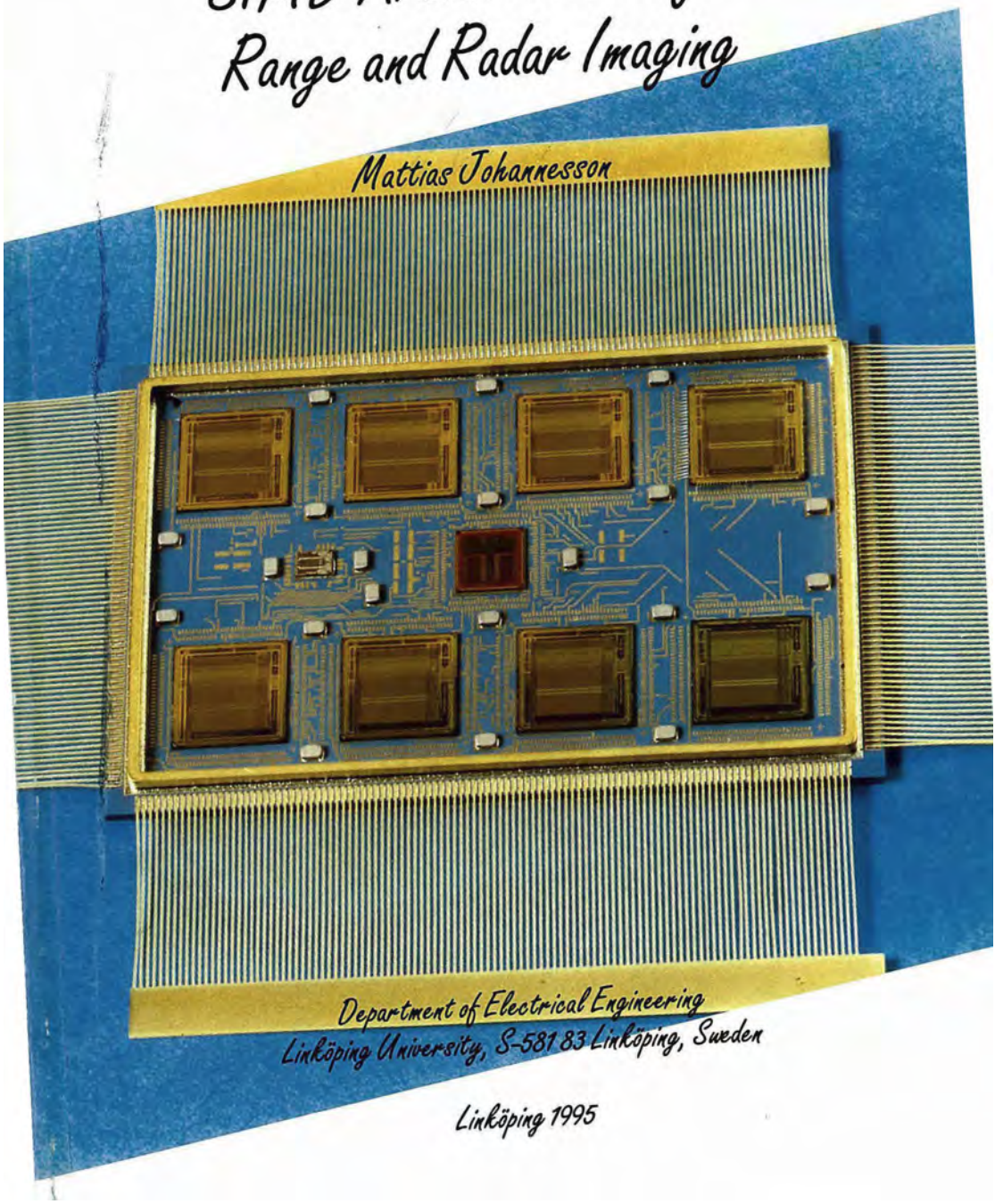
Linköping Studies in Science and Technology.
Dissertations No. 399

SIMD Architectures for Range and Radar Imaging

Mattias Johannesson



Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1995


Mattias Johannesson: SIMD Architectures for Range and Radar Imaging
Linköping 1995

Linköping Studies in Science and Technology. Dissertations No. 399

SIMD Architectures for Range and Radar Imaging

Mattias Johannesson

Akademisk avhandling

som för avläggande av teknisk doktorsexamen vid Tekniska högskolan i Linköping kommer att offentligen försvaras i sal E324 Schrödinger, hus F, Linköpings Universitet, fredagen den 24 november 1995, kl. 10.15.

Abstract

This thesis describes sheet-of-light range imaging and Doppler radar signal processing. The focus is on implementing algorithms on bit-serial SIMD processor arrays.

The reduction of 2D sensor data to 1D position data in sheet-of-light range imaging is a task well suited for smart optical sensors combining image acquisition and image processing on a single chip. Several smart sensor approaches aimed at sheet-of-light range imaging are presented. Among these is a novel sheet-of-light range imaging architecture based on the Near-Sensor Image Processing concept. We predict that this architecture is capable of range imaging with range pixel (rangel) frequencies up to 25 MHz while keeping a relative accuracy in the range of 1/500 and delivering intensity data concurrently with range. We also present several sheet-of-light range imaging algorithm implementations using the commercial smart image sensor MAPP2200. With this system we can reach up to 5 MHz rangel frequency with a relative accuracy of 1/500. General sheet-of-light system setup and calibration issues are also covered.

We also present how three different Doppler radar algorithms can be implemented on different versions of the VIP, Video Image Processor, architecture. VIP is a family of bit-serial SIMD processor arrays aimed at real-time signal processing. We present the design of three VIP architectures, RVIP, FVIP, and MVIP. The Radar VIP, RVIP, is aimed at ground-based surveillance radar applications. The Flight VIP, FVIP, is aimed at airborne search radars. The Matrix VIP, MVIP, finally, is aimed at airborne space-time adaptive processing in phased array antenna applications. Common for all these radars is that the real-time processing demands are very high, ranging from 2 to 40 Giga MAC operations per second, and that the IO data rates are high, up to 1.5 Gbit per second. FFTs are an important part of Doppler radar signal processing and FFT implementations on a VIP array are therefore presented in detail.

The RVIP architecture has been manufactured by Ericsson Microwave systems for use in their future radar systems.

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1995

ISBN 91-7871-609-8 ISSN 0345-7524

Linköping Studies in Science and Technology.
Dissertations No. 399

SIMD Architectures for Range and Radar Imaging

Mattias Johannesson

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1995

Cover illustration: 512 PE RVIP MCM.
Photo courtesy of Ericsson Microwave systems.

ISBN 91-7871-609-8
ISSN 0345-7524
Printed in Sweden by LJ Foto & Montage, Linköping/VTT-Grafiska, Vimmerby 1995

Abstract

This thesis describes sheet-of-light range imaging and Doppler radar signal processing. The focus is on implementing algorithms on bit-serial SIMD processor arrays.

The reduction of 2D sensor data to 1D position data in sheet-of-light range imaging is a task well suited for smart optical sensors combining image acquisition and image processing on a single chip. Several smart sensor approaches aimed at sheet-of-light range imaging are presented. Among these is a novel sheet-of-light range imaging architecture based on the Near-Sensor Image Processing concept. We predict that this architecture is capable of range imaging with range pixel (rangel) frequencies up to 25 MHz while keeping a relative accuracy in the range of 1/500 and delivering intensity data concurrently with range. We also present several sheet-of-light range imaging algorithm implementations using the commercial smart image sensor MAPP2200. With this system we can reach up to 5 MHz rangel frequency with a relative accuracy of 1/500. General sheet-of-light system setup and calibration issues are also covered.

We also present how three different Doppler radar algorithms can be implemented on different versions of the VIP, Video Image Processor, architecture. VIP is a family of bit-serial SIMD processor arrays aimed at real-time signal processing. We present the design of three VIP architectures, RVIP, FVIP, and MVIP. The Radar VIP, RVIP, is aimed at ground-based surveillance radar applications. The Flight VIP, FVIP, is aimed at airborne search radars. The Matrix VIP, MVIP, finally, is aimed at airborne space-time adaptive processing in phased array antenna applications. Common for all these radars is that the real-time processing demands are very high, ranging from 2 to 40 Giga MAC operations per second, and that the IO data rates are high, up to 1.5 Gbit per second. FFTs are an important part of Doppler radar signal processing and FFT implementations on a VIP array are therefore presented in detail.

The RVIP architecture has been manufactured by Ericsson Microwave systems for use in their future radar systems.

Acknowledgments

First I wish to thank my supervisor Professor Per-Erik Danielsson and co-supervisor Doctor Anders Åström for their help, suggestions and valuable discussions.

A very special hats-off thanks to Doctor Mats Gökstorp for all discussions, pep-talk and the offer to proof read large parts of this thesis (of course, I read parts of his so what would you expect...). Special thanks also to Michael Hall for innumerous VIP discussions, especially during the summer of '95 when we both were writing VIP theses.

Thanks to Per Ingelhag, Christer Jansson, Anders Edman and Jan-Erik Eklund at the department of physics, IFM, for help with the VLSI architecture issues, both within the sheet-of-light and the VIP projects.

Thanks to Erik Åstrand who devised several of the algorithms presented in chapter 7, and has also given invaluable input to many other smart MAPP algorithms.

Without the initiative and help of Ragnar Arvidsson, Tor Ehlersson, Bo Lyckegård, Ulf Näsström, and several others at EMW there would never have been any radar VIP architectures, and no second part of this thesis, thank you very much.

I must also thank everybody at IVP for giving me the opportunity to take my research into the "real world" when implementing the RANGER system.

Thanks to everybody who has contributed to the Elauto atmosphere through the years, especially to the "Acats" Mats G., Magnus H., and Mikael W., and to the system administrators and secretaries.

The financial support of CENIIT's (Center for industrial information technique) VLSI for vision project and NUTEK's (Swedish National Board for Technical Development) grant No. 9303459 are gratefully acknowledged.

Last but not least, thanks to Myself, I couldn't have done this without Me.

Contents

About this thesis

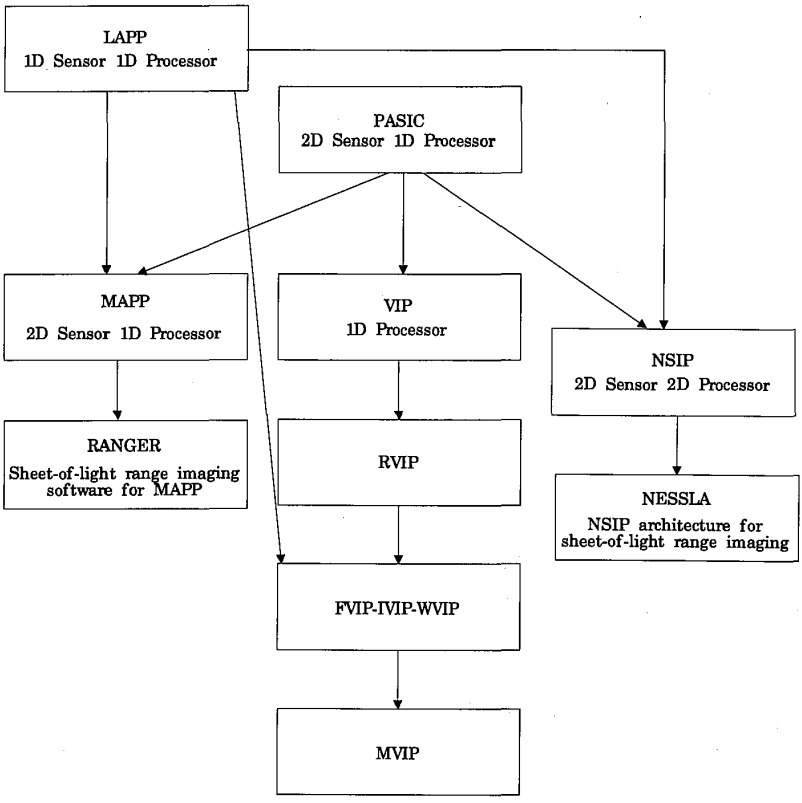
Background

This thesis describes work in two seemingly very disparate areas, triangulation-based range imaging using sheet-of-light illumination, and bit-serial Single Instruction stream Multiple Data stream, SIMD, architectures aimed at Doppler radar signal processing. However, both these projects share the same origins in the SIMD-architecture/Smart-Sensor research at Linköping University [5]. All architectures described in this thesis evolved at least partly from the PASIC project which started in the late eighties at Linköping University [2], [4], [29], see Figure 1. PASIC, which stands for Processor, A/D-converter and Sensor Integrated Circuit, combined a fully addressable 128x128 photo diode sensor with a linear array of A/D converters and bit-serial Processing Elements, PEs, on a single chip. PASIC and LAPP1100, Linear Array Picture Processor, [12] together evolved into the MAPP2200, Matrix Array Picture Processor, [9], [32], a commercial smart sensor from Integrated Vision Products, IVP, in Linköping. MAPP2200 contains a 256x256 photo diode array and a linear array of 256 A/D converters and PEs on a single chip.

A novel imaging and signal processing idea called Near-Sensor Image Processing, originally implemented in the LAPP architecture, then led to the evolution of the 2D NSIP architecture [10], [11], [30], [33]. In the NSIP architecture the sensor and processing element is combined into a compact Sensor-Processor Element, SPE, and the SPEs are arranged in a 2D architecture. The NSIP architecture is capable of general low-level image processing but a simplified version of the NSIP architecture called NESSLA [16], [23] was developed for sheet-of-light range imaging applications. This architecture, together with several sheet-of-light range imaging algorithm implementations on the MAPP2200 sensor [16], [17], [18], [22], [27], [28] are presented in **Part I** of this thesis. We also cover optical design, calibration and general signal processing aspects of sheet-of-light range imaging as well as comparisons with other smart sensor architectures aimed at this particular application. This is an reworked edition of my licentiate thesis [16] together with work on novel MAPP2200 algorithms [18], [27], [28] and calibration [19]. Three of the MAPP2200 algorithms and the calibration routines have been implemented in a sheet-of-light range imaging software package called RANGER commercially available from the MAPP2200 manufacturer IVP since mid 1994 [26]. The NSIP work in [16] have been updated to include experimental results from two general NSIP chips that have been designed and tested to verify the concept of NSIP as a good sheet-of-light range imaging architecture [7], [8].

During the work with PASIC, its processor part was discovered to be very powerful on its own. This resulted in a processor design called VIP, Video Image Processor [3], [6], [31]. VIP is a 1-Dimensional bit-serial SIMD architecture. The combination of many small but powerful Processing Elements, PEs, on a single chip gives a very effective row-parallel image processing computer for demanding low-level real-time applications.

FIGURE 1.



The PASIC-VIP family tree.

The VIP design also proved to be effective for Doppler radar signal processing [15], and **Part II** of this thesis describes the Radar-VIP, RVIP, architecture aimed at ground-based surveillance radar signal processing [1], [25] and its successor for airborne applications, the Flight-VIP, FVIP, [24], [35]. We also present ideas for the Matrix-VIP, MVIP, architecture aimed at future airborne space-time adaptive phased-array processing.

The RVIP chip has been implemented by Ericsson Microwave systems, EMW, (formerly Ericsson Radar Electronics) in Mölndal for use in ground based surveillance radar applications. Two other VIP architectures, the Wood-VIP (WVIP) [13] and the Infra-red-VIP (IVIP) [34], [21] have also been developed but are not presented in depth here. For more information on the IVIP and WVIP architectures we refer to [14]. The FVIP, WVIP and IVIP designs have been developed concurrently and are very similar. We also present implementations of Fast Fourier Transforms, FFTs [20], on VIP arrays since this is an important part of the Doppler radar signal processing.

Thesis overview

Part I describes sheet-of-light range imaging, and the goal is fast imaging. Using a standard video sensor gives us very slow imaging speed, and the work focuses on effective peak detection algorithms and their smart sensor implementations. After the introduction about different active range imaging techniques in *chapter 1,* I present a thorough investigation of the optical design of a sheet-of-light system in *chapter 2.* I derive equations mapping from sensor to world coordinates and describe how to apply the Scheimpflug condition so that the whole light-sheet can be viewed in perfect focus.

In *chapter 3* I present a novel sheet-of-light camera calibration scheme based on approximations of the triangulation equations derived in chapter 2.

Different suitable peak detection algorithms are studied in *chapter 4,* and I present simulations of the accuracy of the algorithms with and without signal degradation by noise.

An overview of different sensor designs aimed at sheet-of-light range imaging are given in *chapter 5.*

In *chapter 6* I present the novel NESSLA sensor architecture. NESSLA is a special purpose NSIP-type architecture with a 2D array of Sensor-Processor Elements and it is capable of very fast sheet-of-light range imaging. Experimental results obtained using a general 16x16 SPE NSIP chip are given to verify the NESSLA concept.

Chapter 7 presents implementations of different peak detection algorithms on the MAPP2200 smart sensor, and I describe how to achieve fast and accurate imaging. The algorithms in sections 7.10-7.13 were mostly devised by E. Åstrand.

I have implemented the presented calibration routine and the algorithms in sections 7.3-7.6 in the RANGER sheet-of-light range imaging software package available from the MAPP2200 manufacturer IVP.

Part I ends with *chapter 8* which is a short discussion of the presented smart sensor architectures.

Part II of this thesis describes the evolution of different VIP architectures for Doppler radar signal processing applications. Much of this work was performed in a large group of people from Ericsson Microwave systems, the VLSI group at Linköping University, and from the Image Processing Laboratory at Linköping University, where I work.

Chapter 1 gives an overview of the history of the VIP architecture and describes two similar linear array SIMD architectures.

Chapter 2 describes a few of the principles and the signal processing algorithms of pulsed Doppler radars.

Chapter 3 describes an LPD algorithm and how this algorithm led to the evolution of the RVIP architecture. The RVIP PE redesign mainly affected two parts of the PE, the IO-register and the Multiplier-Accumulator. I was involved to a high degree in the evolution of the Multiplier-Accumulator design, where I studied several different designs to find the best trade-off between high performance and small area. The combination of the parallel coefficient load port and the dual-ported accumulator increased the PE performance for multiply-accumulate operations three to four times compared with the original VIP PE design. We also present an overview of the system design with several chips and a local micro controller packaged together in a 512 PE system. For this part I worked on the micro controller design.

PAGE 13 OF 204

Chapter 4 then presents an MPD algorithm and how this led to the evolution of the FVIP architecture. There was very little architecture development in FVIP and my work was concentrated on the problem of mapping the 2D input image on a linear array architecture. A complication was that the image size varied continuous between three different aspect ratios, and that the architecture and algorithm mapping had to be able to handle all three formats.

I also studied how 1D FFTs, which are a major part of the MPD algorithm, could be implemented efficiently on FVIP, and this work is presented in *chapter 5*. One of the main problems with the FFT implementation was that the organization of the samples within the array made it impossible to neglect the bit-reverse sorting and butterfly-scheme communication problems.

In *chapter 6* I present ongoing work on how to map a space-time adaptive processing (STAP) algorithm on a new VIP architecture called MVIP. Here I have devised a possible mapping of the 3D data matrix and algorithm on an MVIP array. To solve the difficult communication problems the linear array has been augmented with limited 2D communication capacity. A new floating-point version of the MVIP PE is also presented.

Part II ends with *chapter 7* which compares the VIP performance with two other architectures, and describes possible future work within the VIP project. There is also a short list of abbreviations used in Part II.

Publications

The work presented in this thesis has previously appeared, in different form, in these publications

[i] Johannesson M., Åström A., Danielsson P.E., *An Image Sensor for Sheet-of-light Range Imaging,* Proc. of MVA '92, Tokyo, 1992.

[ii] Johannesson M., Åström A., *Sheet-of-light range imaging with MAPP2200,* Proc. of SCIA '93, Tromsö, 1993.

[iii] Johannesson M., Åström A., Ingelhag P., *The RVIP Image Processing Array,* Proc. of CAMP '93, New Orleans, 1993.

[iv] Johannesson M., *Fast, programmable, sheet-of-light range finder using MAPP2200,* Proc. of San Diego SPIE '94, San Diego, 1994.

[v] Åstrand E., Johannesson M., Åström A., *Two novel methods for range imaging using the MAPP2200 smart sensor,* Proc. of Machine Vision Applications in Industrial Inspection, SPIE '95, San José, 1995.

[vi] Åström A., Johannesson M., Edman A., Ehlersson T., Näsström U., Lyckegård B., *An Implementation Study of Airborne Medium PRF Doppler Radar Signal Processing on a Massively Parallel SIMD processor architecture,* Proc. of RADAR '95, Washington 1995.

[vii] Johannesson M., *Calibration of a MAPP2200 sheet-of-light range camera,* Proc of SCIA '95, Uppsala, 1995.

[viii] Johannesson M., *FFTs on a linear SIMD array,* Proc. of EURO-PAR'95, Lecture Notes on Computer Science 966, Springer Verlag, 1995.

Related, but not direct included, publications are

[ix] Johannesson M., *Can Sorting Using Sheet-of-Light Range Imaging and MAPP2200,* Proc. of IEEE/SMC '93, Le Touquet, 1993.

[x] Johannesson M., Gökstorp M., *Video-rate pyramid optical flow computations on the linear SIMD array IVIP,* Proc. of CAMP '95, Como, 1995.

PAGE 14 OF 204

References

[1] Arvidsson R., *Massively Parallel SIMD Processor for Search Radar Signal Processing*, Proc. of RADAR '94, Paris, 1994.

[2] Chen K., Danielsson P.E., Åström A., *PASIC a Sensor/Processor Array for Computer Vision*, Proc. of Application Specific Array Processors, Princeton, 1990.

[3] Chen K., Svensson C., *A 512-processor array chip for video/image processing*, Proc. of "From Pixels to Features II", Bonas, 1990.

[4] Chen K., Åström A., Danielsson P.E., *PASIC a Smart Sensor for Computer Vision*, Proc. of Pattern Recognition, Atlantic City, 1990.

[5] Danielsson P.E., *Smart Sensors for Computer Vision*, Report No. LiTH-ISY-I-1096, Linköping, 1990.

[6] Danielsson P.E., Emanuelsson P., Chen K., Ingelhag P., Svensson C., *Single-Chip High-Speed Computation of Optical Flow*, Proc. of MVA '90, Tokyo, 1990.

[7] Eklund J-E., Svensson C., Åström A., *Near-Sensor Image Processing, a VLSI Realization*, Proc of Transducers95, Eurosensors IX, Stockholm, 1995.

[8] Eklund J-E., Svensson C., Åström A., *Near-Sensor Image Processing, a VLSI Realization*, Proc of 8th Annual IEEE Int. ASIC Conference., Austin, 1995.

[9] Forchheimer R., Ingelhag P., Jansson C., *MAPP2200, a second generation smart optical sensor*, SPIE, vol 1659, 1992.

[10] Forchheimer R., Åström A., *Near-Sensor Image Processing. A new Approach to low level image processing*, Proc. of ICIP, Singapore, 1992.

[11] Forchheimer R., Åström A., *Near-Sensor Image Processing. A New Paradigm*, In IEEE trans. on Image Processing, november 1994.

[12] Forchheimer R., Ödmark A., *Single chip linear array processor*, in Applications of digital image processing, SPIE, vol 397, 1983.

[13] Hall M., Åström A., *High Speed wood inspection using a parallel VLSI architecture*, Proc. of 3rd Int. Workshop on Algorithms and Parallel VLSI Architectures, Leuven, 1994.

[14] Hall M., *A study of Parallel Image Processing on the VIP Architecture*, Lic Thesis, LiTH-ISY-LIC-1995:41, Linköping, september 1995.

[15] Ingelhag P., Åström A., Chen K., Svensson C., Ehlersson T., *RADAR Signal Processing Using a 512-Processor Array Chip*, Proc. of ICDP91, Florence, 1991.

[16] Johannesson M., *Sheet-of-light Range Imaging*, Lic thesis No 404, Linköping, 1993.

[17] Johannesson M., *Can Sorting Using Sheet-of-Light Range Imaging and MAPP2200*, Proc. of IEEE/SMC '93, Le Touquet, 1993.

[18] Johannesson M., *Fast, programmable, sheet-of-light range finder using MAPP2200*, Proc. of San Diego SPIE '94, San Diego, 1994.

[19] Johannesson M., *Calibration of a MAPP2200 sheet-of-light range camera*, Proc of SCIA '95, Uppsala, 1995.

[20] Johannesson M., *FFTs on a linear SIMD array*, Proc of EURO-PAR'95, LNCS 966, Springer Verlag, 1995.

[21] Johannesson M., Gökstorp M., *Video-rate pyramid optical flow computations on the linear SIMD*

array IVIP, To appear in Proc of CAMP '95, Como, 1995.

[22] Johannesson M., Åström A., *Sheet-of-light range imaging with MAPP2200*, Proc. of SCIA '93, Tromsö, 1993.

[23] Johannesson M., Åström A., Danielsson P.E., *An Image Sensor for Sheet-of-light Range Imaging*, Proc. of MVA '92, Tokyo, 1992.

[24] Johannesson M., Åström A., Edman A., *FVIP, An implementation study of an MPD radar algorithm*, Report LiTH-ISY-R-1698, Linköping, 1994.

[25] Johannesson M., Åström A., Ingelhag P., *The RVIP Image Processing Array*, Proc. of CAMP '93, New Orleans, 1993.

[26] *RANGER RS2200 Product Information*, IVP, Linköping, 1994.

[27] Åstrand E., Johannesson M., Åström A., *Five contributions to the art of sheet-of-light range imaging*, Report No. LiTH-R-ISY-1611, Linköping, 1994.

[28] Åstrand E., Johannesson M., Åström A., *Two novel methods for range imaging using the MAPP2200 smart sensor*, Proc. of Machine Vision Applications in Industrial Inspection, SPIE '95, San José, 1995.

[29] Åström A., *A Smart Image Sensor. Description and Evaluation of PASIC.*, Lic. Thesis. LIU-TEK-LIC-1990:57, Linköping 1990.

[30] Åström A, *Smart Image Sensors*, Ph.D. thesis No. 319, Linköping, 1993.

[31] Åström A., Danielsson P.E., Chen K., Ingelhag P., Svensson C., *Videorate signal processing with PASIC and VIP*, Proc. of 2nd Specialist Sem. on design and Appl. of Parallel Digital Processors, Lissabon, 1991.

[32] Åström A., Forchheimer R., *MAPP2200 Smart Vision Sensor: Programmability and Adaptivity*, Proc. of MVA '92, Tokyo, 1992.

[33] Åström A., Forchheimer R., Ingelhag P., *An Integrated Sensor/Processor Architecture Based on Near-Sensor Image Processing*, Proc. of CAMP '93, New Orleans, 1993.

[34] Åström A., Isaksson F., *Real-time Geometric Distortion Correction and Image Processing on the 1D SIMD architecture IVIP*, Proc of MVA '94, Kawasaki 1994.

[35] Åström A., Johannesson M., Edman A., Ehlersson T., Näsström U., Lyckegård B., *An Implementation Study of Airborne Medium PRF Doppler Radar Signal Processing on a Massively Parallel SIMD processor architecture*, Proc of RADAR '95, Washington 1995.

Part I

Sheet-of-light Range Imaging

Chapter 1

Introduction

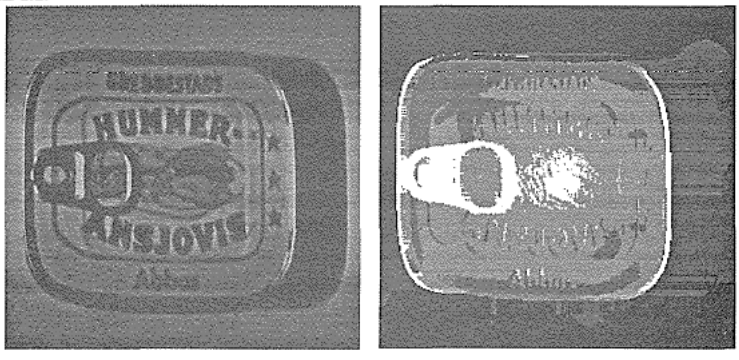
1.1 Range Imaging

Normally, the image from a camera depicts the intensity distribution of the viewed scene as in the left image in Figure 1.1. A range camera on the other hand, depicts the distance from the camera to the objects of the scene as in the right image in Figure 1.1. In this range image the printed text is no longer visible, while the can opener and edges show up because of their difference in height. Interestingly, range and intensity images may also be combined as in Figure 1.2.

Range images are used in many applications, e.g. road surface surveying [9], industrial inspection [4], [50], [45], [65], and industrial robots [63]. In road surface surveys the goal is to measure cracks and ruts in the road surface. In industrial inspection the goals are often to determine if fabricated parts are within the tolerated sizes and/or orientations. In industrial robot applications we need to find locations and sizes of objects for navigation and/or object manipulation purposes.

Range images are obtained in many different ways. Traditionally these are separated into two categories, active and passive range imaging respectively. In active range imaging a dedicated and well defined light source is used in cooperation with the sensor. In passive range imaging no special light (except ambient) is required. The most common passive method is stereo vision where at least two images from different positions are used in a triangulation scheme. More detailed descriptions of different passive range imaging methods can be found in for instance [24], [1] and [22].

FIGURE 1.1.



An intensity and a range image of a fish can. Dark areas are far away from the camera, and light areas close.

FIGURE 1.2.

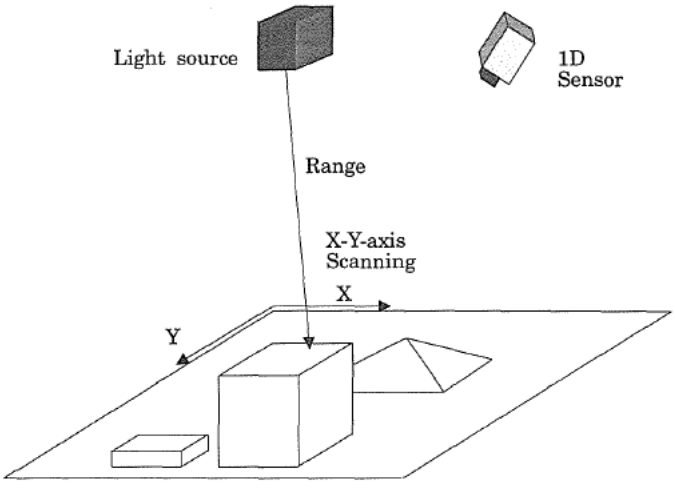
A combined range and intensity image.

1.2 Active range imaging

Many active range imaging techniques use a *triangulation* scheme where the scene is illuminated from one direction and viewed from another. The illumination angle, the viewing angle, and the baseline between the illuminator and the viewer (sensor) are the triangulation parameters. The most common active triangulation methods include illumination with a *single spot, a sheet-of-light* and *gray coded light*. A thorough discussion on the different methodologies can be found in for instance [64], [6] and [47]. The most common non-triangulation method is time-of-flight (radar), where the time for the emitted light pulse to return from the scene is measured. The high speed of electromagnetic waves makes time-of-flight methods difficult to use for high accuracy range imaging since small differences in range have to be resolved by extremely fine discrimination in time.

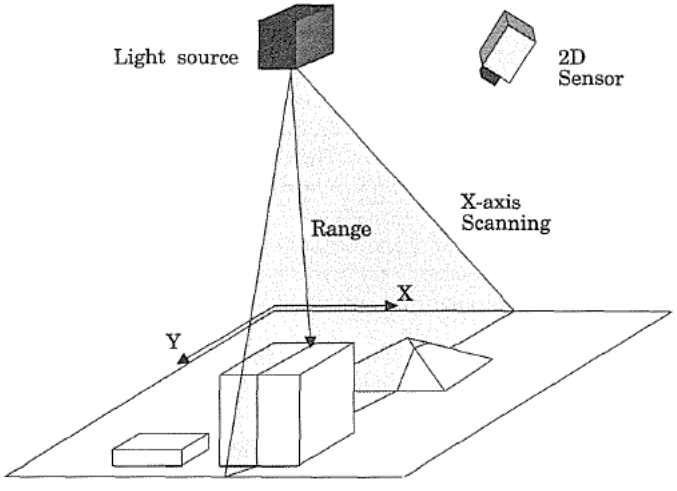
In single spot range imaging a single light ray is scanned over the scene and one range datum (rangel) is acquired for each sensor integration and position of the light, see Figure 1.3. Thus, in order to obtain an *MxN* image *MxN* measurements and sensor integrations have to be made. In sheet-of-light range imaging a sheet (or strip) of light is scanned over the scene and one row with *M* rangels is acquired at each light position and sensor integration, see Figure 1.4. In this case only *N* measurements and integrations are needed for an *MxN* image. Finally in a gray-coded range camera the scene is illuminated with *logN* gray coded patterns and therefore only *logN* integrations are needed to form one *MxN* image, see Figure 1.5.

FIGURE 1.3.



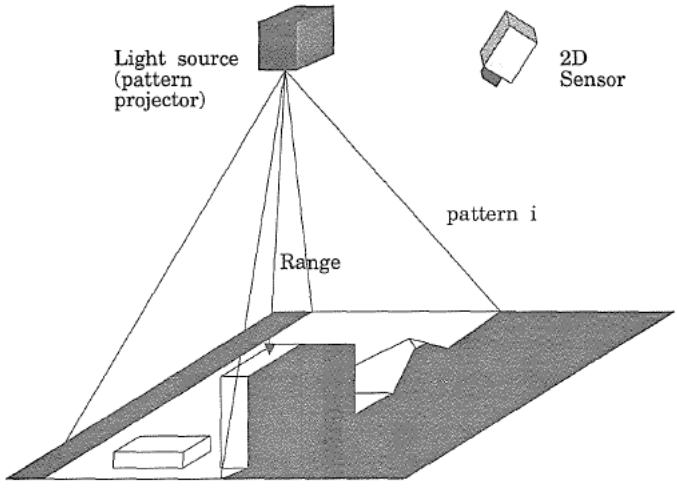
Single spot range imaging. Note that the range can also be defined as the distance between the sensor and the illuminated object instead of the distance between the light source and the object.

FIGURE 1.4.



Sheet-of-light range imaging.

FIGURE 1.5.



Range imaging with gray coded light.

It would seem that the gray coded light method has speed advantages over the other active triangulation methods since fewer sensor integrations are needed. However, one problem seems to be how to make a high intensity projector that can switch between patterns as fast as the sensor can integrate images. Hence, the trouble spot is the design of the illuminator rather than the sensor. Furthermore, any object movements between the integrations can give errors in the range image since the triangulation is performed *after* the *logN* integrations under the assumption that the illuminated scene was static. Admittingly, the errors are minimized and not catastrophic due to the properties of the Gray code. In the other methods, however, each single sensor integration gives one set of range values. Therefore any object movement between integrations only gives a time dependent range image, where each range datum was correct when it was acquired.

In gray-coded light systems ordinary tungsten or halogen lamps are predominant instead of the laser light predominant in the other methods. We note that tungsten and halogen lamps are less harmful for the eyes than laser light and therefore better suited for measurements of human body parts.

The single spot technique requires advanced mechanics to allow the spot to reach the whole scene. At least two synchronous scanning mirrors are required. The advantage is that a relatively simple linear sensor, e.g. of the Position Sensitive Device type, can be used. But, the use of only one single sensing element often reduces the acquisition speed; the system presented by Rioux [49] reaches a rangel (range pixel) speed of 10 KHz. However, a recent system by the same research group actually outputs range data at RS-170 compatible video rate [5].

In the case of sheet-of-light systems, the projection of the light can be made with one single scanning mirror which is considerably simpler than the projector design for gray coded light, or the two mirror arrangement for single spot. Actually, in the sheet-of-light systems presented

in this thesis the sheet-of-light is not swept at all. Instead the apparatus itself or the scene is moving. Thus, the camera system itself has no moving parts. In principle, however, the sensor designs to be presented are just as applicable to sweeping sheet-of-light.

Finally, it should be noted that a moving scene or moving apparatus rule out the gray-coded methods since they require a static scene during the *logN* integrations.

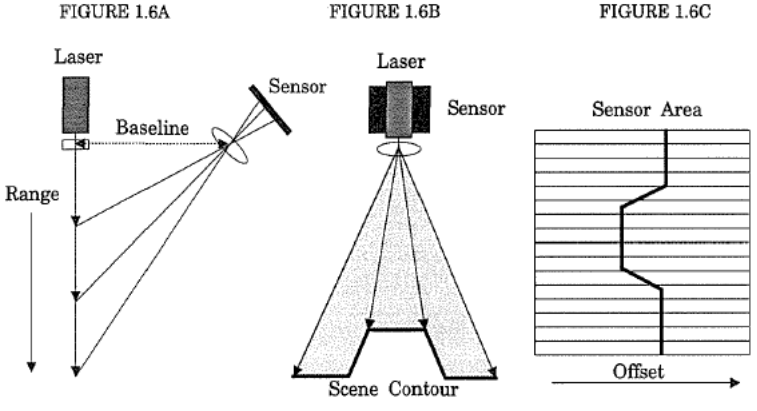
1.3 Sheet-of-light range imaging

In a sheet-of-light system range data is acquired with triangulation. The offset position of the reflected light on the sensor plane depends on the distance (range) from the light source to the object, see Figure 1.6. Using trigonometry we can solve the equations for the range if we know the distance between the laser and the optical center of the sensor (the *baseline*) and the direction of the transmitted ray. The third triangulation parameter, the direction of the incoming light ray, is given by the sensor offset position.

For each position of the sheet-of-light the depth variation of the scene creates a contour which is projected onto the sensor plane, see Figure 1.6B-C. If we extract the position of the incoming light for each sensor row we obtain an offset data vector that serves as input for the triangulations.

To make a sheet-of-light the pencil sharp laser spot-light passes through a cylindrical lens, see Figure 1.6A-B. The cylindrical lens spreads the light into a sheet in one dimension while it is unaffected in the other. A sheet-of-light can also be made using a fast scanning mirror mechanism [4], an array of LED's [45], or with a slit projector [45].

FIGURE 1.6.



A range camera using sheet-of-light. Figure 1.6A shows the illumination with the sheet-of-light perpendicular, Figure 1.6B parallel to the plane of the paper. Figure 1.6C is a snapshot of the sensor area.

As mentioned, two-dimensional range images can be obtained in at least three ways: by moving the apparatus over a static scene as in a road surface survey vehicle, by moving the scene as a conveyor belt, or by sweeping the sheet-of-light over a static 3D-scene using a mirror arrangement. In the two first cases the distance can be computed from the offset position in

each row using a simple range equation or a precomputed lookup table. The third case is more awkward since the direction of the emitted sheet-of-light changes so that a second triangulation parameter becomes involved for each light sheet position.

In any case, for each illumination position and sensor integration, in the first processing step, the output from the 2D image sensor should be reduced to a 1D array of offset values.

Many commercial sheet-of-light range imaging systems are based on video-standard sensors. Such a sensor has a preset frame-rate, which in the PAL standard is specified as 50 Hz. This limits the range profile frequency to 50 Hz assuming that the processing can be made in real-time. A 256x256 range image is then obtained in 5 seconds, and the range pixel frequency is 12.8 KHz. Even if the frame rate could be increased the sensor output is still serial which would require very high output and processing clock frequencies and possibly increase the noise level.

In [6] Besl reports on a few commercial sheet-of-light range imaging systems with CCD video-type sensors, but none of these have higher rangel frequencies than 5 KHz. As will be shown below, combining sensing and processing on the same chip implies that shorter integration times and considerably faster range imaging can be obtained.

1.4 Smart sensors

Part I of this thesis deals mainly with smart optical sensors and some initial comments on the concept seem appropriate. A smart sensor is a design in which data reduction and/or processing is performed on the sensor chip. Sheet-of-light range imaging is well suited for smart sensor architectures since each row of sensor data shall be reduced to one offset position value for the incoming light sheet.

In Linköping the first smart sensor was LAPP1100 [19], conceived in the early eighties. LAPP1100 combines a linear photo diode array with a linear array of bit-serial processing elements. It was followed by the PASIC design [68] in the late eighties. PASIC was a general programmable smart sensor, consisting of a 2D sensor array combined with a 1D array of A/D converters and processing elements. No PASIC chip was built, but the ideas from this design, combined with the LAPP1100 architecture became the MAPP2200 sensor [40], [17] now being manufactured by IVP, Integrated Vision Products, in Linköping.

Chapter 2

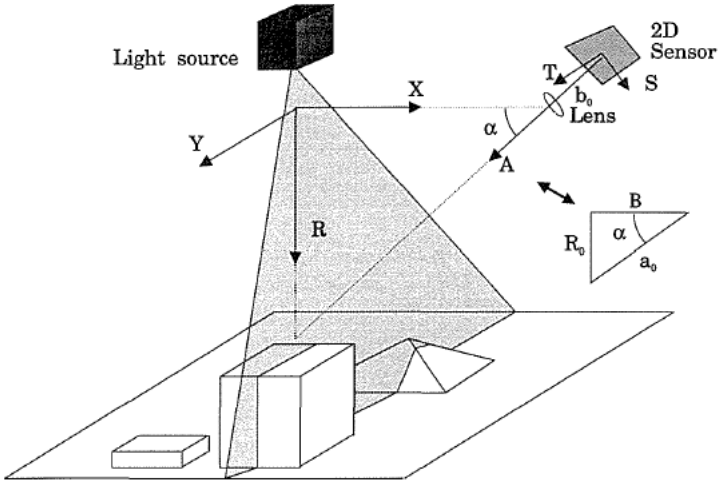
Optical design for sheet-of-light

2.1 Geometry definitions

In Figure 2.1 we show the coordinate system to be used in the sequel. The world coordinate system is *(X,Y,R)* and the sensor coordinate system is *(S,T,A)*. The transformation between the coordinate systems can be described by two translations and one rotation (Translation *B* along the *X*-axis, rotation 90-α around the *Y*-axis and translation -b_0 along the *A*-axis). As will be shown later, there might also be a skew of the *(S,T,A)* system so that *A* is non-orthogonal to the *(S,T)* plane.

Since the *Y* and *T* axes are parallel, range measurements along the *R*-axis are independent of the sensor *t*-coordinate. This reduces all geometric range discussions to 2D, and we can view the system as in Figure 2.2 (for geometry definitions see Table 2.1). Here we only consider a non-scanning light-sheet and therefore the illumination angle γ is set to 90° unless otherwise is stated. This simplifies the range expressions somewhat. Also, the laser axis is considered vertical unless stated otherwise. A discussion on different scanning techniques can be found in [31].

FIGURE 2.1.



A Sheet-of-light system with world and sensor coordinate system definitions.

TABLE 2.1.

Word	Short	Definition
Optical center	OC	Position of the center of the sensor system lens.
Optical axis	OA	The *A*-axis, passing through the optical center.
Laser axis	LA	Optical axis of the laser system (the *R*-axis).
Laser origin	LO	Position along the laser axis closest to the optical center. Origin of world coordinate system.
Baseline	B	Distance between the optical center and the laser origin.
View angle	α	Angle between the optical axis and the baseline. Also the angle between the laser sheet and the normal of the optical axis if $\gamma = 90°$.
Illumination angle	γ	Angle between the baseline and the laser sheet. 90°unless deflected by a mirror.
Sensor angle	β	Angle between the normal of the optical axis and the sensor *s* axis.
Sensor position	s,t	Sensor coordinates along rows and columns. Origin at optical axis
Focal length	f	Focal length of the lens.
Lens aperture	d	Effective diameter of the lens.
Sensor distance	b_0	Distance along the optical axis between the optical center and the sensor.
Focus distance	a_0	Distance along the optical axis between the optical center and the sheet-of-light.
Range	r(s)	Range (distance), parallel with the range axis, from LO to a point in the laser sheet from where a ray emanates and hits the detector at *s*. $r(0) = R_0$ is the range of ray going in the optical axis.
Sensor height	h_0	Height between the baseline-axis and the sensor center column position (s = 0)
Sensor baseline	b_{10}	The base in the triangle made by the optical axis, the sensor height and the baseline-axis.
Pixel pitch	Δx	Inter-pixel distance on the sensor.
Sensor row size	N	The number of pixels in a sensor row.
Light angle	ρ	Angle between a light ray and the optical axis
Max angle	ρ_m	Maximum angular deviation from the optical axis.
Range interval	R_T	Total range covered by light rays offset with $\|\rho\| < \rho_m$ from the optical axis.

FIGURE 2.2.

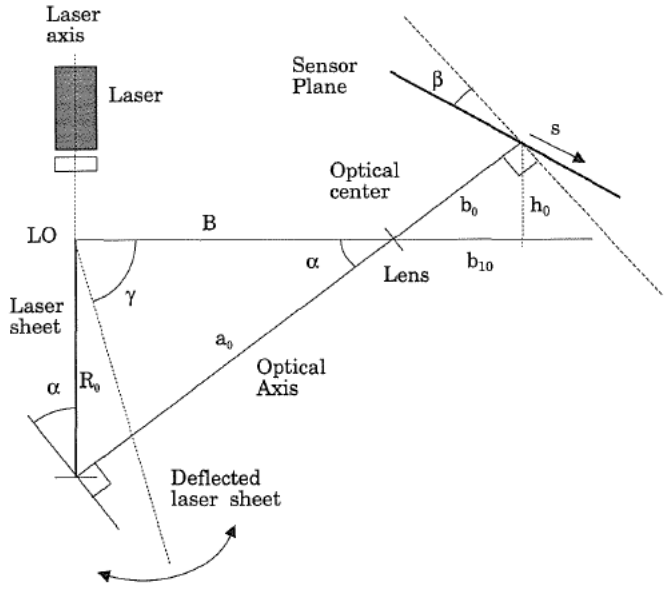


Illustration to the definitions made in Table 2.1.

Numerical examples are used in the sequel to give a better intuitive understanding of the problems and the derived equations. We consider two systems with equal view angles and approximately equal range interval but with different optics. One system utilize a "normal" lens with $f = 18$ mm, the other one a "zoom" lens with $f = 75$ mm. We also consider two apertures for each lens. The light sensitivity of a lens is expressed by the f-stop (relative aperture) f/d. The used f-stops are 5.4 and 1.8 which give the lens apertures 3.3 and 10 mm for the 18 mm lens and 14 and 42 mm for the 75 mm lens. Typical parameters are

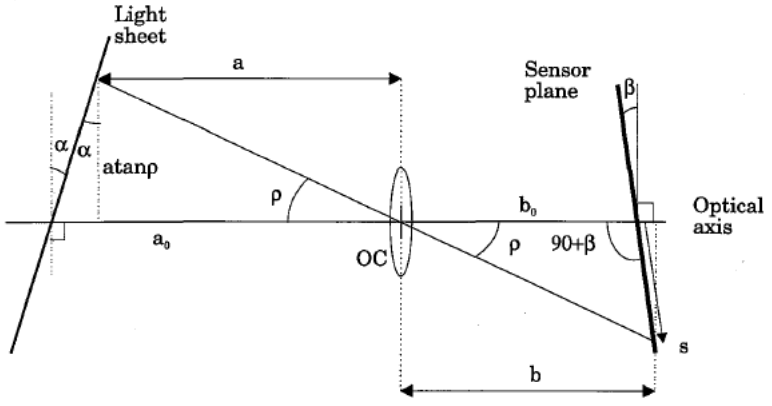
View angle	α	45°
Range interval	R_T	500 mm
Sensor row size	N	256
Pixel pitch	Δx	32 μm
Focal length	f_1	18 mm
Lens aperture	d_A	3.33 mm
Lens aperture	d_B	10.0 mm
Base line	B_1	520 mm
Focal length	f_2	75 mm
Lens aperture	d_A	13.9 mm
Lens aperture	d_B	41.67 mm
Base line	B_2	2282 mm

2.2 The Scheimpflug condition

In Figure 2.3 we see the sheet-of-light geometry for arbitrary laser and sensor angles, and for an arbitrary incoming light ray. The relation between the focal length and the sensor and focus distances is dictated by the lens law

$$\frac{1}{f} = \frac{1}{a_0} + \frac{1}{b_0} \tag{2.1}$$

FIGURE 2.3.



A setup with arbitrary sensor and laser angles and an incoming light ray at an angle ρ.

For an arbitrary light ray making an angle ρ with the optical axis and hitting the sensor at s, we can find the corresponding orthogonal distances a and b from the optical center to the laser sheet and the sensor. The equations are

$$a = \frac{a_0}{(1 + \tan\rho\tan\alpha)} \tag{2.2}$$

$$b = \frac{b_0}{(1 - \tan\rho\tan\beta)} \tag{2.3}$$

If we want the ray going to s to be focused on the sensor plane, a and b must satisfy the lens law (2.1). From (2.2) and (2.3) we get

$$\frac{1}{a} + \frac{1}{b} = \frac{1}{a_0}(1 + \tan\rho\tan\alpha) + \frac{1}{b_0}(1 - \tan\rho\tan\beta) \tag{2.4}$$

$$= \frac{1}{f} + \tan\rho\left(\frac{\tan\alpha}{a_0} - \frac{\tan\beta}{b_0}\right) \tag{2.5}$$

We see that the lens law is satisfied for arbitrary angles ρ if

$$\frac{\tan\alpha}{a_0} = \frac{\tan\beta}{b_0} \tag{2.6}$$

This is satisfied in the special case

$$\alpha = \beta = 0 \tag{2.7}$$

and in general for

$$\tan\beta = \frac{b_0 \tan\alpha}{a_0} \tag{2.8}$$

The condition in (2.7) implies that the sensor is orthogonal to the optical axis and that the optical axis is orthogonal to the laser axis. The condition in (2.8) is known as the *Scheimpflug condition* [6], and defines the tilt of the sensor plane to achieve focus over the whole sensor plane on the sheet-of-light when the optical axis is not orthogonal to either plane. In normal cameras the sensor angle is fixed and orthogonal to the optical axis, which means that only the special condition in (2.7) can be satisfied.

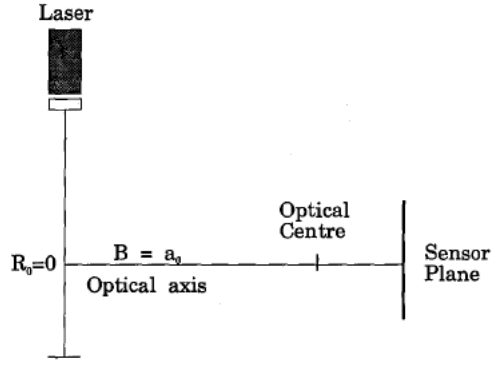
If we have a setup which satisfies the Scheimpflug condition in (2.8), how large is the angle β? Numerical examples are given in Table 2.2. As expected we find that β is very small for view angles below say 70°. Notice also that the angle is almost independent of the lens parameters for a given range interval and view angle.

TABLE 2.2.

f	b_0	a	b
18 mm	18.4 mm	45°	1.43°
75 mm	76.8 mm	45°	1.36°
18 mm	18.4 mm	63°	2.81°
75 mm	76.8 mm	63°	2.67°
18 mm	18.4 mm	85°	15.98°
75 mm	76.8 mm	85°	15.21°

As said before, in normal cameras the sensor plane is orthogonal to the optical axis and if we focus the camera on the light plane as in Figure 2.4 the focus condition in (2.7) is satisfied. Unfortunately this arrangement is not very attractive. If we assume that the objects in the scene are planar and parallel with the optical axis, then when an object is higher than the range R_0 the laser light will by all likelihood be occluded, and no light will reach the sensor.

FIGURE 2.4.

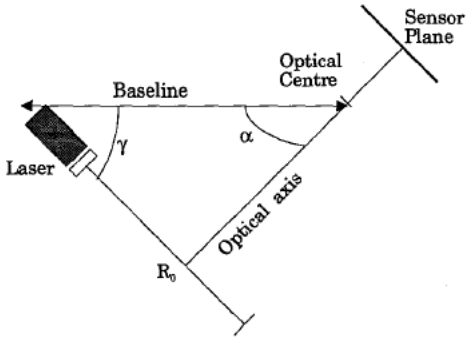


A system with the sensor aligned with the optical axis orthogonal to the light plane.

However, there might be applications where the surface orientations of the objects have little variation. The system in Figure 2.4 should then be arranged so that the object surface normal approximately bisects the 90° angle between the laser sheet and the optical axis. For a case where the 3D scene moves horizontally the setup should be tilted 45° as shown in Figure 2.5.

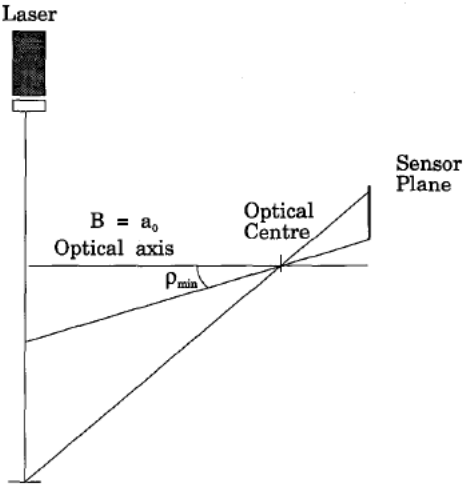
If we use the setup in Figure 2.4 but move the sensor so that the whole sensor is above the optical axis we still satisfy (2.7). We then get the setup in Figure 2.6. However, normal lenses attenuate the intensity across the image plane with the factor $\cos^4\rho$ [24]. Therefore, in Figure 2.6 where we utilize a shifted interval of the sensor plane off the optical axis, there is a price to pay in the form of uneven sensitivity along the sensor plane. Also, lenses are often less than ideal for light rays far away from the optical axis, which results in distorted images.

FIGURE 2.5.



The system in Figure 2.4 tilted 45°.

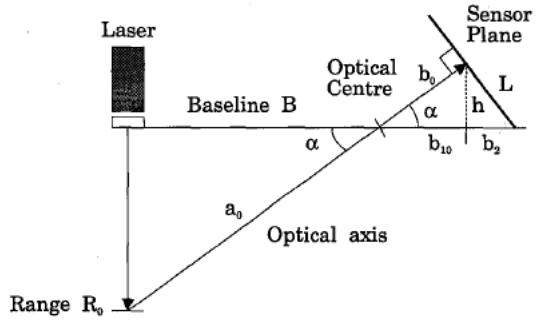
FIGURE 2.6.



A system with the sensor offset from the optical axis.

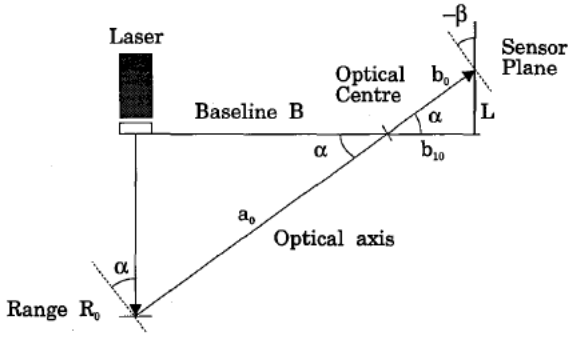
The most common arrangement for sheet-of-light systems is to tilt the camera and its optical axis so that the view angle α is substantially larger than zero, typically in the range of 30-60°, see Figure 2.7. This reduces the problem of occlusion, but the conditions in (2.7) or (2.8) are not satisfied. Furthermore, the linearity found between range r and sensor position s in the previous cases is lost since linearity is only found when the sensor and laser planes are parallel. To regain linearity we can tilt the sensor as in Figure 2.8. Here, the sensor and view angles α and β are equal but with opposite signs. Unfortunately this setup dissatisfies the conditions in (2.7) and (2.8) even more, and therefore gives more unfocused (unsharp) images. If we want a focused design we should use a setup as in Figure 2.7 but instead tilt the sensor according to the Scheimpflug condition (2.8).

FIGURE 2.7.



A system with the sensor aligned with the view-angle approximately 40°.

FIGURE 2.8.



A setup for linearity between the sensor position and the range data.

2.3 Equations for range

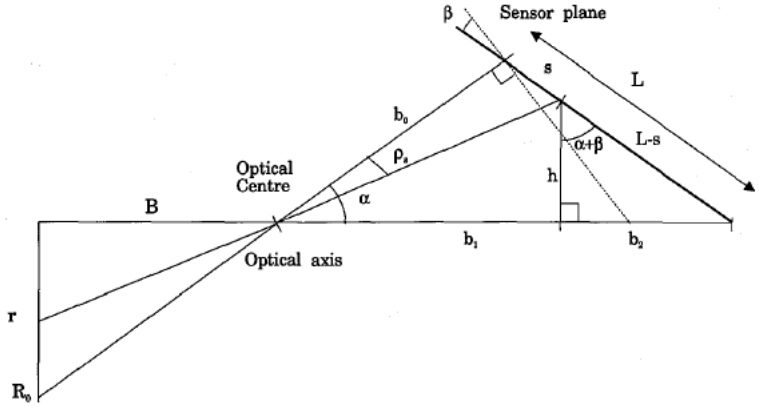
From Figure 2.2 we have

$$R_0 = \frac{Bh_0}{b_{10}} \tag{2.9}$$

For an arbitrary light ray, offset from the optical axis with an angle ρ_s and hitting the sensor at s, we get the geometry in Figure 2.9 where

$$r = \frac{Bh}{b_1} \tag{2.10}$$

FIGURE 2.9.



Geometry for an arbitrary sensor alignment, and an arbitrary input ray.

The geometry for the triangles give

$$b_1 + b_2 = \frac{b_0 \sin(90+\beta)}{\sin(90-(\alpha+\beta))} = \frac{b_0 \cos\beta}{\cos(\alpha+\beta)} \tag{2.11}$$

$$L = \frac{b_0 \sin\alpha}{\sin(90-(\alpha+\beta))} = \frac{b_0 \sin\alpha}{\cos(\alpha+\beta)} \tag{2.12}$$

$$h = (L-s)\cos(\alpha+\beta) \tag{2.13}$$

$$b_2 = (L-s)\sin(\alpha+\beta) \tag{2.14}$$

Using (2.11) - (2.14) in (2.10) yields

$$r = \frac{B\left(\frac{b_0 \sin\alpha}{\cos(\alpha+\beta)} - s\right)\cos(\alpha+\beta)}{\frac{b_0 \cos\beta}{\cos(\alpha+\beta)} - \left(\frac{b_0 \sin\alpha}{\cos(\alpha+\beta)} - s\right)\sin(\alpha+\beta)} \tag{2.15}$$

This rather cumbersome equation holds for all sensor and laser alignments including the Scheimpflug geometry. For the special geometry in Figure 2.4 we have $\alpha = \beta = 0$, which simplifies (2.15) to

$$r = \frac{-Bs}{b_0} \tag{2.16}$$

For the geometry in Figure 2.7 we have $\beta = 0$, which reduces (2.15) to

$$r = \frac{B(b_0 \tan\alpha - s)\cos\alpha}{\frac{b_0}{\cos\alpha} - (b_0 \tan\alpha - s)\sin\alpha} = B\frac{b_0 \tan\alpha - s}{b_0 + s\tan\alpha} \tag{2.17}$$

For the geometry in Figure 2.8 we have $\alpha = -\beta$ which gives the range equation

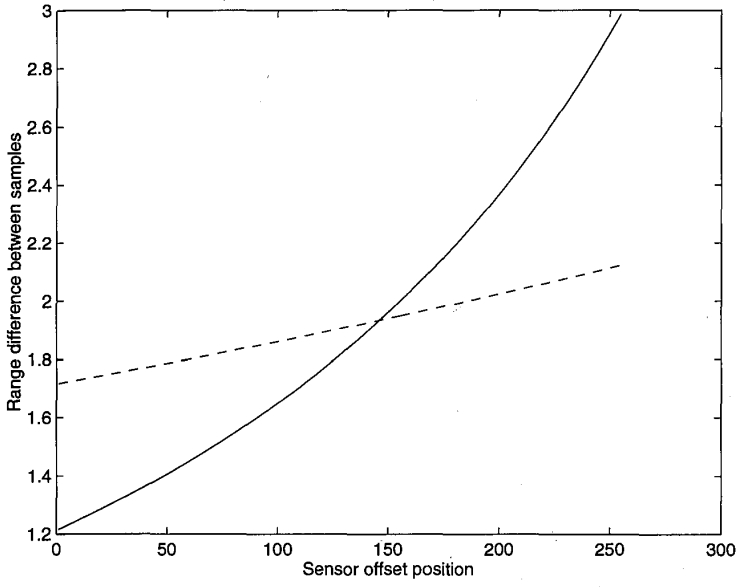
$$r = \frac{B(b_0 \sin\alpha - s)}{b_0 \cos\alpha} \tag{2.18}$$

As mentioned above the most common setup is the one in Figure 2.7, with the non-linear range equation (2.17). The non-linearity means that a constant precision Δs in the sensor position s results in a variable precision Δr

$$\Delta r = \Delta s \frac{-Bb_0}{(b_0 \cos\alpha + s\sin\alpha)^2} \tag{2.19}$$

The system should be designed with this in mind. The function $|\Delta r|$ is plotted in Figure 2.10 for the two cases $f = 75$ mm and $f = 18$ mm and the range interval ~350 mm. Here we see that the difference in Δr is much larger for the smaller f value. This is an indication that a larger f (a zoom lens) gives better linearity.

FIGURE 2.10.



Plot of range increments $|\Delta r|$ as a function of the sensor offset using two different lenses $f_1 = 18$ (solid) and $f_2 = 75$ (dotted). $s = 0$ is sensor offset position 128.

To find the range interval R_T for a given setup we insert the extreme values of s (+/- $N\Delta x/2$) into equation (2.17) and after simplification we obtain

$$R_T = \frac{4Bb_0N\Delta x\,(1 + (\tan\alpha)^2)}{4b^2{}_0 - (N\Delta x\tan\alpha)} \approx \frac{4BfN\Delta x\,(1 + (\tan\alpha)^2)}{4f^2 - (N\Delta x\tan\alpha)} \tag{2.20}$$

This equation can be useful if we want to determine the range interval of a given system, or for instance if we want to redesign a system keeping the same range interval but using a different focal length or view angle.

2.4 Equations for width

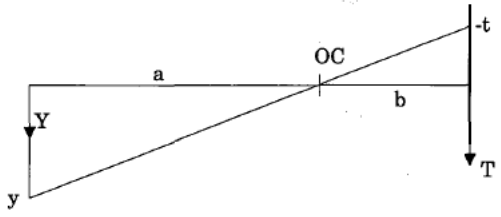
In Figure 2.11 we see the correspondence between sensor coordinate t and world coordinate y which can be expressed as

$$y = -\frac{a \cdot t}{b} \tag{2.21}$$

where a and b as functions of ρ are defined in (2.2) and (2.3) and as shown in Figure 2.12 $\tan\rho$ is found using

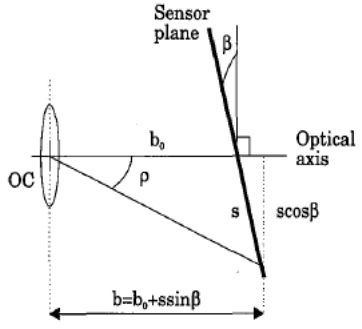
$$\tan\rho = \frac{s\cos\beta}{b_0 + s\sin\beta} \tag{2.22}$$

FIGURE 2.11.



The width geometry.

FIGURE 2.12.



The focus distance as a function of the incoming light angle ρ.

If we combine these equations we obtain

$$y = \frac{-t \cdot B}{\cos\alpha\,(b_0 + s\sin\beta)\left(1 + \frac{s\cos\beta\tan\alpha}{b_0 + s\sin\beta}\right)} \tag{2.23}$$

If β=0 (2.23) can be simplified to

$$y = \frac{-t \cdot B}{b_0\cos\alpha\left(1 + \frac{s\tan\alpha}{b_0}\right)} = \frac{-t \cdot B}{b_0\cos\alpha + s\sin\alpha} \tag{2.24}$$

From (2.23) and (2.24) it is evident that y decreases for increasing s and constant t.

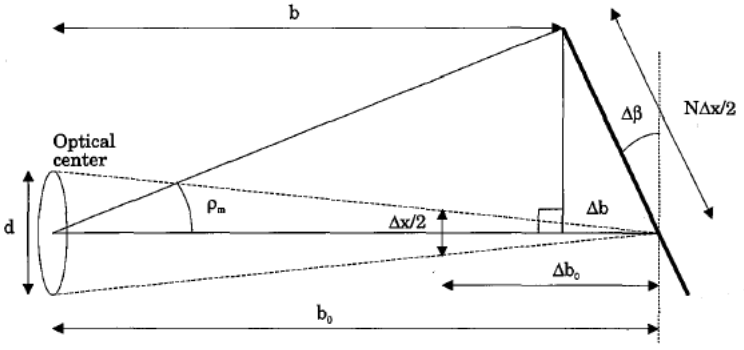
As we have shown the range r and width y can be determined from the sensor offset position (s,t) if we know the system parameters. However, in practice these range equations are not often used to obtain the range in a sheet-of-light range imaging system. Instead the range values (r,y) and their corresponding offset values (s,t) are registered in a calibration procedure, and stored in a look-up table which later can be addressed with the sensor offset positions. The advantage of the calibration solution is that it may automatically compensate for lens and sensor aberrations [55] which are not accounted for in the formulas above. Sheet-of-light range camera calibration is discussed in chapter 3. Nevertheless, the formulas (2.15) - (2.24) should give a solid qualitative understanding of the design problems for sheet-of-light based ranging systems.

2.5 Depth of focus

The lens parameters affect the depth of focus for the system. To correctly design a system we must know the tolerances for the sensor and sheet-of-light alignment. Specifically, we want to know how much the sensor can deviate from the Scheimpflug condition (2.8) without loosing much focus. To this aim we define the sensor to be in focus when the focusing ray-cone has a cross section width which is less than 1/2 pixel pitch, see Figure 2.13.

To be in focus, the sensor must be aligned so that the maximum deviation Δb from the correct distance b_0 is smaller than the depth of focus distance Δb_0. For simplicity, in these computations we only consider the case β=0 but the results should be valid for most β within the narrow range given in the first four rows of Table 2.2 above.

FIGURE 2.13.



Depth of focus geometry. The broad line depicts the sensor plane.

The depth of focus distance Δb_0, where the ray-cone has a cross section diameter which is half the pixel pitch Δx, is

$$\Delta b_0 = \frac{b_0 \Delta x}{2d} \tag{2.25}$$

The length Δb as a function of the sensor mis-alignment $\Delta\beta$ is

$$\Delta b = \frac{N\Delta x \sin\Delta\beta}{2} \tag{2.26}$$

and it denotes how much the distance b deviates from b_0 at the edge of the sensor. If this change in b is smaller than Δb_0 according to (2.25) the sensor is in focus, which gives a condition for the maximum deviation angle $\Delta\beta_m$

$$\sin\Delta\beta_m = \frac{b_0}{Nd} \approx \frac{f/d}{N} \tag{2.27}$$

The only parameters that determines $\Delta\beta_m$ is the sensor row size N, the lens diameter d and the focus distance b_0 (the sensor row size N and the f-stop f/d). Numerical examples are given in Table 2.3. We notice that the sensor angular alignment is rather critical, especially for small f-stops f/d, but that for f/d = 5.4 $\Delta\beta_m$ is almost as large as β. We can conclude the depth of focus discussion by noting the following. If we have high laser intensity (good signal to noise ratio) we can use a large f-stop, and then we can possibly achieve sharp (focused) images without

using Scheimpflug sensor geometry. If, on the other hand, we are forced to use a small f-stop or a large view angle we should definitely align the sensor according to the Scheimpflug condition to achieve sharp range images.

TABLE 2.3.

f/d	$\Delta\beta_m$	Δb_0
1.8	0.41	0.03
5.4	1.24	0.09

2.6 Accuracy limitations

The limited resolution of the sensor limits the possible ranging accuracy. The range uncertainty Δr as a function of the sensor uncertainty Δx (= Δs if no sub-pixel accuracy is used) and the sensor position s is obtained as follows, see also Figure 2.14.

$$s = b_0 \tan\rho \tag{2.28}$$

$$\Delta\rho_s = \frac{\Delta x (\cos\rho)^2}{b_0} = \frac{\Delta x b_0}{b_0^2 + s^2} \tag{2.29}$$

$$r = B\tan(\alpha + \rho) \tag{2.30}$$

$$\Delta r = \frac{\Delta(\alpha+\rho)B}{(\cos(\alpha+\rho))^2} = \Delta\rho_s \frac{B^2+r^2}{B} = \frac{\Delta x b_0}{b_0^2+s^2}\frac{B^2+r^2}{B} \tag{2.31}$$

Let us increase the angle α and assume $r >> B$, $b_0 >> s$ and $b_0 \approx f$. Then (2.31) reduces to

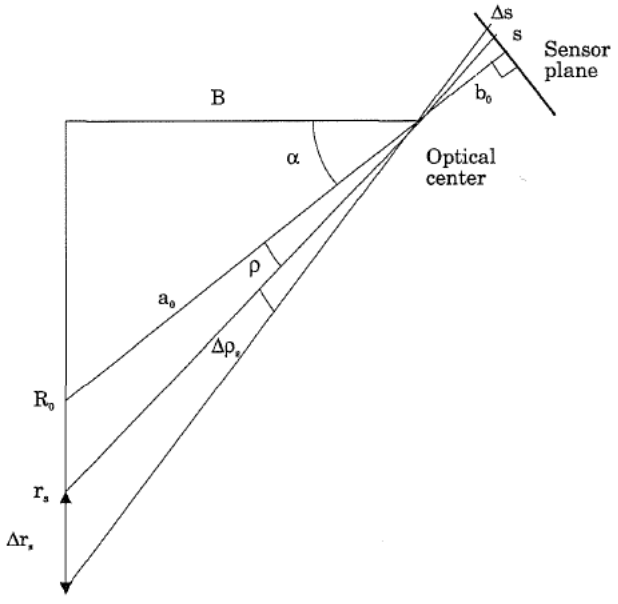
$$\Delta r = \Delta x \frac{r^2}{fB} \tag{2.32}$$

This equation shows us that the range uncertainty increases proportionally to the square of the range r. This makes it clear that range imaging based on triangulation only is attractive for short range distances where the baseline distance is at least in the same order of magnitude as the range distances. Further discussion on accuracy limitations due to sensor resolution can be found in [28].

Another limiting factor is the thickness of the laser sheet, see Figure 2.15. To give an unambiguous range value the reflected laser light should only illuminate *one* pixel in each sensor row. Each pixel interval Δx corresponds to a range increment Δr according to (2.32). For a given maximum view angle $\alpha+\rho_m$ and resolution Δr we get a maximum laser sheet thickness m as

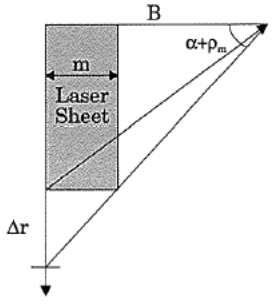
$$m \le \Delta r \tan(90 - \alpha - \rho_m) \tag{2.33}$$

FIGURE 2.14.



The accuracy limitations due to the sensor resolution.

FIGURE 2.15.



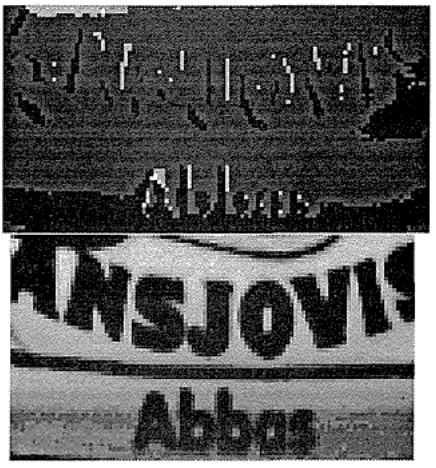
The resolution limit due to the thickness of the laser sheet.

In [44] a theoretical value for the maximum range error due to laser sheet width is given, but it is only valid for center-of-gravity sensor offset position computations. However, we can observe that the larger baseline we have, the smaller view angle α we get, and therefore better accuracy for a given laser sheet thickness. Figure 2.16 gives an example of what happens with a laser sheet which is thicker than the limit in (2.33). We see that the range values are incorrect where the maximum intensity change to/from very low values. When one part of the laser sheet illuminates an area of low reflectance and the other part illuminates an area of high reflectivity the maximum intensity along the sensor row gets shifted left or right away from the

center-plane of the sheet as shown in Figure 2.17. Off course, if the object reflectivity is constant a wider laser sheet can be used and therefore the offset position can be obtained with very fine sub-pixel accuracy, see section 4.1.

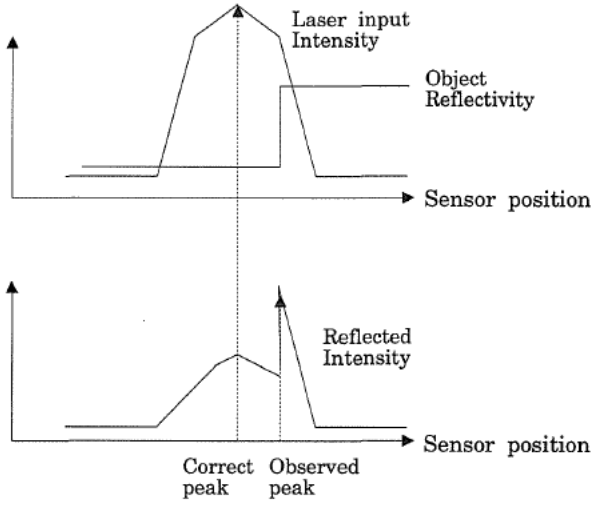
Another problem with a thick laser sheet is the risk for multiple reflections when the laser illuminates two different ranges.

FIGURE 2.16.



A range image where the laser sheet width is larger than the limit in (2.33). The range image depicts a flat surface. The top image is the range and the bottom image the intensity.

FIGURE 2.17.



Response from a thick laser sheet illuminating an object with strong reflectivity contrast.

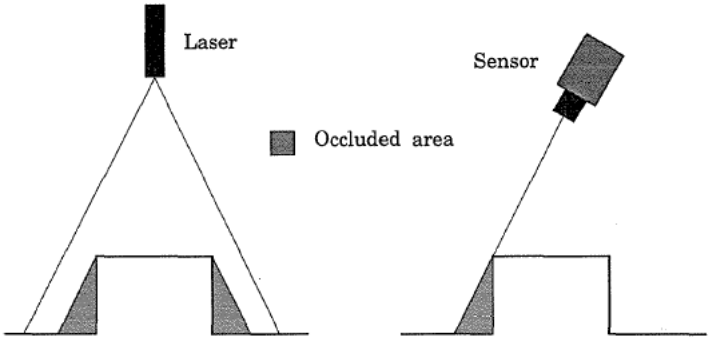
In theory the laser can be focused to a very thin sheet, limited only by the wavelength of the light. Unfortunately, some materials such as plastics spread the light so that a thick line is reflected nevertheless. If the 3D-object is made of a material which does not spread the light the potential range increment Δr may well be in the 20 μm range [45].

Since the laser sheet thickness changes over the range it is not possible to focus the laser precisely over a very large distance. In [44] dynamic laser focusing is suggested, but this requires that two range images are acquired. The first one is to achieve approximate range values so that the laser sheet focus distance can be adjusted and the second one to obtain the exact range.

2.7 Occlusion

A major problem in all triangulation methods is occlusion. In our case the problem manifests itself as one of the two following cases, see Figure 2.18. Either the laser light does not reach the area seen by the camera (laser occlusion) or the camera does not see the area reached by the laser (camera occlusion). In both cases the maximum reflection peak found in the sensor row data is the result of noise and/or ambient light. Any range value computed from such a maxima is totally insignificant as seen in Figure 2.19. We see large spurious range values computed for a large occluded area to the right of the can and smaller occluded areas around the other edges.

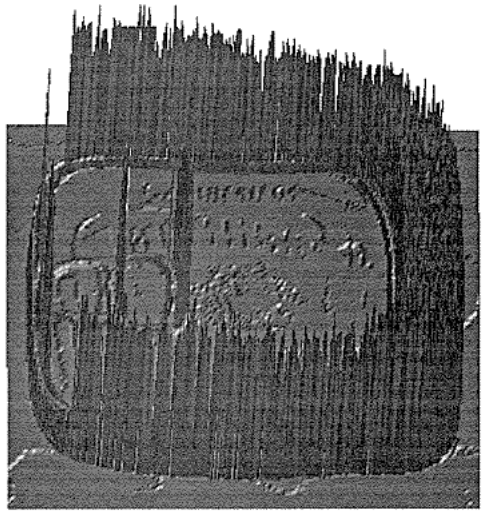
FIGURE 2.18.



Laser (left) and camera (right) occlusion.

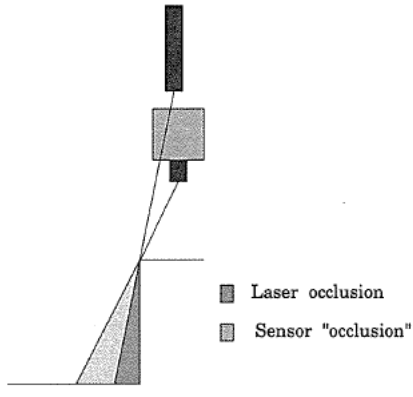
Both laser and camera occlusion can be minimized by careful placement of the laser and sensor. Laser occlusion is avoided by assuring that the laser reaches all areas seen by the sensor. Ideally, the baseline should be small so that the sensor and the laser can be considered as being in the same plane. Laser occlusion is then avoided by ensuring that the optical centre of the laser lens is further away from the scene than the optical centre of the sensor system, see Figure 2.20. Here the divergent sheet-of-light reaches all areas seen by the sensor since for any occluding object edge the area not illuminated by the laser is smaller than the area not seen by the sensor. Laser occlusion can also be avoided with multiple laser sources each illuminating the scene a little from the side, see Figure 2.21.

FIGURE 2.19.



A3D plot of an unfiltered range image of a fish conserve can illustrating the effects of occlusion.

FIGURE 2.20.

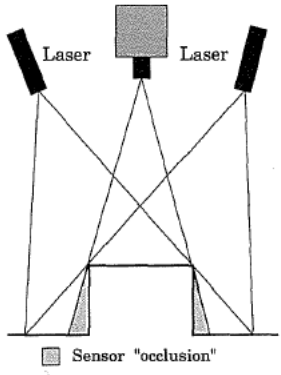


Laser occlusion avoidance.

Sensor occlusion occurs when the baseline increases from the ideal zero value. This can only be avoided using multiple sensors as seen in Figure 2.22. If two sensors are used, each viewing with a baseline B, but from separate sides of the laser most of the sensor occlusion is avoided. For more on the problems and benefits of a two-sensor system see for instance [51]. The only remaining sensor occlusion comes from deep holes where both sensors are occluded [57]. The system can be designed with only one sensor and mirrors reflecting the light from two virtual baselines as in Figure 2.23 [43]. The sensor must alternate between imaging as the

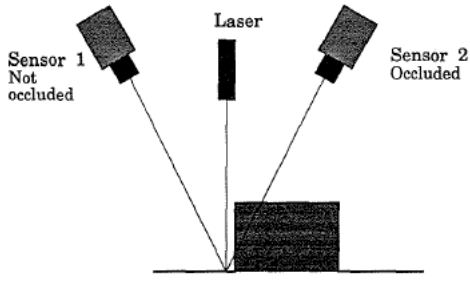
two virtual sensors since the sensor offset position will be different for the different virtual sensors. This technique also reduces the physical width of the system since the utilized baseline is wider than the system.

FIGURE 2.21.



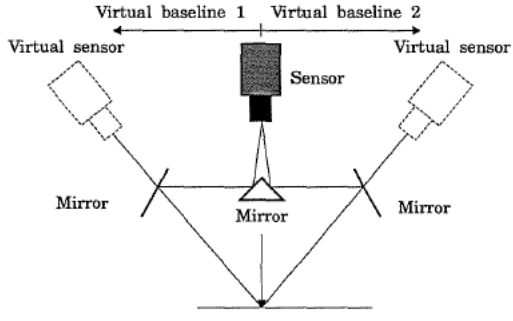
Laser occlusion avoidance using two lasers.

FIGURE 2.22.



Sensor occlusion avoidance using two sensors.

FIGURE 2.23.



Using one sensor with two virtual baselines.

Chapter 3

Sheet-of-light range camera calibration

3.1 Calibration Procedures

As was shown in chapter 2 the mapping from sensor to world coordinates can be found using trigonometric and geometric calculations. Often the geometry parameters are hard to measure, especially the interior camera parameters, which are the focal distance and sensor center offset position relative to the optical center. An alternative to setup measurements is to calibrate the system to find all unknown parameters in the equations. No direct (non-iterative) solution exists to find all the parameters, only iterative solutions [23]. However, a direct (non-iterative) solution exists to find 4x4 homogenous transformation matrix describing the sensor-to-world transformation [51], [55]. Alternatively, if the interior camera parameters are known all transformation parameters in the equations can be determined [41].

Another approach is to find the precise matching for a number of points in the scene and make an interpolation for all points in-between [57]. This method takes all lens and setup distortions into account, but it is often cumbersome, both in data acquisition and in the mapping from sensor to world coordinates. If this mapping is made as a direct lookup, a table with MxN entries is needed. A smaller table can be used, but then a 2D interpolation step is needed for each 3D value that is computed. For example, if $N = 256$ and $M = 512$ a complete table with 32-bit data precision would require 1Mb.

A third approach, which is used here, is to calibrate the system and find a polynomial approximation of the desired functions for range and width derived in chapter 2. This approach results in two small 1D lookup tables (one with N and one with M values) and requires only two table lookups and one multiplication to find the desired world coordinates for range and width. Another reason to choose this approach is that the resulting tables can be used with the existing *RANGER* range camera software [48]. However, this calibration scheme does not account for lens and setup distortions.

3.2 Function approximations

In our calibration we find the best polynomial approximations of the range and width equations (2.17) and (2.24) using least-square techniques. We wish to use a low order polynomial since it allows us to use fewer calibration points, and it minimizes the risk that the approximation fits closely to calibration data corrupted by noise. This is because a high-order polynomial tends to follow the noise. To find the suitable polynomial orders we have evaluated the maximum deviations between the correct equations and the least-square fitted polynomial functions. We used a setup with f=23 mm and B =300, and using the view angles α=30°,45° and 60°. In Table 3.1 we give the maximum deviations between the least-square fitted polynomial and the correct range equation (2.17). In this example each pixel corresponds to a range of roughly 0.5-3 mm and we aim at an accuracy of the resulting system which is within 1 pixel (the range camera algorithms used has a resolution of 1/2 pixel). Thus we propose to use at least second, third and fourth order approximations for the three cases α=30°,45° and 60° as this can give an approximation accuracy within +/-0.25 mm.

TABLE 3.1.

	Maximum approximation error (mm)				
View-angle	App deg 1	App deg 2	App deg 3	App deg 4	App deg 5
30°	4.3	0.24	0.01	0.0001	< 0.00001
45°	12.3	1.2	0.12	0.01	0.001
60°	50.9	9.0	1.5	0.25	0.04

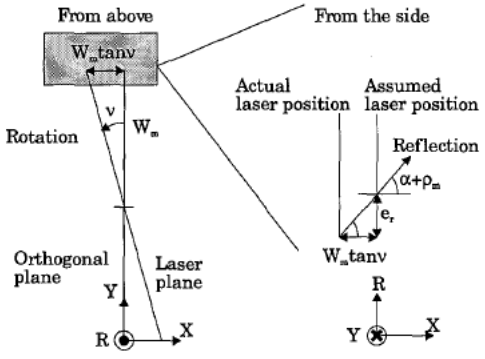
If we compare with Table 3.1 the maximum width error is approximately 10% of the maximum range error. Therefore we chose to use a first order approximation. However, for angles above ~45° it might be worthwhile to use a second order approximation.

3.3 Error sensitivity

In our calibration scheme we assume that the laser plane is parallel to one axis of the sensor. How sensitive is the system to violations of this assumption? If the laser plane is rotated an angle v around the range axis (laser optical axis) this gives a translation $W_m$tanv of the laser plane reflected at the edge of the sensor, see Figure 3.1. This translation gives a projection range error which is

$$e_r = W_m \tan\upsilon \tan(\alpha + \rho_m) \tag{3.1}$$

FIGURE 3.1.



The projected range error when the laser plane is rotated.

If we input the system parameters, using the approximation that b_0 is approximately equal to f into equation (3.1) we obtain

$$e_r = \frac{BN\Delta x \tan\upsilon}{2f\cos\alpha_p^{\,2}} \tag{3.2}$$

where

$$\alpha_p = \alpha + \rho_m = \alpha + \text{atan}\left(\frac{N\Delta x}{2f}\right) \tag{3.3}$$

If we use the system considered before, with B= 300 and f=23, we find that for a 1° degree rotation the error is 1.6, 2.8 and 8 mm for the cases α = 30°, 45° and 60° degrees respectively. Thus, we see that the laser plane must be aligned very carefully, which has proved to be very hard.

3.4 Calibration procedure

To calibrate the system we need to know the corresponding world and sensor coordinates for at least one more point than the highest order of the approximations. We use a calibration object with bright patches at well defined 3D positions placed in the laser plane. In order to achieve an accurate calibration the calibration object should cover as large part of the field-of-view as possible.

A segmentation program implemented on the MAPP2200 smart sensor is then used to accurately find the center-of-gravity of these patches [69]. Least-square adaptions are then made to find the best approximations of the transfer functions. Since both functions are approximations around the origin and the exact location of the optical center (camera coordinate system origin) is unknown, we iterate the least-square adaption over several possible sensor offset positions and use the offset which gives the best least-square fit to the model. The whole procedure takes less than 1 minute using a standard PC. The resulting polynomials are then used to create lookup-tables compatible with the *RANGER* software.

3.5 Experiments

We give results for two setups, one with a baseline B = 290 mm, view angle α = 45° and one with a baseline B = 170 mm and view-angle α = 30°, both with an f = 23 mm lens. The first system has a total range resolution of approximately 200 mm and each pixel corresponds to approximately 0.7-1.1 mm. The second system has a total range resolution of 80 mm where each pixel corresponds to approximately 0.3-0.7 mm. The fit between the calibration data set and the second and third order approximations are found in Table 3.2. We see that both the maximum error and the standard deviation is more than halved using the third order approximation compared to the second order ditto.

In Table 3.3 we have collected the corresponding results when the calibration lookup tables were applied to actual range data. We can see that the resulting accuracy of the calibrated system is close to the theoretical (measured) system in the first case (in the second case no accurate setup measurements were made). Thus we believe that the observed errors are errors in the range data acquisition rather than in the calibration. In the second experiment the range acquisition was more accurate, i.e. the thickness of the sheet-of-light was wider, giving a more stable range algorithm output, and we see that the second order approximation actually gives better results than the third order ditto. This is probably due to the low number of points (5) used in the calibration. Here the calibrated range accuracy is within 1 pixel on the sensor.

TABLE 3.2.

Model	Max Error (mm)	Std dev (mm)	# points
2-deg app 1	1.18	0.56	12
3-deg app 1	0.46	0.24	12
2-deg app 2	0.28	0.17	5
3-deg app 2	0.08	0.05	5

TABLE 3.3.

	System 1		System 2	
Calibration	Max error (mm)	Std dev (mm)	Max error (mm)	Std dev (mm)
Equation	1.6	0.7	-	-
2-deg app	1.8	0.7	0.25	0.15
3-deg app	1.2	0.6	0.48	0.2

We feel that the resulting accuracy, around 0.5% of the measurement range is adequate and comparable to those reported in [57], [55] and [51]. However, we have noticed that our calibration scheme is very sensitive to sensor and calibration (laser) plane rotations. That is, if the Y and T axes are non-parallel no correct calibration can be made since the range values then depend both on the s and t axis position of the sensor data. We have not presented any results on the y-axis calibration accuracy but from our experiments we believe that it works as good as the range calibration.

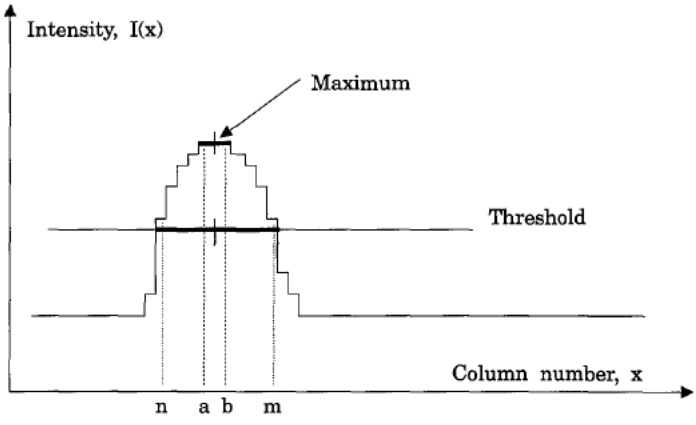
Chapter 4

Signal processing for sheet-of-light

4.1 Localizing the impact position on the sensor

Assume that the scene contour is reflected vertically along the image array as in Figure 1.6C. Along a row in the sensor the sheet-of-light reflected from the 3D scene may produce a signal as in Figure 4.1. Various rules and algorithms may be formulated to derive a unique impact position (*pos*) from this 1D signal. The most natural may be to find the position of the maximum intensity. Alternatively, we can threshold the signal and find the mid position of the pixels above the threshold. Analog devices such as position sensitive devices (see section 5.1) measure the center-of-gravity of the incoming light and this can also be computed with ordinary pixel-based sensors.

FIGURE 4.1.



An illustration of two different ways to find the laser reflection.

As indicated in Figure 4.1, even if the signal is well-behaved, the localization of the maximum of a discrete and quantized signal might be ambiguous. Assuming that the signal maximum is flat in the interval a≤x≤b the position (*pos*) would naturally be chosen as

$$pos = \frac{a+b}{2} \tag{4.1}$$

In some smart sensors [11], [38] the position is actually given as either a or b whichever is easier to compute. Obviously, this simplification is allowed only if it is ensured that the interval b-a is small.

If we use thresholding the peak mid position is given by

$$pos = \frac{n+m}{2} \tag{4.2}$$

where n and m are the first and last positions with a pixel value above the threshold respectively, see Figure 4.1. For signal peaks with a width of an even number of pixels equations (4.1) and (4.2) result in a position halfway between two pixel centers. This tells us that these formulas give answers with half-pixel resolution (sub-pixel resolution).

The center-of-gravity position using the total signal is given by

$$pos = \frac{\sum_x xI(x)}{\sum_x I(x)} \tag{4.3}$$

where x is the pixel column position and $I(x)$ the pixel intensity.

In [4] an accuracy of 1/20000 is claimed for a sensor with 388 pixels/row (which is 1/50 pixel sub-pixel resolution) using a White Scanner [54] with center-of-gravity computations. We find these claims somewhat doubtful and certainly impossible to hold up in the presence of noise and changing object reflectiveness. In [44] 1/16th of a pixel resolution is reported, but it is also stated that 1/2 pixel is a realistic value when viewing metallic surfaces with specular reflections. This resolution is acquired with a twin laser-sheet illumination scheme, and approximating (fitting) the signal bolus (c.f. the signal in Figure 4.1) with a Gaussian before the center-of-gravity computation. The double-sheet illumination design utilizes two lasers of slightly different wavelengths displaced slightly from each other and this scheme is used to avoid effects of abruptly changing object reflectiveness (c.f. Figure 2.17).

Both the maximum and center-of-gravity techniques can be combined with thresholding, so that only values above a threshold are considered in the computations. This makes the computations less sensitive to noise and background illumination. A compromise between center-of-gravity and thresholding, where multiple thresholds are used and the *pos* value is chosen as the mean center-of-gravity of the binary peaks obtained in the thresholding, is suggested in [20]. Maximum finding can also be combined with multiple thresholds to increase the position resolution.

A well known technique for finding the exact location of a peak is to find the position of the zero crossing of the first derivative. Using this for laser triangulation was suggested in [7]. Here several different derivative operators were evaluated using linear interpolation to determine the exact zero crossing location. The filter that gave the best performance was an eight-order FIR filter, with the convolution kernel [1 1 1 1 0 -1 -1 -1 -1]. Using this a resolution of 1/5 of a pixel was achieved in their experiments. If we assume a certain smoothness of the imaged scene it might be worthwhile to use filter kernels that cover several rows, for instance 3x3 or 5x5 Sobel filters, but this has not been evaluated.

Another elaborate scheme to find the range is to threshold the data at half of the maximum intensity and then use a linear interpolation as in [10]

$$pos = \frac{1}{2}\left[n - 1 + m + \frac{T - I_{n-1}}{I_n - I_{n-1}} + \frac{I_m - T}{I_m - I_{m+1}}\right] \tag{4.4}$$

The portion above the threshold is n to m, see Figure 4.1. According to the authors this gives very good resolution, but they also claim that their thresholding and center-of-gravity methods produced equally correct values. Furthermore, maximum finding was found to be more noise sensitive than thresholding.

In the MAPP2200 chapter we will discuss the implementation of an ad-hoc peak-detection algorithm called the odd-even method, OEM [66]. In this method the odd and even rows of the sensor are processed separately. Starting with the odd rows the charge from all pixels is added until the sum is above a certain threshold. The same technique is applied for the even rows, but starting on the other edge of the sensor. The idea is that each measurement gives the start position of the peak from one direction of the sensor, and that the correct peak position is the mean of these values. The algorithm is sensitive to noise since a small noise contribution from each pixel can give a large contribution to the sum which is compared with a threshold. The algorithm and its implementation is discussed in more detail in section 7.13.

To evaluate these different algorithms we have performed two sets of experiments. First we simulate the algorithm peak detection accuracy with and without the addition of white noise. Then, we test the algorithms on real data, where the noise cannot be said to be white. In the simulation results we can give very accurate error estimates, but the evaluation of the real data experiments is only qualitative.

Simulations

The simulations were performed using Matlab. The Signal-to-Noise Ratio, SNR, of an image row is measured as

$$SNR = 10\log\frac{\sigma_s^2}{\sigma_n^2} \tag{4.5}$$

where σ_s is the standard deviation of the signal and σ_n is the standard deviation of the Gaussian noise. The signal has 8-bit dynamic range. Since the algorithm performances might be dependent on the peak width we evaluated a narrow and a wide peak. The narrow peak covered approximately 2 pixels at half the maximum intensity and the broad peak 5. Results for the sub-pixel accuracy of two different peak-widths are found in Table 4.1 (narrow) and Table 4.2 (wide). In Table 4.1 the columns indicate maximum absolute error and mean error in pixels for zero noise, 43 and 10 dB SNR, left to right, and in Table 4.2 they indicate zero noise, 66dB SNR and 23 dB SNR. The added noise component is the same in the two simulations, but the peak width is changed, thus changing the SNR.

The simulated algorithms are thresholding, maximum finding, maximum finding with four extra thresholds, center-of-gravity with and without thresholding, the odd-even method, the interpolation method and the derivative method. The derivative method has been simulated using four different filter kernels, three derivative of Gaussian filters and the filter suggested in [7]. The derivative of Gaussian filters have the size three, five and seven pixels, and they are [1 0 -1], [1 2 0 -2 -1] and [1 3 3 0 -3 -3 -1] respectively.

TABLE 4.1.

	Algorithm accuracy measured in pixels					
SNR	**∞ dB**		**43 dB**		**10dB**	
Algorithm	**E-max**	**E-mean**	**E-max**	**E-mean**	**E-max**	**E-mean**
Thresholding	0.49	0.05	0.47	0	94.3	0.7
Max-position	0.49	0.04	0.76	0.03	1.01	0.03
Max-multi thr	0.18	0	0.3	0	0.49	0.01
CoG	0.01	0	5.15	0.66	6.6	0.70
CoG w thr	0.20	0.02	0.34	0	13.96	0.1
Odd/even	0.73	0.05	7.67	0.6	10.97	0.64
Interpolation	0.02	0	0.23	0	114.4	1.4
Der 3	0.03	0	0.18	0.01	0.41	0
Der 5	0.02	0	0.17	0.01	0.36	0
Der 7	0.02	0	0.19	0	0.35	0
Der 8	0.01	0	0.22	0	0.44	0

TABLE 4.2.

	Algorithm accuracy measured in pixels					
SNR	**∞ dB**		**66 dB**		**23 dB**	
Algorithm	**E-max**	**E-mean**	**E-max**	**E-mean**	**E-max**	**E-mean**
Thresholding	0.48	0.05	0.66	0.01	1.68	0.01
Max-position	0.8	0.05	1.36	0.05	1.89	0
Max-multi thr	0.19	0	0.44	0	0.79	0.01
CoG	0.01	0	4.2	0.73	6.37	0.74
CoG w thr	0.20	0.02	0.50	0.01	4.38	0.04
Odd/even	0.75	0.05	6.89	0.88	14.6	0.29
Interpolation	0.01	0	0.38	0	3.89	0.03
Der 3	0.05	0	0.65	0.01	1.20	0.02
Der 5	0.03	0	0.36	0	0.74	0.02
Der 7	0.02	0	0.28	0	0.61	0.01
Der 8	0.01	0	0.24	0.01	0.48	0.02

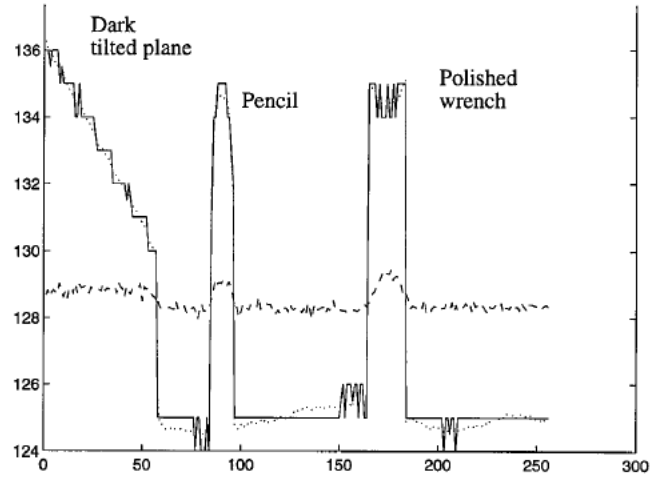
In the leftmost columns in these tables we see how the algorithms perform under ideal conditions. As can be expected the more elaborate algorithms give high accuracy, and the simple maximum and thresholding around 1/2 pixel accuracy. If we study the middle rows of the tables we see how the algorithms perform under fairly reasonable SNR values. We see that the center-of-gravity, CoG, and odd/even methods give large errors, and that maximum finding on a broad peak also gives quite large deviations. We also see that the claim from [7] of 1/5 of a

pixel should be possible with the derivative algorithm and an appropriate derivative kernel. If we study the rightmost columns we see how the algorithms perform under low SNR. We can see that maximum finding combined with multiple thresholds gives fairly good accuracy and that the derivative methods perform very well, but for the narrow peak the Gaussian derivative filters give slightly better accuracy than the filter suggested in [7]. We also note that the interpolation method and center-of-gravity with thresholding, which works well under moderate noise conditions deteriorates under heavy noise. As a conclusion we can state that maximum finding combined with multiple thresholds or a derivative method seems to give the best peak detection accuracy under very noisy conditions.

Experiments

We have tested the algorithms above on real images acquired using the MAPP2200 sensor. The images illustrate several of the error sources in sheet-of-light range imaging i.e. occlusion, uneven object reflectiveness, secondary reflections and illumination from ambient light. An example of range profiles acquired using maxima, center-of-gravity and derivative-based maxima detection can be found in Figure 4.2. As was expected the center-of-gravity is very inaccurate but the other two methods give similar, more correct, profiles. We also tried the algorithms subtracting two images, one obtained with the laser on and one with the laser off. This gives almost noise free images and all algorithms give performance in accordance to the table above. Another effective way to reduce the problem with ambient illumination is to use $\lambda/4$ optical interference wavelength filters which only allows light with a wavelength close to the laser wavelength. More experiments using real data are reported in [31].

FIGURE 4.2.



An example of range profiles. The scene contains three objects, a dark tilted plane, a wooden pencil and a polished wrench on a flat surface. The profiles were obtained using maximum (solid) derivative (dotted) and center-of-gravity (dashed) methods.

4.2 Multiple maxima considerations.

With all the above algorithms we get problems if two or more maxima appear in one sensor row. The most natural scheme would then be to simply remove or disregard the range results, or mark them as uncertain. However, this is not possible in all sensors. Some sensors/algorithms output the position of one of the maxima [11], or the mid-position between the two [2], [60].

In some applications it is desirable not to discard the values but to read out the range values from several peaks on one row, and leave the decision on which datum is correct to the next computational level. The reason that several peaks occur might not always be background illumination noise but secondary reflections from the laser reflecting on specular or transparent surfaces. In those cases more global information is needed to distinguish between the correct and the false reflections [46].

4.3 Post processing

The main purpose of the post processing is to detect, remedy, or alleviate the effects of occlusion and multiple reflections. Of course the true offset position for an occluded position can never be obtained, but once detected it might be approximated and/or interpolated, for instance using so called normalized convolution [39]. Alternatively the range data can be labeled as occluded (uncertain). We have used two different cues to detect erroneous (occluded) peak positions. *Error filtering* uses the error value, which we define as the number of detected maxima (peaks) in the sensor row. If occlusion occurred, the background illumination should be almost homogeneous and thus the probability that several peaks are found is large. This filtering is also very effective for detecting multiple reflections. Thus, when the number of maxima are more than one we know that the range datum is unreliable. Another way to detect and alleviate effects of occlusion is to use only those sensor rows where some pixels has an intensity above a certain maximum value. We define this as *intensity filtering*. If the laser is occluded very little light reaches the sensor row and the maximum intensity is low. Actually, if we use a thresholding algorithm intensity filtering is an integral part of the algorithm. An alternative to intensity filtering is to use the intensity as a confidence measurement in normalized convolution. When discussing different smart sensors and smart sensor implementations we will also discuss their ability for intensity and/or error filtering. Further discussion on different post processing techniques can be found in [31].

Chapter 5

Sensor designs for sheet-of-light

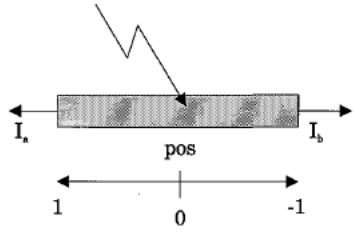
5.1 Position Sensitive Devices

A laterally extended photo diode is shown in Figure 5.1. This device is known as a Position Sensitive Device, PSD. The incoming light produces a photo-electric current in the p-n junction of the diode, and by measuring the currents flowing out from the edges of the diode the centroid position of the illuminated area can be obtained with the formula

$$pos = \frac{I_a - I_b}{I_a + I_b} \tag{5.1}$$

The sum I_a+I_b is proportional to the total incoming light intensity.

FIGURE 5.1.



A 1D PSD photo diode.

Sheet-of-light range finders based on use of multiple 1D PSDs has been presented by Verbeek et. al. [53], [60], [61] (1985) and Araki et.al. [2], [3] (1990), see Figure 5.2.

From Verbeek et. al. a 96-strip PSD array chip has been fabricated for which is reported 2 Mrangels/second with an accuracy of 1%. However, as of present, due to a bottle-neck in the analog processing only 8 of the 96 strips are utilized. The sensor chip from Araki et. al. has an array of 128 PSDs which is said to produce range images at 32 Hz with a range accuracy within 0.3%. If 128^2 images are obtained the range pixel frequency is 500 KHz. However, it is also stated that the PSD response time is 1 µs. Assuming that the processing is no bottleneck the theoretical maximum rangel frequency is in the 100 MHz range. But given the processing complexity a more realistic maximum rangel frequency ought to be more in the order of 3-10 MHz.

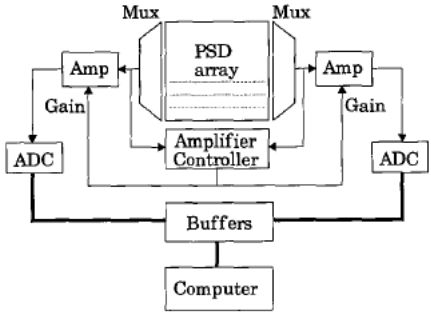
We notice that the primary sensor output from a PSD is the two analog values I_a and I_b which have to be transferred (in Figure 5.2 also multiplexed) and A/D converted. Analog data transfers are more sensitive to noise than digital data transfers which is a potential problem for PSD range finders.

Another problem is the following. If the background light intensity is high in parts of the scene, the centroid position is no longer the same as the peak position value as pointed out in section 4.1. To overcome this problem a PSD-system needs very high signal to noise ratios, and this can only be acquired by either increasing the sheet-of-light intensity and/or quelling

the ambient illumination. This is reflected in the use of a very high powered laser, 200 mW, in the case of [2], [3], and very short imaging distance, 50 mm, in the case of [53], [60], [61]. An image subtraction technique similar to the one discussed in section 4.1 can also be used with PSDs [59]. First the outputs of the array are recorded without any laser illumination. Then, when the sheet-of-light sweeps over the scene these values are subtracted from the sensor outputs prior to the pos calculations.

Since the intensity values might be read out they can be used as a confidence measurement. It is not possible however, to detect multiple maxima.

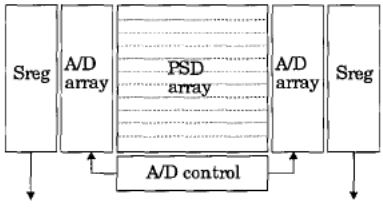
FIGURE 5.2.



A range finder architecture from Verbeek et. al. [60]. The amplifier control is used to continuously adapt the analog values to the working range of the eight-bit A/D converters.

To overcome the drawback of analog data transfers and the A/D conversion bottle neck in the proposed PSD array architectures we can combine a PSD array sensor with on-chip A/D conversion using row-parallel A/D converters, see Figure 5.3. The A/D converter and shiftregister layouts are very similar to the MAPP2200 design [17], [25], [27]. In [27] C. Jansson describes a 16-bit A/D converter design suitable for array implementations, and this precision would certainly be enough for high accuracy sheet-of-light range imaging. The conversion speed of the described circuit is 78 µs, giving an effective sample frequency of 3.3 MHz in a 256 cell array, but it is also stated that higher speeds are possible. Thus, this architecture should easily be able to output digital samples corresponding to 50 Hz 256^2 range image frame rate. Higher speeds are probably possible, especially if lower A/D conversion precision is used.

FIGURE 5.3.



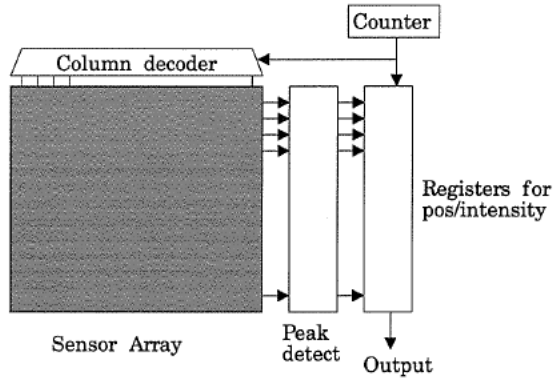
A novel sheet-of-light sensor combining a PSD array and on-chip A/D conversion.

5.2 Smart sensor architectures

We will present three devices below all of which attempts to compute position (offset) from a 2D discrete sheet-of-light sensor using on-chip circuitry. To be able to compare these non-implemented architectures we have assumed that we need 10 μs integration time and that the on-chip processing and analog data transfer frequency is 10 MHz.

The *column readout peak detection sensor* was originally described briefly in [11], and it has not been implemented. The chip combines an ordinary photo-diode or CCD sensor array with a column of peak detection circuits, see Figure 5.4. After integration of one sheet-of-light position in the sensor array the analog pixel values are read out column-wise, and the maximum positions are obtained during the column readout by the analog peak detection circuits. Thus, one column of data from all sensor rows are processed in parallel, not in series as is the case with a standard video sensor.

FIGURE 5.4.

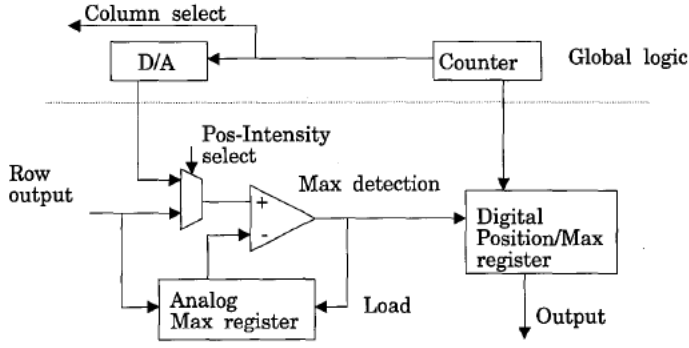


Column by column readout with peak detection.

The peak detection circuits is suggested to consist of a column of comparators which compare each row output with the previously recorded maximum intensity for that row, see Figure 5.5. If a new maximum is found, the analog value is saved in an analog register and the digital column position is saved in a position/max register.

After the position has been obtained the analog maximum registers contain the intensity values for the range data. To obtain the digital maximum intensity values a D/A converter instead of the row output is connected to the comparator so that the maximum value can be compared with a global ramp (staircase) signal from the D/A converter. When the ramp signal reaches the value of the analog max register the counter value is loaded to the corresponding register as seen in Figure 5.5. Thus, one column of range values for one sheet-of-light position is captured in a time frame which consists of integration plus sensor readout and peak detection plus position result output plus (optional) detection and readout of intensities. By doubling the digital registers sensor readout and peak detection can be made in parallel with the position result output. By again doubling the digital registers and also doubling the max-detection circuits and analog registers the intensity detection can be made in parallel with the next sensor read out and peak detection.

FIGURE 5.5.



One peak detection circuit with maximum value output.

If we have an *NxN* sensor one *NxN* range and intensity image can then be obtained in a time defined as

$$T_{im} = N\left(T_{Ini} + \frac{N}{f}\right) \tag{5.2}$$

where f is the processing frequency. If we assume that the integration time is 10 μs, the processing frequency is 10 MHz and $N = 256$, a complete 256^2 range image can be obtained in approximately 9 ms. This is a range and intensity pixel frequency of 7.2 MHz.

If we redesign the sensor so that we can integrate and process in parallel we can achieve a range and intensity image frequency of approximately 150 Hz, corresponding to a rangel/ intensity frequency of 10 MHz. This method might introduce errors since the maximum finding is done on columns integrated at slightly different time intervals, but we believe that this is no great problem. It is also possible to include peak width and multiple peak detection in the read-out circuits to decrease the noise sensitivity. If we preload the analog max registers with a threshold value before the sensor readout we obtain an automatic intensity filtering.

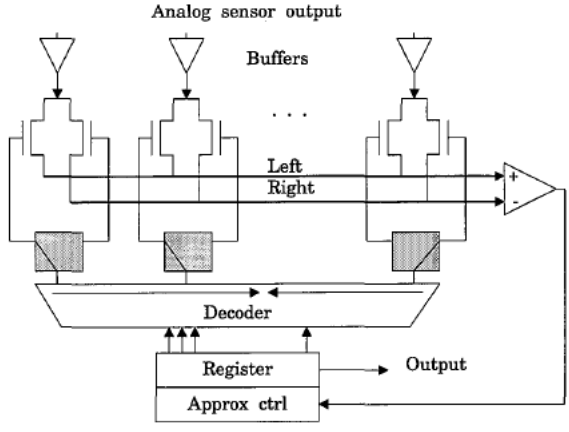
The *row readout successive approximation architecture* was also described briefly in [11], and it has not been implemented. It utilizes row-by-row readings of analog sensor data from an ordinary photo diode or CCD sensor array. For each row of analog sensor output we find the *mid-point position of the incoming light profile*. The analog values from the sensor outputs are multiplexed and summed to one of the two inputs of a comparator, see Figure 5.6. The output of the comparator controls a sequential network of type successive approximation which moves the approximative mid-point position left or right. First the array middle is tested as midpoint. If the left sum is larger than the right sum the assumed mid-point is moved so that the leftmost 25 percent are summed and compared to the rightmost 75 percent e.t.c. To get the output for an N bit row the network has to do *logN* approximations. After the approximations the register indicates the pixel position where half of the total intensity is to the left and half to the right, that is, the mid-point. The mid-point is the integer approximation of the center-of-gravity position.

Sensor integration and read-out can take place in parallel, but the readout speed is not so high since *logN* cycles are needed for each rangel output. Another disadvantage is that no intensity information can be obtained. If we have an *NxN* sensor an *NxN* range image can be obtained in a time defined by

$$T_{im} = N\left(max\{T_{Int}, \frac{N\log N}{f}\} \right) \tag{5.3}$$

If we assume a 256x256 sensor using an integration time of 10 μs, having a clock frequency of 10 MHz, the maximum range pixel frequency is limited by the processing and is 1.25 MHz. This is a range image frequency of 20 Hz. Obviously the design is sensitive to background illumination since all pixels in a row always contribute to the result.

FIGURE 5.6.



Row-by-row read-out with analog/digital processing to find the mid-point of a sensor row.

An *analog processing sensor cell* design differs from "standard' sheet-of-light range imaging techniques which integrate and read out for each sheet-of-light position. With analog processing cells a whole 2D range, or at least point in time, image is captured before the 2D sensor array is read out. (The sensor cells measure the instant light intensity at any given time.) The light-sheet sweeps over the whole scene and each sensor cell records at which sweep position the maximum light intensity occurred. That is, each sensor cell records events in the scene along a ray from the sensor cell through the optical center (OC), the sight line. Using this and the recorded impact time translated to illumination angle a triangulation can be made to obtain the range value, see Figure 5.7. Thus, each full sweep single "integration" gives a complete 2D range image of the scanned scene.

A design for this purpose was proposed by Kanade et. al. [21], [38], see Figure 5.8. Peak detection and registration of the time when the peak occurred is done within each sensor cell. Actually, Figure 5.8 represents the simplified circuit implemented on a 28x32 sensor chip [21] and not the more complex circuit implemented with discrete components in [38]. Among other things the difference is that the latter had true peak detector and also the ability to register the

maximum intensity in each sensor cell. During the "integration" a well-defined analog ramp signal is distributed to the sensor cells. This value defines the time, and the ramp value is stored when a maximum occurs.

As pointed out true maximum detection is not performed. Rather a fixed threshold is used and the ramp (time) value is saved when the sensor output crosses the threshold the first time. This makes the circuitry sensitive to background illumination and secondary reflections since any peak can be chosen as the maximum. It also gives inaccurate results for a thick light sheet reflection since any part of the sheet can give a positive result.

In [21] it is argued that the time domain sampling, restricted by the time resolution in sampling of the ramp signal, in this circuit gives higher range accuracy than the traditional space domain sampling, restricted by the inter-pixel pitch of the sensor. However, this reasoning is only valid if the peak is found accurately, and this is not ensured since no real peak detection is made in the circuit. Furthermore, in space domain sampling it is often possible to determine the width of the peak to correct for laser sheet width and uneven reflectiveness as discussed in section 4.1.

FIGURE 5.7.

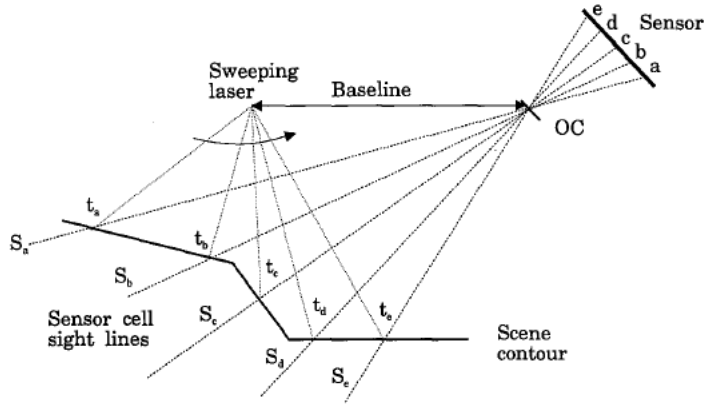
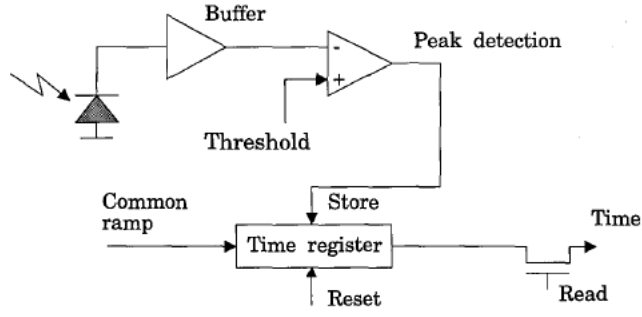


Image acquisition while the light sheet sweeps over the scene.

FIGURE 5.8.



Sensor cell due to Kanade et. al.

As we mentioned each sensor cycle gives data for one complete range image. Hence, every pixel in the 2D sensor must be read out in-between cycles which takes time and decreases the duty cycle for the system. Furthermore, analog processing is needed outside the chip to convert the analog time values to digital range ditto. Different triangulation parameters are needed for every sensor cell since each sensor cell represents one unique direction of the incoming light.

Due to the extensive hardware in each sensor cell this design is very hard to implement in standard VLSI. A recent report mentions a fabricated 28x32 array [21], where each sensor cell is 0.25 x 0.25 mm. Nevertheless, Kosuke Sato at the University of Osaka, who previously worked with Kanade, states that a similar design with 128^2 sensor cells is currently being manufactured by their laboratory [52].

In [21] it is stated that a range image speed of up to 1000 images per second can be achieved. 1000 images per second gives a range pixel frequency of approximately 900 KHz with the 28x32 sensor. If the same image speed is obtained in an 128^2 sensor the range pixel frequency is 16 MHz, and the requested readout frequency is 32 MHz if equal time is used for sweep and readout. Accurate readout of analog values at such high clock frequencies might be a problem. Even so, we admit willingly that this architecture has a potential for very high range image speeds. It is claimed [21] that the range accuracy is 1/256, based on eight bits of resolution in A/D conversion of the registered impact times.

Chapter 6

The NESSLA architecture

6.1 The NSIP paradigm

Near-Sensor Image Processing (NSIP) [18], [69], [70] is a new image processing concept where the individual sensor operation is an integrated part of the signal processing algorithm. In conventional smart sensors and image processing systems, after a fixed integration time, the sensor data is A/D converted and then processed digitally. In NSIP however the *binary* (thresholded) sensor cell outputs are repeatedly interrogated and processed *during* the integration and the results are obtained directly when the integration is completed. The use of binary signals seem like a disadvantage, but by processing the binary signals as they evolve over time we can nevertheless emulate grey-scale image processing. Actually, if we want a grey-scale output, by repeated interrogation of the binary outputs we can measure the time from the start of the integration until each binary sensor output switches from "0" to "1". The time until the sensor output switches is inversely proportional to the accumulated light intensity.

A general 2D NSIP architecture [69], [71] is capable of many low level image processing algorithms, but the architecture we present here is specialized to implement a row wise maximum finding algorithm. Thus the *Sensor-Processing-Element* (SPE) is much simpler and easier to implement than the general case. The architecture to be presented below can be implemented in full-scale with current CMOS technology. We call the architecture *NEar-Sensor Sheet-of-Light Architecture*, NESSLA [36], [37].

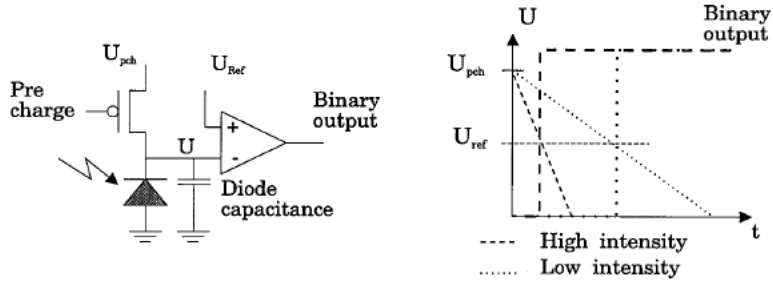
6.2 The NSIP sensor element

The basic NSIP sensor element is shown in Figure 6.1. It consists of a pre-charged photodiode connected to a high impedance voltage comparator with a binary output. If the diode voltage is below the reference voltage the comparator output is high (*one*), otherwise low (*zero*). After the diode precharge the incoming light will induce a current in the diode, proportional to the intensity. This current will discharge the capacitor of the diode and decrease the voltage. The time from the precharge of the diode to the time the diode voltage crosses the reference voltage of the comparator is inversely proportional to the light intensity, see Figure 6.1. Further discussions on NSIP sensors can be found in [69].

6.3 The NESSLA Sensor-Processing-Element

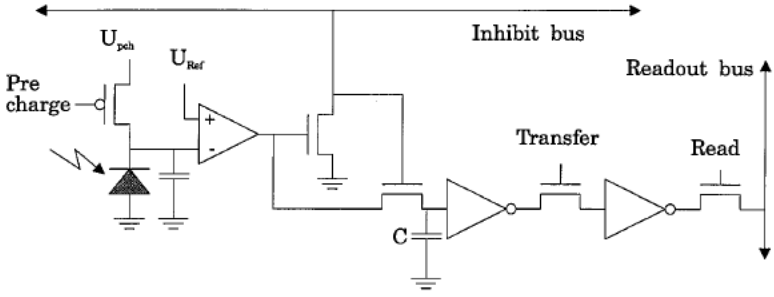
The design of a complete individual NESSLA Sensor-Processing-Element (SPE) is shown in Figure 6.2. The sensor area consists of a two-dimensional array of SPE's interconnected by inhibit and readout buses. An inhibit bus connects SPE's in a row for the purpose of maximum finding and a readout bus connects SPE's in a column for data readout. We shall see later that in an alternative design the readout bus is made horizontal, but this does not affect the design of the SPE itself.

FIGURE 6.1.



The NSIP sensor cell design. Circuit diagram for the diode and comparator (left). Voltages as a function of time and intensity (right).

FIGURE 6.2.



A complete single NESSLA SPE. The inhibit and readout buses are precharged high.

The integration starts when the photo diode capacitors in the SPE's have been precharged, after which they discharge proportionally to the incoming light intensities. When a photo diode voltage reaches the reference voltage its comparator output switches from low to high, and the inhibit bus goes low from its precharged high. Thus, at this moment all comparator outputs in the row are forced to freeze their potentials at the capacitor C. Interpreted as binary signals these values are *zero* except for the cell with the highest intensity and which took command over the inhibit bus. This cell has attained the value *one*.

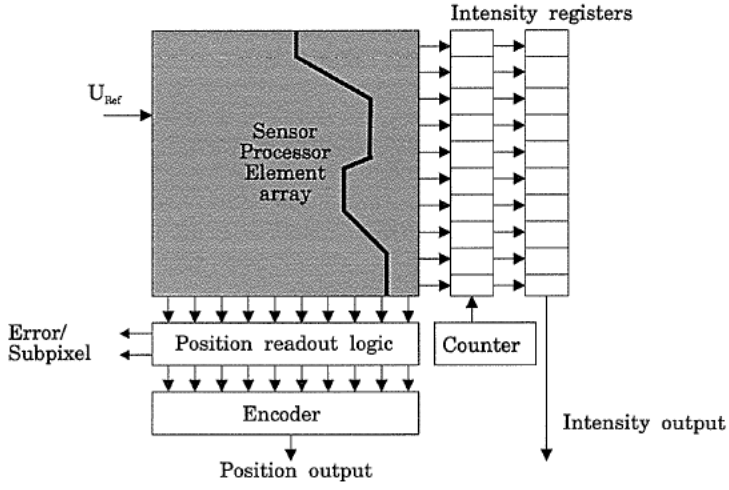
When the transfer transistors are enabled the binary data are moved from the first inverter to the second, and a new integration can begin in the photo diodes. The readout is then made concurrently with the next integration cycle in the sensor array.

We notice that the NESSLA SPE of Figure 6.2 so far is only capable of producing a maximum intensity *position* indication and we will see shortly how this is taken care of to obtain range data. However, already now we may also notice that if we would be able to record the point in time *when* the inhibit bus switches to low in a certain row, this information should be possible to translate to intensity. Hence, the NESSLA cell harbors the same possibilities for intensity *and* range data fusion as those presented in previous chapters.

6.4 Global architecture

The global architecture presentation is split in two parts; the position readout and the intensity readout respectively. An overview of the chip layout is shown in Figure 6.3. From the 2D-array of SPE's binary position data is read one row at a time, and the binary code of the position of the single *one* is obtained with an encoder. The encoder is preceded by the *position readout logic*, which inhibits all but a single *one* in a way defined below. Intensity values are read out to the right and stored in the intensity registers during integration. They are then read from the intensity registers concurrently with the position readout. As already indicated above, what is called intensity here is really time intervals inversely proportional to intensity.

FIGURE 6.3.



The NESSLA chip layout. Max position data are read out downwards from the sensor and intensity data are read out to the right.

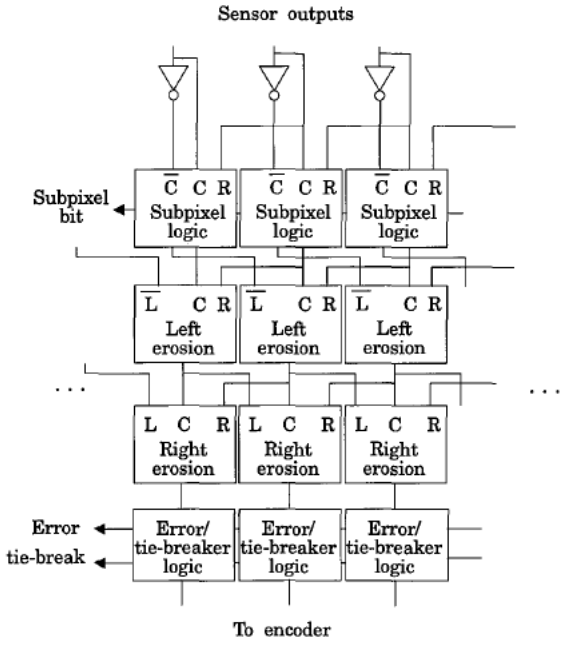
The SPE row is designed so that in most cases the resulting binary position vector contains only a single *one* which corresponds to the position of the maximum intensity. However, if several SPE's in a row are subjected to the same or nearly the same intensity (possibly because of a thickened, unfocused sheet-of-light or because of multiple reflections) more than one SPE could obtain a binary *one* output. Hence, some binary post-processing is necessary to obtain a clear-cut position of the maximum intensity.

Another case which must be accounted for is the case where no maximum intensity exists in the row, or rather when all intensities are so low that none of them are able to bring the photo-diode voltage down to U_{ref} in the allotted integration time, see Figure 6.1.

If the output from a SPE row contains one consecutive run of ones the correct output ought to be the mid-point value of this run. Ideally, the vector is expected to have one or maybe two ones if the light is focused and the object is a good reflector. We have chosen to accept up to three ones as an acceptable run while more than three consecutive ones will be indicated as an error. This rule comes from empirical experiments using MAPP2200 [29], [35].

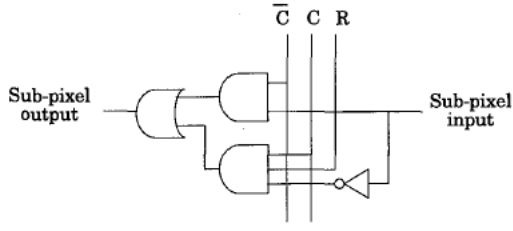
The position readout logic is pipelined into four stages, see Figure 6.4. The first operation on the output vector is a subpixel ripple chain. This is used to produce a sub-pixel position bit which extends the precision of the final position vector with an extra least significant bit, see Figure 6.5. The sub-pixel output is set to *one* if the sensor output vector has two consecutive ones. In such a case the maximum position is more likely to be the mid-point between these two pixels than the position of the leftmost or the rightmost of the two. This is exactly what the extra bit signifies.

FIGURE 6.4.



The position read-out logic pipeline. C stands for center pixel, L for left neighbor and R for right neighbor.

FIGURE 6.5.

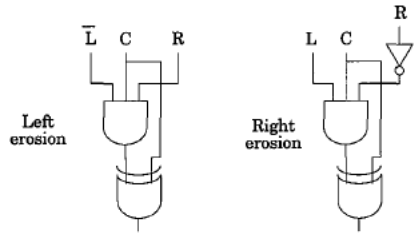


The sub-pixel ripple chain logic.

The second and third operations are one-dimensional erosion from the left and the right respectively, implemented with the logic seen in Figure 6.6. These stages erode two and three consecutive *ones* to a single *one*.

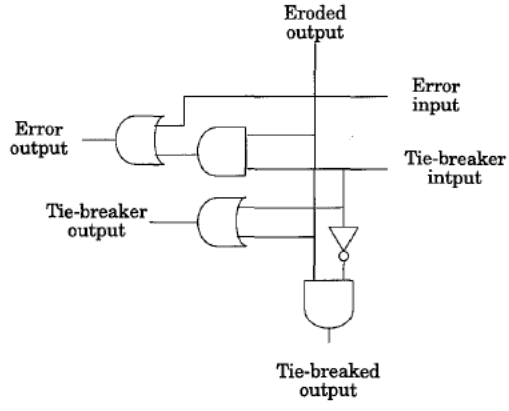
The last pipeline stage is the error indicator and tie-breaker logic, seen in Figure 6.7. The error indicator checks if there is more than one bit set to *one* after the erosion stage. Thus, the error bit will be set if there are more than three consecutive *ones* or more than one run of *ones* in the row read-out from the sensor. The tie-breaker ripple chain blocks all eroded outputs except the rightmost *one* to ensure proper function of the subsequent encoder in Figure 6.3. Hence, if there are more than a single *one* in the input vector to the tie-breaking logic it nevertheless gives the encoder a chance to deliver a properly coded position response. At the same time, potential multiples of *ones* in the input are indicated by the error output and the programmer can therefore decide wether the possibly flawed position value should be used or not. If no column position data is set to *one* the tie-breaker chain will deliver a *one* in the last position as a default. Hence, the encoder will always have a well defined output.

FIGURE 6.6.



One-dimensional erosion from the left and from the right. Erosion from the right is only performed if more than two consecutive ones are found.

FIGURE 6.7.

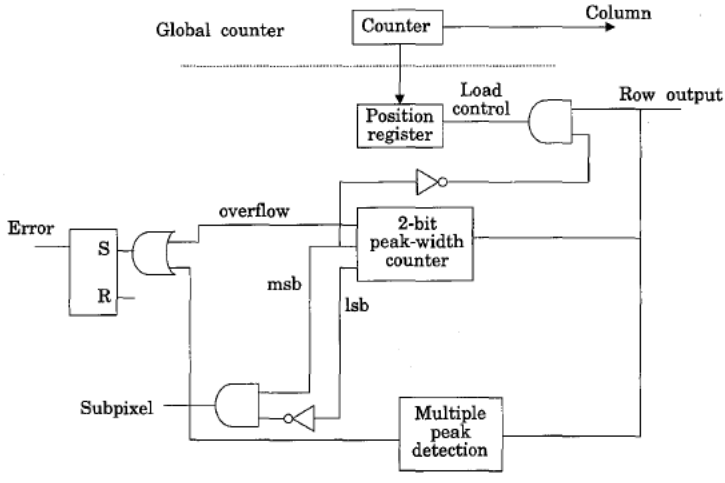


Error indicator and tie-breaker ripple chain logic. The error output is one if there is more than one bit set to one after the erosion stage.

Since the position readout logic has a gate-depth which is at least as large as the width of the sensor it is hard to use high clock frequencies. We must set aside time for the signals to ripple through the subpixel, error and tie-breaker ripple chains. One way to allow higher clock frequencies is to read out the position values column-wise and replace the position encoder by a serial peak detection in a manner that is vaguely similar to the column readout sensor in section 5.2. In fact, in the present case the peak detection is digital (binary) and therefore much easier to implement. The error and subpixel logic are already given as iterative networks and therefore ready to be implemented in a serial fashion.

One such column-wise readout circuit can be seen in Figure 6.8. To ensure that the position register receives a valid information, loading is only allowed twice. Thus a three pixel wide peak will get the correct position value. Error is indicated if more than three *ones* are found, that is when the two-bit width counter overflows, or when multiple peaks are detected.

FIGURE 6.8.



Serial column-wise position readout logic.

As pointed out before the time from the start of the integration to the activation of the inhibit bus is inversely proportional to the light intensity in the maximum intensity position. This intensity depends on the reflectance (brightness) of the surface illuminated by the sheet-of-light. Thus, a full reflectance map, an intensity image, can be obtained by saving these points of time for each row. To this effect the intensity registers in Figure 6.3 are utilized as follows.

A counter is started when the sensor diodes are precharged, and when the inhibit bus of a certain row is triggered the corresponding register is loaded with the counter value. These registers are then addressed simultaneously with the row output so that both the position and its intensity value are output concurrently. To be able to use this design together with the integrations, the registers have to be doubled so that one set of registers can be loaded from the inhibit bus while the other set is used for output.

6.5 NESSLA implementation

The implementation discussion is based very much on the results of two general NSIP chip prototypes presented in [13] and [14]. The two prototypes have 16x16 and 32x32 SPE cells respectively.

To achieve high sensor aperture the sensor should cover a large part of the area. However, to be able to implement the SPE logic effectively the area of the photo diode must be reduced. This is possible while actually increasing the aperture if we apply focusing micro lenses on top of the chip [15], [42], [58]. Such a lens is able to increase the effective light sensing area from around 1% to almost 100%. Other possible solutions are 3D VLSI processes [56] or thin-film amorphous silicon sensors applied on top of the chip [16]. The use of micro lenses is currently being tested on a prototype chip.

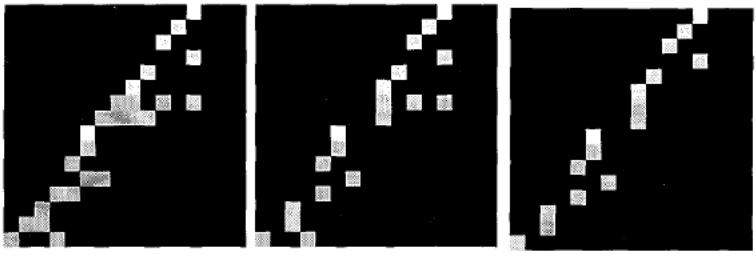
The comparator must be compact, use low power and be compensated against fabrication variations. The low power requirements makes it impossible to use standard current mirror comparators [26] and the proposed solution is to use an inverter with some sort of offset compensation [31], [13], [14]. The fabrication variations gives different threshold voltages for all transistors, and thus different thresholds for different inverters. The compensation can, for instance, be implemented by short-circuiting each inverter before the pre-charge, driving each inverter to its own individual threshold voltage.

Using the above comparator solutions in a 0.8 µm process a *general* 32x32 NSIP chip has been designed with 118x118 µm SPE size. The expected clock frequency is up to 100 MHz. For a NESSLA design the SPEs can probably be implemented on a 60x60 µm area, making it possible to implement a 256x256 SPE chip on a 15x15 mm chip [12]. The 16x16 prototype NSIP chip has been programmed for sheet-of-light range imaging, and an example of the results can be seen in Figure 6.9. For a 128x128 chip the predicted performance is 500000 processed frames per second, giving 64 MHz rangel frequency [14], [13]. However, if we consider the integration time the *NxN* range image speed for an *NxN* NESSLA sensor is given by the equation

$$T_{im} = N \cdot Max\{T, \frac{N}{f}\} \tag{6.1}$$

Hence a 256x256 SPE sensor array, 10 µs integration time and 100 MHz processing speed this gives a range image speed of 390 Hz, and a 25 MHz pixel frequency for both range and intensity values.

FIGURE 6.9.



An example of a sheet-of-light range profile acquired with a 16x16 general NSIP prototype. Left shows maximum finding results, mid after erosion and right after tie-breaking. Here the inhibit bus, left-right erosion and tie-breaker is implemented on the SPE array using general NLU and GLU operations. The grey levels of the pixels indicate the reflected brightness.

Chapter 7

MAPP2200 as range image sensor

7.1 MAPP2200 architecture

The smart sensor MAPP2200 [40], [17], is a 256x256 photo diode sensor array combined with a linear array of 256 A/D converters and 256 bit-serial processing elements (PE's) on one chip, see Figure 7.1. Each PE has an eight-bit A/D converter output register, an eight-bit parallel shiftregister, 96 bits of memory and an ALU, all connected via a one-bit bus. The A/D converters have direct outputs to the A/D registers so that pixel-row A/D conversions can be made independent of, and parallel with, the PE processing. The shiftregister can also run autonomously so that row transfers to/from MAPP2200 are independent of the processing. Each ALU has a Point Logic Unit (PLU) for 1-3 input logic operations, a Neighborhood Logic Unit (NLU) for left-right neighbor dependent binary operations and a Global Logic Unit (GLU) which sees MAPP2200 as one 256 bit wide processor. The sum of the accumulator contents of the ALUs (the number of *ones*) is called COUNT and can be read via MAPP2200s status register. The Global Or (GOR) output is *one* if the contents of the accumulator register is non-zero.

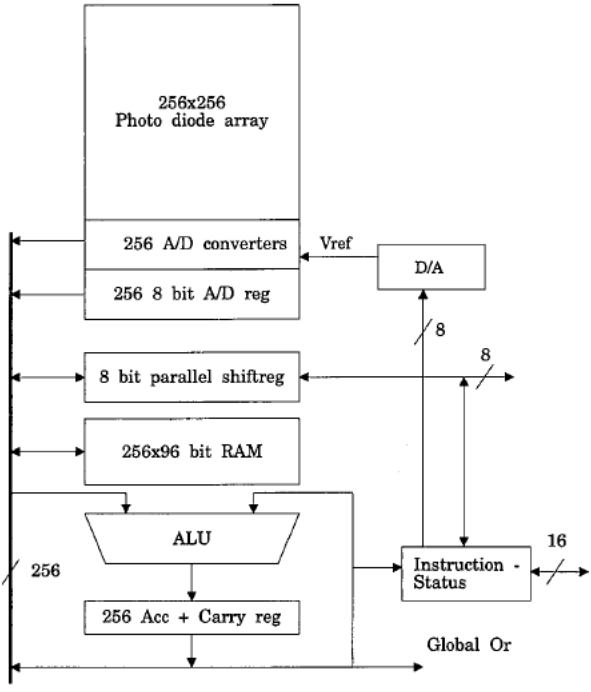
The row parallel Single Instruction stream Multiple Data stream, SIMD, PE array in MAPP2200 can be used to process the sensor data with a variety of low level image processing algorithms. For algorithms such as gradient (Sobel), high-pass (Laplace), low-pass and median filters video rate processing with eight-bit data precision is possible with 3x3 kernels. In [69] MAPP2200s performance for many low level algorithms are listed. Bit-serial addition of *b by b* bit words requires *3b* operations and an increment of a *b* bit word requires *2b* operations. However, if we use a Gray coded counter we can implement an increment operation with only an exor operation in 2 clock cycles. A conversion from Gray code to ordinary binary representation requires *2b* operations.

During normal sensor row readout the read transistors are only activated a short time to allow the photo diode voltage to be polled, see Figure 7.2. Then as long as the integrate transistors are activated the pixel charge is stored in the capacitance C. V_{ref} changes linearly from 0-5 V (0-255) via a counter and an internal D/A converter. The D/A counter value is stored when the PD-output switches from zero to one. The A/D converter resolution is programmable, so that any A/D conversion precision *b* between 1 and 8 bits can be used. A *b*-bit A/D conversion requires 2^b clock cycles.

A special mode allow MAPP2200 to be used for Near-Sensor Image Processing. The processor capacity is hardly sufficient to give NSIP support to the full 2D-array of sensor elements. However, it is possible to repeatedly interrogate the binary comparator outputs while integration is going on in one row of pixels. The readout stage of one MAPP2200 pixel column can be seen in Figure 7.2. One row of sensor elements are activated by one row of read transistors. By keeping both the read and integrate transistors *on* and V_{ref} constant the comparator output (PD-out) is the NSIP cell output for the activated sensor row. Thus, we can emulate NSIP architectures on a row by row basis. One thing to note about the sensor layout is that the sensor rows are grouped two and two, i.e. the distance between consecutive rows alternate between

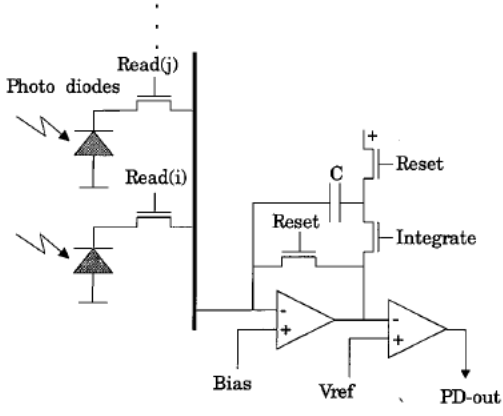
two different values. This will affect the performance of some algorithms. Also, the pixel aperture, i.e. effective sensor area, is only around 50%.

FIGURE 7.1.



Block diagram of MAPP2200 chip.

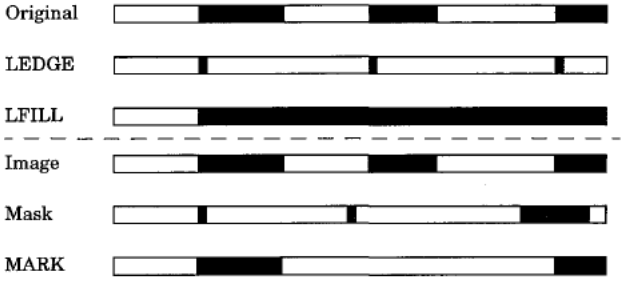
FIGURE 7.2.



Sensor column readout design on MAPP2200.

The NLU/GLU operations in MAPP2200 to be used in the sheet-of-light implementations are demonstrated in Figure 7.3. The NLU operation LEDGE is used to find the number of left edges, that is, the number of separate runs of ones, in a binary vector. We see that after the GLU operation LFILL the number of ones in the accumulator is the same as the position of the leftmost one in the original binary row. This number is the COUNT value. The MARK operation operates on two vectors, *image* and *mask*, and objects in image vertically connected to any set bit in mask are kept. The MARK operation can for instance be used to extract single objects from a binary vector (if the *mask* register is the result of an LFILL and a LEDGE operation the leftmost object will be extracted). The NLU operations LSHIFT and RSHIFT shift the binary vector one PE left or right respectively. This can be used in convolutions since it can be used to input data from the left/right neighbor to a PE. Thus it is easy to implement 3x1 convolution kernels using the NLU.

FIGURE 7.3.



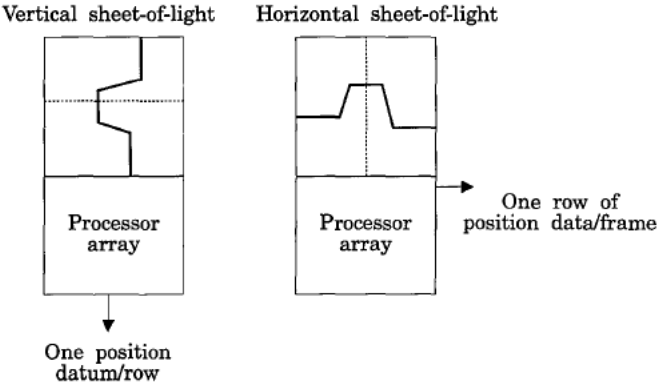
The NLU-GLU functions used in this chapter.

MAPP2200 is connected to a controller, normally a PC, which writes instructions to the camera and reads camera output data. A special semi-autonomous video generator can be connected to the shiftregister output so that the camera can generate a video signal. This unit also contains four external vram image memories connected to the shiftregister. MAPP2200 is also available as a *smart camera* which contains a MAPP2200 chip and an embedded 186 PC processor in a compact package. This camera can be used as a stand-alone image processing system. The smart camera communicates via four binary IO channels and a serial PC-port.

MAPP2200 is designed for a maximum clock frequency of 4 MHz, but our experiments have shown that clock frequencies up to 8 MHz is possible on selected chips. The maximum A/D conversion and shiftregister clock frequencies possible are 23 and 10 MHz respectively [27].

MAPP2200 can be used for sheet-of-light range imaging either with the light-sheet projected horizontally or with the light-sheet projected vertically on the sensor. Both ways are illustrated in Figure 7.4. When the sheet-of-light is vertical the PEs are used as one 256-bit processor to find the position of the sheet-of-light impact position of each row in a serial fashion. When the sheet-of-light is horizontal the PEs are used as 256 independent bit-serial processors, where all PEs find their own sheet-of-light impact positions in their respective columns in parallel. Thus, with a vertical sheet-of-light each sensor row processing gives one position value, whereas with a horizontal sheet-of-light each processor is expected to deliver one position datum after processing of all sensor rows.

FIGURE 7.4.



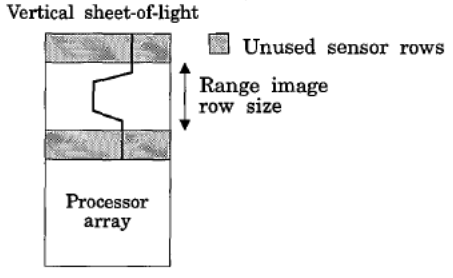
Vertical and horizontal sheet-of-light range imaging on MAPP2200.

In the pseudo code examples given we use syntax similar to the programming language C. All MAPP2200 specific instructions are written in capitals, and MAPP2200 macros start with a capital letter. All other instructions are executed by the controller.

7.2 Trading resolution for speed

The programmability of MAPP2200 makes it possible to generate range images of arbitrary size or range resolution. With a vertical sheet-of-light, range images with less than 256 samples per range image row are obtained by only utilizing a fraction of the sensor rows, see Figure 7.5. This does not change the range pixel frequency, and thus the range image frequency increases. For instance, if we generate 128^2 range images instead of 256^2 ditto the image frequency increases with a factor 4. The readout can also easily be interlaced, so that all sensor rows are utilized, but only every second integration [31].

FIGURE 7.5.

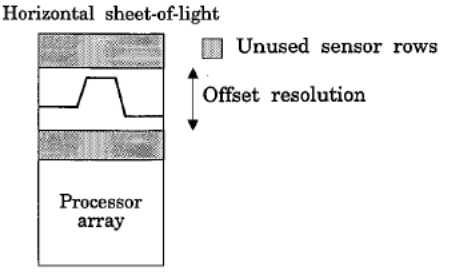


Increasing the range image speed by decreasing the image size.

With a horizontal sheet-of-light it is possible to *increase* the range pixel frequency by limiting the offset position resolution to a fraction of the sensor, see Figure 7.6. This can be made in two ways, either by using maximum accuracy over a smaller range interval, or by using lower accuracy over the maximum range interval. If we only process a fraction of consecutive rows we obtain maximum accuracy over a fraction of the range interval. In MAPP2200 it is

also possible to address several sensor rows concurrently and read out the analog sum of the rows to the A/D converters. If we process, say, the analog sum of pairs of rows we obtain range images with half of the maximum range resolution over the whole range interval.

FIGURE 7.6.



Increasing the range pixel speed by decreasing the range resolution.

7.3 Vertical NSIP algorithm

MAPP2200 can be programmed to emulate the NSIP maximum finding algorithm in the NESSLA architecture (see chapter 6). The performance is of course not as good as in NESSLA since each row of sensor data must be processed in series and not in parallel.

The principle is to integrate each row until the first pixel charge reaches the preset threshold. The pixel which first reaches the threshold is the one which received the highest intensity. Thus, after the maximum finding loop the binarized sensor row vector has a *one* at this PE position. The position of the *one* is then found using the GLU function LFILL. In pseudo code the program can be written

```
    /* NSIP sheet-of-light algorithm on MAPP */
SETV 20                 /* An appropriate threshold */
for (i=x;i<y;i++) {     /* Loop N = (y-x) rows */
   SETR i               /* Set sensor row to read */
   j=100                /* Reset integration counter */
   RESET                /* Reset PD register */
   INTEG                /* Start integration */
   LDI PD               /* Load comparator output */
                        /* Loop until threshold is reached */
   while ((GOR == 0) && (j > 0)) {
       j--
       LDI PD}          /* j = 0 Indicates no range data */
   width[i] = COUNT     /* Width of maxima */
   LD LEDGE Acc         /* Find no of maxima */
   err[i] = COUNT       /* Read error value */
   LFILL Acc            /* Find leftmost pixel pos */
   pos[i] = COUNT       /* Read position values */
   int[i] = j           /* Read intensity */
}
```

The algorithm output consists of four values for each sensor row, the leftmost position of the maxima (*pos*), the maxima width (*width*), the error value (*err*) and the counter value corresponding to the maximum intensity (*int*). The *width* value is used to compensate for the maxima width using the formula

$$pos[i] = pos[i] - \frac{(width[i] - 1)}{2} \tag{7.1}$$

Thus, we reduce the *pos* value with half the maxima width to the mid maxima position value and get half pixel sub-pixel accuracy. If the *err* value is larger than one we know that there were more than one maxima, and that the *pos* datum is uncertain. If we use up to 100 NSIP interrogations for each sensor row the possible range pixel frequency is 15-30 KHz, which corresponds to frame rates of 0.2-0.5 Hz for the image size 256^2.

Unfortunately, the design of MAPP2200 makes it hard to use for NSIP range imaging. This is because the read transistors attached to the column readout bus also act as photodiodes, see Figure 7.2. When we use MAPP2200 for NSIP processing we activate one row of read transistors and activate the integrate transistors in the read-out circuits. This opens a path for charge transport from the capacitor C, via the read bus to the photo diode. However, additional charge is leaking from C via the parasitic photo diodes attached to the bus. This gives the result that if high intensity reaches several sensors in the same column this light might discharge its column capacitor C more than the higher but single intensity can do in another column in the same row. Since this leakage increases the generated intensity levels using the intensity as a confidence measure often fails. As a whole, the leakage effect often leaves errors which are hard to detect and remedy, such as contours getting expanded over occluded areas [31]. The only real remedy would be to redesign the read-out circuitry. The leakage effect is also visible in some of the other algorithms we have implemented, but since the time the analog PD register is connected to the bus without a reset is shorter the effect is smaller. Examples of images obtained using this algorithm can be found in [29] and [31].

7.4 Vertical successive approximation algorithm

This implementation uses a successive approximation algorithm previously described in [29], [37], [35] and [31] to find the maximum intensity in each row. The intensity is converted to an eight bit integer. The successive approximation is done by first setting the threshold to the middle of the interval, i.e the most significant D/A bit set to *one* ($10000000_2 = 128_{10}$). If any PE has a pixel value above this threshold we know that the maximum intensity is higher and we keep that D/A bit set to *one* and then test again with the next D/A bit set to *one* ($11000000_2 = 196_{10}$). If no PE has a pixel above the threshold we know that the maximum intensity is lower, and we reset that D/A bit to *zero* and test again with the next D/A bit set to *one* ($01000000_2 = 64_{10}$). Thus, we half the possible maximum intensity interval at each iteration and after eight iterations we obtain the maximum intensity for that sensor row. If we threshold the row with the found maximum we should, normally, obtain a binary vector with a single *one* at the position of the maximum. The position of the *one* is then found using the GLU function LFILL. In pseudo code the program can be written

```
        /* Vertical Successive approximation */
        /* Sheet-of-light algorithm */
SETR i                                  /* Start row */
for(i=x;i<y;i++) {                      /* Loop over N = (y-x) rows */
   INTEG                                /* Read pixel to PD reg */
   HOLD                                 /* Hold pixel value */
   volt = 128                           /* Initial approximation */
   for(j=7;j>=(8-b);j--) {              /* Succ. app. loop */
        SETV volt                       /* Set A/D voltage */
        LDI PD                          /* Load binary row */
        if GOR                          /* If "hit" */
         ST PD,R(0)                     /* Save binary vector */
        volt -= ((1-GOR)<<j)            /* Change volt */
        volt += (1<<(j-1))              /* Set next bit */
   }
   LD R(0)                              /* Load best approx vector */
   width[i] = COUNT                     /* Read width value */
   LD LEDGE ACC                         /* Detect left edges */
   err[i] = COUNT                       /* Read number of maxima */
   LFILL ACC                            /* Fill from leftmost one */
   LD GLU                               /* Read fill result */
   pos[i] = COUNT                       /* Read position value */
   RESET,+                              /* Reset PD/pixel, new row */
   int[i] = volt                        /* Read intensity value */
   }
```

The time to acquire one *NxN* range image is approximately

$$T_{im} = \frac{N^2(4b+10)}{f_p} \tag{7.2}$$

where b is the number of steps in the successive approximation loop. Fewer steps in the loop gives higher ranging speed, but the accuracy of the range data might decrease. Also, the resolution of the intensity data decrease. With *b=8* the maximum range data frequency is 190 KHz, and the 256^2 image frequency is 2.9 Hz. With *b=4* the corresponding figures are 308 KHz and 4.7 Hz. With a dedicated PAL controller we reached approximately 160 KHz rangel frequency [35], [31]. Without the error and peak width filtering the speed increase around 10%. In chapter 4 peak finding using thresholding was found to be slightly less noise sensitive than maximum finding. It might therefore be worthwhile to threshold the peak with for instance half the peak intensity instead of with the peak intensity. This actually increases the algorithm speed, since the store of the binary vector in the inner loop can be omitted, but in our tests no accuracy improvement was detected. The store of vectors in the inner loop is made to remove errors which occurred when a final thresholding at the maximum found intensity gave only zeros due to analog noise in the readout circuitry [31].

An example of the combined range and intensity output of the successive approximation algorithm can be seen in Figure 7.7.

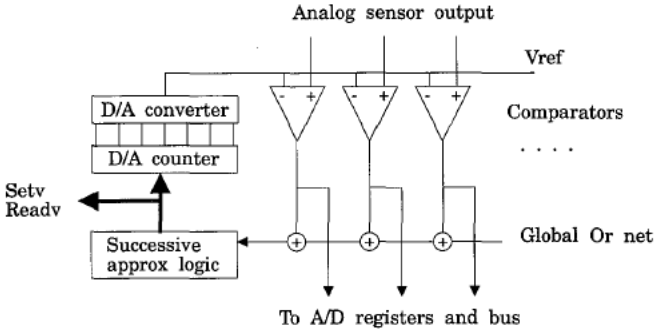
FIGURE 7.7.

A 256x256 range image plotted as stacked height curves and with the intensity information as plot intensity.

Improvements in MAPP2200 architecture

If the successive approximation loop could be performed without transfers to the accumulator the inner loop of the algorithm could be speeded up with a factor of at least three since only one instead of three-four instructions would be needed. This implies that a Global Or net, wired to the comparator outputs and input to a small sequencer to control the A/D converter, is added, see Figure 7.8. To make it possible to read out the D/A counter value (the maximum intensity) another modification is also needed. These modifications should not be too difficult to implement since the existing automatic A/D conversion net is rather similar. This modification should be useful in other MAPP2200 applications such as finding the brightest pixel over the whole sensor.

FIGURE 7.8.



A modified A/D converter design with automatic maximum detection.

7.5 Vertical thresholding algorithm

The pseudo code for an implementation of a vertical thresholding algorithm on MAPP2200 can be written

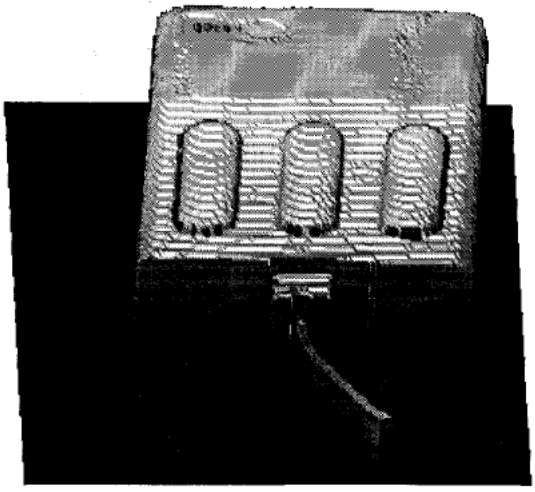
```
        /* Vertical thresholding algorithm */
SETV volt               /* Set A/D threshold voltage */
SETR i
for(i=x;i<y;i++) {      /* Loop over N = (y-x) rows */
   INTEG                /* Read analog sensor row */
   LDI PD               /* Load thresholded row */
   RESET,+              /* Reset PD/pixel, new row */
   width[i] = COUNT     /* Read the peak width */
   LD LEDGE Acc         /* Detect no of peaks, error */
   err[i] = COUNT       /* Read the error val */
   LFILL Acc            /* Fill from leftmost one */
   LD GLU               /* Read fill value */
   pos[i] = COUNT       /* Read position value */
}
```

The time to acquire one *NxN* range image is approximately

$$T_{im} = \frac{9N^2}{f_p} \tag{7.3}$$

If we use a clock frequency of 8 MHz we achieve approximately 880 K rangels per second (an image frequency of 13 Hz) with error/peak width correction and 1.3 MHz without. A complete range image acquired with the thresholding algorithm can be seen in Figure 7.9.

FIGURE 7.9.

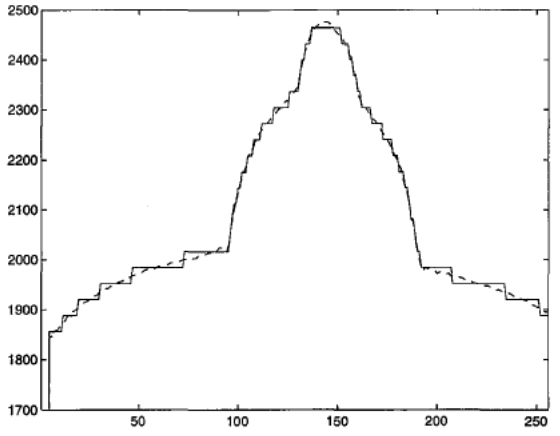


A 3D plot of a complete range image of a computer mouse acquired with the thresholding algorithm.

7.6 High accuracy combination algorithm

By augmenting the successive approximation maximum finding algorithm with a number of thresholdings, the extra thresholds selected uniformly between the found maximum and the supposed noise level, we obtain higher accuracy. Each extra thresholding gives a new range value estimate, and by averaging all estimates we can get higher accuracy than with only a single maximum finding or thresholding. Of course the range imaging speed decreases, but in some applications high accuracy is more important than high speed. We feel that the combination of maximum finding and thresholding is better than to just use several pre-set multiple thresholds since we always use the whole available signal. We applied this technique using 5 extra thresholds, and a resulting combination profile with 14-bit resolution and the maximum-finding profile with 9-bit resolution can be seen in Figure 7.10.

FIGURE 7.10.

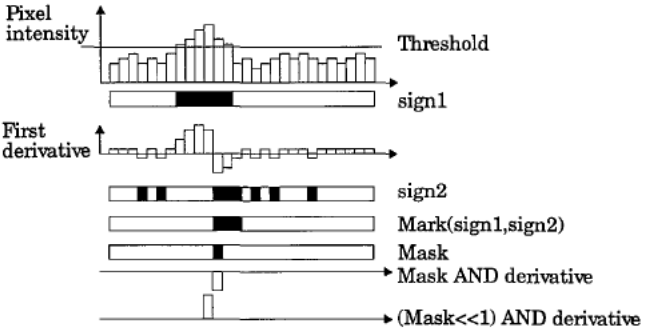


A comparison of maximum finding (solid) and maximum finding combined with multiple thresholds (dashed). The profile depicts a cross-section of a jet-fighter model.

7.7 Vertical first-derivative method

In [7] a peak detector based on the position of the zero-crossing of the first derivative was presented. This is a similar implementation on MAPP2200. We use a thresholding algorithm to find the initial peak position and we use the NLU and GLU to read out the intensity values at the zero crossings of the first derivative, see Figure 7.11. Then we interpolate a sub-pixel value from the first derivative.

FIGURE 7.11.



The peak detection in the vertical first-derivative method using the GLU/NLU.

In our implementation we use a 5x1 Sobel-type binomial filter to approximate a smoothing first derivative operator and in pseudo code the algorithm looks like

```
/* First derivative peak-detection algorithm */

for (i=x;i<y;i++) {                  /* Read N = (y-x) rows */
   pix = ReadAD                      /* Read A/D converter b */
   Sub(pix,thresh,sign1)             /* save only sign, 2b */
                                     /* Derivative filter */
   Add(pix,left_pix,tmp)              /* 1 1 kernel 3b */
   Add(tmp,right_pix,tmp)            /* 1 1 -> 1 2 1 */
   Add(tmp,left_pix,tmp)             /* 1 1 -> 1 3 3 1 */
   Sub(tmp,right_pix,tmp)            /* 1 -1 -> 1 2 0 -2 -1 */
                        /* sign of tmp = sign2, possible pos */
                        /* Find zero crossings within peak */
   MARK(sign1,sign2)                 /* 2 clock */
   LFILL
   pos[i] = COUNT                    /* Read position of peak */
   LEDGE                             /* Mask for positive val */
   ST MASK                           /* Store mask */
   for (j=b;j>0;j--) {               /* Read nval, 3b */
       LD MASK
       AND der(j)                    /* Mask out bit */
       nval += GOR<<j                /* Read bit to controller */
   }
   LD (LSHIFT) MASK                  /* Pos for positive val */
   Read(pval)                        /* Read positive value 3b */
}
```

The sub-pixel peak position is then found as

$$pos[i] = pos[i] - \frac{nval}{pval - nval} \tag{7.4}$$

note here that nval is negative. Since we need time to A/D convert the next row while processing the present one the time for an *NxN* image is approximately

$$T_{im} = N^2\left(Max\left(\frac{2^b}{f_{AD}}, \frac{21b+7}{f_p}\right) + \frac{b}{f_p}\right) \tag{7.5}$$

This gives a rangel speed of approximately 44 KHz if *b* is eight. With a 3x1 derivative filter the corresponding speed is 67 KHz, but it will also be more sensitive to noise. The range frequencies obtained are not very high, but on the other hand this algorithm is, as was shown in section 4, capable of high accuracy even under noisy conditions.

7.8 Horizontal maximum finding algorithm

A horizontal maximum finding algorithm over *a* rows is obtained by the following

```
/* Horizontal maximum finding sheet-of-light
   algorithm over part of the sensor */
/* Position counter is Gray coded */
pos = 0                              /* Reset counter; loga */
Max = 0                              /* Reset max; loga */
for (i=x;i<y;i++) {                  /* Read a = (y-x) rows */
   pix = readAD                      /* Read A/D converted row b */
   if (pix > Max){                   /* Test for maximum 2b */
```

```
        Max = pix                    /* Save new maximum 3b */
        pos = rowno                  /* Save position 3loga */
    }
    rowno++                          /* Increment pos 2 */
}
Convert(pos)                         /* Convert from Gray code */
Move(pos,Sreg)                       /* Readout pos loga */
SAVE                                 /* Shift out position */
Move(Max,Sreg)                       /* Readout Max b */
SAVE                                 /* Shift out intensity */
```

Since we need time to A/D convert the next row while processing the present one the time for an *NxN* image with the range resolution *a* is approximately

$$T_{im} = NMax\left(\frac{N}{f_s} + \frac{\log a}{f_p}, aMax\{\frac{2^b}{f_{AD}}, \frac{6b + 3\log a + 2}{f_p}\} + \frac{b + 2\log a}{f_p}\right) \quad (7.6)$$

If we choose $a = 64$ and $b = 8$ the A/D conversion sets the speed limit with an image frequency of 4.75 Hz. The range and intensity pixel frequency is approximately 310 KHz. If we evaluate equations (7.3) and (7.6) we note that if the range accuracy needed is more than 6 bits the vertical algorithm is faster, but for 6-bit accuracy and less the horizontal algorithm is preferable. Note that if the intensity as well as the range is to be read out the limit-frequency is halved (~ 5MHz), and that no error and peak width corrections are implemented.

7.9 Horizontal thresholding algorithm

In this implementation, previously described in [34], [32] each PE finds the row position of the first pixel above the threshold (= the start position of the peak) by counting the number of rows before any pixel is above the threshold, and the row position of the last pixel above the threshold (= the end of the peak) by counting the number of rows before the end of the first peak is found. The peak position is then computed with half pixel sub-pixel accuracy as the sum (mean) of the two counters.

In pseudo code the horizontal thresholding algorithm is described as follows

```
        /* Horizontal thresholding sheet-of-light algorithm */
SETV volt                            /* Set threshold voltage */
Count1 = Count2 = 0                  /* Reset counters 2logn */
START = 1                            /* Reset Mask bits */
ISTOP = 0
ERR = 0
SETR x                               /* Set row */
for (i=x;i<y;i++) {                  /* Read a = (x-y) rows */
   INTEG                             /* Read analog row value */
   LD PD                             /* Read inverted row */
   RESET,+                           /* Reset PD, new row
   ST Rx(i)                          /* Store inv. row */
                        /* Peak start counter */
   AND START
   ST START
   XOR Count1(i)                     /* Change counter bit */
   ST Count1(i)                      /* Counter Gray coded */
                        /* Peak end counter mask */
```

```
   LD Rx(i)                        /* Load invert row */
   AND Rx(i-1)                     /* And prev row */
   OR ISTOP                        /* Or with stop */
   ST ISTOP                        /* Save stop */
                        /* Error filtering */
   AND Rx(i)
   OR ERR
   ST ERR
                        /* Increment end counter */
   LDI ISTOP                       /* Read count 2 cond */
   XOR Count2(i)                   /* Change counter bit */
   STORE Count2(i)
}
Convert(Count1)                    /* from Gray 2loga */
Convert(Count2)                    /* from Gray 2loga */
Add(Count1,Count2,Sum)             /* add 3loga */
Move(Sum,Sreg)                     /* Load sreg loga */
sreg(8) = ERR                      /* Load err bit to sreg */
SAVE                               /* Shift out row */
```

The START bit in each PE is used to detect the beginning of a peak. If a peak is found it will be set to *zero* for the duration of the sensor readout. When the START bit is *zero* no start increment of *Count1* is possible in this PE and *Count1* will retain the peak start position value. The ISTOP bit in each PE is used to detect the end of a peak. If the end of a peak is found ISTOP is set to *one* for the duration of the sensor readout. When ISTOP is *one* no increment of *Count2* is possible and *Count2* will retain the peak stop position value. In the code we output the error value, but it is also possible to make the error filtering in MAPP, i.e. use the error bit to AND with the data bits. This requires approximately log*a* cycles.

Since we need time to shift out the result of the previous sensor processing while processing the previous the time for an *NxN* (N = 256) image with the range resolution 2log*a* is approximately

$$T_{im} = N\left(Max\{\left(\frac{N}{f_S} + \frac{1+\log a}{f_p}\right)\left(\frac{18a}{f_p} + \frac{9\log a}{f_p}\right)\} \right) \tag{7.7}$$

For n = 128 (8 bit range accuracy) the range image frequency is 13 Hz and the rangel frequency 860 KHz, i.e. the speed is very similar to the vertical implementation. However, for lower range accuracy than 8 bits the horizontal algorithm has higher speed. For instance, if the range accuracy is 6 bits the rangel frequency is 3.3 MHz. Also, when the accuracy is less than 5 bits it is possible to output 2 rangels during each 8-bit shift, so the rangel frequency can actually increase above the 10 MHz shiftregister transfer frequency.

7.10 Horizontal thresholding with USM

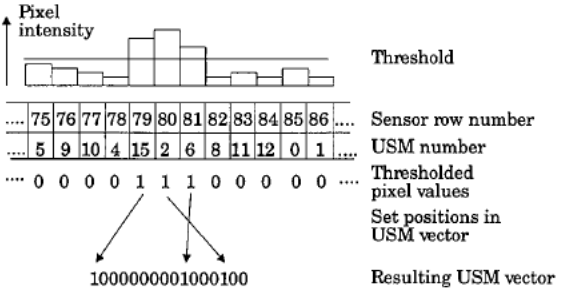
The Unique Sequence Method, USM, was originally described by Åstrand et. al. in [66] and [67]. Although it is technically a horizontal thresholding algorithm, the USM implementation is very different from the previous algorithms, giving radically different performance.

To understand how the USM algorithm works, study Figure 7.12. With each sensor row we have associated a *USM number*, say between 0 and 15. The sequence of USM numbers is called the *USM sequence*. How the USM sequences can be generated will be discussed below.

Now, if we threshold a peak which covers n consecutive pixels we thereby select n USM numbers from the sequence. Due to the characteristics of the USM sequence the combination of USM numbers thus selected is unique, and can be used to determine the peak position.

When implementing USM the USM numbers are represented by bit positions in a 16 bit word, called the *USM vector*. When thresholding each sensor row we thereby obtain a bit indicating if this bit in the USM vector should be set or not, and by OR-ing this bit with the previously stored bit we obtain a new bit value in the USM vector. The peak position indicated by the USM vector after all rows have been thresholded can then be found by a look-up table.

FIGURE 7.12.



The USM peak detection.

USM sequence generation

We have generated USM sequences using a computer program. It iteratively tries to add new USM numbers (i.e. more sensor rows) to the generated sequence while ensuring the uniqueness of the USM vectors generated by all possible peak positions in the sequence. The parameters that control the sequence generation is the length of the USM vector, i.e. the number of bits in the code, and the minimum and maximum peak widths n that should be possible to detect. The program then tries to find the longest possible sequence of USM numbers within the given constraints. There is no guarantee that the generated sequence is optimal, but we believe that no significantly longer sequences will be possible for a given set of input data. The sequence generation is covered in more detail in [66].

Ideally we want to be able to detect peaks with a variety of widths using as few bits in the vector as possible. Using few bits in the vector facilitates the conversion from the USM vector to the peak position. So far we have not implemented this conversion in MAPP2200, instead all USM vectors are output to the controller where a table lookup is performed. Since the MAPP2200 shiftregister is eight bits wide it seems natural to chose USM vector lengths as multiples of eight, and since 8 bits gives very short sequences we use 16 and 24 bit vectors. Data for four generated sequences can be found in Table 7.1. We see that for a 16 bit USM vector we can choose between detecting 3-9 pixels wide peaks using 182 sensor rows or 5-11 pixels wide peaks over the whole sensor. Using a 24 bit USM vector we can detect peak widths down to 2 pixels. The first sequence in the table is the one we have used in our experiments.

TABLE 7.1.

Sequence Length	USM vector length	Minimum peak width	Maximum peak width
182	16	3	9
256	16	5	11
256	24	2	9
232	24	2	11

Since the number of valid USM vectors is around 10% the possible vectors the inversion lookup tables contains a large number of non-valid entries. This gives a limited form of error detection capability. If the peak is thinner than the minimum width it is always guaranteed that no valid USM vector is generated, but if the peak is wider or a noise induced pixel position is set to one the generated USM vector may or may not be part of the valid set. In our experiments we have verified that the probability that a correct USM vector is generated from an incorrect peak is very small.

USM implementation

USM can be implemented on the MAPP2200 sensor in several ways. The most straightforward is to address the sensor rows in sequential order and threshold each row with the pre-specified threshold. The resulting bit is then OR-ed with the data in the USM vector position given by the row number. In pseudo code this will be

```
/* USM Implementation 1 */
/* tab[i] contains the USM number for row i */
Reset(R)                          /* Reset the USM vector, L */
SETR 0                            /* Set start row */
for (i=0;i<a;i++) {
   INTEG                          /* Read pixel value */
   LDI PD                         /* read thresholded bit */
   RESET,+                        /* Reset pixel */
   OR R(tab[i])                   /* Or with USM bit in RAM */
   ST R(tab[i])                   /* Store new bit in RAM */
}
Move(R,Sreg)                      /* move vector to sreg */
SAVE                              /* Shift out USM vector */
```

This implementation requires five MAPP2200 instructions in the inner loop. However, by utilizing the capability of analogue addition of charge from several rows before the thresholding we can implement the algorithm in a much more effective way. If we consider the charge in a pixel as zero if the peak does not illuminate that pixel and above the threshold if it does, then we can add the charge from all pixels who have the same USM number by addressing the corresponding sensor rows, and then threshold the result, getting the logic OR of the thresholded sensor rows as before. Thus, instead of the table with the USM sequence we generate a table which contains the row indices for the rows with USM number zero, the rows with indices one and so on. This table is then used in the implementation which looks like

```
/* USM Implementation 2 */
/* tab[i][0..(J-1)] contains the row numbers for
       the USM number i */
/* The number of rows in each USM bin is J */
Reset(R)                          /* Reset the USM vector, L */
for (i=0;i<L;i++) {    /* for all L USM numbers */
   RESET                          /* Reset PD register */
   SETR(tab[i][0])                /* select first row */
   INTEG                          /* Start integration */
   for (j=1;j<J;j++) {
       SETR(tab[i][j])            /* Select rows */
   }
   ST PD, R(i)                    /* Store thresholded bit */
}
Move(R,Sreg)                      /* move vector to sreg */
SAVE                              /* Shift out USM vector */
```

Using this implementation there is only one MAPP2200 instruction in the inner loop and the time to acquire one *NxN* range image using an *L* bit code word is approximately

$$T_{im} = N\left(Max\left(\left\lceil\frac{L}{8}\right\rceil\cdot\frac{N}{f_s}, \frac{a+5L}{f_p}\right)+\frac{L}{f_p}\right) \tag{7.8}$$

where a is the number of rows in the sequence. If a is 182 and L is 16 the range image frequency is 74 Hz and the range pixel frequency 4.8 MHz.

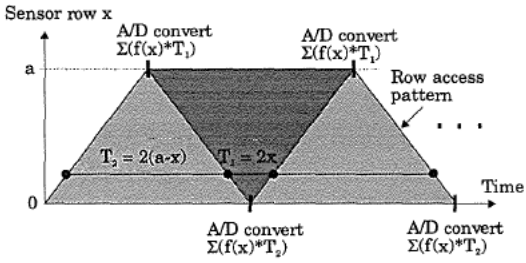
USM drawbacks

A large drawback of the USM algorithm is the limited peak widths that can be detected. This sets a limit in how much the reflectivity of the viewed object can change. Thus, the USM is not suited for applications where the object reflectivity changes very much. Also, the use of analog addition before the thresholding makes the algorithm more sensitive to background noise. In this implementation the shiftregister communication is the bottleneck, not the processing. Thus, if an effective way of translating the USM vector to a peak position can be implemented in MAPP2200 the performance can increase the range pixel speed up to almost 8 MHz.

7.11 Horizontal center-of-gravity algorithm FBM

In [31] we presented a straight-forward implementation of a center-of-gravity peak finding algorithm. The implementation has very low performance, less than 100 KHz rangel frequency, and has never been implemented. The Forward Backward Method, FBM, however is a very effective way to implement a horizontal center-of-gravity algorithm on MAPP2200. FBM was originally described by Åstrand et. al. in [66] and [67]. Just as the USM it utilizes the ability to add charge from several sensor rows before thresholding or A/D conversion. To understand the FBM study Figure 7.13. We scan the sensor rows in a forward-backward pattern, adding analog charge from all accessed rows, and at each turning point the collected intensity sum is A/D converted.

FIGURE 7.13.



The row access pattern and A/D conversion in the FBM.

For each row x the integration time is alternating between $2x$ and $2(a-x)$. The intensity sum achieved after a forward scan written as an integral is

$$S_1 = \int 2xf(x)\,dx \tag{7.9}$$

and correspondingly the intensity sum achieved from a backward scan is

$$S_2 = \int 2(a-x)f(x)\,dx \tag{7.10}$$

Now, if we combine these two sums we get

$$S_1 + S_2 = \int 2xf(x)\,dx + \int 2(a-x)f(x)\,dx = 2a\int f(x)\,dx \tag{7.11}$$

and thus we can compute the center-of-gravity position of $f(x)$ using

$$P = \frac{\int xf(x)\,dx}{\int f(x)\,dx} = a\frac{S_1}{S_1 + S_2} \tag{7.12}$$

In pseudo code the FBM implementation looks like

```
/* Horizontal center-of-gravity using FBM */

RESET
laser(on)
INTEG                              /* Start integration */
for (i=0;i<a;i++) {                /* Forward scan */
   SETR i                          /* Access gives integration */
}
laser(off)
HOLD
INITAD                             /* A/D convert S1 */
WAITAD
Move(AD,Sreg)
SAVE
RESET
laser(on)
INTEG                              /* Start integration */
for (i=a-1;i>=0;i--) {
   SETR i
}
```

```
laser(off)
HOLD
INITAD                                  /* A/D convert S2 */
WAITAD
Move(AD,Sreg)
SAVE
```

If the position P is to be calculated within MAPP, a bit-serial division is required which takes approximately $6b^2$ cycles where b is the number of bits. To avoid this the two intensity values are shifted out and the range values are then computed in the controller. A feature is that intensity data, i.e. the sum of the two components, is acquired as well as range data. Note that we modulate the laser to avoid integration during the A/D conversion.

The time for an NxN range image is then approximately

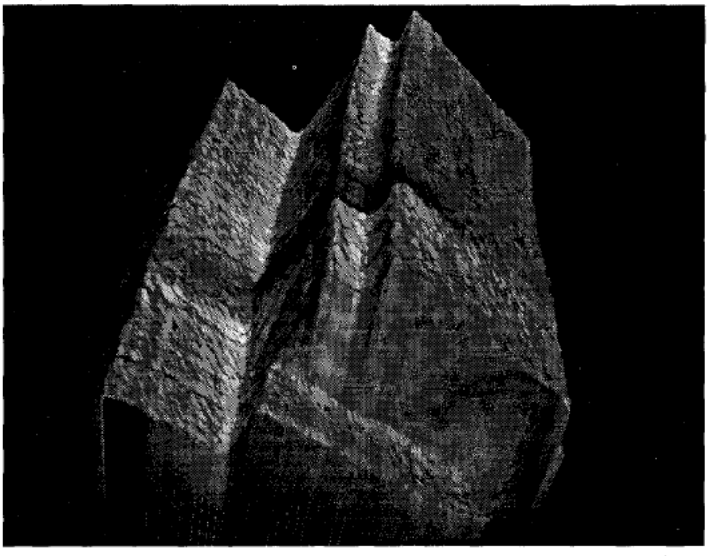
$$T_{im} = N\left(max\left(\frac{2N}{f_s}, \left(\frac{2a+18}{f_p} + \frac{2\times 2^b}{f_{AD}} \right) \right) + \frac{2b}{f_p} \right) \tag{7.13}$$

The data output using the shiftregister limits the rangel frequency to 4.8 MHz if the maximum A/D converter precision is used. Actually, by computing a new range vector for each forward or backward scan readout the rangel frequency can be doubled, giving 9.6 MHz rangel frequency. To reduce the noise sensitivity we can apply an image subtraction using an image acquired without the laser illumination. This requires two additional scans and A/D conversions, and two bit-serial subtractions. Since the computations can be performed concurrently with the A/D conversions and shifts, and the maximum A/D converter frequency is more than double the shift frequency, the performance loss for the image subtraction is very low.

FBM has a large potential for sub-pixel accuracy peak detection. The limiting factor is the A/D conversion, limited to eight bits, which constrains the theoretical position resolution. According to Matlab simulations around 9 bits of accuracy can be attained. However, this resolution is independent of the number of sensor rows actually used. Experiments have shown that around eight bits of resolution can be obtained using as few as eight sensor rows. In Figure 7.14 we give an example of an FBM image with eight bit resolution acquired using FBM on 16 sensor rows.

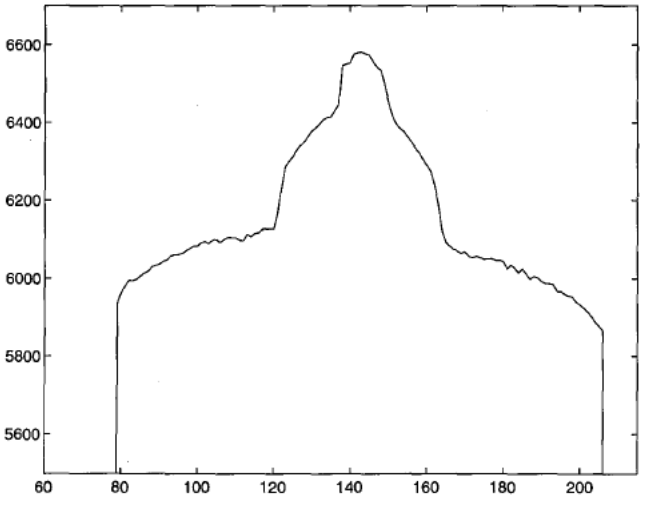
Now, since we can achieve around eight bits of resolution using FBM on a few sensor rows FBM has potential for very high accuracy if several consecutive scans over different parts of the sensor are combined. That is, we obtain a coarse resolution by noting at which row interval the laser illumination was detected, and a fine resolution using the output from this interval. An example where FBM has been used iteratively over the sensor is shown in Figure 7.15. 32 FBM scans over the sensor has been combined into one complete range profile with 5+8 = 13 bit resolution. We observe some noisy areas which occur when the scale factor a does not match between the different iterations. Part of this comes from the fact that a depends on the width of the laser line, and due to the perspective projection this width changes over the sensor. Thus it is very hard to trim the transitions between different scans. Note that the ranging speed is reduced proportionally to the number of scans over the sensor.

FIGURE 7.14.



A 3D plot of an FBM range and intensity image acquired using 16 sensor rows.

FIGURE 7.15.



An FBM profiles with 13 bit resolution. Compare with the profile in Figure 7.10.

7.12 Horizontal integrating algorithm I, TIM

The Total Integration Method, TIM, was originally described by Åstrand et. al. in [66]. Implementation-wise it is very similar to the horizontal thresholding algorithm without peak-width and error computation. However, instead of resetting the readout register after each row access the charge is added and we compare the sum with the threshold. If we consider the rows not illuminated by the laser as containing zero charge we get the same algorithm as with horizontal thresholding. Furthermore, after the scan of all rows the sum of all row charges is found in the registers, and after A/D conversion we have the total integrated column intensities. Thus we get both range and intensity information. In pseudo code the algorithm looks like

```
/* Horizontal total integrating method, TIM */
/* Gray(i) returns the correct counter bit pos */
Count = 0                          /* Reset position counter */
SETV volt                          /* Set threshold voltage */
SETR x                             /* Select start row */
RESET                              /* Reset readout register */
for (i=0;i<(y-x);i++) {            /* Read a = y-x rows */
   INTEG,+                         /* Access row i */
   LD PD
   XOR R(Gray(i))                  /* Change counter bit */
   ST A,R(Gray(i))
}
HOLD
SETV 0                             /* Reset A/D Start value */
INITAD                             /* A/D conversion */
Convert(Gray)                      /* 2loga */
Move(Counter,Sreg)
SAVE
```

The time to acquire one NxN image of range and intensity data is

$$T_{im} = Nmax\left(\left(\frac{b+\log a}{f_p}+\frac{2N}{f_{sh}}\right),\left(\frac{6+3\log a+4a}{f_p}+\frac{2^b}{f_{AD}}\right)\right) \tag{7.14}$$

and if only range data are acquired this is reduced to

$$T_{im} = Nmax\left(\left(\frac{\log a}{f_p}+\frac{N}{f_{sh}}\right),\left(\frac{2+3\log a+4a}{f_p}\right)\right) \tag{7.15}$$

For $b=8$ and $a = 256$ the range pixel frequency is almost 2 MHz (1.8 MHz if the intensity also is read out). If the resolution is reduced to 6 bits or less the shiftregister communication limits the performance and the rangel frequency increases to almost 10 and 5 MHz respectively.

Although TIM is very fast it has several drawbacks. It is very sensitive to background illumination since the total column charge is compared with the threshold, and the acquired peak position depends heavily on the peak width, and therefore on the peak intensity. If the peak intensity is acquired it might be possible to compensate for this, but no accurate compensation scheme has been devised. Despite all this the image quality can be quite good. We have successfully experimented with increasing the threshold incrementally during each sensor scan to reduce the sensitivity to accumulated background illumination noise.

7.13 Horizontal integrating algorithm II, OEM

The Odd Even Method, OEM, was originally described by Åstrand et. al. in [66] and is an extension of the TIM. In OEM we integrate the odd and even lines separately in order to overcome the intensity dependance and to extract some sub-pixel information. The algorithm works in two phases, one for odd rows and one for even rows. Each phase is identical to the TIM algorithm, the difference being that the even rows are accessed in reverse order. By reversing the scan order we get one measurement for each edge of the peak. It is then possible to approximate the peak position with their mean value. Since we still integrate all pixel charges we can find the total intensity by adding the intensities from the two passes. In pseudo code the implementation looks like

```
* Horizontal total integrating method, TIM */
/* Gray(i) returns the correct counter bit pos */
Count_o = Count_e = 0              /* Reset counters */
SETV volt                          /* Set threshold voltage */
RESET                              /* Reset readout register */
INTEG                    /* Start the even integration */
for (i=0;i<(y-x);i=i+2) {          /* a = (y-x) */
   SETR x+i                        /* Access row */
   LD PD
   XOR R(Gray_o(i))
   ST A,R(Gray_o(i))
}
HOLD
SETV 0                             /* Reset A/D Start value */
INITAD                             /* A/D conversion */
WAITAD
Move(AD,Reg1)

SETV volt                          /* Set threshold voltage */
RESET                              /* Reset readout register */
INTEG                    /* Start the odd integration */
for (i=0;i<(y-x);i=i+2) {          /* Read a = y-x rows */
   SETR y-i/* Access row */
   LD PD
   XOR R(Gray_e(i))
   ST A,R(Gray_e(i))
}
HOLD
SETV 0                             /* Reset A/D Start value */
INITAD                             /* A/D conversion */
WAITAD
Move(AD,Reg2)
Convert(Gray_o)                    /* Convert counters */
Convert(Gray_e)
Add(Odd,Even,Sum)                  /* Mean of counters is peak */
Move(Sum,Sreg)                     /* Shift out mean peak pos */
SAVE
Add(Reg1,Reg2,Sum)                 /* Add intensities */
Move(Sum,Sreg)                     /* Shift out intensity sum */
SAVE
```

The time to acquire one range image using OEM is then

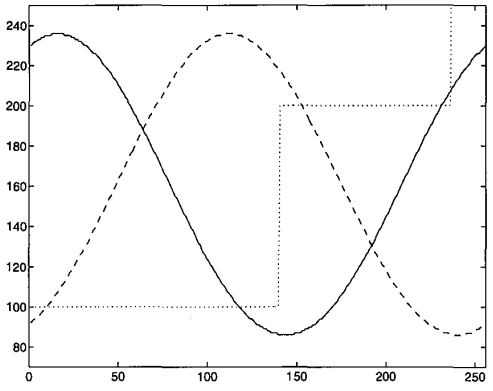
$$T_{im} = Nmax\left(\left(\frac{b+\log a}{f_p}+\frac{2N}{f_{sh}}\right),\left(\frac{14+2b+4a}{f_p}+\frac{2\cdot 2^b}{f_{AD}}\right)\right) \quad (7.16)$$

The OEM performance is slightly lower than for the TIM, and for b=8 and a = 256 the range pixel frequency is 1.8 MHz (1.5 if the intensity is readout as well). For less than 6 bit resolution the shiftregister limits the performance and the rangel speeds are close to 10 and 5 MHz respectively.

In the pseudo code we compute the intensity sum and transfer that out to the controller. However, it might be advantageous to shift out the intensity and peak position components instead as they contain sub-pixel and error information. If the two positions indicated by the counters are dissimilar one of them is probably affected by noise and the absolute difference can be thresholded into an error value.

That the intensity components can be used for sub-pixel information can be understood by noting the following. Assume that the peak is approximately one pixel wide. Then, if the intensity components are equal the peak is centered between two pixels, and if the even sum is much larger than the odd sum the peak is centered around an even pixel. A simulation of the intensity components as a function of the peak position can be found in Figure 7.16. In the simulation we have used MAPP2200 sensor geometry, and the uneven row distance is the cause of the non symmetric intensity curves. Now, if we can recover the "phase" of the two components we should be able to estimate the peak position more accurately.

FIGURE 7.16.

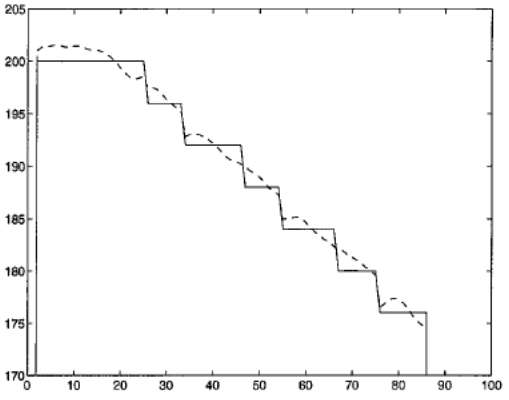


Odd and even pixel intensities for different peak positions. The integer peak position increment is shown amplified.

In Figure 7.17 we show the result when we have used the intensity components to try to recover sub-pixel accuracy. In the figure we see that the method fails at every second integer transition, but otherwise the results looks good. This is because the acquired sub-pixel information and the integer position values are "out-of-phase", i.e. the change in integer position does not occur simultaneously with the change from plus to minus sub-pixel information. In this example we have managed to select the threshold so that one transition is in-phase with the sub-pixel information but the other is not. Thus, it is hard to use the intensity components for

sub-pixel accuracy ranging. However, if a continuous surface is viewed any change in distance may be determined with high sub-pixel accuracy from the intensity components, but not the absolute peak position. Also the amplitude of the sinusoids depends heavily on the peak width, which makes the method unsuitable for use with materials having large intensity variations. Actually, if we only study the intensity components the results are very similar to the FBM results, but in the FBM the unsymmetrical sensor row positioning gives no non-symmetry in the components since the odd and even sensor rows not are processed separately.

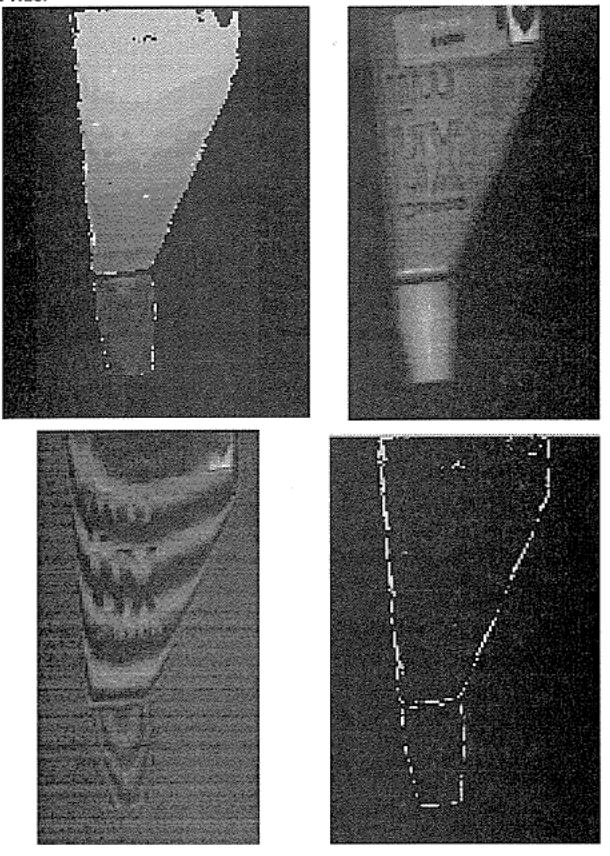
FIGURE 7.17.



The integer and recovered sub-pixel range for a range profile. Note that the predicted non-symmetry of the integer position increments is clearly visible.

Four image results from one OEM range image acquisition is shown in Figure 7.18.

FIGURE 7.18.



Range (top left), intensity (top right), sub-pixel information (bottom left) and error (bottom right). The sub-pixel information is the difference between the intensity components and the error image is the difference between the range components.

7.14 Multiple peak readout

Multiple peak readout can easily be incorporated into the MAPP2200 vertical sheet-of-light programs using the PEs to mask out the peaks from the binary position vector one after the other as in the code example below.

```
    /* Multiple peak redout from MAPP2200,binary peak row in Acc*/
ST Acc, R0              /* Store temporary row */
while (GOR)             /* While any peak left */
   LFILL R0             /* Detect leftmost pixel */
   LD GLU
   pos[i] = COUNT       /* Read position */
   LEDGE                /* Find leftmost pixel */
```

```
    MARK R0              /* Mask out leftmost peak */
    LD GLU
    width[i] = COUNT     /* Read peak width */
    XOR R0               /* Remove peak from temporary row */
    ST Acc, R0           /* New row as temporary */
  endwhile
```

The width and position data is in fact the run-length code of the thresholded row, and the extra execution time is approximately 1.5 μs for each additional peak readout (at a PE clock frequency of 8 MHz).

7.15 RANGER RS2200

The RANGER RS2200 range sensor is a MAPP2200 sensor with sheet-of-light range imaging software and it has been commercially available from IVP since mid-1994. So far systems have been sold in for instance Sweden, USA, Italy and Israel.

The system is based on the results presented in [31] and it incorporates the vertical algorithms in section 7.3 - section 7.6, i.e. NSIP maximum finding, successive approximation maximum finding, and thresholding algorithms. Both the successive approximation and the thresholding algorithm support multiple thresholdings to increase the range resolution. The system also contains tool functions for system accuracy prediction, calibration and range image viewing. With the PC version maximum rangel frequency of between 90-150 KHz can be obtained, and with the Smart Camera version the maximum rangel frequency is approximately 200 KHz.

Chapter 8

Comparisons and conclusions

In this part of the thesis we have presented a total system overview of sheet-of-light range imaging. We have covered geometric design and calibration as well as several algorithms. A novel smart sensor architecture for range imaging called NESSLA has been presented, and several algorithm implementations have been studied using the MAPP2200 sensor. Here we give a comparison between our results and those for other smart sensor implementations presented in the literature.

The *range pixel speed* differences among the presented smart sensor architectures are not very large. Most of them are capable to output position data at around 1-2 MHz, which is 50-100 times faster than what is possible with a standard video sensor. The best predicted performance, up to and above 5-10 MHz, was found for the NESSLA architectures and for certain MAPP2200 implementations.

The NESSLA architecture has digital data transports from the pixel cells which should be less *noise sensitive* than the analog transfers in all other designs. However, the extensive use of logic in the pixels makes it hard to have a large sensor aperture, and thus the light sensitivity might suffer. The NESSLA design automatically adapts for different light intensities and therefore the risk for sensor saturation is very low. Sensor saturation can otherwise be a problem, especially when we get specular reflections of the laser light.

The Analog processing sensor cells architecture is the most *complex* design with the analog signal processing in each sensor cell. The NESSLA design is also quite complex but we believe that it is possible to fabricate a working 256^2 sensor with standard 0.8 μm CMOS VLSI process design. Furthermore, the NESSLA design does not need any analog hardware outside the sensor area which makes it rather VLSI friendly. The PSD designs seems to be easy to implement but they need extra analog processing to obtain the offset (range) data from the sensor outputs. The analog bottleneck in the previously presented designs can be avoided by using the design with on-chip A/D conversion presented here. Both the Row readout successive approximation and Column readout peak detection architectures utilize a standard sensor area and are therefore not very complex to implement.

The NESSLA architecture, the Column read-out peak detection architecture and several of the MAPP2200 implementations can all output grey level *intensity images* concurrently with the range data. Theoretically, the PSD architecture can output reflectance maps, but no reports of this are available. Intensity information is very useful both as image information and as a confidence measurement.

We believe that a sheet-of-light range imaging system using the NESSLA architecture can output range images with range image frequency and accuracy comparable to or better than the architectures found in the literature. NESSLA also has a built-in error correction which does not exist in any of the others except some of the MAPP2200 implementations. The Column readout architecture can also be very effective and it is fairly easy to implement in VLSI.

The MAPP2200 implementations are maybe not as fast as the special purpose architectures for the same accuracy, but the overwhelming advantage of this sensor is that it is commercially available. With MAPP2200 a sheet-of-light range camera system can be built today, whereas

the other designs are prototypes at the best. Also, with MAPP2200 it is possible to trade resolution for speed and obtain low resolution images at high speed. An additional feature of MAPP2200 is its general programmability which makes it possible to implement not only the sheet-of-light range finding algorithm but also subsequent processing of range data as has been shown in a can sorting application example [30]. This alleviates the need for extra hardware and makes the overall system installation easier

References

[1] Andersson (Gökstorp) M.J., *Range from Out-of-Focus Blur,* Lic. Thesis. LIU-TEK-LIC-1992:17, Linköping 1992.

[2] Araki K.,Shimizu M., Noda T., Chiba Y., Tsuda Y., Ikegaya K., Sannomiya K., Gomi M., *A range finder with PSD's*, Machine Vision and Applications, MVA '90, Tokyo 1990.

[3] Araki K., Tsuda Y., Shimizu M., Ikegaya K., Noda T., Sannomiya K., Chiba Y., Gomi M., *High Speed and Continuous 3-D Measurement System, 11th Int.* Conf. on Pattern Recognition, Vol 4, Hague 1992.

[4] Beeler T.E., *Producing Space Shuttle Tiles with a 3-D Non-Contact Measurement System,* In "Three-Dimensional Machine Vision" ed. by T Kanade, Kluwer Academic Publishers 1987.

[5] Beraldin J.A., Rioux M., Blais F., Cournoyer L., Domey J., *Registered intensity and range imaging at 10 mega-samples per second,* Optical Engineering 31 (1), pp 88-94, 1992.

[6] Besl P. J., *Active Optical Range Imaging Sensors*, Advances in Machine Vision, Ch. 1, Springer, 1988.

[7] Blais F., Rioux M., *Real-time numerical peak detector*, Signal Processing 11, pp 145-155, North Holland, 1986.

[8] Bracewell R.N, *The Fourier transform and its applications,* McGraw-Hill, 1986.

[9] Burke M. et. al., *RST, Road Surface Tester*, Report from OPQ systems, Linköping 1991.

[10] Case S.K. Jalkio J.A. Kim R.C., *3-D Vision System Analysis and Design,* In "Three-Dimensional Machine Vision" ed. by T Kanade, Kluwer Academic Publishers 1987.

[11] Danielsson P.E., *Smart Sensors for Computer Vision*, Report No. LiTH-ISY-I-1096, Linköping 1990.

[12] Eklund J.E., *Oral communication,* Linköping 1995.

[13] Eklund J.E., Svensson C., Åström A., *Near-Sensor Image Processing, a VLSI Realization,* Proc of Transducers95, Eurosensors IX, Stockholm, 1995.

[14] Eklund J.E., Svensson C., Åström A., *Near-Sensor Image Processing, a VLSI Realization,* Proc of 8th Annual IEEE Int. ASIC Conf., Austin, 1995.

[15] Erhardt, *Silicon Cylindrical Lens Arrays for Improved Photoresponse in Focal Plane Arrays,* SPIE Vol. 501, 1984.

[16] Fischer H., Lulé T., Schneider B., Schulte J., Böhm M., *Analog image detector in Thin Film on ASIC (TFA)-technology,* EUROPTO - Sensors and Control for Advanced Automation II, Frankfurt, 1994.

[17] Forchheimer R, Ingelhag P, Jansson C, *MAPP2200, a second generation smart sensor*, SPIE vol 1659, 1992.

[18] Forchheimer R., Åström A., *Near-Sensor Image Processing. A New Paradigm,* IEEE trans. on Image Processing, vol. 3, no. 6, november 1994.

[19] Forchheimer R., Ödmark A., *Single chip linear array processor*, Applications of digital image processing, SPIE, Vol. 397, 1983

[20] Garcia D. F., del Rio M. A., Diaz J. L., Fransisco J. S., *Flatness defect measurement system for steel industry based on a real-time linear-image processor,* Proc of IEEE/SMC'93, Le Touquet 1993.

[21] Gruss A., Carley R., Kanade T., *Integrated Sensor and Range-Finding Analog Signal Processor,* IEEE Journal of Solid State Circuits, Vol 26, pp 184-191, 1991.

[22] Gökstorp M., *Range Computation in Robot Vision,* Ph.D. Thesis No. XX, Linköping, 1995.

[23] Haralick R.M., Shapiro L.G., *Computer Vision, Vol II,* Addison Wesley, 1993.

[24] Horn B. K. P., *Robot Vision,* MIT press 1986.

[25] Ingelhag P., *A contribution to high performance CMOS circuit techniques*, report LiU-TEK-LIC-1992:35, Linköping, november 1992.

[26] Jansson C, *Oral communications*, Linköping september 1993.

[27] Jansson C., *CMOS Signal Conditioned Sensor Arrays and A/D-D/A Conversion Techniques,* Ph.D. thesis No. 377, Linköping, 1995.

[28] Jiang X-Y, Gunke H, *Quantization Errors in Active Range Sensing*, Proc of SCIA-93, pp 913-920, Tromsø 1993.

[29] Johannesson M., *Sheet-of-Light range imaging experiments with MAPP2200,* Report No. LiTH-ISY-I-1401, Linköping 1992.

[30] Johannesson M, *Can Sorting Using Sheet-of-Light Range Imaging and MAPP2200*, Proc of IEEE/SMC'93, Le Touquet 1993.

[31] Johannesson M., *Sheet-of-light Range Imaging,* Lic thesis No. 404, LIU-TEK-LIC-1993:46, Linköping, 1993.

[32] Johannesson M., *Fast, programmable, sheet-of-light range imager using MAPP2200*, Proc. of San Diego SPIE '94, San Diego, 1994.

[33] Johannesson M.,*Calibration of a MAPP2200 sheet-of-light range camera,* Proc of SCIA'95, Uppsala, 1995.

[34] Johannesson M., Åstrand E., Åström A., *A Fast and Programmable Sheet-of-light Range Imager Using MAPP2200*, Proc. of SSAB '94, Halmstad, 1994.

[35] Johannesson M, Åström A, *Sheet-of-Light range imaging with MAPP2200,* Proc of SCIA -93, pp 1283-1290, Tromsø 1993.

[36] Johannesson M., Åström A., Danielsson P.E., *An Image Sensor for Sheet-of-Light Range Imaging*, proc of MVA '92, Tokyo 1992.

[37] Johannesson M, Åström A, Ye Q.Z, Danielsson P.E., *Architectures for Sheet-of-Light Range Imaging,* Report No. LiTH-ISY-I-1335, Linköping 1992.

[38] Kanade T, Gruss A, Carley L.R, *A Fast Lightstripe Rangefinding System with Smart VLSI Sensor,* Machine Vision for Three-Dimensional Scenes, Ed. H Freeman, Academic Press, 1990.

[39] Knutsson H., Westin C.F., *Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data,* Proc of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, New York, 1993.

[40] *MAPP2200 Technical Description*, IVP Integrated Vision Products AB, Linköping,1992.

[41] Melen T., Sommerfelt A., *Calibrating Sheet-of-Light Triangulation Systems,* Proc of SCIA '95, Uppsala 1995.

[42] *Micro-Optics Monolithic lenslet modules,* Prospect from United Technologies Adaptive Optics Associates, 1992.

[43] Moss J.P., McCane M., Fright W. R., Linney A. D., James D. R., *A three-dimensional soft tissue*

analysis of fifteen patients with Class II, Division 1 malocclusions after bimaxillary surgery, Am. Jour. of Orthodontics and Dental Orthopedics, Vol 105, 1994.

[44] Mundy J.L, Porter G.B. III, *A Three-Dimensional Sensor Based on Structured Light,* In "Three-Dimensional Machine Vision" ed. by T Kanade, Kluwer Academic Publishers 1987.

[45] Nakagawa Y, Ninomiya T, *Three-Dimensional Vision Systems Using the Structured-Light Method for Inspection Solder Joints and Assembly Robots,* In "Three-Dimensional Machine Vision" ed. by T Kanade, Kluwer Academic Publishers 1987.

[46] Nygårds J, Wernersson Å, *Gripping Using a Range Camera: Reducing Ambiguities for Specular and Transparent Objects,* Proc from Robotikdagar, Linköping 1993.

[47] Poussart D., Laurendeau D., *3-D Sensing for Industrial Computer Vision,* Advances in Machine Vision, Ch. 3, Springer, 1988.

[48] *RANGER RS2200 Technical Description,* IVP, Integrated Vision Products, Linköping, 1994.

[49] Rioux M, *Laser Range finder based on synchronized scanners,* In "Robot sensors Vol 1 Vision" ed. by A Pugh, Springer Verlag 1986.

[50] Rowa P., *Automatic visual inspection and grading of lumber,* Seminar/Workshop on scanning technology and image processing on wood, Skellefteå, 1992.

[51] Saint-Marc P., Jezouin J.L., Medioni G., *A Versatile PC-Based Range Finding System,* IEEE Trans on Robotics and Automation, Vol 7, No 2, 1991.

[52] Sato K, *Oral communications,* Osaka december 1992.

[53] Stuivinga M., Nobel J., Verbeek P.W., Steenvoorden G.K., Joon M.M., *Range-Finding Camera Based on a Position-Sensitive Device Array,* Sensors and Actuators 17, pp 255-258, 1989.

[54] Technical Arts Corp, *White Scanner 100a User's Manual,* Seattle, USA 1984.

[55] Theodoracatos V, Calkins D, *A 3-D Vision system Model for Automatic Object Surface Sensing,* In "Journal of Computer Vision", 11-1, pp 75-99, 1993.

[56] Toborg S., *A 3-D Wafer Scale Architecture for Early Vision Processing,* Proc. of Int. Conf. on Application Specific Array Processors, pp 247-258, Princeton 1990.

[57] Trucco E., Fischer R. B., *Acquisition of Consistent Range Data Using Local Calibration,* Proc of IEEE Int Conf. on Robotics and Automation, San Diego, 1994.

[58] Veldkamp W., *Binary Optics,* McGraw-Hill Yearbook of Science and Technology 1990, McGraw-Hill 1991.

[59] Verbeek P, *Oral communications,* Linköping april 1993.

[60] Verbeek P.W., Nobel J., Steenvoorden G.K., Stuivinga M., *Video-speed Triangulation Range Imaging,* NATO ASI Series, Vol F 63, pp 181-186, Springer-Verlag Berlin, Heidelberg 1990.

[61] Verbeek P.W., Groen F.C.A., Steenvoorden G.K., Stuivinga M.E.C., *Range cameras at video speed based on a PSD-array or on CCD-interpolation,* Proc. of IASTED Int. Symp. ROBOTICS AND AUTOMATION, pp 67-69, Lugano 1985.

[62] Weste N., Eshraghian K., *Principles of CMOS VLSI Design,* Addison Wesley, 1985.

[63] Westelius C-J, *Preattentive Gaze Control for Robot Vision,* LIU-TEK-LIC-1992:14, Linköping 1992.

[64] Ye Qin-Zhong, *Contributions to the Development of Machine Vision Algorithms,* Ph.D. Thesis No. 201, Linköping University 1989.

[65] Åstrand E.,*Detection and Classification of Surface Defects in Wood A Preliminary Study,* Report No. LiTH-ISY-I-1437, Linköping 1992.

[66] Åstrand E., Johannesson M., Åström A., *Five contributions to the art of sheet-of-light range imaging*, Report No. LiTH-R-ISY-1611, Linköping, 1994.

[67] Åstrand E., Johannesson M., Åström A., *Two novel methods for range imaging using the MAPP2200 smart sensor,* Proc of Machine Vision Applications in Industrial Inspection, SPIE '95, San José, 1995.

[68] Åström A, *A Smart Image Sensor. Description and Evaluation of PASIC.*, Lic. Thesis. LIU-TEK-LIC-1990:57, Linköping 1990.

[69] Åström A, *Smart Image Sensors,* Ph.D. thesis No 319, Linköping 1993.

[70] Åström A, Forchheimer R, *Near-Sensor Image Processing A new approach to low level image processing*, Proc. of 2nd Singapore Int. Conf. on Image Processing, 1992.

[71] Åström A., Forchheimer R., Ingelhag P., *An Integrated Sensor/Processor Architecture Based on Near-Sensor Image Processing,* Proc of CAMP'93, New Orleans, 1993.

Part II

Radar Signal Processing using the VIP Architecture

Chapter 1

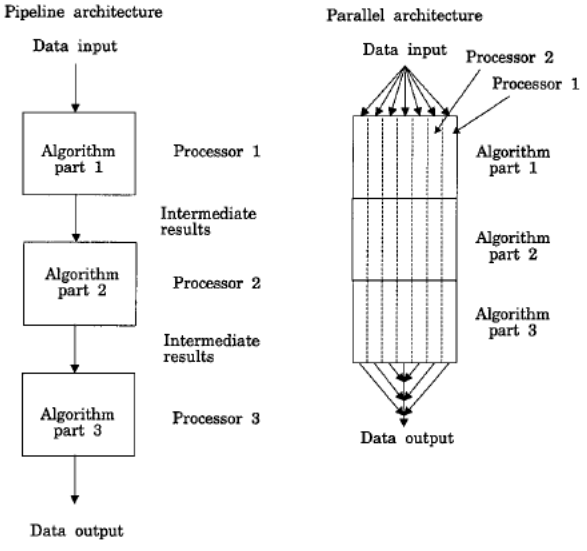
Introduction

1.1 Parallel processing

High data rates and large computational demands are typical for real-time image- and signal-processing applications. Already for a standard black and white video signal the data rate is in the order of 50 Mbit/s, and, if typical radar signals are processed, this figure may well be several hundreds of Mbit/s. With these kind of data rates even the simplest real-time signal processing task requires hundreds of millions of operations per second (Mops), and with a more complicated algorithm thousands of millions of operations per second (Gops) performance is needed. Since no single processor comes near this kind of performance (yet!) the processing has to be broken down into smaller tasks which can be executed by several processors in parallel.

Two different ways to parallelize a task are: 1. A pipeline of processors, each one solving its part of the algorithm on all input data. 2. An array of parallel processors, where each processor executes the whole algorithm for a limited amount of data. See Figure 1.1.

FIGURE 1.1.



A comparison between pipelined and parallel multiprocessor systems.

In a pipelined system it is always possible in principle to optimize the processor design for each sub-task, but the partitioning problem itself might be non-trivial. Also, each processor must still process the data at the input data rate. Although not conceptually identical this kind of parallel system is often called Multiple Instruction stream Single Data stream (MISD) machine.

In an array of parallel processors each processor processes only a sub-set of all data. Normally this will lower the data rate to each processor and therefore a less complex processor can be used. Two different kinds of parallel systems, Multiple Instruction stream Multiple Data stream (MIMD) and Single Instruction stream Multiple Data stream (SIMD), can be used to implement this very general idea. In an MIMD system the processors run, possibly, but not necessarily, individual programs asynchronously and data dependently. In an SIMD system all processors execute the same program synchronously. Since one single control unit can support many processors in an SIMD-machine, the processors can be made extremely simple, lending themselves to massive parallelism. The disadvantage with SIMD processing is that since all Processing Elements, PEs, execute the same instructions, data dependent operations are cumbersome. In this thesis we concentrate on SIMD systems. Further discussion on different parallel systems can be found elsewhere, for instance in [10], [37], [38], [40], [49] and [53].

The seemingly most natural SIMD architecture to process 2D data is the 2D processor array. Although numerous 2D architectures have been presented, for instance GAPP [29], the Connection Machines CM1 and CM2 [67], CLIP [22], MPP [6], Blitzen [8], and MasPar [56], the number of 2D arrays used for real-time signal processing is low. One common reason is that the IO bandwidth of the system is too low, and this limits the performance before the actual processing capacity. Also, even with the current state-of-the-art technology it is hard to make a compact (and reasonably cheap) 2D array with say 512x512 processors. One recent design, scheduled for release in early 1996, has compacted a 128x128 PE MasPar 32-bit SIMD RISC processor design into a two cubic feet package [66]. The system has a predicted peak performance of 34 GFlops.

Instead of using a complete 2D array we may use a 1D row-parallel array. Examples of such 1D array designs are LUCAS [62], AIS-5000 [68], IMAP [25], SVP [55], CRAM [18], [19] and LAPP1100 [21]. The LAPP1100 also contains a linear image sensor on the processor chip. It is easier to implement a row-parallel linear array than an image parallel 2D array, but of course, each of the processors must then execute the chosen task for a column of pixels in the image rather than for one pixel.

1.2 VLSI limitations

Even though the CMOS VLSI technology evolution is very fast there are still many physical limitations on chip design. The most apparent limit is the chip area, which effectively limits the number of transistors on a chip. However, with the current sub-micron feature size fabrication processes combined with chips larger than 200 mm^2 this limit is less important and designs using several millions of transistors are common. Furthermore, the trend towards even smaller feature sizes still continues.

More acute is the limited number of IO connections. While the feature sizes of the state-of-the-art CMOS VLSI processes used today are well below one micron the size of an IO pad remains in the order of at least a hundred microns. This, together with a maximum chip circumference of around 4-6 cm, limits the number of connections to a few hundred. This makes it hard to design large processor arrays with large IO bandwidth per processor. A lot of effort is

put into the problem of increasing the number of interconnections between chips by distributing the IO pads over the chip surface rather than around the circumference. Those methods are not very common however. An overview of different state-of-the-art chip interconnection strategies can be found in [32].

Also, as the clock frequency and therefore the power consumption increases, the power supply has to be connected over many input pins which further limits the number of available signal pins. For instance, the Alpha microprocessor, clocked at 300 MHz, has a power consumption of 50W, giving currents above 10 A [3].

1.3 VIP history

The Smart Sensor project at Linköping University started around 1988 aiming at a smart sensor named PASIC (Parallel A/D converter Sensor Integrated Circuit) [72]. PASIC was a 2D sensor with a 1D SIMD array of bit-serial processing elements on a single chip. Some of the ideas in PASIC have subsequently been implemented in the commercial smart image sensor MAPP2200, fabricated by the company IVP in Linköping [20]. The processor array of the PASIC architecture was found to be a very powerful signal processing architecture on its own due to the effective but small Processing Element (PE) design. This modified architecture was then called Video Image Processor, VIP [74].

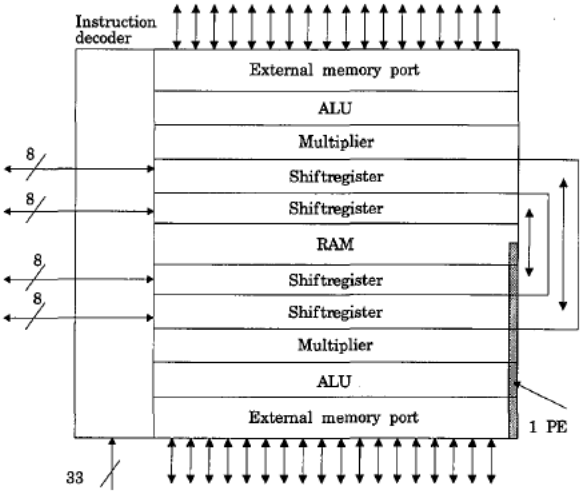
The row-parallel bit-serial VIP processor was designed with real-time video-rate input signal processing in mind. However, a preliminary study showed that a modified VIP could be used for Low Pulse repetition frequency Doppler (LPD) radar signal processing as well [43]. Following this study a joint project between the University of Linköping and Ericsson Radar Electronics was started in 1991, which resulted in the Radar VIP (RVIP) architecture described here and in [47], [48] and [4]. The RVIP is currently being implemented by Ericsson Microwave systems, EMW, (formerly Ericsson Radar Electronics) and a prototype VLSI-system is in operation since mid 1995.

After the success of the RVIP, further studies were launched in 1993 to find other suitable applications for the VIP design. The new applications studied so far are airborne Medium Pulse repetition frequency Doppler (MPD) radar signal processing (Flight VIP, FVIP) [76], [46], infrared video signal processing (IR VIP, IVIP) [33], [35], [75] and "secondary" wood inspection (Wood VIP, WVIP) [35], [36]. In 1994 we also started a study of a space-time adaptive phased array antenna application, the Matrix VIP. Only the RVIP, FVIP and MVIP designs will be described in this thesis.

The original VIP

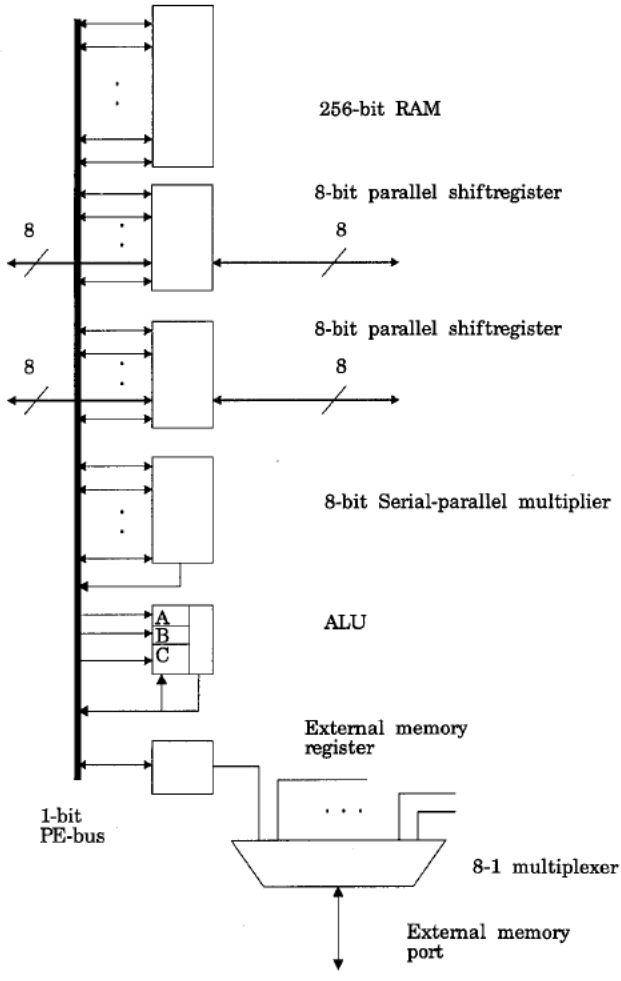
The original VIP architecture, [74], [43], is an SIMD architecture with a linear array of possibly 512 bit-serial processing elements (PEs) on a single chip, and the layout of the chip can be seen in Figure 1.2. In the floorplan the linear array is folded so that (possibly) 256 PEs are facing two opposite sides of the chip. Each PE has a bit-serial multi-functional ALU with an 8-bit serial-parallel multiplier, 256 bits of memory, two 8-bit shiftregisters and an external memory port, all connected via a one-bit bus, see Figure 1.3. Each external memory port is shared by eight PEs in order to reduce the number of IO connections on the chip.

FIGURE 1.2.



The layout of the original VIP chip.

FIGURE 1.3.



The processing element in the original VIP.

The VIP was designed for 20 MHz clock frequency, and the 8-bit size of the shiftregisters and multiplier indicates that the target application was video image processing of 8-bit greys-cale images. The predicted performance of one 512 PE VIP chip was approximately 420 Mops for 8+8 bit additions and 320 Mops for 8x8 bit multiplications.

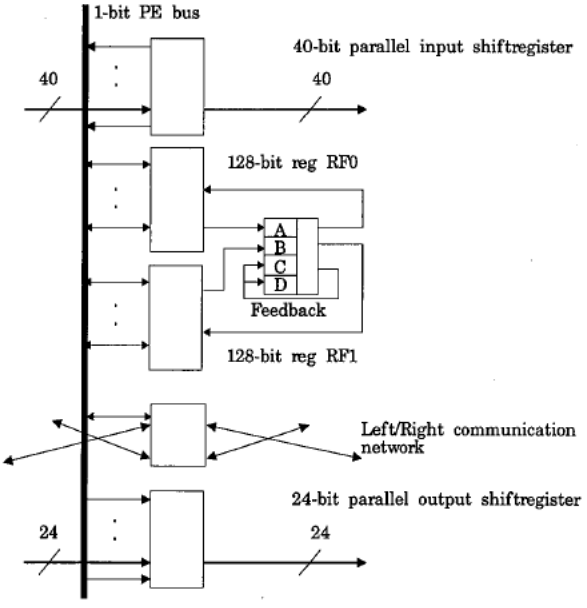
A 256 PE prototype, without the multipliers, was implemented in 1990-1991, and it was successfully integrated in a test system with a micro controller, compiler, simulator and frame grabber [45]. The compiler and simulator package was later updated and used to simulate the RVIP architecture and algorithms [47].

1.4 Other related architectures

Many 1D SIMD architectures have been presented throughout the years. We feel that two of these share a lot of ideas with the VIP concept, and have somewhat comparable performance. They are therefore presented below. The most notable difference between these architectures, and older architectures not presented here, is the extended IO capacity which makes it possible to implement more IO intensive operations.

A row-parallel SIMD architecture called Serial Video Processor (SVP), has been presented by Texas Instruments [55]. The target application is digital television applications and 768-1024 Processing Elements on each chip are proposed for different TV standards. Data IO is supported by two parallel shiftregisters, 40 and 24 bits wide respectively, running at 33 MHz clock frequency asynchronously with the PE processing. Each PE consists of 2x128 bit data registers (register files), a bit-serial ALU with four internal registers and a communication network allowing each PE to reach four left-right neighboring PEs, see Figure 1.4. The performance for a 1024 PE chip is stated as 1250 Mops for 8+8 bit addition and 155 Mops for 8x8 bit multiplication at a PE clock frequency of 20 MHz. The SVP is currently available in Japan, but we do not have any detailed information of the current version [28].

FIGURE 1.4.

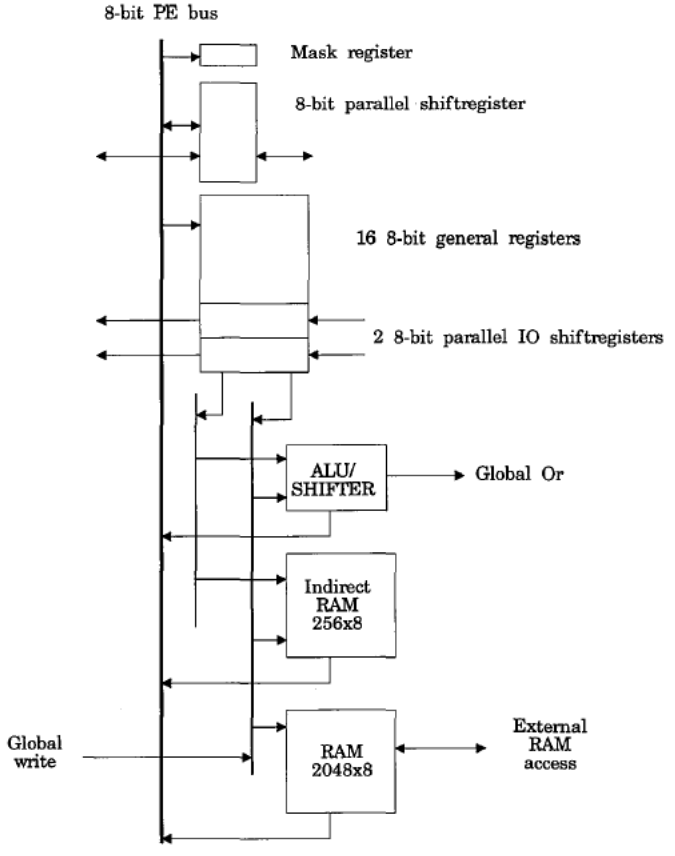


The processing element of the SVP processor array (adapted from [55]*).*

An architecture for video input processing, called alternatively Integrated Functional Memory, IFM, and Integrated Memory Array Processor, IMAP, has been presented by NEC [24], [25], [26], [27], [65]. This chip combines a large dual-ported VRAM and an up to 512 PE SIMD array of 8-bit PEs. Each PE has an 8-bit ALU with a barrel shifter using input data from sixteen 8-bit registers, see Figure 1.5. IO communication is supported by two 8-bit parallel shiftregisters and a third 8-bit shiftregister is used for neighbor PE communication. The inter-

nal memory is 2-4 Kbyte and since it is dual-ported it can be accessed by the external controller as well as by the PEs. A mask bit can be used to enable/disable each PE which allows for a limited form of data dependent operations. A special feature of this architecture is the 256 word indirect access memory which speeds up histogram computations, multiplications, and image rotations.

FIGURE 1.5.



The IFM (IMAP) processing element (adapted from [25]).

Two prototypes have been implemented. The first one employs a 0.8 µm process with 8 PEs on each chip running at 25 MHz and reaching 7.7 Gops for a 64-chip 512 PE system. The second one employs a 0.55 µm BiCMOS SRAM process with 64 PEs on each chip, reaching 3.84 Gops per chip at a clock frequency of 40 MHz. The latter design contains 11 million transistors on a single 235 mm^2 chip.

The chip performance for 8+8 bit addition is 2.56 Gops and 8x8 bit multiplication 230 Mops. 3.84 Gops is attainable by counting additions and memory accesses as separate operations. The large internal dynamic memory, which originally had a cycle time of 150 ns [24], has been exchanged for a faster static memory in the latest prototype, but the memory bandwidth is still a bottleneck as long as the access time is at least 2 clock cycles.

Chapter 2

Pulse Doppler Radar

2.1 General[1]

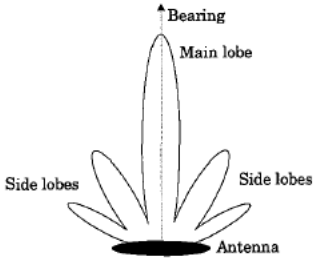
This part of the thesis investigates VIP architectures suitable for pulse Doppler radar signal processing. Therefore this chapter presents some of the basics of a radar, i.e. how range and velocity can be detected using reflected microwave pulses. To understand this we also present the function of parts of the required hardware, i.e. the antenna and the receiver. We then give an overview of the typical signal processing algorithms which are used in the radars in the following chapters.

RAdio Detection And Ranging, RADAR, is a ranging technique based on time-of-flight measurement which was developed in the late nineteen thirties. The radars discussed here use electromagnetic microwave pulses with a wavelength λ of around 1-10 cm, i.e. a frequency of 3-30 GHz, as a carrier wave. The applications are military ground-based or airborne radars where the goal is to detect and measure the position of airborne *targets*, e.g. aeroplanes, helicopters and missiles. In principle it is also possible to use ultra-sound or visible light pulses in a radar, but microwaves are better suited for accurate long distance ranging.

The normal radar transmitter antenna emits energy in a direction called *bearing*, and the receiver, which often uses the same antenna, detects the echoes returning from targets in the bearing direction. To resolve the direction to the targets accurately, the antenna should only transmit energy in one direction, i.e. the direction of the *main lobe*. However, small parts of the energy is always transmitted in several other directions, known as the *side lobe* directions, see Figure 2.1. When the same antenna is used both for the transmitter and the receiver the side lobe characteristics are of course the same. Echoes returning from these side lobes may be detected. Unwanted echoes are called *clutter*, and a large part of the signal processing task is aimed at detecting a weak target signal in the presence of strong clutter. Clutter is not only caused by the sidelobe echoes, but often the main lobe produces clutter from the ground, which in fact can be much larger than the target echoes. Another type of unwanted signals comes from active *jammers*, i.e. enemy transmitters, trying to disturb the function of the radar. Anti-jamming techniques are not employed in the radar applications discussed here.

1. *Much of the contents of this chapter has been collected from the book "An introduction to airborne RADAR" by G.W. Stimson [61].*

FIGURE 2.1.



A cross section of the sensitivity and directivity of a typical radar antenna.

2.2 Radar ranging

Radar ranging is performed by transmitting a modulated continuous wave (CW radar) or a pulsed wave in one direction (bearing). The range r to a reflecting object is then proportional to the round-trip time t of the wave, i.e. the time until the echo is received, such that

$$r = \frac{c \cdot t}{2} \tag{2.1}$$

where c is the speed of light. The range resolution Δr is inversely proportional to the receiver sampling frequency f_s

$$\Delta r = \frac{c}{2f_s} \tag{2.2}$$

For each transmitted pulse R samples, called *range bins*, are collected over an instrumented range $\Delta r \cdot R$, giving one row of range data. The number of samples R is limited by the *pulse repetition frequency*, *prf* or f_p, and the sampling frequency f_s so that

$$R \leq \frac{f_s}{f_p} \tag{2.3}$$

This implicates that we can only detect targets at ranges up to the unambiguous range r_u

$$r_u = \Delta r \cdot R = \frac{c}{2f_p} \tag{2.4}$$

If the target distance r is larger than r_u aliasing will give a detected apparent range, r_a, which is

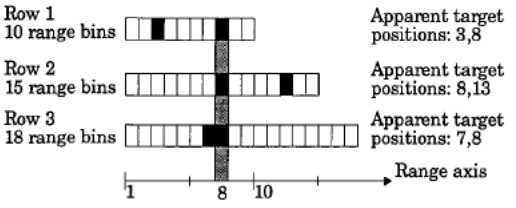
$$r_a = r - r_u \cdot n \tag{2.5}$$

where n is an integer value describing the number of times the range value has been folded around the ambiguity limit r_u. n is also the number of pulses that has been transmitted since the detected signal was transmitted.

Prf switching is a technique which is used to distinguish between aliased and non-aliased target echoes. By changing the prf while acquiring several rows of data from the same bearing we obtain data where we know that non-aliased target echoes appear in the same range bin in all of the acquired rows. Aliased echoes will, however, since the ambiguity limit changes, move to different range bins in the different rows. This is illustrated in Figure 2.2. We see that

a target echo appears in range bin 8 in all three rows, and thus a target detection is made for that bin. The other echoes are discarded since they do not coincide, and are either generated by noise or targets at a distance greater than r_u.

FIGURE 2.2.



Data from three consecutive rows from the same bearing acquired with different prfs. One target is found in range bin 8, and the other echoes are aliased and therefore discarded.

The distances to targets at ranges greater than r_u can be found by further exploiting prf switching with a technique called *resolving*. By definition a *Coherent Processing Interval, CPI,* contains data sampled from one or more pulses transmitted with a constant prf. If we discard data immediately after a switch to a new prf we know that all echoes in the collected data from each *CPI i* satisfies equation (2.5) for some integer n_i. Thus with X CPIs we get X equations

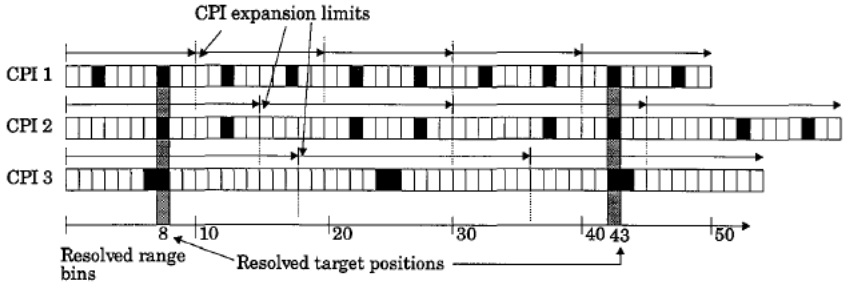
$$r = r_{ai} + r_{ui} \cdot n_i \tag{2.6}$$

with the unknown variables n_i and r, i.e. we have X equations and $X+1$ unknown variables. A value of r which satisfies the equation system can be found by a trial and error search. To the data in Figure 2.2 this gives one possible solution as

$$r = 3 + 10 \cdot 4 = 13 + 15 \cdot 2 = 7 + 18 \cdot 2 = 43 \tag{2.7}$$

Alternatively we can use the technique illustrated in Figure 2.3, where data from the CPIs from Figure 2.2 is resolved.

FIGURE 2.3.



Resolving data from the three CPIs in Figure 2.2. The ranges of the two targets are 8 and 43.

Here we unfold data from each CPI around its respective ambiguity limit and find the coincidences analogous with the prf switching technique described above. In the example each CPI contains between ten and eighteen range bins and the resolving is performed out to fifty range

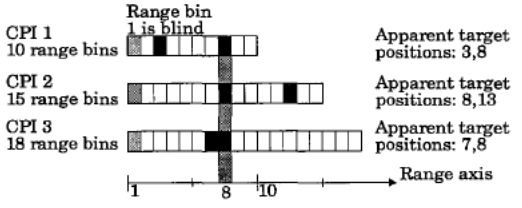
bins. Thus, in this example the resolving increases the maximum distance to a correctly detected target with a factor between three and five compared with the maximum target distances of the individual CPIs.

For various reasons only a fraction x, usually around 0.75-0.9, of the available time is spent sampling the receiver so that in practice the number of range bins is reduced to

$$R = x\frac{f_s}{f_p} \tag{2.8}$$

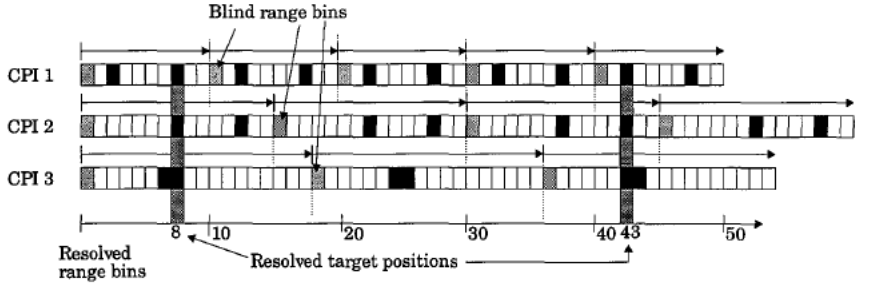
One reason for an x smaller than 1 is that the transmitter is active around 10% of the time and we cannot sample simultaneously with transmission if transmitter and receiver use the same antenna. We denote x as the *receiver duty factor*. When the receiver is switched off we get blind range bins, or blind zones, see Figure 2.4. A blind zone close to the radar is often not a problem, but during resolving blind zones can appear further away as is shown in Figure 2.5 The impact of the blind zones is minimized by careful selection of the used prfs.

FIGURE 2.4.



Three CPIs with the first range bin blind since the receiver is switched off.

FIGURE 2.5.



Resolving the three CPIs in Figure 2.4. Note that two of the three CPIs are blind in the resolved range bin 31.

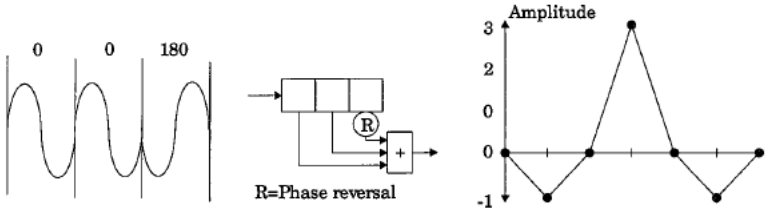
2.3 Pulse code compression

To be able to detect target radar echoes they must contain a certain amount of received energy. Ideally, the transmitted pulse should be a very short pulse with very large peak power, giving a large reflected energy at a well defined point in time. In practice however, the peak power of the transmitter is limited and the energy must be distributed over longer pulses. The technique to compress the energy of a long received pulse into a shorter and stronger response to be saved in a single well-defined range bin is called *pulse code compression* and is illustrated in

Figure 2.6. Here the 3 element pulse is phase coded with a binary phase code, and thus the relative phase of the carrier wave is either 0° or 180°. To detect the pulse the received signal is correlated with a discrete three-point kernel which is equal to the code. The pulse response is strong when the kernel matches the incoming signal. In Figure 2.6 the ratio between the correct peak and the largest sub-maxima (compression side lobe) is three to one. By using longer codes, possibly with more phase positions and/or amplitude modulation, higher ratios between true maxima and sub-maxima are acquired. Another common modulation technique is frequency modulation.

It may be noted that coding by means of amplitude modulation is uncommon since amplitude modulation reduces the pulse energy. With phase or frequency modulation the radar is always using the transmitter peak power. A typical pulse length (*transmitter duty factor*) is 10% of the total number of range bins, which gives a pulse duration time of $10/f_s$.

FIGURE 2.6.

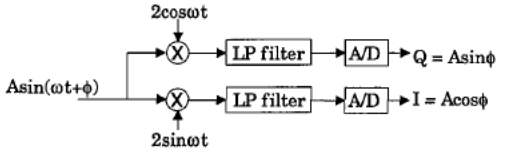


Pulse code compression. A 3 sample code (left), its decoder (middle) and the pulse response of the decoder (right).

2.4 Synchronous detection

To be able to perform pulse code compression we must be able to retrieve both the amplitude and the phase of the return echoes. The amplitude and phase information are also needed to determine the speed of the target relative to the receiver. To this effect the signal is received in a so called *synchronous detector*. In the synchronous detector the input signal is multiplied with two reference signals with the same frequency ωt as the carrier frequency of the radar signal and then low-pass filtered, see Figure 2.7.

FIGURE 2.7.



A synchronous detector.

The received signal, which has an amplitude A and a phase ϕ relative to the reference signal will in the in-phase channel, I, be converted to

$$I = LP\,(2\sin\omega t \cdot A\sin\,(\omega t + \phi)\,) \;= \tag{2.9}$$

$$= LP\,(2A\sin\phi\sin\omega t\cos\omega t + 2A\cos\phi\,(\sin\omega t)^2) \;= \tag{2.10}$$

$$= LP\left(A\sin\phi\sin 2\omega t + 2A\cos\phi\left(\frac{1}{2}+\frac{1}{2}\cos 2\omega t\right)\right) = \tag{2.11}$$

$$= A\cos\phi \tag{2.12}$$

Analogously, the quadrature channel, Q, will become

$$Q = A\sin\phi \tag{2.13}$$

Thus, the synchronous detector demodulates the received signal and makes it possible to extract both the amplitude A and the relative phase ϕ of the echo.

2.5 Target velocity detection

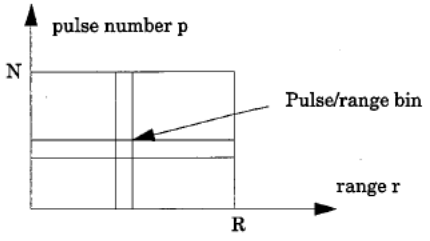
When a target is moving relative to the radar the received phase will change between consecutive echoes. This is called the *Doppler effect* or *Doppler shift*. For instance, if the distance between the target and the radar changes $d = \lambda/4$ meters between two pulses the phase between the echoes will shift δ=1/2 period. The Doppler shift δ is then proportional to the relative target velocity according to

$$\delta = \frac{-2d}{\lambda} = \frac{-2v}{\lambda \cdot f_p} \tag{2.14}$$

where d is the distance moved between the pulses and v the speed of the target. The negative sign in the equation shows that if the target is approaching (v is negative) the received phase will increase.

To extract the target speed, echoes from N pulses are collected during a CPI. Each sample in this CPI is then collected in a 2D array of pulse and range bins, see Figure 2.8.

FIGURE 2.8.



A CPI array of NxR pulse and range bins.

An object not moving a distance longer than Δr during the acquisition time N/f_p will give echoes that appear in the same range bin for all pulses. However, the phase between any two consecutive samples within that range bin will be proportional to the relative velocity of the target as shown in Eq. (2.14). Between the first and the last sample in the range bin the signal will shift $N\delta$ periods. Note that with a wavelength λ in the order of centimeters and a range resolution Δr in the order of tens of meters the total phase shift $N\delta$ may very well be several hundred periods while the target moves within one range bin. This total phase shift can be interpreted as a (Doppler) frequency along the pulse bin axis

$$f_d = \frac{N\delta}{t} = N\delta \cdot \frac{f_p}{N} = \frac{-2v}{\lambda} \tag{2.15}$$

Thus, by *Fourier transforming* the N samples in each range bin the Doppler frequency, and therefore the object velocity, can be detected as

$$v = \frac{-\lambda f_d}{2} \tag{2.16}$$

The Fourier transform will have a frequency resolution of f_p/N over the frequencies

$$|f_d| < \frac{f_p}{2} \tag{2.17}$$

This gives the velocity resolution Δv as

$$\Delta v = \frac{\lambda f_p}{2N} \tag{2.18}$$

The maximum unambiguous velocity v_u that is possible to detect will then be

$$|v_u| < \frac{f_p \cdot \lambda}{4} \tag{2.19}$$

If the target velocity v is above v_u aliasing will give an apparent velocity, v_a, that will be detected. The apparent velocity will be, analogously to the apparent range r_a,

$$v_a = v \pm m \cdot \frac{f_p \cdot \lambda}{2} \tag{2.20}$$

Here, m is an integer which describes the number of times the velocity has been folded into the detected frequency spectrum.

After the Fourier transform the pulse bins are called *velocity bins*. The samples from N pulses from a single bearing processed in this way to obtain simultaneous velocity and range information is called a Coherent Processing Interval, CPI, and the CPI-time, t_{CPI}, is

$$t_{CPI} = \frac{N}{f_p} \tag{2.21}$$

The accurate detection of velocities larger than v_u is possible by processing data from several consecutive CPIs acquired using different prfs analogous with the range resolving described in section 2.2.

2.6 Multi channel antennas

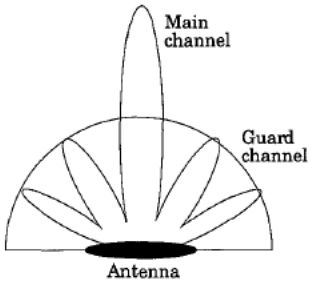
Many of the radars used today have multiple antenna elements. The output of each antenna element is then sampled in its own separate channel.

Guard horn

A *guard horn* (or guard antenna) can be used to distinguish between main and side lobe echoes. In Figure 2.9 we see typical lobe diagrams for the main and guard antennas, and by comparing the respective outputs main lobe target echoes, which should be significantly stronger in

the main channel, can be distinguished from side lobe target echoes which typically has similar strength in the two channels. Note that the signal from the guard channel should be processed analogous with the main channel, and thus the signal processing demands are doubled.

FIGURE 2.9.

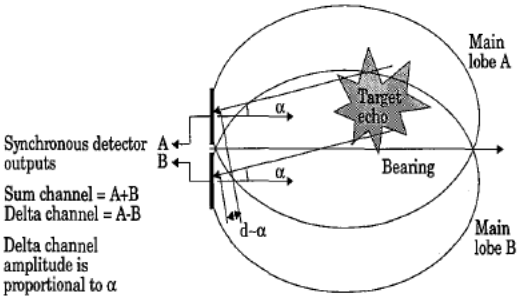


The lobe diagrams for main and guard antennas.

Monopulse antenna

Another common multichannel antenna uses a technique called *monopulse*. The idea with a monopulse antenna is to transmit several similar, but not identical lobes simultaneously, see Figure 2.10, and then process the returning signals to obtain a more accurate estimate of the target direction than what is possible with just one single transmitted lobe. A one-dimensional example of phase comparison monopulse antenna can be seen in Figure 2.10. By comparing the phase of the echoes from the two channels the target direction relative to the main lobe bearing can be determined very accurately. Typically a mono pulse antenna transmits two or four lobes, giving one- or two-dimension target direction information. Note that the processing of a delta channel is made analogous with the sum channel, and thus the signal processing demands increase considerably with a monopulse antenna.

FIGURE 2.10.



A two-lobe phase comparison monopulse antenna. The distance difference d between the target and the two antenna elements gives a phase shift of the echoes which is proportional to the angle α.

Phased array antennas

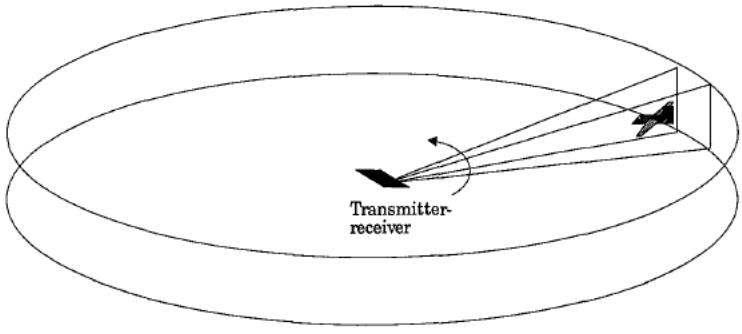
A phased array antenna can be viewed as an extension of a monopulse antenna. In a phased array antenna many individual antenna elements are arranged in a 1D or 2D array. By controlling the relative phase of the transmitted pulses the main lobe bearing can be controlled without moving parts. When receiving, the main lobe direction can be steered analogously. Furthermore it is possible to attain zero sensitivity in other directions, i.e. almost totally block out jammers. This beam forming can be made either direct at the microwave receiver, or digitally in the subsequent signal processing. The radar signal processing of phased array antenna data is often called *space-time* processing [5].

2.7 Radar image generation

For each CPI a 1D range vector is acquired. To obtain 2D range images the radar bearing is changed. Several different sweep strategies are used. A typical ground based surveillance radar system sweeping in circles can be seen in Figure 2.11. With this technique 2D images covering 360° around the radar are obtained. A typical airborne search and tracking radar setup can be seen in Figure 2.12. Here the radar sweeps in a 2D scan pattern. It is also possible to lock the radar on a selected target to track its trajectory.

The radar sweep velocity is slow so that several consecutive CPIs can be considered as being acquired in the same bearing.

FIGURE 2.11.



A typical ground-based surveillance radar sweep pattern.

2.8 Doppler radar system classification

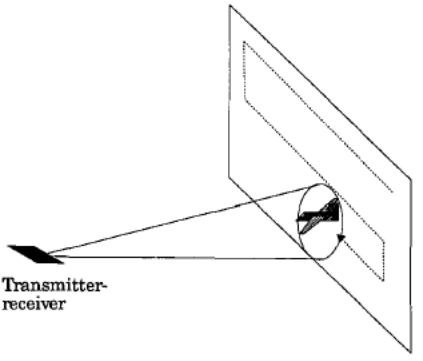
Pulsed Doppler radar systems are commonly categorized in Low, Medium and High Pulse repetition frequency Doppler, called *LPD*, *MPD* and *HPD* respectively. Typical pulse repetition frequencies for the different modes are found in Table 2.1. Each has its own distinctive use and characteristics. Many radar systems can switch between the three modes by changing the prf and the subsequent signal processing.

If we combine equations (2.8) and (2.21) we find that

$$R \cdot N = x f_s t_{CPI} \tag{2.22}$$

which shows that for a given CPI time, receiver sampling frequency and receiver duty factor the product *RN* is constant. Typical CPI times range from 10-30 ms. The correspondence between the number of range and pulse bins used in each CPI with the different prfs can be studied in Figure 2.13. An LPD radar often has a range resolution in the order of 1000 bins and a velocity resolution in the order of 10 bins, and vice versa for the HPD radar. With the MPD radar we have in the order of 100 range and velocity bins.

FIGURE 2.12.

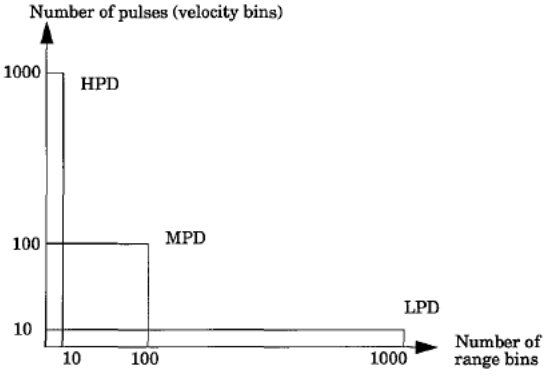


A typical airborne search radar sweep pattern.

TABLE 2.1.

	LPD	MPD	HPD
PRF (KHz)	0.25-4	10-20	100-300
Pulses per CPI	~10	~100	~1000
Range bins per CPI	~1000	~100	~10

FIGURE 2.13.



Typical Coherent Processing Interval (CPI) matrices for LPD, MPD and HPD.

LPD

An LPD radar system is designed for unambiguous range detection. That is, a new pulse is not sent out until all possible (or probable) echoes from the previous pulse have returned. Thus, the lower the prf is, the further away the radar can see, as shown in Eq. (2.4). On the other hand, the low prf and the low number of pulses in each CPI makes it harder to accurately compute the Doppler shift between echoes from the same target, and thus the velocity resolution and the maximum unambiguous velocity is low as was shown in equations (2.18) and (2.19). LPD systems are often found in ground-based search radar where long detection range is important.

MPD

An MPD system is designed so that both the received range and the received velocity is ambiguous. That is, we know that the conditions in the equations (2.4) and (2.19) might be violated. However, using the resolving technique described earlier, it is possible to process several consecutive CPIs acquired using different prfs and find the unambiguous range and velocity of each target.

Note that the resolving increases the number of range and velocity bins beyond the unambiguous range and velocity of the individual CPIs. Thus, after resolving both the number of range and the number of velocity bins might be around 1000. MPD systems are often found in airborne radar systems where it is important to accurately estimate the range and velocity of the targets.

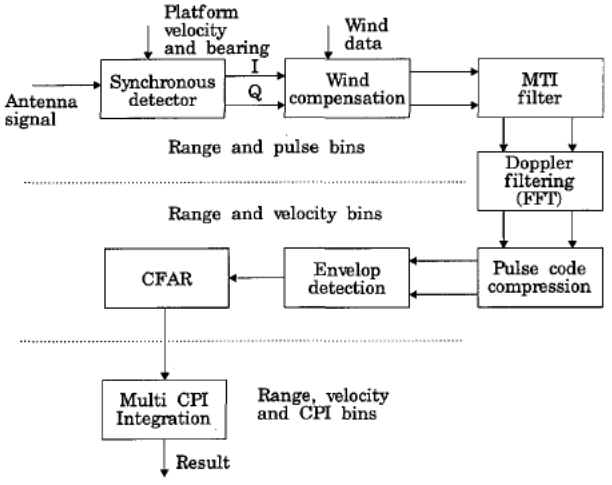
HPD

An HPD system is designed for unambiguous velocity detection. That is, the high prf gives a high maximum detectable velocity, and the large number of CPI pulses N make it possible to compute the relative target velocity accurately as was shown in equations (2.18) and (2.19). However, the fact that many pulses are in the air at the same time makes it hard to determine the range to a target unambiguously. HPD systems are often found in tracking radar systems where it is important to accurately predict the trajectory of a target.

2.9 Radar signal processing

All Doppler radars have many processing steps in common and a typical radar signal processing algorithm can be seen in Figure 2.14. Observe that the MTI filter, the Doppler filtering and the pulse code compression often are implemented as linear filter operations which implicates that their relative order in the algorithm can be exchanged.

FIGURE 2.14.



A typical radar signal processing chain.

Synchronous detector

If the radar platform is static there is no processing in the synchronous detector. However, if the platform is moving there is a motion compensation that shifts the Doppler frequency of the ground clutter to the DC-component. This simplifies the MTI filter as will be described below.

Wind compensation

When a wind blows, rain and clouds will move with the wind and give clutter which has non-zero Doppler frequency. This means that this so called *wind clutter* not is removed by the MTI filter. To ensure that the MTI filter removes both the ground and the wind clutter the spectrum of the return signal is shifted in the wind compensation so that both the ground and the wind clutter falls inside the frequencies removed by the MTI filer, see Figure 2.15. The wind compensation is performed by a vector rotation

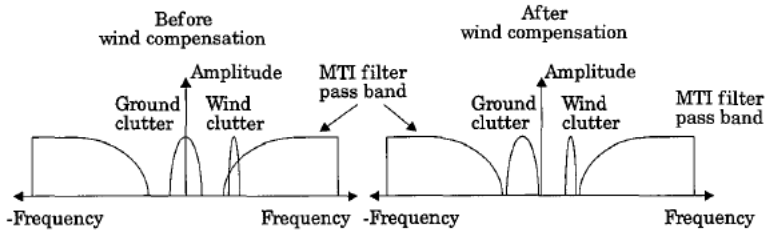
$$\begin{bmatrix} I_{WC} \\ Q_{WC} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} I \\ Q \end{bmatrix} \tag{2.23}$$

where the angle α is range and/or pulse number dependent. The angle α is range dependent if the Doppler frequency of the rain clutter is range dependent.

MTI filter

The Moving Target Indicator, MTI, filter is used to remove clutter echoes arising from static targets. The goal of the radar is to detect small moving targets, but at the same time the ground, which is static, gives very strong main lobe clutter.

FIGURE 2.15.



The shift of the signal frequency spectrum to ensure that wind clutter is rejected by the MTI filter.

If the radar platform is static the MTI filter can be implemented with a filter operating along the pulse axis and removing the DC component of the signal. However, if the platform is moving the ground clutter will not be the DC component. Instead the ground clutter will have a frequency which depends on the platform velocity and lobe direction. Instead of changing the frequency response of the filter to compensate for the platform velocity and lobe bearing, a compensation so that the frequency spectrum of the signal is shifted so that the DC component always corresponds to the ground clutter frequency is done in the synchronous detector.

The MTI filter is often implemented as a high-pass convolution in the pulse direction.

Doppler filtering

The Doppler filtering transforms the pulse bins into velocity bins. This is often implemented using a Fast Fourier Transform, but if only a few Doppler channels are extracted it can be performed using Doppler filter banks (convolutions) instead. If only one Doppler channel is processed, as may be the case in LPD radars, the Doppler filter can be combined with the MTI filter into a single pulse direction convolution.

Since the DFT/FFT by definition operates on a circularly repeated signal, a pre-weighting (or windowing) is often applied before the transform to meet this condition. The result is a reduction of side lobes, i.e. signal response in an incorrect velocity bin. The window typically reduces the amplitude of the samples from the first and last pulses.

Pulse code compression

The goal of the pulse code compression, pcc, is to collect all received energy from one target into a single range bin. This is made with a correlation with the transmitted phase or frequency code which normally has a length of R/10 range bins. Since the pulse codes might be several hundred bins long the computation is sometimes performed in the Fourier domain, i.e. an FFT is performed along the range bins, the correlation is then implemented as a Fourier domain multiplication and then an inverse FFT is performed. For each sample a pcc convolution has a complexity of $\sim R/10$ and an FFT implementation $\sim \log R$ operations. This makes a Fourier domain implementation favorable if R is larger than approximately 64-128. However, the overhead often found in parallel FFT implementations might increase this limit considerably. In our LPD implementation to be presented in the next chapter we implement the pulse code compression with a convolution even though R is 4000. In chapter 5 we investigate when a Fourier domain implementation might be favorable in VIP.

Envelop detection

The envelop detection removes the phase information from the samples since it is not needed after the FFT and/or pulse code compression has been performed.

CFAR

The Constant False Alarm Ratio, CFAR, processing is made to reduce the number of possible targets in each CPI. The name comes from the goal to only allow detection of a certain number of false targets in a specified amount of time.

Since each target often gives several echoes in neighboring bins the CFAR works in a range and/or velocity neighborhood. Often it is implemented as a local maximum detection, i.e. only local maxima are considered as targets. Sometimes a thresholding is a part of the CFAR, but if not, local maxima are amplified relative to their neighbors so that they can be extracted by a subsequent thresholding operation.

Multi CPI integration

The integration, or accumulation, of results from several CPIs is made to further increase the probability of detecting a target while reducing the probability of false target detection. The integration is often combined with the prf switching and resolving techniques described earlier in this chapter. The integration can either be made on thresholded data, called *binary integration*, or before thresholding, called *video integration.*

In binary integration the CPIs are often acquired using slightly different prfs. The different prfs makes it possible to perform resolving of ambiguous target velocities and/or ranges, but it is also used to move the blind zones of the radar. That is, the range/velocity areas where no target detection is possible due to clutter and the fact that the receiver duty factor is less than 100% are moved around in the integration matrix. The detection criteria is then typically that at least 3 of 9 coinciding indications are needed. The integration can either work on batches of A CPIs, delivering a result every A:th CPI, or use a gliding window of the A latest CPIs, giving a new result for every acquired CPI.

Video integration is often performed as a correlation over several neighboring CPIs (and bearings), and the length of the correlation kernel reflects the number of CPIs that a target is expected to be within the main lobe and thus appear in the CPI data.

2.10 VIP in RADAR signal applications

Intuitively, both LPD and HPD algorithms seems to fit well into a linear SIMD architecture since they are open to a large degree of parallelization in either range or pulse/velocity direction, see Figure 2.13. MPD however, has only "medium" parallelism in two dimensions and therefore an implementation in a 2D architecture is attractive whereas a linear SIMD array implementation is much less straightforward. In this thesis we limit our studies to LPD and MPD implementations on linear VIP arrays. One of our LPD implementations uses a phased array antenna and space-time adaptive processing.

Chapter 3

The Radar VIP

3.1 General

The Radar VIP, RVIP, was designed for Low Pulse repetition frequency Doppler, LPD, radar signal processing. This application differs in many ways from the image processing for which the original VIP was designed. Therefore we present an overview of a typical LPD algorithm and then show how this has affected the PE design. After that we present the overall system design and the result of the LPD implementation on the modified RVIP architecture.

3.2 An LPD algorithm

An overview of one of the studied LPD algorithms, which is very similar to the generalized signal processing chain described earlier, can be found in Figure 3.1. The application is a ground-based surveillance radar.

Typical CPI parameters used are collected in Table 3.1. We see that the sampling frequency f_s is 5 MHz. The prf f_p is around 900 Hz and the number of samples R in one bearing (row) up to 4000. The sampling frequency and number of samples acquired for each pulse/bearing gives the radar a range resolution Δr of 30 m over an instrumented range of 120 km.

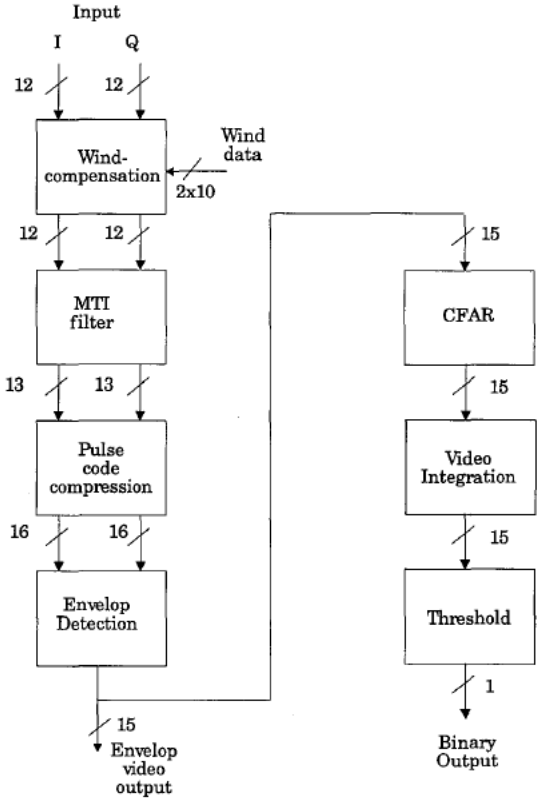
The actual algorithm uses only one pulse in each CPI and therefore no Fourier transformation and subsequent separation into different velocity bins is performed. Instead all moving targets extracted by the MTI filter are considered. That is, we can view the MTI filter as a Doppler filtering, where we only are interested in, and compute, one frequency bin. This is the main difference to the more general scheme in Figure 2.14.

The input is I and Q data sampled as 2x12 bit 2-complement integers. The precision grows to a maximum of 2x16 bits after the pulse code compression. All filter coefficients have 10 bit precision. The wind compensation is range dependent and the coefficients for each range bin are input concurrently with the radar data.

TABLE 3.1.

LPD algorithm parameters	Value
Sampling frequency, f_s	5 MHz
Pulse repetition frequency, f_p	< 900 Hz
Unambiguous range, r_u	< 167 km
Number of pulses/CPI	1
CPI time	> 1.1 ms
Range resolution, Δr	30 m
Number of range bins	< 4000
Instrumented range	< 120 km

FIGURE 3.1.



An overview of an LPD algorithm.

Algorithm complexity

We have found it convenient to approximate the algorithm complexity in terms of real-valued MAC-operations per input sample. One MAC (Multiply and Accumulate) is typically executed on 10-16-bit fix-point integer data. Without going into too many details we are now going to give such estimates for the various parts of the LPD-algorithm in Figure 3.1. For more details on the different steps in the algorithm we refer to the material in chapter 2.

The complexity of the vector rotation in the *wind compensation* is 4 MAC operations, see equation (2.23).

The *MTI filter* is a real-valued FIR filter over eight consecutive CPIs (bearings)

$$(I+jQ)_{MTI}(r,p) = \sum_{i=0}^{7} C_{MTI}(i) \cdot ((I+jQ)_{WC}(r,p-i)) \qquad (3.1)$$

and the complexity is 2x8=16 MAC operations. Since the filter is in the bearing direction seven old bearings (rows) must be stored in internal RAM memory.

The correlation window in the *pulse code compression* is 383 range bins and the coefficients are all real valued. The convolution can be written

$$(I+jQ)_{PC}(r,p) = \sum_{i=0}^{382} C_{PC}(i) \cdot ((I+jQ)_{MTI}(r-i,p)) \tag{3.2}$$

This is by far the most computationally demanding part of the algorithm, requiring 2x383 = 766 MAC operations.

The *envelop detection* is the operation

$$A(r,p) = \sqrt{I^2_{PC}(r,p) + Q^2{}_{PC}(r,p)} \tag{3.3}$$

Since taking the square root is a time consuming operation in a bit-serial architecture the envelop detection is approximated with the operation

$$X(r,p) = max(|I_{PC}(r,p)|, |Q_{PC}(r,p)|) \tag{3.4}$$

$$Y(r,p) = min(|I_{PC}(r,p)|, |Q_{PC}(r,p)|) \tag{3.5}$$

$$A(r,p) = max(X(r,p), 0.875X(r,p) + 0.5Y(r,p)) \tag{3.6}$$

This approximation gives a result within three percent of the actual value. The envelop detection requires approximately 5 MAC equivalent operations.

The *CFAR* operation is implemented by dividing each sample *A(r)* with a mean value of 16 of its nearest range neighbors. The definition is

$$S_{early}(r,p) = \frac{1}{16}\sum_{i=2}^{17} A(r-i,p) \tag{3.7}$$

$$S_{late}(r,p) = \frac{1}{16}\sum_{i=-2}^{-17} A(r-i,p) \tag{3.8}$$

$$S_{max}(r,p) = MAX(S_{early}(r,p), S_{late}(r,p)) \tag{3.9}$$

$$A_{cfar}(r,p) = \frac{A(r,p)}{S_{max}(r,p)} \tag{3.10}$$

If we examine the equations (3.7) and (3.8) we note that $S_{late}(r,p)$ equals $S_{early}(r+19,p)$. This means that if we compute $S_{early}(r,p)$ for all range samples we have also computed $S_{late}(r,p)$ for all samples. For simplicity we assume that the complexity of a division equals that of a MAC operations and then the CFAR processing requires approximately 17 MAC operations.

The *video integration* consists of two FIR filters in the bearing direction. First the CFAR result is averaged and down-sampled a factor eight,

$$A_{tmp}(r,p') = \frac{1}{8}\cdot\sum_{i=0}^{7} A_{CFAR}(r,p'-i) \tag{3.11}$$

where

$$p' = 8 \cdot p \tag{3.12}$$

The down-sampled signal A_{tmp} is then input to a 16-tap FIR filter in the bearing direction

$$A_{VI}(r,p') = \sum_{i=0}^{15} C_{VI}(r,p') \cdot A_{tmp}(r,p') \quad (3.13)$$

Due to the down-sampling the 16-tap FIR filter is only evaluated every eighth bearing. For simplicity we approximate the complexity of the video integration as 8+16=24 MAC operations, i.e. we can evaluate the FIR filter at every bearing (CPI). Row buffering of 16 intermediate (down sampled) results is needed.

The data is *thresholded* using a threshold determined by other parts of the system and the result indicates if a target is present in the range bin or not. The complexity is 1 MAC operation.

The total computational complexity is then approximately 830 MACs for each sample in each bearing. Thus, the total processing demand is around 830x4000x900 = 3 Giga MACs per second. If each sample is to be processed before the next sample arrives, eliminating the need for input buffers and not using the empty part of the receiver duty cycle, the complexity is $830x5x10^6$ = 4.15 Giga MACs per second. In principle, by computing the pulse code compression in the Fourier domain this demand can be decreased with one order of magnitude.

The data input is 2x12 bits of radar data and 2x10 bits of wind compensation coefficients, which gives a total of 44 bits for each radar sample. Thus the input capacity to the system must be at least 220 Mbits/s. The required output is the envelop detected video signal and one bit every eighth CPI, which equals less than 70 Mbits/s. The needed internal RAM is approximately 8x2x12 for the MTI filter and 16x15 bits for the video integration, which gives a total of 432 bits.

Variations

Ericsson also intended to use some algorithm versions where the IO and memory capacity had to be 3-4 times larger than what is shown here. One of those versions is described in [34].

Algorithm mapping

The mapping of the algorithm onto the VIP architecture is made straightforwardly, by assigning one PE to each range bin, which means that the PE array will require at least 4000 processing elements (although not on one chip). Since we propose to use a PE array where the number of PEs is an even power of two the array will have 4096 PEs. With the approximate MAC complexity given above and a target PE clock frequency of 50 MHz we find that one MAC operation must be performed in less than 70 clock cycles. In the original VIP architecture the optimum MAC performance is *8b,* i.e. 8 times the word size *b*, and with a data precision between 10-16 this indicates that the PE performance must be increased in order to meet the real-time requirements.

The actual number of samples per CPI varies continuously, which must be taken into account in the chip implementation. Also, the algorithm contains parameters which change as the number of samples and the pulse code change. The use of different pulse codes (and possibly different carrier wave frequencies) is necessary to find the currently best combination of target detection and clutter suppression.

All IO communication can be made transparent to the processing. This is achieved by processing one CPI (row) while the next is being input and the previous result is being output.

3.3 PE design

Clock frequency

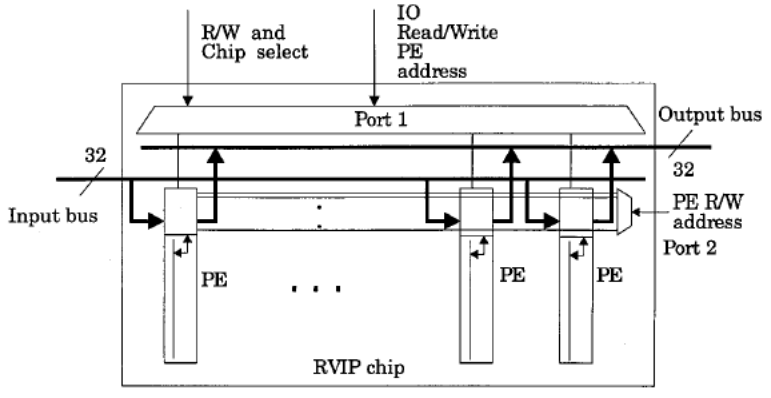
It was predicted that an on-chip clock frequency of 50 MHz would be attainable. With a prf of 900 Hz this leaves 55000 VIP clock cycles for the processing of one CPI in the preceding LPD-algorithm.

The IO register

In the original VIP design, described in section 1.3, the IO-system consisted of synchronous shiftregisters. When, as in this application, the IO data frequency differs from the PE clock frequency such a design requires large amounts of buffer memory. Therefore, an asynchronous solution was chosen, and for RVIP we decided to replace one of the shiftregisters with a dual ported (orthogonal) memory, see Figure 3.2. IO data is written into and read from the array via port one, while the PEs access the data via port two. More details will be given below.

The addressable memory makes the IO-system more flexible since data may be written into the array in any order, e.g. bit-reversed for FFT operations, but if the addresses are sequential the functionality equals that of an asynchronous shiftregister. In the LPD algorithm the dual-ported design is especially favorable when all PEs are not in use. A shiftregister would have to spend idle cycles bypassing empty PE positions while the dual port system can be reprogrammed and stay fully efficient. Furthermore, the size of the dual-ported cells are smaller than the shiftregister cells and the power consumption is much less. However, the additional address decoding hardware needed on the chip evens out some of the area gain [42].

FIGURE 3.2.



The IO register design on the chip level.

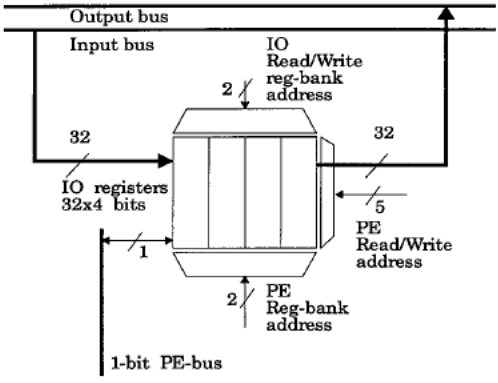
To obtain a large enough IO bandwidth with a moderate number of IO connections the IO memory is organized as 4x32 bits in each PE. This means that each PE contains four 32-bit words, called register banks, see Figure 3.3. In this way each IO cycle contains one read and one write operation of 32-bit words. It is natural to input and output 128 bits of data to/from each PE concurrently with the processing of one row (bearing). That is, while data is being processed in the PEs, the next row of data is input and the previous result output. Then, in the blank interval when no data is input, the PEs read the new input data from the IO register into

the RAM and write the result of the previous bearing into the IO register. To ensure that the external IO controller does not access a certain register bank at the same time as the PEs are accessing it, a synchronization of the register bank access must be implemented by the micro controller.

Note that in the presented algorithm only two of the four 32-bit registers are needed.

The maximum data sampling frequency is expected to be approximately 5 MHz. Including some future needs that requires all four parallel register banks the required IO data rate is 20 MHz. This IO data rate and the width of the IO interface gives an input *and* output capacity of 20*32 = 640 Mbit/s for each chip, i.e. a total IO capacity of 1.2 Gbit/s per chip.

FIGURE 3.3.



The RVIP IO register and register bank layout in the PEs.

The internal shiftregister

The internal shiftregister in RVIP has been augmented to 16 bits to accommodate for the higher data precision used in radar signal processing compared to greyscale image processing. When longer words than 16 bits are used several shift operations are needed. It is not possible to run the shift operation synchronously at the internal RVIP clock speed of 50 MHz since the signal may pass a circuit board and possibly a back-plane to the next chip. The maximum shift frequency will probably be around 25 MHz. This should not be a serious bottle-neck, however, since most of the shifts are short, and longer shifts can often be made concurrently with the processing. If the shift frequency is 25 MHz the number of 50 MHz cycles needed to shift b bits of data to a PE A steps away is

$$N = 2A\left\lceil \frac{b}{16} \right\rceil + 2b \tag{3.14}$$

where the $2b$ factor comes from loading and unloading the shiftregister.

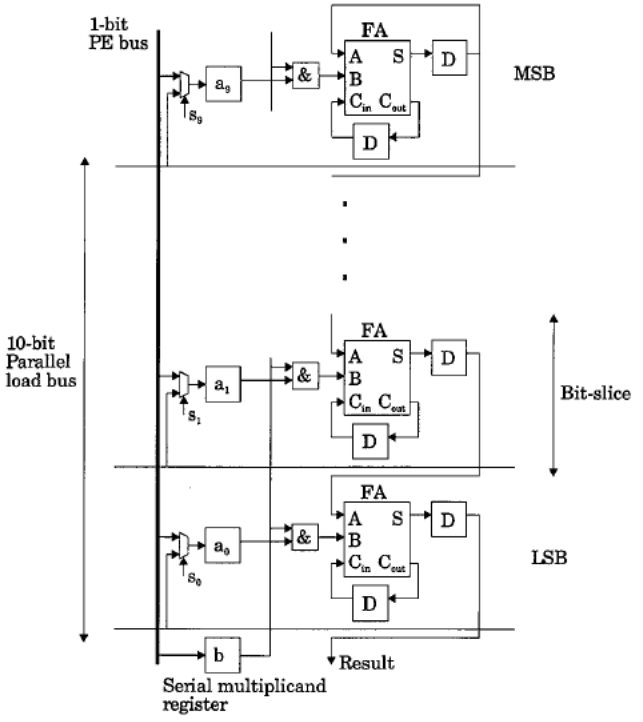
The internal RAM

The size of the internal RAM was set to 2048 bits per PE since at least 1700 bits are needed in some versions of the LPD algorithm [47]. Although a dynamic memory requires less area the RVIP memory is static to avoid implementing the refresh logic needed for dynamic RAMs. For reset purpose one memory cell is a ROM containing the constant 0.

The serial-parallel multiplier

A serial-parallel multiplier is a hybrid between a bit-parallel and a bit-serial architecture [41]. Since there is no carry propagation in such a design it can easily utilize very high clock frequencies [14]. Another important feature is that the length of the multiplicand is arbitrary whereas the coefficient length in RVIP is ten bits. The multiplier can be designed to accept both unsigned and two-complement integers, but we have chosen to hard-wire it for two-complement operation. To avoid confusion we denote the multiplicand in the parallel part of the multiplier the *coefficient* and the multiplicand applied serially the *multiplicand*, see Figure 3.4.

FIGURE 3.4.



The serial-parallel multiplier. The select signals s_i are used to input either one bit from the PE bus to the multiplier coefficient register a_i or a 10 bit coefficient from the external parallel load bus to $\overline{a}$. The feedback of the sum in the sign (MSB) bit-slice makes two-complement multiplication possible.

Initially all internal *D* registers are reset to zero. The ten-bit coefficient is loaded to the parallel register a_0-a_9. Then, the multiplicand is applied bit by bit starting with the least significant bit, the lsb, to the one-bit register *b*. For each new bit in the multiplicand the new sum is shifted one step and the lsb of the result is output. After the last multiplicand bit (the sign bit) has been applied the multiplications continue using this bit until all 2b-1 bits have been output.

A common coefficient can be loaded to all PE multipliers concurrently and bit-parallel from an external array controller via the external load bus. Of course, the coefficient can also be loaded bit-serially from the PE bus. The parallel load bus reduces the execution time for multiply operations from *4b* to *3b* cycles when all PEs use the same coefficient (as in convolutions and correlations).

The ALU-Accumulator

The most common operation when using RVIP for radar signal processing is convolution and correlation (a correlation is a convolution with a non-rotated convolution kernel). This operation can be written

$$h(x) = \sum_{\alpha} f(x-\alpha)\, g(\alpha) \tag{3.15}$$

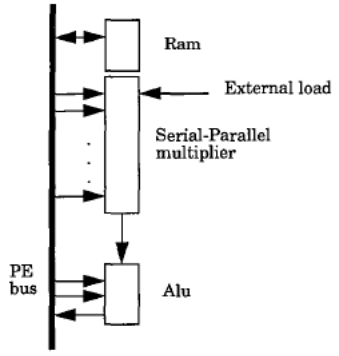
In RVIP a typical convolution is made with $f(x)$ in the shiftregister so that different $f(x - \alpha)$ are obtained by shifting. The coefficients $g(\alpha)$ are common for all PEs and are loaded direct to the parallel port of the multiplier from the external controller as described above. The total convolution consists of multiply and accumulate, MAC, operations. In the original VIP a MAC operation required *8b* cycles for *bxb* bit multiplication and *2b* bit accumulation (without further bit-expansion in the accumulation and with serial coefficient load). This is illustrated in the pseudo code below

```
        /* MAC operation: m = a*c, acc += m */
load a                  /* load coefficient a */
for(i=0;i<b;i++)        /* loop over multiplicand */
   mult = c(i)          /* load multiplicand bit */
   alu = m(i)           /* move mult-res bit to alu */
   alu = acc(i)         /* move acc-bit to alu */
   acc(i) = sum         /* store new acc-bit */
end
for(i=b;i<2b;i++)       /* loop over mult content */
   alu = m(i)           /* move mult-res bit to alu */
   alu = acc(i)         /* move acc-bit to alu */
   acc(i) = sum         /* store new acc-bit */
end
```

The parallel coefficient load reduces the MAC complexity to *7b*, and the LPD algorithm would then require approximately *830*7b* = 70000 clock cycles with *b=12*, which this still exceeds the available 55000 clock cycles. The limiting factor is the PE bus which only can transfer one bit at the time. In the above program we notice that the partial results *m(i)* are transferred from the multiplier to the ALU over the PE bus. Therefore the redesign aimed at transferring *m(i)* without the use of the PE bus.

First we redesigned the connections between the multiplier and the ALU so that the multiplier output connects directly to the ALU input, see Figure 3.5. This gives a gain of *2b* cycles since the ALU can be loaded with both the previous accumulation result and the multiplier output simultaneously. However, the decrease to *5b* for a MAC operation still gives around 50000 clock cycles for the LPD algorithm example, and further improvements were needed to ensure that the RVIP actually could implement the algorithm within the given specifications.

FIGURE 3.5.



The accumulator-ALU design with the multiplier output connected direct to the ALU.

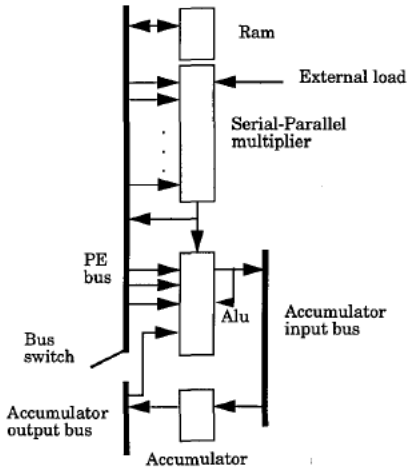
The second redesign incorporates a dual-ported 32-bit accumulator register. This makes it possible perform the *accumulation in parallel with the multiplication* without loss of clock cycles. The accumulator is separated from the ram memory and PE bus as seen in Figure 3.6. Using this architecture the pseudo code for the MAC operation will be

```
        /* MAC operation: m = a*c, acc += m */
Load a                  /* load coefficient a */
mult = c(0)             /* load one bit of b */
for(i=0;i<b-1;i++)      /* loop over multiplicand b */
            /* save new acc bit, load mult bit and alu */
            /* performed in one clock cycle */
   alu = acc(i), alu = m(i), acc(i) = sum(i),
   mult = c(i+1)
end
for (i=b-1;i<2b;i++)    /* loop over multiplier */
                        /* sum to acc, load alu */
   alu = acc(i), alu = m(i), acc(i) = sum(i)
end
```

We see that there is a one step (one clock cycle) pipeline between the multiplier and the ALU (accumulator). That is, the result from the previous multiplication step is used at the same time as the next is computed.

The cycle count for the MAC operation is then close to $2b$ for parallel coefficient load and $3b$ for serial coefficient load. This accumulator-bus structure also improves multiple addition accumulations from $3b$ to b cycles per b-bit operand. Thus the 830 MAC operations should be possible to compute in less than 30000 clock cycles (exact cycle counts will be given below).

FIGURE 3.6.

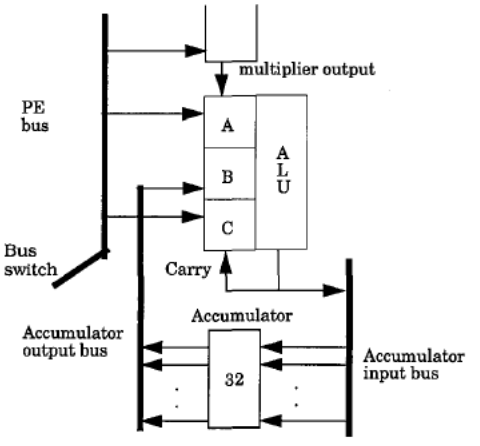


The accumulator-ALU design with the accumulator separated from the PE bus.

The ALU has three inputs as seen in Figure 3.7. The connections within the ALU are rather inflexible to keep down the width of the instruction word. Input A is dedicated to the PE bus and the multiplier output. Input B is dedicated to the accumulator output, and input C is dedicated to the PE bus and to internal carry-feedback.

To unload the accumulator the output bus may be connected to the PE bus with the bus switch. However, all data input to the accumulator must pass the ALU.

FIGURE 3.7.



The design of the Multiplier-ALU-Accumulator connections.

The implemented ALU functions and their micro-codes are found in Table 3.2. All arithmetic operations use a carry bit stored in the carry register C. The Multiplex (Mux) operation is used for data dependent choices such as maximum finding, and the xor operation is used for absolute value computations.

The four-bit micro-code word has room for more logic functions but the instructions in Table 3.2 was found to be enough in the LPD algorithms. Two-input *AND* and *OR* functions can easily be implemented using the carry output. With C pre-set to zero an Add will give *A AND B* as a carry result. And with C pre-set to one an Add will give *A OR B* as a carry result. Thus, we can implement arbitrary Boolean expressions.

TABLE 3.2.

Code	A	B	C	Out	Function
0	-	-	-	-	Nop
1	Peb	-	-	A	Load acc
2	Peb	-	-	$\overline{A}$	Load acc inverted
3	Mult	-	-	A	Load acc from mult
4	Mult	-	-	$\overline{A}$	Load acc inv. w. mult
5	-	-	Peb	-	Load c
6	Peb	Accb	Cf	A+B	Add from bus
7	Peb	Accb	Cf	B-A	Sub bus from acc
8	Mult	Accb	Cf	A+B	Add from mult
9	Mult	Accb	Cf	B-A	Sub mult from acc
10	Peb	-	-	A xor C	Xor A and C
11	Peb	Accb	-	CA+$\overline{C}$B	Mux with c to acc

PE performance

The number of cycles required for a number of arithmetic/logic functions are collected in Table 3.3. We give cycles counts with and without bit-expansion. Thus, in the left cycle column a *b+b* and a *bxb* operation gives a *b* bit result and in the right cycle column a *b+b* operation gives a *b+1* bit result and a *bxb* operation a *2b* result. We see that a division has $O(b^2)$ performance, but since the LPD algorithm only contains one division this is acceptable. For the complex operations the notation is that *p* is the precision and *b* the total data size, i.e. if the complex data has 2x16 bits *b* is 32 and *p* is 16.

A complex division is performed by first multiplying divisor and dividend with the conjugate of the dividend, thus achieving a division with a real valued integer. Thus, the complexity equals the complexity of two complex MAC operations and two division operations.

TABLE 3.3.

Operation	Data location			Cycles	
	Op 1	Op 2	Result	No bit expansion	Bit expansion
Add/Sub	RAM	RAM	RAM	3b+1	3b+2
Add/Sub	RAM	Acc	Acc	b+1	b+2
Absolute value	RAM	-	RAM	3b+1	-
Absolute value	RAM	-	Acc	2b+1	-
Mux	RAM	RAM	RAM	3b+1	-
Mux	RAM	Acc	Acc	b+1	-
Max/Min	RAM	RAM	RAM	5b+2	-
Max and Min	RAM	RAM	RAM	8b+4	-
Mult	RAM	RAM	RAM	4b+1	5b+1
Mult	RAM	RAM	Acc	3b+1	4b+1
Mult	RAM	External	RAM	3b+2	4b+2
Mult	RAM	External	Acc	2b+2	3b+2
MAC/MSUB	RAM	RAM	Acc	3b+1	3b+1
MAC/MSUB	RAM	External	Acc	2b+1	3b+1
Complex mult	RAM	RAM	RAM	14p+4	16p+4
Complex mult	RAM	External	RAM	10p+8	12p+8
Complex MAC/ MSUB	RAM	RAM	RAM	16p+4	20p+4
Complex MAC/ MSUB	RAM	RAM	Acc	12p+4	12p+4
Complex MAC/ MSUB	RAM	External	RAM	12p +8	16p+8
Complex MAC/ MSUB	RAM	External	Acc	8p+8	8p+8
Division	RAM	RAM	RAM	~$7b^2$	-
Complex division	RAM	RAM	RAM	~$14p^2$	-

External memory

The original VIP architecture incorporated external memory ports shared by eight PEs. This means that with standard eight-bit wide memory chips we need two per each 128 PE, using one-bit memory chips we need 16 per each 128 PE. Both these solutions have drawbacks and the memory bandwidth is still rather small, approximately only 6 Mbits per PE. Increasing the bandwidth to external memory chips is difficult and costly due to pin count limitations on the RVIP chip. Thus, instead of this external port design we chose to extend the internal RAM and the IO capability as shown above. This only increases the memory bandwidth to around 10 Mbits but it simplifies the VLSI layout of the RVIP chip.

3.4 The chip design

In the LPD algorithms the number of samples in each CPI (i.e. one image row or bearing) is large, up to 4000. This means that up to 4096 Processing Elements are needed in the system if each PE is to process one sample in a row-parallel fashion. With the current VLSI technology it is not possible to integrate that many RVIP PEs on a single chip. Therefore an array of chips connected in series is needed. This does not change the design of the individual PE, but it puts special demands on the overall system design.

RVIP was designed to be implemented in a 0.8μm CMOS SRAM process. The SRAM process has a second, high resistance, poly-silicone layer which is used to design 4-transistor NMOS SRAM cells with pull-up resistors instead of p-transistors. This makes it possible to design the SRAM cells with only a third of the area of a 6-transistor cell designed with p-transistors in a standard (2-metal) CMOS process. A 2-phase (pseudo 4-phase) clock is used. The chip area was maximized to around 100 mm^2. The estimated areas for each part in a 128 PE chip are collected in Table 3.4. The total chip area was estimated to be 115 mm^2 including instruction decoders and IO pads [47]. The power consumption was estimated to 1.8 W at 5V. The total number of connections on each chip is around 250, of which approximately 150 are used for IO and control and 100 for power distribution.

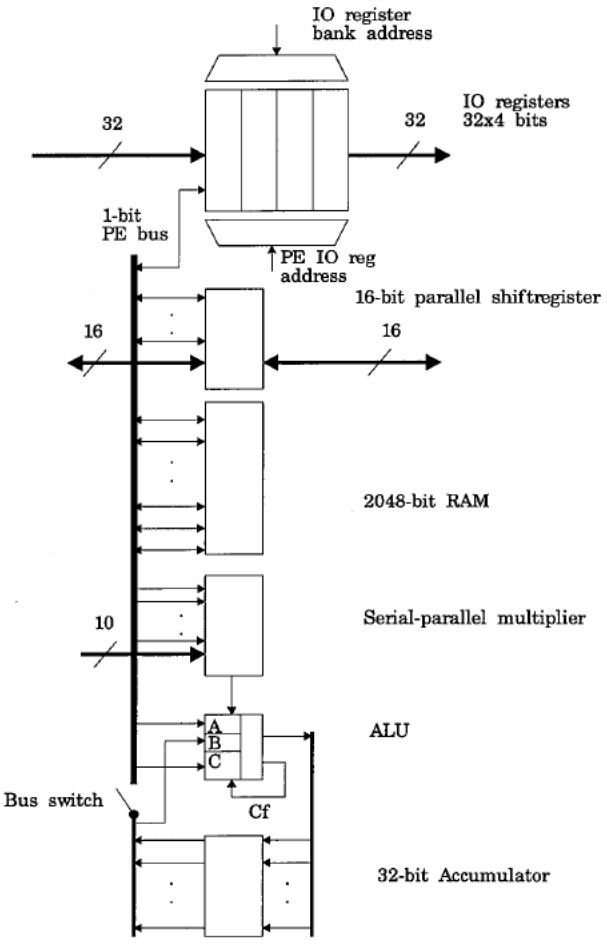
TABLE 3.4.

Chip part	Area mm^2	Area% of total
IO reg	9.4	12
S-reg	6	7.5
Multiplier	15	19
ALU	4	5
Acc	7.5	9
SRAM	38	47.5
SUM	**80**	**100**

The RVIP chip layout

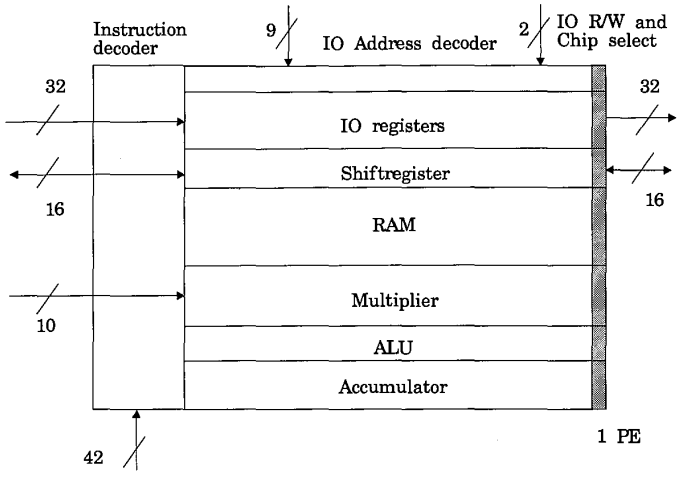
The final RVIP PE and chip layouts are shown in Figure 3.8 and Figure 3.9, respectively. Each chip contains 128 PEs. The peak performance in each 128 PE chip is around 320 MMAC operations per second for 10x10 bit data. The peak performance of a 32 chip 4096 PE RVIP array is around 10 GMAC operations per second for 10x10 bit operations.

FIGURE 3.8.



The RVIP PE.

FIGURE 3.9.



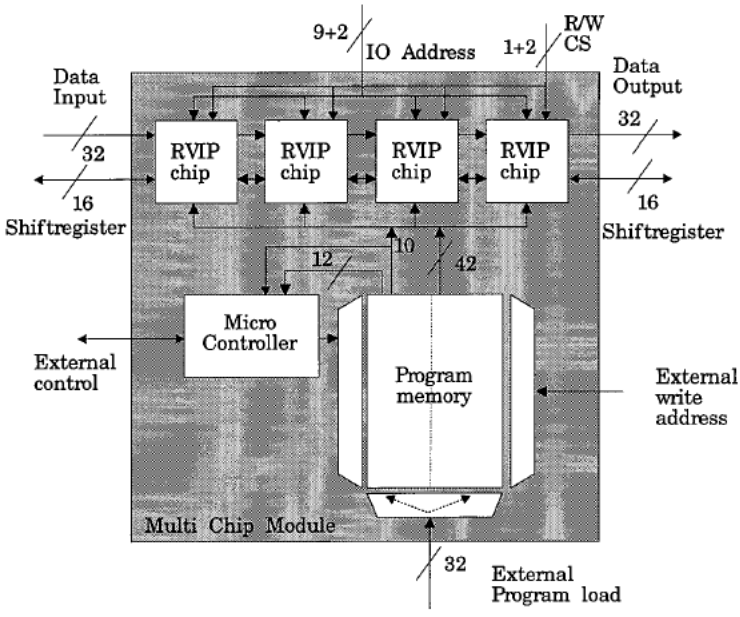
The layout of the RVIP chip.

3.5 System design

An RVIP LPD radar system consists of 4096/128 = 32 chips connected in series. If a standard, e.g. pin-grid, chip package is used the package size would be much larger than the chip due to the large number of external connections. To reduce the total system size it was decided to use a Multi Chip Module, MCM, package where four chips are bonded together on a 2"x2" ceramic wafer, see Figure 3.10. This reduces the system size since many of the pins on the chips can be shared on the MCM level. Furthermore, a micro controller, which is described below, was included in each MCM.

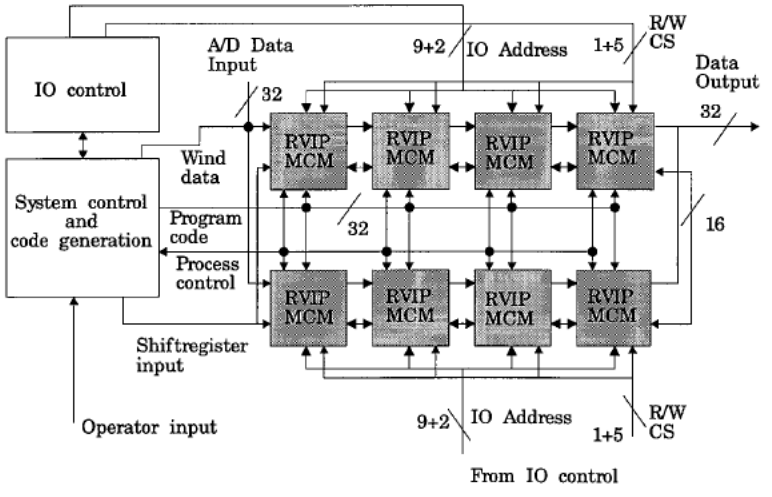
Thus, the RVIP system will be built with a number of Multi Chip Modules, MCMs. Each MCM contains 512 RVIP PEs and a complete 4096 PE system will then use eight modules in series, see Figure 3.11. Internally each module runs at 50 MHz, but due to synchronization issues all communication to and between modules is limited to 20 MHz. The system controller selects the current program and distributes the micro-code to the modules synchronously.

FIGURE 3.10.



RVIP Multi Chip Module packaging. Each module is a stand-alone 512 PE RVIP module with an internal micro controller.

FIGURE 3.11.



The RVIP system layout with multiple MCM modules, system controller and IO controller.

Micro controller design

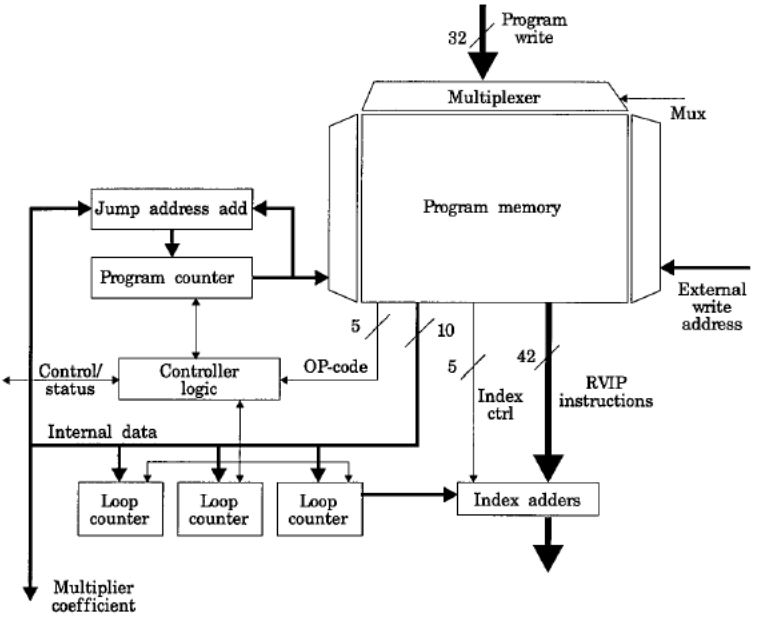
The size and complexity of an RVIP system makes it very hard to have a common instruction sequencer distributing instructions synchronously at the RVIP clock frequency 50 MHz. Therefore we included a local micro controller on each MCM so that each module can run at full speed. To make this scheme work it must be possible to either load the complete current program code to the modules and then execute the program, or to load the program code piecewise to the modules during execution. Since the RVIP program is constantly changing it is impossible to use the first method and we must be able to load new code to the MCM program memories during execution.

At first this might seem impossible by the same reason it was impossible to have a common instruction sequencer. However, if we make the MCM micro controller capable of expanding loops in the code we can obtain a lower data rate to the MCMs. Since the RVIP architecture is bit-serial, loops with only a few instructions repeated over the word size b are very common. This gives a ratio of approximately b between the number of instructions the controller receives and the number of instructions the controller issues. In the simulations of an LPD radar algorithm, with b in the order of 10-16, the ratio was indeed 1/10 between the number of non-expanded micro-code instructions and the number of executed instructions. Thus, it is possible to distribute the VIP code to the micro controller(s) at a much lower data rate than the instruction rate. This lower data rate is used both to decrease the width of the program load bus and the data rate on this bus so that in RVIP we distribute the 64-bit RVIP instruction words over a 32-bit wide bus at 20 MHz and still run the RVIP array at 50 MHz.

The controller reads the micro code and unwinds low level loops. Three nested loops can be used, and one of these can be indexed, see Figure 3.12. The micro controller also communicates with the external high level controller to synchronize the use of the register banks in the IO registers. Loading multiplier coefficients to the RVIP chips are made from the micro controller with data in the internal data field.

The index hardware uses two counters, one loop counter and one index counter. The loop counter is decremented for each lap in the loop and the loop ends when it reaches zero. The index counter starts at zero, and it is incremented when the loop counter is decremented. The index counter value is then added to the respective address fields to be indexed. The index adders can also be used to issue a NOP instruction, which is useful if VIP must wait for synchronization of for instance IO transfers. The micro controller loop logic uses simultaneous decrement and test for zero, and thus the loop will end when the counter is decremented from zero to minus one. This scheme is used to save code, since only one instruction instead of two is needed when test and decrement are separated. Thus, a bit-serial loop can be written with only one instruction.

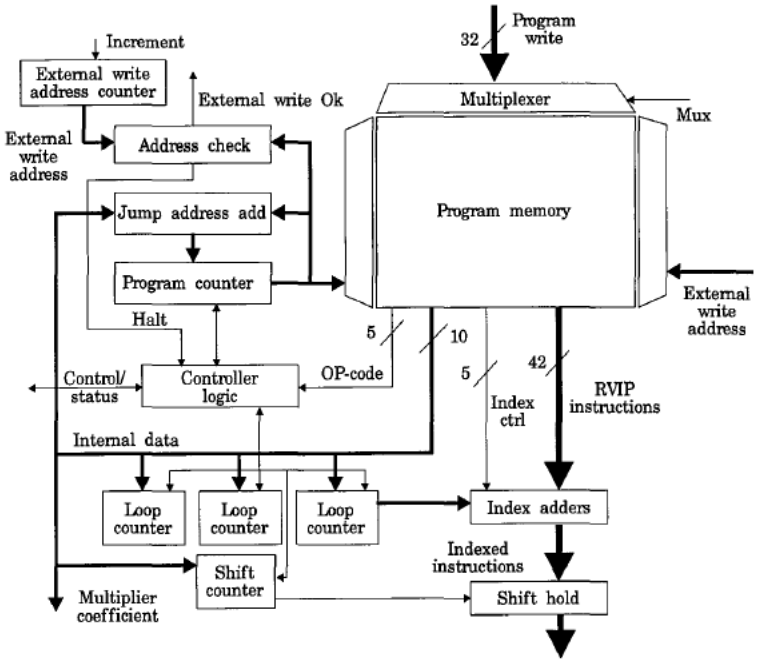
FIGURE 3.12.



A micro controller with loops and external program write.

The program counter address the two-ported instruction queue in a circular FIFO fashion, and new instructions are written asynchronously from the external controller, see Figure 3.13. External write of new instructions is made at addresses which are at a fixed distance below the current program counter value to avoid overwrite of loop starting points. How large the queue and the writing distance need to be will be determined using a simulator. If the queue is empty the controller automatically waits until a new code word is written. A shift counter is used to be able to synchronize long shift operations concurrent with the rest of the program. The counter is loaded, and as long as it is greater than zero the shift instruction is held in the register. The controller can be programmed to wait until the shift counter reaches zero, and until then the RVIP instructions are set to be NOP instructions. An overview of the complete micro controller design can be seen in Figure 3.13.

FIGURE 3.13.



A more complete description of the micro controller. Small arrows depict control signals and wide data signals.

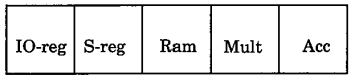
The functions needed in the micro- controller are summarized in Table 3.5 below.

TABLE 3.5.

Code	Mnemonics	Function
0	Nop	No control op
1	Load1	Load loopcounter 1
2	Load2	Load loop counter 2
3	Load_index	Load index-loop counter
4	Load_shift	Load shift counter
5	Load_mult	Load multiplier coeff
6	Dec&Jmp1	Decrement lc1, jmp if not zero
7	Dec&Jmp2	Decrement lc2, jmp if not zero
8	Dec&JmpI	Decrement index lc, jmp if not zero
9	Wait_shift	Wait for shift operation to finish
10	Wait_IO	Wait for external IO to finish
30	Break	Simulator break point
31	Halt	Stop execution, signal user

The micro controller needs a 10 bit wide internal data bus apart from the 4 bits needed for the micro controller instructions (the use of 5 bits here is to allow for future extensions of the controller functions). The internal data bus must be able to hold multiplication coefficients and relative jump addresses. However, the relative jump addresses can be left out if the controller keeps track of the loop start positions. The indexation can be applied independent to the different parts of RVIP, and thus five bits are needed to decode which PE part that is indexed. The coding of the indexation vector can be seen in Figure 3.14. This vector can probably be decoded into a smaller vector since all combinations are not meaningful which has been subject to further studies regarding newer VIP designs [35].

FIGURE 3.14.



The coding of the index vector.

RVIP micro-code

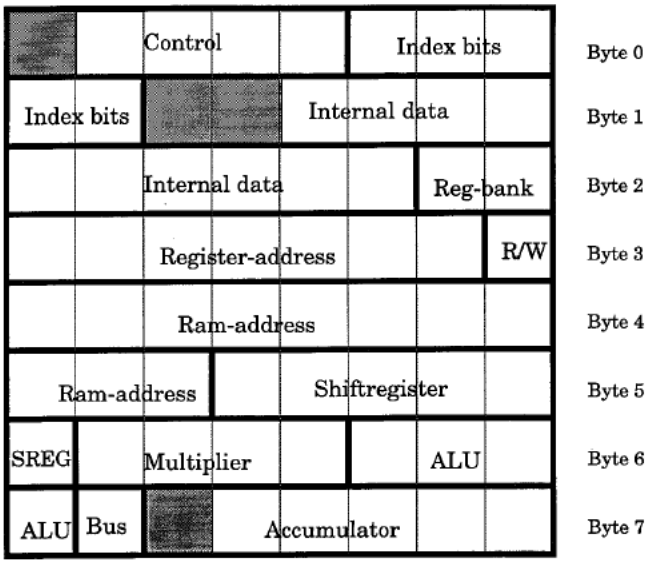
The total number of bits needed for the RVIP micro code word is found in Table 3.6. We have coded the micro word so that all parts of the PE can be used independently. This makes it possible to use broadcasting, i.e. one unit is writing to the bus and several units are reading, but other possibly more compact and/or effective codings have been proposed [35], [54].

TABLE 3.6.

Part	# bits
Control	4
Index vector	5
Data	10
IO-reg	9
shiftreg	6
Mult	4
ALU	4
Acc	5
Bus-switch	1
Ram	12
SUM	60

The instruction field coding is shown in Figure 3.15. We have augmented the micro word to 64 bits by extending micro controller field to 5 bits, the internal data bus to 12 bits and the accumulator address to 6 bits. This allows more functions in the controller, the multiplier to be extended to 12 bits, and the accumulator to be extended to 64 bits without re-coding the instruction field.

FIGURE 3.15.



The coding of a VIP instruction. The shaded areas represent the extensions of the micro word mentioned in the text.

3.6 EMW RVIP modifications

Ericsson Microwave systems made a few changes to the RVIP design when implementing the chip. Due to increased precision requirements the parallel part of the multiplier was increased to 12 bits. A one-bit activation register was inserted in the PE. This register is used for data dependent operations and to "turn off" unused PEs when less than 4096 samples are collected in each CPI. Compared with the estimates in Table 3.4 the RAM now requires approximately 60% of the PE area due to the use of a 3-metal CMOS process without high-resistance poly-silicone. To increase the yield and to reduce the risk of the project the final chip has only 64 PEs instead of the planned 128, but instead eight chip are used in each MCM module, still giving 512 PEs in each module [4]. Also, JTAG boundary scan logic has been included to faciliate testing of the chip and modules. The expected power consumption for a 64 PE chip is around 2 W. The number of transistors on each chip is around 1.1 million.

The design of the RVIP chip, micro controller and system started early 1994, and the first chips have been successfully tested during the summer of 1995.

3.7 Results

With the modified RVIP architecture given above the implementation of the algorithm is straight-forward. The total number of cycles needed are summarized in Table 3.7.We see the RVIP array can make the processing in less than half of the pre-specified maximum number of clock cycles 55000. This indicates that it actually is possible to process two range bins in each PE, using only 2048 PEs to implement this particular LPD algorithm.

TABLE 3.7.

Algorithm part	RVIP cycle count
Input data transfer	24
Wind comp	188
MTI-filter	410
Pulse compression	20714
Envelop detection	384
CFAR	2061
Video integration	499
Thresholding	30
Output data transfer	16
Sum	**24326**

In section 3.2 we found that the complexity of the LPD algorithm was approximately 3-4 G MAC operations. If we assume that we wish to implement the LPD algorithm using word-parallel Digital Signal Processors, DSPs, we can then find an approximation of the number of such processors required to reach the given performance. To make the comparison fair we assume that the clock frequencies of the two systems are equal, i.e. 50 MHz, and that each DSP is ideal and can execute one MAC operation each clock cycle. The number of required DSPs, N_{DSP}, is then between

$$\frac{3\times 10^9}{50\times 10^6} = 60 \le N_{DSP} \le 83 = \frac{4.15\times 10^9}{50\times 10^6} \tag{3.16}$$

That is, the algorithm which could be solved by 2048/128 = 16 RVIP chips requires at least 60 ideal DSPs. In reality however, no ideal 50MHz DSP processor exists and we have not considered the problem of partitioning the LPD algorithm into sub-algorithms suitable for single DSPs. Thus, very conservatively we claim that for this application each RVIP chip is at least three times as powerful as a sequential word-parallel processor operating at the same clock frequency. Note however, that if the pulse code compression is implemented in the Fourier domain the number of ideal DSPs required decreases to around 8-10.

The internal memory required by this algorithm is less than 500 bits. The motivation for a 2Kbit memory is the ability to store intermediate results from several different lobes (bearings) at the same time, and, as in the MPD case studied later, to use one array to compute several input channels of radar data concurrently.

Chapter 4

The Flight VIP

4.1 General

A Medium Pulse repetition Doppler, MPD, algorithm is studied in this chapter. The algorithm is presented and implemented in two parts. The first part is the CPI processing and the second part is the resolving and integration of data from the nine latest CPIs. We call the architecture suitable for CPI processing FVIPa. For the resolving processing we propose a slightly different architecture, called FVIPb. The only difference between the two FVIP architectures is the size of the IO registers and the internal memory.

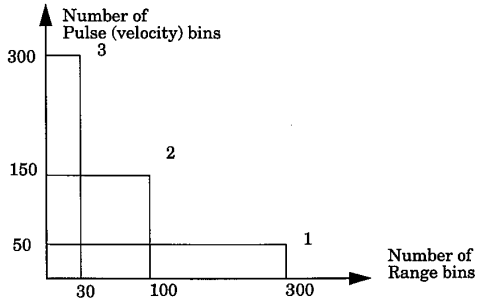
First we present the CPI and resolving algorithms, and find suitable mappings of the algorithms on a linear array. Then we present the required changes in the PE architecture, and finally the results.

The real-time requirement is that the delay between the acquisition of a CPI and the output of target information from the resolving must be less than 100 ms.

4.1.1 CPI processing

The different CPI formats to be accommodated for can be seen in Figure 4.1 and data for them are collected in Table 4.1. The formats differ in trade-off between range and velocity resolution. Within each CPI format prf switching between three prfs is used, see section 2.2. In the table we have indicated the maximum number of range bins used during the switching. The sampling frequency is fixed at 2 MHz. All transmitted pulses are not sampled as is indicated in the table, and we see that the number of samples processed in each CPI range from 7710 to 12900. The number of samples actually processed are selected so that the number of pulse bins for each range bin is an even power of two at the beginning of the FFT transform. We denote the number of range and pulse bins processed within a CPI as R and P.

FIGURE 4.1.



The three different CPI formats.

TABLE 4.1.

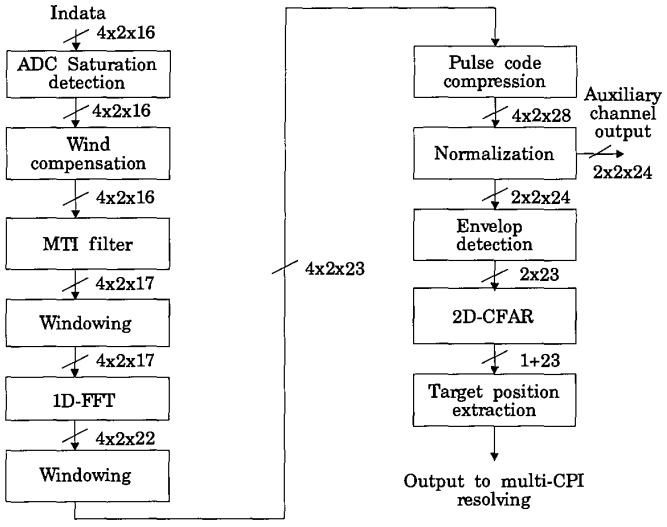
CPI format	1	2	3
Sampling frequency, f_s	2 MHz	2 MHz	2 MHz
PRF, f_p	~6 Khz	~18 KHz	~60 KHz
Prf switching	3	3	3
Range resolution, Δr	75 m	75 m	75 m
Range bins, R	<300	<100	<30
Processed range	< 22.5 km	< 7.5 km	< 2.25 km
Unambiguous range, r_u	<25 Km	<8.3 km	<2.5 km
Transmitted pulses	50	150	300
CPI time	8.3 ms	8.3 ms	5 ms
Processed pulses, P	33	129	257
Velocity resolution, Δv	<0.78 km/h	<0.6 km/h	<1 km/h
Unambiguous velocity, v_u	<12.5 km/h	<37.5 km/h	<125 km/h
Samples	**<9900**	**<12900**	**<7710**

An overview of the CPI algorithm, which is very similar to the general radar signal processing algorithm given in section 2.9, can be seen in Figure 4.2. It does not exactly represent any of the algorithms used by EMW today, but it has been designed to represent a typical MPD radar algorithm in computational complexity. The input to the CPI processing is four quadrature *(I,Q)* channels from a two-dimensional monopulse antenna with a guard horn, see section 2.6. We call the four channels *main*, *guard*, *auxiliary1*, and *auxiliary2*. All four channels are processed independently but identically until after the normalization when the auxiliary channels are output as quadrature video data. This means that the actual monopulse processing to acquire additional target bearing information is handled by the system control processor.

The input channels are sampled with 16-bit precision and the maximum intermediate precision is 28 bits.

Analogous to the LPD algorithm studied previously we also study the MPD algorithm complexity, i.e. the number of MAC operations required. Using this result we can find an appropriate FVIPa parallelization.

FIGURE 4.2.



An overview of the CPI processing algorithm.

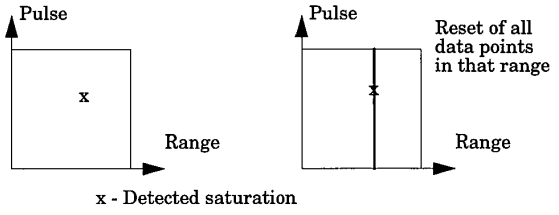
ADC saturation detection

The ADC saturation detection is a function where all samples in a range bin are set to zero if any sample in that range bin has an envelop above a global threshold, see Figure 4.3. This is used to suppress very strong direct radar reflections. Formally this can be expressed as

$$\forall r\Big(\exists p\Big(\sqrt{I^2(r,p)+Q^2(r,p)} > Thresh\Big) \Rightarrow \forall p\,(\,(I+jQ)_{ADC}(r,p\,) \rightarrow 0)\,)\Big) \qquad (4.1)$$
$$r \in [1,R], p \in [1,P]$$

This operation requires that a global OR of all test results within each range bin is distributed to all PEs containing data from that range bin. This indicates that special processing or a new PE design is required for our SIMD implementation. The computational complexity for each channel depends on the number of pulses in the CPI, since in the worst case a result must be distributed to P different processors or memory locations. The envelop detection and thresholding requires approximately 5 MAC operations and we approximate the total complexity to $P+5$ MAC operations for each channel.

FIGURE 4.3.



The ADC saturation detection.

Wind compensation

The wind compensation is a vector rotation with an angle α which is common for all samples in that CPI. The complexity is, as in the LPD algorithm, 4 MAC operations for each channel.

MTI filter

The MTI filter is a first-order differentiation in the pulse direction

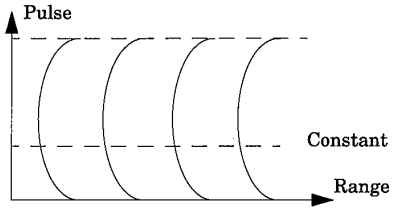
$$(I+jQ)_{MTI}(r,p) = (I+jQ)_{WC}(r,p+1)-(I+jQ)_{WC}(r,p) \quad (4.2)$$

That is, the output is the result of a subtraction between two neighboring pulse values. The complexity is 2 MAC operations for each channel. After this operation the last sample in each pulse is discarded, giving a number of pulse samples which is a power of two.

Windowing

The window is real-valued and varies in the pulse direction, but it is common for all range bins, see Figure 4.4. Up to *P* different window values must be distributed to the array.The complexity is 2 MAC operations for each channel.

FIGURE 4.4.



Approximate envelop of the window parameters.

Fourier transform

The complexity of a *P*-point FFT operation is approximately *(PlogP)/2* butterfly operations, which approximately equals *4PlogP* real-valued MAC operations. For each of the *P* data points this equals a complexity of *4logP* MAC operations. The implementation of FFTs on FVIPa will be treated in more detail in chapter 5 below.

Windowing

This is a real-valued window, analogous with the previous window, varying in the frequency direction and it has a complexity of 2 MAC operations for each channel.

Pulse code compression

The pulse code compression correlation kernel is complex and its length is approximately 10% of the number of range bins.

$$(I+jQ)_{PC}(r,p) = \sum_{i=0}^{\frac{R}{10}} (C+jD)_{PC}(i) \cdot (I+jQ)_{PC}(r-i,p) \qquad (4.3)$$

The complexity for each sample in each channel is approximately *R/10* complex MAC operations, which equals approximately *4R/10* real-valued MACs.

Normalization

The normalization is a scaling to equalize the signal strength between the different CPIs. The peak strength after the pulse code compression is proportional to the pulse length which should be compensated for. The normalization factor is a complex number common for all samples in a CPI. The complexity is 2 MAC operations for each channel. After this step the auxiliary channel results are output while the main and guard channels are processed further.

Envelop detection

The envelop detection complexity is 5 MAC equivalent operations for each of the two remaining channels.

CFAR

The constant false alarm ratio, CFAR, processing is computed in a 6x6 range-pulse neighborhood of the main channel, and the following three criteria must be true in order to be a target.

```
The main sample must be more than X dB larger than
   7 of the 35 neighbor samples.
The main sample must be a local maximum in a 3x3 neighborhood.
The main sample must be more than Y dB larger than
   the guard sample.
```

To avoid the logarithm computation the first condition can be computed as

```
The main sample must be larger than at least
7 of the 35 neighbor samples scaled with the constant 10^(X/10).
```

The third condition can be computed analogously. The first and second conditions implements the actual CFAR (local maxima) computation, and the third condition is used to remove side lobe echoes detected by the guard horn, see section 2.6. The complexity of the CFAR is approximately 47 MAC operations since we need to perform 45 comparisons and two multiplications to compute the criteria above.

Target extraction

The target extraction delivers the range, velocity and envelop of the targets indicated by the CFAR to the resolving. One comparison is required to determine if the range/pulse bin contains a target. We note that to extract target position information from a VIP array a new PE design is required.

The number of targets extracted from each CPI is limited to one hundred. If more targets are present they are discarded. A natural way to select which targets to discard is to always process the targets that are closest to the radar platform.

CPI algorithm complexity

The computational complexity of the CPI processing is approximately $125+4(P+4\log P+4R/10)$. For $P=256$ and $R=30$ this gives a maximum of 1325 MAC operations for each sample acquired at 2 MHz. The computational demand is then 2.6 G MAC operations per second. If we allow for computation during the whole CPI interval this demand decreases to 2 G MAC operations per second.

CPI algorithm mapping on FVIPa

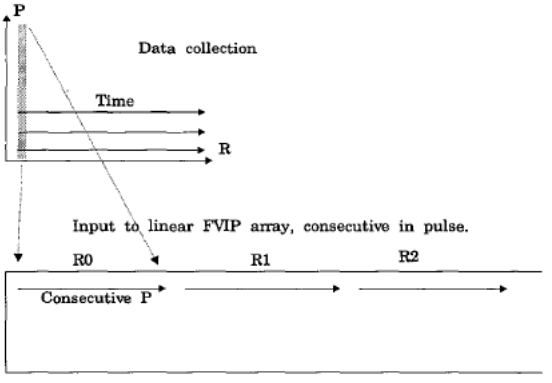
With a precision of 20 bits an FVIPa PE, using the same ALU-Accumulator-Multiplier design as RVIP, has a peak performance of around 1 Mega MAC operations per second. With a processing demand of 2 G MAC operations at least 2000 FVIPa PEs are needed to implement the CPI processing. However, we must find a suitable mapping of the CPI data matrices onto the FVIPa array. Each 2D CPI array contains between 7710 and 12900 samples, and if we process one sample per PE at least 12900 PEs are required. If we process two samples per PE at least 6450 PEs are required, and if we process four samples per PE at least 3225 PEs are required. This indicates that it might be possible to process four samples in each PE. However, since our linear array must emulate a 2D array we cannot hope to implement the algorithm without a large communication overhead, and therefore we propose to process two samples in each PE, giving at least 6450 PEs.

When emulating a 2D PE array with a 1D PE array there are at least two ways of organizing the data in the array. Either the linear array is aligned along the rows or along the columns of the 2D array. In the present application this means that neighboring PEs contains either neighboring range bins or neighboring pulse (velocity) bins. In any case neighbor communication in one dimension is straight-forward, but in the other dimension neighboring data are found up to several hundred PEs away. In FVIPa neighbor communication is accomplished by the shiftregister, causing overhead proportional to the shift length. Thus we should map the data onto the array so that the long shift operations are minimized.

Communication with neighboring pulse/velocity or range bins are needed in five parts of the algorithm. In the ADC saturation detection, the MTI filter and in the FFT the communication is with neighboring pulse bins. In the pulse code compression the communication is with neighboring range bins. In the CFAR the communication is with neighboring pulse and range bins and between the sum and guard channels. Of all these the communication in the FFT is the most complicated, and the total communication complexity is much larger in the pulse than in the range direction. Therefore, as shown in Figure 4.5, we have chosen to organize the linear array so that neighboring PEs have data from neighboring pulses within the same range bin, thus minimizing long shifts of data. This indata organization is possible since the IO registers

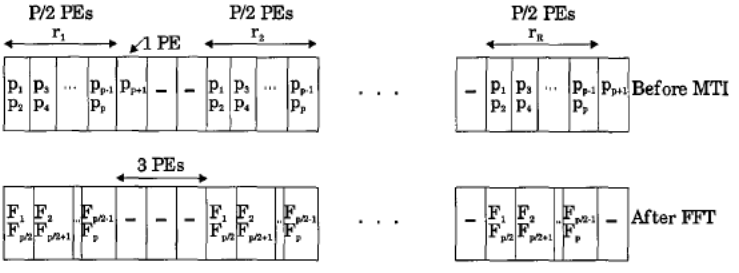
can be addressed in arbitrary order. As mentioned above, each PE is assigned to process two samples, and thus each PE contains two neighboring pulse samples, see Figure 4.6. Note that the sample positions are changed during the FFT, and this creates an extra overhead in the CFAR. This, together with two PEs without input data between each range bin makes the communication overhead to neighboring range bins *P/2+4* PEs. The use of empty PEs between range bins facilitates the implementation of the ADC saturation detection and the CFAR, as will be shown below, but it increases the maximum number of required PEs to (P/2+1)R+2R. However, this is still less than 7168 which is the closest multiple of 1024. The system layout will be discussed in more detail below.

FIGURE 4.5.



The indata and PE organization.

FIGURE 4.6.



A more detailed view of the data organization in FVIPa. The number of unused PEs between each range bin increases to three after the MTI filter when the last sample in each pulse is discarded.

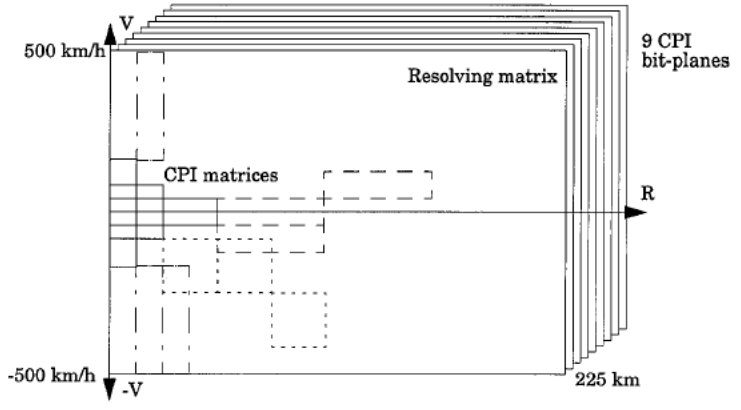
4.1.2 Resolving

The resolving is made out to 3000 range bins and 1000 frequency bins, extending the unambiguous range to 225 km and the unambiguous velocity to +/- 500 km/h, see Figure 4.7. The resolving can be described and implemented in several ways, and we describe the algorithm using an easy to understand implementation.

The input to the resolving is the target positions extracted from the nine latest CPIs. Initially the resolving matrix is reset to zero, and for each target extracted from a CPI a one is written into the resolving matrix at the correct positions. Note that since each CPI matrix is folded around its respective ambiguity limits in range and velocity several times within the resolving matrix each target gives several ones in the resolving matrix, compare with the range resolving example in section 2.2.

The resolving is then performed by finding all positions within the resolving matrix where at least three of the nine CPIs indicate a target. We note that a target echo may only contribute to one resolved target position. However, since it is represented at several positions within the resolving matrix it might also contribute to erroneous target detections. Therefore it is crucial that once a target has been detected all duplicates of the echoes causing it must be removed from the resolving matrix. If we look at the example from section 2.2, reprinted in Figure 4.8, we see that the two echoes gives four possible target positions indicated by at least two of the three CPIs.

FIGURE 4.7.

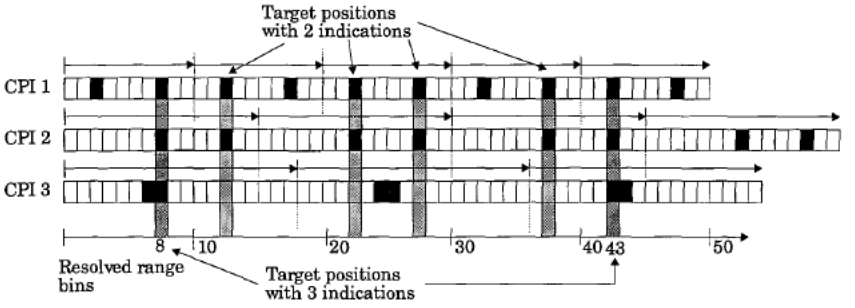


The resolving matrix. The individual CPI matrix sizes and some of their expansions around their respective ambiguity limits are also shown.

Thus, the algorithm needs to detect all coincidences with nine occurrences and then remove those target echoes from the matrix. After that all coincidences of eight are found and removed and so forth until all coincidences of three or more have been found. Additional constraints, for instance a similarity in echo strength, might be applied for a target to be determined as correct. This can be handled by the system control processor which has access to the envelop information extracted by FVIPa.

Note that due to the different Δv in the different CPIs an interpolation step is required in the frequency resolving. The interpolation is implemented using the nearest neighbor value.

FIGURE 4.8.



An example showing that the targets echoes with three CPI coincidences at range 8 and 43 also gives two CPI coincidences at four other positions.

Resolving complexity

The complexity of the resolving is difficult to approximate since many different implementation approaches are possible. The matrix contains 3000 range bins and 1000 velocity bins, and each bin contains 9 binary values. A target detection in each bin is performed by counting the number of ones among the 9 binary values. Seven consecutive tests are performed, and the brute force complexity is then 3000x1000x9x7 = $190*10^6$ (binary) operations. This must be performed in 5 ms (before the arrival of the next CPI) which requires a computing power of 38 Gops.

A more effective implementation is to first test which of the bins that contain at least three hits, and then only poll these in the subsequent processing. After the first test only a fraction of the resolving matrix bins need to be polled in the subsequent processing. This can be understood by noting that since each CPI matrix has a maximum of 100 targets and around 10000 bins only around 1% of the bins in each bit-plane of the resolving matrix contain a target. Therefore no more than 1%, and probably much less, of the resolving bins can contain more than three target indications. The initial test requires 3000x1000x9 binary operations and the subsequent six tests less than 3000x1000x9x6/10 and the total requirements are around 8.5 Gops. This scheme does not fit well in an SIMD implementation however, and is not used, but we will use it in out comparisons with a DSP implementation of the resolving.

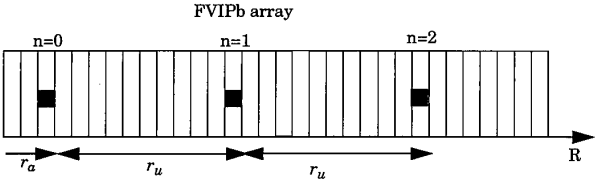
Resolving mapping on FVIPb

The resolving is implemented within a 1000x3000 data matrix. In the complexity analysis we approximated the complexity of an SIMD suitable implementation to be 38 Giga binary operations. If a binary operation requires 3 clock cycles each FVIPb PE has a peak performance of 16 Mega operations per second. This means that at least 2300 FVIPb PEs are required for the resolving. Therefore it is only feasible to align the PEs along the 3000-bin range axis, giving at least 3000 PEs. With this PE layout each PE is assigned to process one resolved range bin (column) of the resolving matrix. For each target at the range r_a extracted from CPI i the possible resolved ranges are given by equation (2.6). Thus it shall be input to the PEs

$$PE = r_a + r_{ui} \cdot n \tag{4.4}$$

The variable n is in the range from 0 to the maximum number of data multiples used, see Figure 4.9.

FIGURE 4.9.

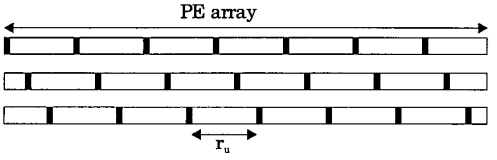


The PE positions where data with the extracted range r_a should be input.

To input the data to the correct PEs we use the shiftregister and PEs in the following manner. First we store all possible r_u:s in the internal RAM of the array. Or rather, vectors with ones at the r_u interval are stored, see Figure 4.10. We then shift the appropriate r_u vector r_a steps and read the result to the PE RAM. Since r_a can be up to 300 we initially also store shifted versions of r_u so that by selecting the appropriate pre-shifted vector a shift longer than three PEs never is needed. This gives a total of 3*300/5+3*100/5+3*30/5 = 258 r_u bit-vectors stored in the RAM of each PE. These bit-vectors are then used analogously to indicate the PE positions where CPI target data should be removed from the resolving matrix.

Instead of storing the complete resolving matrix columns in the PEs we store the columns of the nine CPI matrices. For each velocity to resolve the corresponding target positions within the CPI matrices are given by equation (2.20). Thus, for each resolved frequency we find the appropriate CPI matrix entries and we can perform the velocity resolving without explicit storage of the complete columns of the resolving matrix. The position extraction once a target detection vector has been obtained requires the same PE modifications as in FVIPa. These modifications are described below.

FIGURE 4.10.

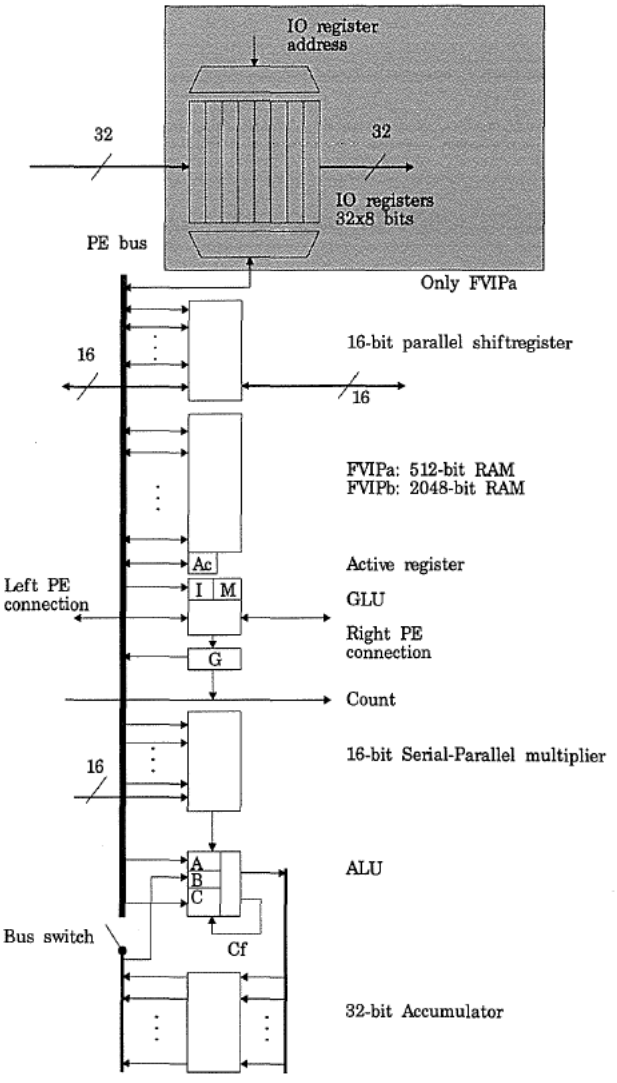


An r_u vector, and its phased shifted version stored in internal RAM.

4.2 The FVIPa/b PE

The FVIPa and FVIPb PE layout can be seen in Figure 4.11. Since the FVIPa and FVIPb designs are very similar the design overhead to implement both chips is very low compared to designing only one chip. The FVIPa/b PE design is similar to the RVIP PE, and below we discuss the modifications made and how they are used to implement the problems encountered in the algorithm descriptions. We still assume that the PE array is clocked at 50 MHz.

FIGURE 4.11.



The layout of one Processing Element of FVIPa/b. The difference is that the IO-register only is implemented in FVIPa and that the size of the RAM differs.

The IO register

If we input two samples to each FVIPa PE we need at least 2x4x16x2= 256 bits in the IO register. Thus, the FVIPa PE has no less than eight 32-bit register banks in the IO register. We note that the input channel capacity is used fully which means that any additional input must be performed by other means. The IO-register is not used in FVIPb and can therefore be omitted. Instead we can use the shiftregister for input of the required mask vectors.

The Shiftregister

The shiftregister is identical to the RVIP, and just as in the RVIP design we assume that the shiftregister is clocked at half the PE clock, i.e. 25 MHz. However, since the RVIP system implemented by EMW has reduced the shiftregister clock frequency from our estimate of 25 MHz to 12.5 MHz we have also studied the effect of using only 12.5 MHz shiftregister frequency.

The RAM

The data in the CPI processing in FVIPa expands to 28 bit precision, and thus up to 2x4x2x28 = 448 bits are required to store the data. All parameters are input when needed and require only temporary storage. Thus, 512 bits of RAM is sufficient in FVIPa. In the resolving the input data is binary data from nine CPIs. This is 3*(256+128+32) = 1248 bits. As was shown earlier an additional 258 bits was required for r_u vectors and we implement 2048 bits of RAM in FVIPb.

The Multiplier

Compared to RVIP the multiplier has been extended to 16 bits, allowing for higher coefficient precision. We have also found that we without hardware modifications can use the multiplier for counting purposes. The inner loop of the resolving procedure consists of counting the number of ones present in a word in the RAM. This loop is performed 7000 times for each resolving, and thus the counting operation must be efficiently implemented. Normally, the operation to count the number of ones in a b-bit word using the FVIP ALU has the complexity

$$C(b) = \sum_{i=1}^{b} \lceil ({}^2\log(i+1)) \rceil \tag{4.5}$$

clock cycles. For instance, counting the bits in a 16-bit word requires 50 clock cycles. This operation can be performed more effective using the existing serial-parallel multiplier hardware in the following way.

In every multiplication step the serial-parallel multiplier performs a parallel 16-bit carry save addition of the contents of the parallel coefficient register and the internal parallel register. The sum is also shifted down one step before the next addition. Thus, if we load the parallel coefficient register with 2^{b-1} and multiply with one bit from the vector, we will get the result 2^{b-1} if the bit was one and zero otherwise. Now, if we re-load the parallel register with 2^{b-2} and multiply with the next bit in the vector, without resetting the internal state of the multiplier, the multiplier will contain the sum of the two bits multiplied with 2^{b-2} since the previous partial result is down-shifted one step. These steps are then repeated for the remaining bits and after that the parallel register of the multiplier contains the number of ones. Thus, the addition of ones in a b-bit word can be performed in approximately b cycles if the external load of the

coefficient is made in parallel with the load/mult of the currently counted bit. For b=16 we need 16 cycles instead of 50 as above. Of course, if we want to store the result in the internal ram this requires at least $^2\log(b)+1$ cycles.

The ALU

Since the FVIPb array is used for logic rather than arithmetic processing we have augmented the ALU functionality with the logic operations AND and OR. This can be done without increasing the size of the ALU instruction field given for RVIP in section 3.3.

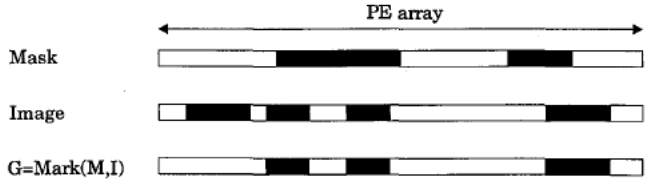
The GLU

The Global Logic Unit, GLU, is a new hardware feature used for global data dependent operations and data extraction. This type of operations are needed to implement the ADC saturation detection efficiently and to extract the target positions. The functional description of the FVIPa/b GLU is derived from the LAPP chip [21] and the VIP implementation is described in more detail in [15], [35]. The GLU has two binary input registers in each PE called *Image, I,* and *Mask, M,* and one binary output register *G.* Connected to the GLU we also receive the status information COUNT and Global Or (GOR) to the micro controller. COUNT equals the number of *ones* in the *G* registers of all PEs in the array and GOR is one if any PE have a bit set to *one* in the *G* register.

Two GLU operations are implemented in hardware, MARK and LFILL. The MARK operation is illustrated in Figure 4.12. Those objects (runs of ones) in the image which overlap the mask in at least one position are kept. The LFILL operation is illustrated in Figure 4.13. All *G* registers to the right of the leftmost PE containing a *one* are set to one. We note that after an LFILL operation the number of ones in the *G* register equals the distance from the leftmost *one* to the right edge of the array. Thus, by reading the COUNT value we find the PE position of the leftmost *one* in the array. In the LAPP GLU the LFILL also set a *one* in the leftmost PE containing a *one*. Our re-design makes it possible to implement an operation called LEDGE, giving a single *one* in the PE with the leftmost *one,* by a bit-wise AND between the LFILL input vector and the inverse of the LFILL output vector.

The MARK operation is used in the ADC saturation detection. As we pointed out, the Global Or of the thresholding results within each range bin is to be distributed to all PEs containing data from that particular range bin. Assume that we have a bit-vector B with ones at the PEs containing input data, see Figure 4.14. A MARK operation, with the result R of the thresholding operation input to the mask register and B input to the image register gives the desired Global Or distribution.

FIGURE 4.12.

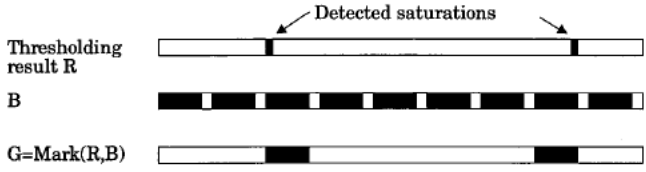


The GLU operation MARK.

FIGURE 4.13.

The GLU operation LFILL and the LEDGE operation.

FIGURE 4.14.



The use of the MARK operation to distribute a Global Or to selected PEs.

The LFILL operation is used for target position extraction in FVIPa and FVIPb. Assume that we have a bit-vector R with ones in the PEs where a target was detected, see Figure 4.15. After an LFILL operation the COUNT value, which indicates the PE position, and therefore the range/frequency, of the leftmost target, is read by the micro controller. To extract the positions of all other targets in R we remove the leftmost bit from R by an XOR operation between R and the LEDGE result and perform a new LFILL operation. Furthermore, if we wish to extract envelop information for each target this can also be accommodated. For each bit in the envelop of each target we perform an AND operation with the L vector and input this to the image register and read the Global Or output. If the Global Or is one the selected bit was *one* otherwise it was *zero*.

FIGURE 4.15.

The use of LFILL and LEDGE operations to extract target positions.

The speed of the GLU operations will be dependent on the maximum length of the signal propagation across the array. In the MARK operations we know that the maximum object length is $P_{max}/2$ which is 128, and then we assume that MARK will be ready within 160 ns, which is 8 FVIP clock cycles [15]. The GOR function can be made very fast if implemented as a tree structure on each chip. The Global-Or output from one chip can then also be used as a look-ahead signal for the LFILL if it is connected to the left edge input of the LFILL ripple chain on the next chip. With this construction we expect the LFILL/GOR ripple to be ready within 80 ns, which is 4 FVIP clock cycles, on a 6600 PE array. The COUNT net work in par-

allel with the LFILL net and will be ready within another 80 ns, giving a total of 160 ns. The COUNT net only ripples through one MCM, and is then added to the result of the other MCMs externally. The GLU implementation is discussed in more detail in [15] and [35].

The Active register

In SIMD architectures it is common to allow PEs to be turned of conditionally via a PE active or mask register [7], [24], [29], [62]. This is a deviation from strict SIMD processing since all PEs do not perform the same instruction. Such an active register is very useful for parameter distribution in FVIPa. We illustrate the parameter input using the active register with an example.

Assume that we have an *N*-PE array and wish to distribute one unique 16-bit constant to each of them. Assume also that we in the PE RAM have an *N*-bit-vector which contains a single one at the leftmost PE, compare with the r_u vectors in Figure 4.10. If we load a multiplier coefficient using the parallel load port it is normally stored in all PE multipliers. If we instead load the bit-vector to the active registers and perform the multiplier coefficient load conditionally only the leftmost PE will load the coefficient. Now, if we input the bit-vector to the shiftregister and perform *N-1* right shifts, active register stores and coefficient loads each PE will have a unique constant stored in the parallel register of the multiplier. Thus, if a one PE shift requires 2 clock cycles, distributing one unique parameter to each of the *N* PEs requires *3N* clock cycles. Without the active register the distribution can be implemented in a similar fashion using mux operations and the number of cycles is then approximately $(b+1)N = 17N$. In FVIPa we have two real-valued window parameters and *logP* twiddle factors that should be distributed. Since all parameters are constant along the range axis each parameter has no more than *P* unique values. The twiddle factors actually has less than P/2 unique values and when *P=256* all parameters can be distributed using around 8000 clock cycles. Without the active register the corresponding figure is 43000, and thus the gain with the active register is around 35000 clock cycles.

The active register also makes the CFAR communication more effective. In section 4.1.1 we noted the data organization after the FFT. To allow each PE to process data within a +/-3 pulse window we use the shiftregister and active registers in the following way. We load the vector with the high frequency samples to the shiftregister and shift *P/2* steps to the right. We then conditionally load the shiftregister with the low frequency samples using a mask-vector indicating the active PEs. Now the shiftregister contains the *P/2+3* lowest frequencies and the CFAR processing of the *P/2* lowest frequency bins is straight-forward. The samples required for CFAR processing of the *P/2* highest frequencies is obtained analogous.

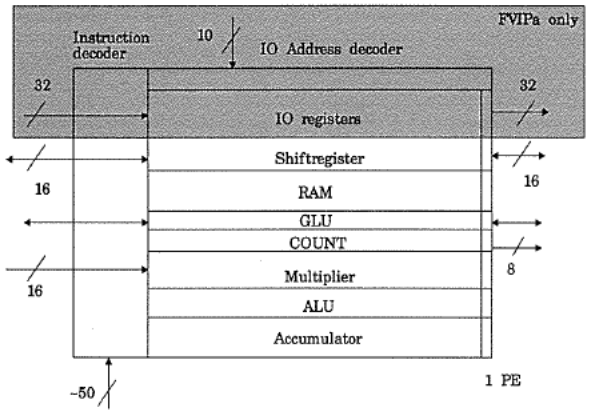
4.3 FVIPa/b chip and system design

The expected chip area for FVIPa and FVIPb chips have been estimated using available data from the RVIP estimates in [42]. The area estimates for 128 PE chips are given in Table 4.2. The total estimated chip areas including instruction decoders and I/O pads are 100 mm^2 for an FVIPa chip and 113 mm^2 for an FVIPb chip. The expected transistor counts for the FVIPa and FVIPb chips are 540000 and 1.1 million respectively, and if a non-SRAM process is used the corresponding estimates are 730000 and 1.6 million. The layout of an 128 PE FVIPa/b chip can be seen in Figure 4.16.

TABLE 4.2.

PE part	FVIPa		FVIPb	
	Area mm^2	Area%	Area mm^2	Area%
IO	16	23	0	0
SR	6	8	6	7
GLU	1.5	2	1.5	2
Mult	25	34	25	30
ALU	4	5	4	5
ACC	7.5	10	7.5	10
RAM	12.5	18	38	46
Total	**72.5**	**100**	**82**	**100**

FIGURE 4.16.



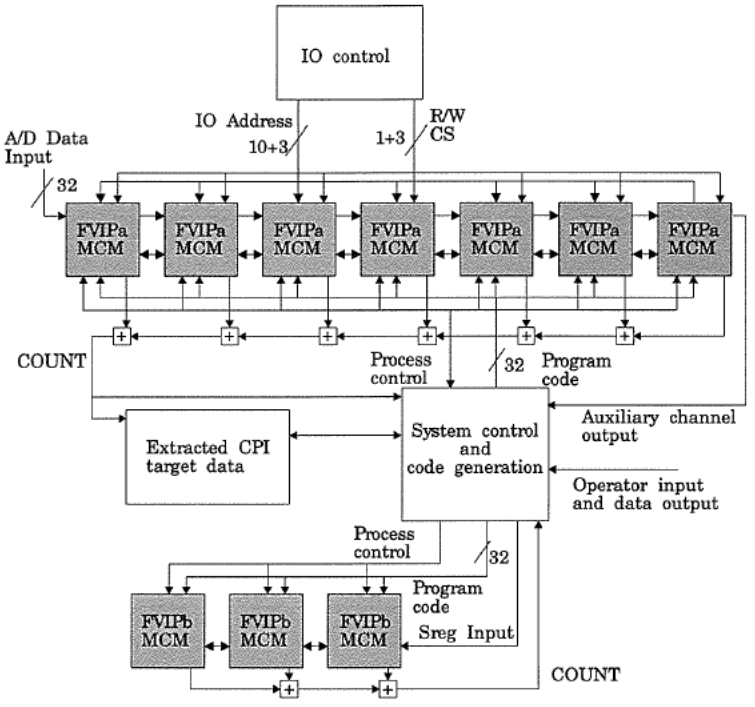
The layout of the FVIP chip.

System design

Each FVIPa/b chip will consist of 128 PEs and eight such chip and one local controller will be mounted inside one Multi Chip Module, MCM, compare with the RVIP MCM layout in section 3.5. Thus, each FVIPa/b module will have 1024 PEs and it will run at a clock frequency of 50 MHz. The IO and shift register frequency however, is limited to 25 MHz. The peak performance for one MCM is then approximately 1.6 Giga MAC operations on 16-bit words. In the CPI processing 7168 PEs are used which gives a total of 7 MCM modules.

In the resolving approximately 3000 PEs are used which gives a total of 3 MCM modules. The FVIPb system will have approximately 50 Gops performance for binary operations. Thus, the total system will require ten MCM modules and a system overview is given in Figure 4.17.

FIGURE 4.17.



An overview of the FVIP system.

The feedback from the FVIPa/b MCMs is more complicated than in RVIP where no feedback occurred. Thus, a more complicated system controller is needed. The external controller will probably be based on some standard signal processor architecture, for instance a TI C40. The design of a more powerful controller and different approaches to code the micro word efficiently are discussed in [35].

We expect the total FVIP system to be around 3 dm^3 in size and need no more than 200 W. The limits for an airborne system is 25 dm^3 and 500 W and we are well within those limits.

4.4 FVIP performance

The CPI time is between 5 and 8.3 ms which at a clock frequency of 50 MHz leaves at least 250000 clock cycles for the processing of a CPI. The two-step pipeline makes the delay between the acquisition of a CPI and the presentation of a result less than two times the CPI time, i.e. less than 17 ms.

4.4.1 CPI processing in FVIPa

The FVIPa computational results for the three different CPI cases are collected in Table 4.3. Instead of presenting the cycle counts for all parts of the algorithm we have separated the cycle count into processing, parameter input using the multiplier port, target position extraction

using the GLU and communication requirements. We also note how many of the communication cycles that can be hidden by the processing. We see that the FVIPa array requires around 50% of the available 250 000 clock cycles. If the shiftregister clock frequency is reduced to 12.5 MHz the number of required communication clock cycles doubles, increasing the number of clock cycles to around 200000, and thus it should be possible to implement the CPI processing in FVIPa even if the shiftregister clock frequency is reduced to 12.5 MHz.

TABLE 4.3.

Algorithm part	**CPI format**		
Number of pulses, P	**32**	**128**	**256**
Number of range bins, R	**300**	**100**	**30**
Processing	109496	72632	60824
Parameter input	912	3392	8848
Data extraction	13300	13300	13300
Communication	31564	53132	62780
Parallel comm/proc	-22968	-19504	-4048
Clock cycle sum	**132304**	**122952**	**141704**

4.4.2 Resolving in FVIPb

The results for the resolving algorithm implementation in FVIPb are given in Table 4.4. We are closer to the 250 000 cycle limit than with FVIPa, but still around 30% below the limit. This corresponds reasonably well with the predicted PE performance and algorithm complexity.

TABLE 4.4.

Algorithm part	**# Clock cycles**
CPI data input	14000
Resolving	156000
Sum	**170000**

4.4.3 Architecture comparisons

FVIPa

Earlier we approximated the CPI processing, solved by 7168 FVIPa PEs in 56 chips, to have a complexity of 2 Gops. Using the same ideal DSP assumption as in section 3.7, a DSP implementation would require 40 ideal 50 MHz DSPs. That is, the number of FVIPa and ideal DSP processor chips are roughly the same. We believe however, that the overhead to implement the MPD algorithm on a DSP array or pipeline is very large, and that performance of each DSP would be far from ideal.

If we look at the FVIPa PE efficiency we use more than three times as many PEs as is theoretically needed. This is because a large overhead is required for the communication, but there is also a large overhead due to our implementation of the parameter input and the data extraction. These cycles could be reduced if the IO and memory capacity in the PEs are increased, and if extra external hardware is used for the extraction.

If we instead of a linear SIMD array try to implement the algorithm using a 2D FVIPa array we can approximate the number of required clock cycles by using the linear array results but removing the communication overhead. The number of cycles required in one CPI is approximately 124000. If we also remove the parameter input and data extraction overhead the cycle sum is around 110000. We see that the gain with a 2D array is very small and designing a 2D layout and corresponding VIP architecture suitable for this application is not trivial since the CPI formats vary very much. Furthermore, the design of a 2D FVIPa chip with full 2D communication capacity is not feasible with the current chip packaging technology.

FVIPb

The complexity of the resolving was approximated as 8.5 Gops in section 4.4.2. Using our usual DSP approximation 170 ideal 50 MHz DSPs are required to implement this. This is much more than the 24 chips which the FVIPb implementation requires. That the FVIPb implementation is very successful is because a large part of the processing is bit-manipulations which are effective in a bit-serial architecture, whereas a DSP is effective for word-parallel operations.

Chapter 5

Fourier transforms in FVIPa

5.1 Introduction

Fast Fourier Transforms, FFTs, are very common in signal processing applications. Since FFT computations are very demanding many ways of speeding them up has been studied, see for instance [31], [37], and [40]. Different speed-up techniques include application specific processors [13], [57], systolic arrays [64], MIMD arrays [1], [60] and SIMD arrays [1], [50], [56], [62].

Linear SIMD arrays without any shuffle interconnection network, such as the FVIPa, are not the obvious or ideal choice for FFT implementations. The butterfly scheme is inherently global in nature with a variety of communication needs across the array.

5.2 FVIPa design prerequisites

In this chapter we will study FFT implementations using an FVIPa PE array. Apart from using the exact FVIPa design we will also investigate how a wider shiftregister and increased multiplier precision affects the FFT performance. The used PE parametrizations can be found in Table 5.1.

When we study 2D FFTs we give one solution where we assume that the internal PE RAM is 16 Kbits. With the VLSI technology available today we can only implement 1-2 Kbit memory in each PE, but within 2-3 years we believe that up to 16 Kbit RAM/PE is feasible.

We use the notation for complex numbers introduced in section 3.3, i.e. the precision of the data is p and therefore the size of a complex word is b where $b=2p$.

TABLE 5.1.

Parameter	Values
Shiftregister width, W	16-32
Shiftregister cycle speed, R	2
Multiplier size, M	16-32
Accumulator size, $2M$	32-64
Precision, p	16-32
Data size, $b = 2p$	32-64

5.3 Fourier transforms

The discrete Fourier transform, DFT, of a 1D discrete signal sampled from *0* to *N-1* is defined as [9]

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\, e^{\frac{-i2\pi ux}{N}} = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\, \omega^{ux} \tag{5.1}$$

where

$$\omega = e^{\frac{i2\pi}{N}} \tag{5.2}$$

The phase factor ω^{ux} is a complex number called twiddle factor. Since the computation of each of the N fourier components requires N complex multiplications the complexity of the algorithm is $O(N^2)$. An implementation in an N-PE FVIPa array is straight-forward and requires around $16pN$ clock cycles.

However, the DFT can be computed with $O(N\log N)$ complexity with a technique called Fast Fourier Transform, FFT [9], [58], and an implementation on an $O(N)$ PE array should ideally be able to compute an FFT with $O(\log N)$ operations. However, since the communication of the N data points includes $O(N)$ distances there is a risk that the ideal speed up is heavily offset by the communication across the array.

The FFT algorithm exploits the fact that an N-point DFT can be computed with $O(N)$ operations from two $N/2$-point DFTs on the original data. This subdivision can then be carried out recursively $\log N$ steps until only $N/2$ DFTs of the length 2 remains (if N is an even power of 2). This FFT technique is called decimation-in-time radix-2 and the butterfly scheme for an 8-point FFT can be seen in Figure 5.1. A 2-point DFT is computed by a so called butterfly operation

$$C = A + B \cdot \omega^i \tag{5.3}$$

$$D = A - B \cdot \omega^i \tag{5.4}$$

where ω^i is the twiddle factor. Thus, each butterfly is composed of one complex multiplication and two complex additions and the whole FFT algorithm can be computed with $(N \cdot \log N)/2$ complex multiplications and $N\log N$ complex additions/subtractions. Other radices than two can also be used, but for SIMD implementations the number of cycles for the computations are minimized with radix-2 transforms [1]. Decimation-in-frequency implementations will have the same complexity as the decimation-in-time algorithms studied here.

If the input data is real-valued, the resulting transform is an even function, and if input data is imaginary the resulting transform data is an odd function [9]. This can be exploited by computing the FFT of two real-valued N-point sequences in one transform, setting one as the real part and one as the imaginary part of the input. Afterwards the DFTs of the different input sequences is obtained by separating the output into one odd and one even sequence by the operations

$$A(u) = \frac{1}{2}(F(u) + F^*(-u)) = \frac{1}{2}(F(u) + F^*(N-u)) \tag{5.5}$$

$$B(u) = j\frac{1}{2}(-F(u) + F^*(-u)) = j\frac{1}{2}(-F(u) + F^*(N-u)) \tag{5.6}$$

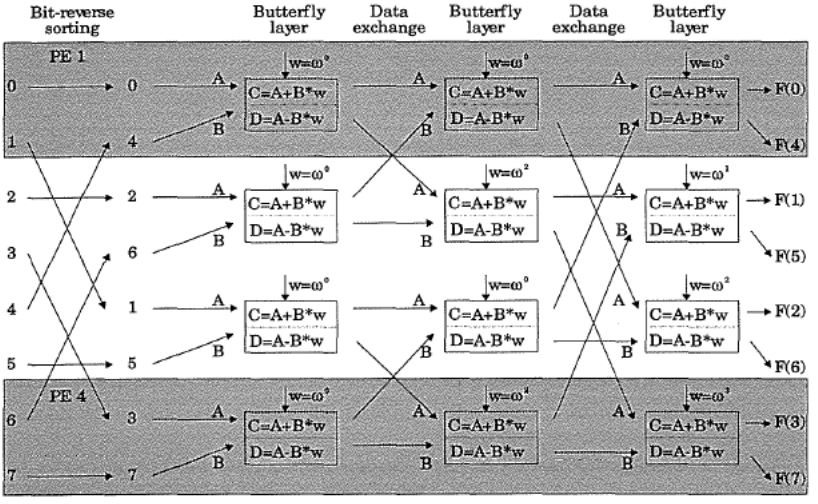
A 2D Fourier transform is usually computed by taking one 1D transform in each direction, so called *row-column* FFT. If we assume that the input (image) data is real-valued we know that the 2D FFT result is Hermitian (has conjugate symmetry) [9], i.e.

$$F(u, v) = F^*(-u, -v) = F^*(N-u, N-v) \tag{5.7}$$

This shows that all the information in the NxN 2D Fourier transform of a real-valued image can be extracted from $N/2$ columns in the Fourier domain image. Thus only $N/2$ column transforms need to be computed.

We have studied an 1D FVIPa FFT implementation using *N/2* PEs. Using *N/2* PEs is more effective than *N* since each butterfly layer contains *N/2* butterflies and because the communication distances decrease [44]. We have also studied two alternative approaches to 2D FFT implementations. One where the column FFT is performed within each PE, and one where it is performed by a second array after a corner turning operation.

FIGURE 5.1.



A decimation in time radix-two FFT algorithm implementation of an 8-point sequence on a four-PE array.

5.4 FFT employing N/2 PEs

In Figure 5.1 we showed the butterfly scheme for an 8-point FFT. We note the following. Before the first butterfly layer data is sorted in bit-reverse order. In each of the *logN* butterfly layers *N/2* butterflies are computed. Between each butterfly layer there is a pair-wise exchange of data between butterfly nodes at a distance of 1, 2, 4, up to *N*/4.

The FVIPa FFT implementation can be broken down into four separate parts. Bit-reverse sorting of the input data, shifting data to the correct PEs before each butterfly layer, distributing twiddle factors ω^j and computing the butterflies.

Bit-reverse sorting data

If the input data to the transform are brought in from the outside of the chip these can be written into the array bit-reversed since the IO registers can be addressed arbitrary. If the FFT-input is an intermediate result the sorting operation must be done within the FVIPa array. If the sorting is performed within the array each PE initially contains one even-indexed data and one odd-indexed data, and after the sorting each PE contains one *A* and one *B* input data, see Figure 5.1. We denote the distance between the reversed and the in-order index *d*. If *d* is positive the corresponding input sample should be fetched *d/2* PEs to the left of the PE and if it is negative the sample should be fetched *d/2* PEs to the right. Furthermore, if *d* is even the input

sample to the *A* sequence is to be fetched from the even samples and if it is odd from the odd numbered samples. Vice versa holds for the samples in the *B* sequence. We have studied the required shift lengths empirically and found that for the *A* sequence no positive odd distances appear and for the *B* sequence no negative odd distances appear. Thus, the *A* sequence can be generated as in the pseudo code

```
        /* Generate the A-sequence */
Move(Even,Sreg,A)       /* Move even data to sreg and A, b */
for (i=1;i<D_e1;i++) { /* Pos even shift lengths D_e1 */
   Shift(left,len_e1(i)) /* Shift left*/
   active = maske1(i)   /* Load mask */
   Cond_load(Sreg,A)    /* Load shifted data, b */
}
Move(Even,Sreg,A)       /* Move even data to sreg and A, b */
for (i=1;i<D_e2;i++) { /* Neg even shift lengths D_e1 */
   Shift(right,len_e2(i)) /* Shift right */
   active = maske2(i)   /* Load mask */
   Cond_load(Sreg,A)    /* Load shifted data, b*/
}
Move(Odd,Sreg,A)        /* Move odd data to sreg, b */
for (i=1;i<D_o1;i++) { /* Neg odd shift lengths Do2 */
   Shift(right,len_o1(i)) /* Shift right */
   active = masko1(i)   /* Load mask */
   Cond_load(Sreg,A)    /* Load shifted data, b */
}
```

The *B* sequence can be generated analogously, using only positive shift lengths for the odd shift lengths. The total number of non-zero shift lengths, *D*, required for each of these operations is less than 10% of *N* for *N* larger than 512. For instance, if $N = 16$ *D* is five and the *A* sequence shift lengths are -9,-7,-5,-2,2 and the *B* sequence shift lengths -2,2,5,7,9. The maximum shift length is less than *N/2* in each direction. With the data length *b* the number of clock cycles needed for the bit-reverse operation can be approximated with

$$C_{BR} = 3b + D \cdot b + 3R \cdot \frac{N}{2}\left\lceil \frac{b}{W} \right\rceil \approx N\left(\frac{b}{10} + \frac{3R}{2}\left\lceil \frac{b}{W} \right\rceil \right) \qquad (5.8)$$

One mask vector for conditional load is required for each shift length and sequence that is generated, giving a total of *2D* mask planes. We note that the bit-reverse sorting has *O(N)* complexity on FVIPa.

Shifting data to the correct PEs

As seen in Figure 5.1 cube-interconnected PEs, i.e. PE pairs with only one bit different in the PE index, exchange data between the *logN* butterfly layers. This is done in an operation similar to the bit-reverse sorting, but only one positive and one negative shift length appear in each layer. In pseudo code one shift operation is

```
/* Shift operation before butterfly layer i */
/* A is either C from PE i or D from PE i+L(i) */
/* B is either D from PE i or C from PE i-L(i) */
Move(D,Sreg)            /* Move D to Sreg */
Move(C,A)               /* Move C to A, b */
Shift(right,L(i))       /* Shift right */
```

```
active= mask(i)          /* load active register */
Cond_load(Sreg,A)        /* Load A conditionally, b */
Move(C,Sreg)             /* move C to Sreg */
Move(D,B)                /* Move D to B, b */
Shift(left,L(i))         /* Shift left */
active =!mask(i)         /* load active reg w/ inverse */
Cond_load(Sreg,B)        /* Load B conditionally, b */
```

The shift lengths between the different butterfly layers are 1,2, 4, up to *N/4* long, which adds up to less than *N/2* shifts in each direction. The number of cycles needed to complete the data shifting for an *N*-point FFT is approximately

$$C_S = 6b(\log N - 1) + R \cdot N\left\lceil \frac{b}{W} \right\rceil \tag{5.9}$$

One mask pattern is needed for each butterfly layer, which means that *logN* patterns are required for an *N* point FFT. The shift operation has *O(N)* complexity.

Distributing twiddle factors

The twiddle factors can be distributed to the PEs in at least two ways, via the IO register or via the parallel load port on the multiplier. The reason that the multiplier is feasible to use comes from the fact that many PEs share the same coefficient. In the first layer all twiddle factors are equal (one), in the second only two different twiddle factors occur, in the third there are four and so on up to *N/2*. The use of the multiplier port is even more attractive when, as in chapter 4, a large array is used to compute several smaller FFTs in parallel because then the twiddle factors are common for all the parallel Fourier transforms in the array. Another attractive feature with this technique is that twiddle factors are input when needed and do not occupy *b*log*N* bits of RAM which they will if they are brought in via the IO register.

Using the parallel register of the multiplier for IO we need, as discussed in section 4.2, approximately *(R+1)N* cycles to distribute one unique word to each PE and then *b* cycles to move data to RAM. The number of different twiddle factors in each butterfly layer is 1,2,4 up to *N/2*, and thus the total number of cycles is less than

$$C_{TW} = (R+1)N\left\lceil \frac{b}{M} \right\rceil + b\log N \tag{5.10}$$

A mask pattern is needed for each butterfly layer giving log*N* masks to determine which PE is receiving the current twiddle factor. The twiddle factor distribution using the multiplier has *O(N)* complexity.

Butterfly computation

We illustrate the butterfly operation on FVIPa with a pseudo code example showing the computation of the real part of the result

```
        /* Butterfly computation pseudo code */
        /* Re(C) = Re(A)+Re(B*w); Re(D) = Re(A)-Re(B*w) */
        /* Re(B*w) = Re(B)*Re(w)-im(B)*Im(w) */
Mult_coeff = Re(w)       /* Coefficient load p/1 */
Acc = Mult*Re(B)         /* Multiply 2p */
Mult_coeff = Im(w)       /* Coefficient load p/1 */
Acc += Mult*Im(B)        /* MAC 2p */
```

```
Tmp = Acc              /* Store Re(B*w) p */
Acc += Re(A)           /* Add Re(A) p */
Re(C) = Acc            /* Store result p */
Acc = Re(A)            /* Load Re(A) p */
Acc -= Tmp             /* Subtract temp p */
Re(D) = Acc            /* Store result p */
```

If we sum the number of clock cycles we find that with serial coefficient load 12p and with parallel coefficient load 10p clock cycles are required to compute the real part of the result. Each twiddle factor is used twice, and since the number of cycles required to store/retrieve a parameter from RAM is less than the number of cycles required to load the multipliers with different coefficients externally we assume that serial coefficient load is used. Thus, here a butterfly operation requires approximately 2*12p cycles. The twiddle factors have at the most M bit precision, where M is the size of the parallel part of the multiplier. The complexity for all butterfly operations is then approximately

$$C_{BF} = 24p\log N = 12b\log N \tag{5.11}$$

Total FFT complexity

With input data in correct bit-reversed order and twiddle factors in internal RAM the number of clock cycles required for an FFT operation is approximately

$$C_{FFT1} = 22b\log N + R \cdot N\left\lceil \frac{b}{W} \right\rceil \approx [p,\ W=\ 16,\ R=\ 2] \approx 704\log N + 4N \tag{5.12}$$

With twiddle factor distribution and bit-reverse sorting the number of clock cycles required for an FFT operation increases to approximately

$$C_{FFT2} = 23b\log N + N\left(\frac{b}{10} + \frac{3R}{2}\left\lceil \frac{b}{W} \right\rceil\right) + R \cdot N\left\lceil \frac{b}{W} \right\rceil + (R+1)N\left\lceil \frac{b}{M} \right\rceil \approx \tag{5.13}$$

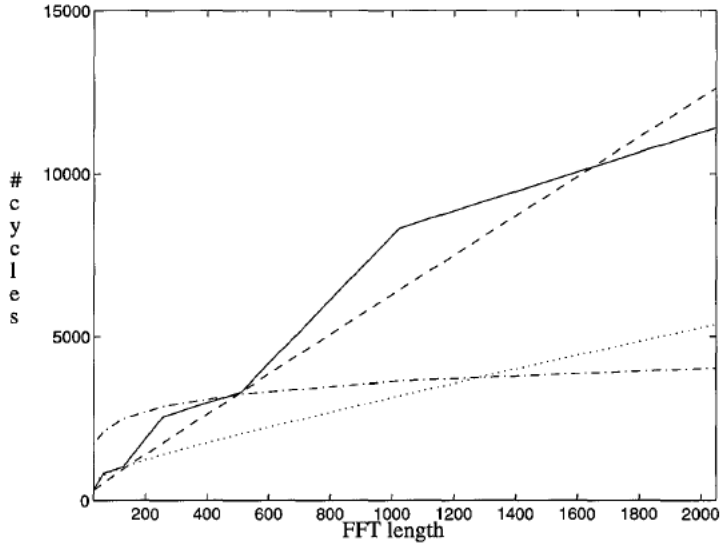
$$\approx [p,\ W, M=\ 16,\ R=\ 2] \approx 736\log N + 20N \tag{5.14}$$

In Table 5.2 we see the number of clock cycles for FFTs of different lengths. The numbers are for 16-bit precision. We note that for long transforms the gain of implementing a wider shiftregister/multiplier is clearly visible. In Figure 5.2 we graphically show the complexity of the different parts of the FFT implementation. We see that for transforms longer than 512 the linear terms for the twiddle factor distribution and the bit-reverse sorting are larger than the logarithmic twiddle factor computation term.

TABLE 5.2.

FVIPa FFT	128	512	2048	8192
FFT1, W = 16	4160	6848	14144	39872
FFT1, W = 32	3904	5824	10048	23488
FFT2, W = 16	6528	15640	44288	154176
FFT 2, W = 32	5888	12480	34048	113216
FFT2, W/M =32	5504	10944	27904	88640

FIGURE 5.2.



The number of clock cycles required for the different algorithm parts. The plot shows bit-reverse sorting (solid), shifting (dotted), twiddle factor distribution (dashed) and butterfly computations (dashed-dotted).

A comparison of 1D FFT execution times in milliseconds on a 8192 PE FVIPa array and a 2D 8192 PE MasPar-1 array [56] can be found in Table 5.3. The data precision is 32 bits and the total number of data points used is always 16384. Thus, either 256 64-point transforms are computed or 32 512-point transforms and so on. We have used an FVIPa model with 32-bit shiftregister and multiplier. We see that the FVIPa array performs 2-15 times better if the bit-reverse sorting and twiddle factor distribution is implemented, but with the twiddle factor distribution and bit-reverse sorting is implemented the gain is smaller. The difference of the two architectures makes it hard to compare the implementations, the FVIPa clock frequency is four times the MasPar's, but on the other hand the MasPar-1 has 4-bit processing elements.

TABLE 5.3.

FFT size	64	512	8192	16384
MasPar-1	2.5	4	5.4	6.1
$\text{FVIPa}_{\text{FFT1}}$	0.12	0.22	0.92	1.6
$\text{FVIPa}_{\text{FFT2}}$	0.16	0.39	3	6.4

Note that in the FFT, when the computational complexity increases because of the long shift operations, the twiddle factor generation complexity also increases. However, the shift operations can be performed concurrently with the twiddle factor distribution, and the total complexity might therefore actually be smaller than the sums indicated above.

5.5 2D FFTs in FVIPa

The most common way to compute a 2D FFT is the row-column method, i.e. first all rows are transformed using 1D FFTs and then the columns of that image are transformed using 1D FFTs. We have two strategies to compute an *NxN* row-column 2D FFT with FVIPa, with or without external corner turning.

With external corner turning

The most straight-forward way of implementing 2D FFTs in FVIPa is to pipeline two arrays with a corner turning in-between. Thus, the first array computes the row transforms and the second the column transforms. It is then easy to exploit real-valued input data and conjugate symmetry, see Figure 5.3. This halves the work loads for the two arrays. We use one *N*-PE array for the odd-even separation. The odd-even separation is two complex add/sub operations, requiring 3b cycles each, and we approximate that with 16-bit precision the separation takes 200 clock cycles.

Now, if we use one *N*-PE array it can be used for all the processing. First the array computes the row transforms four rows at the time, then the odd-even separations and last the column transforms two at the time as in the pseudo code

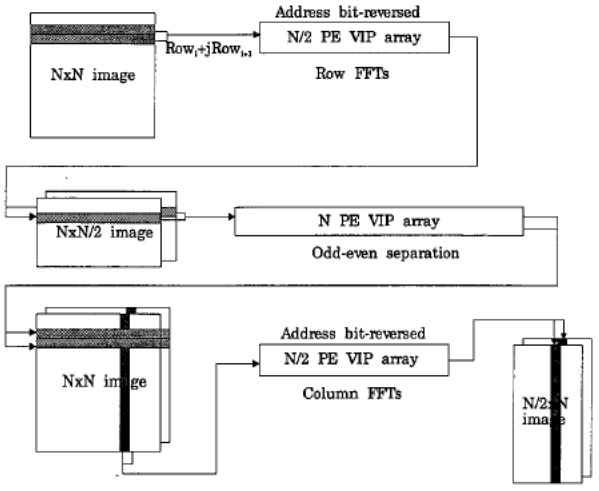
```
        /* 2D FFT using one N-PE array and corner turning */
for (i=0;i<N/4;i++)                  /* Transform N rows */
   FFT({r4i+jr4i+1}{r4i+2+jr4i+3})   /* Four rows concurrent */
for (i=0;i<N/2;i++)                  /* Separate odd-even data */
   OE_separate(Ri)                   /* Each row gives 2 rows */
for (i=0;i<N/4;i++)                  /* Transform N/2 columns */
   FFT({R2i}{R2i+1})                 /* Two columns concurrent */
```

Thus, the processing needed for an *NxN* 2D FFT transform with 16-bit precision and twiddle factors stored in internal RAM is then using equation (5.12) approximately

$$C_{2DFFT} = 2 \cdot \frac{N}{4}(704\log N + 4N) + \frac{N}{2} \cdot 200 = N(100 + 352\log N + 2N) \qquad (5.15)$$

If we compute 512^2 transforms with 16 bit precision (W=32) in a 512 PE array the processing of each image 28 ms, giving a frame rate of 35 Hz. However, several external VRAMs are required for the corner turning.

FIGURE 5.3.



Computing 2D FFT with FVIPa using external corner turning.

Without external corner turning

To compute an *NxN* 2D FFT within an *N*-PE 1D array we may first compute *N* different row FFTs along the array followed by one column FFT within each PE. However, if the input image data is real-valued the 2D FFT result is Hermitian, and we can use *N/2* PEs to first compute *N N*-point row FFTs as described above, and then compute *N/2 N*-point column FFTs by computing one transform within each PE.

To compute a complete FFT within one PE no communication overhead is required. The computations needed in each PE are *(NlogN)/2* butterflies, and since the twiddle factors can be input directly to the multiplier one butterfly takes *20p* cycles. Thus, one FFT requires approximately

$$C_{PE-FFT} = 20p \cdot \frac{N}{2} \cdot \log N = 5bN\log N \approx 160N\log N \qquad (5.16)$$

clock cycles. Thus, using equations (5.12) and (5.16) the computation of one *NxN* 2D FFT in an *N/2* FVIPa array takes

$$C_{2DFFT} = N(704\log N + 4N) + 160N\log N \approx N(864\log N + N) \qquad (5.17)$$

A 512^2 transform in a 256-PE array takes 82 ms. If we increase the shiftregister width to 32 the time decreases to 71 ms. A 512 PE array computes two 512^2 transforms in 82 or 71 ms, giving 41 or 36 ms for each transform. Hence, one 1024 PE FVIPa MCM module can perform 512^2-point FFTs at 50 Hz frame rate. However, the internal memory requirement is a problem. A FVIPa chip with 16 Kbit SRAM/PE, which is required for 512^2 transforms, is not feasible with current VLSI technology.

The Micro-Grain Array Processor, MGAP, running at 25 MHz is capable of computing a 32x32 2D FFT in a 32x32 PE array in 2.15 ms [59]. A 16 PE FVIPa array would require 2 ms for the same task. Of course, the FVIPa implementation only uses 16 PEs whereas the MGAP uses 1024 PEs.

5.6 Convolution-FFT comparison in FVIPa

In section 2.9 we noted that the pulse code compression convolutions in radar signal processing are often very long, and that Fourier domain implementations theoretically can be favorable. With the results obtained above we may compare signal domain convolution and Fourier domain multiplication implementations in a typical FVIPa array.

The shiftregister and multiplier widths are set to 16 and the data precision is 16 bits.

1D convolutions

The most common case when using a linear SIMD array for row-parallel processing is to process one sample in each PE. If we want to replace a 1D convolution along the array with a Fourier domain implementation we must implement an N-point FFT in an N-PE array. In FVIPa this can be made analogous with the $N/2$ PE implementation discussed in section 5.4. The main difference is that the communication distances increase a factor two, and the total number of cycles required with bit-reverse sorting and parameter input is approximately [44]

$$C_{N-FFT1} = 450\log N + 35N \tag{5.18}$$

and without bit-reverse sorting and parameter input

$$C_{N-FFT2} = 415\log N + 8N \tag{5.19}$$

Computing a convolution in the Fourier domain requires one forward transform, one pointwise multiplication and one backward transform. Thus, the number of cycles for a Fourier domain convolution implementation is approximately twice that of the results in equations (5.18) and (5.19).

In a convolution each kernel coefficient requires one MAC operation and for a complex 1D convolution kernel with L coefficients the convolution cycle count is approximately

$$C_C = 8p \cdot L = 256 \cdot L \tag{5.20}$$

If we study these functions we find that the complexity is approximately equal if the convolution kernel lengths are 15% of N. Thus, in RVIP and FVIPa it is not effective to implement typical pulse code compressions, with kernels who are 10% of N, in the Fourier domain. If the twiddle factors are stored in internal RAM and if the bit-reverse is not needed either, the break even length is around 5% and a typical pulse code compression is favorable to implement in the Fourier domain.

If input data and kernels are real-valued the results will be approximately the same since the correlation complexity decrease and two input rows can be Fourier transformed in the same transform, roughly halving the complexity in the two cases.

2D convolutions

Large 2D convolutions are not common in radar signal processing, but they might appear in typical image processing applications. The typical linear SIMD system layout in an image processing application is an *N*-PE array processing *NxN* images. The image data and convolution kernels are often real-valued rather than complex as in the radar signal processing examples.

The convolution of one *NxN* image with an *mxm* real-valued kernel then requires

$$C_c = N \cdot m^2 \cdot 2b = 32N \cdot m^2 \tag{5.21}$$

clock cycles. Using external corner turning the total Fourier domain computation requires approximately twice that reported in equation (5.15) which is

$$C_F = 2N(100 + 320\log N + 2N) \tag{5.22}$$

clock cycles. Using these figures we find that with an image size of 512^2 convolutions with kernels up to approximately 15x15, i.e. with a total of around 225 kernel coefficients, are more efficient to compute in the signal domain than in the Fourier domain. Note however that several VRAMs are required for the Fourier domain implementation.

Chapter 6

The Matrix VIP

6.1 Introduction

The use of a phased array antenna with several parallel channels increases the signal processing complexity considerably compared with non-array antennas. In the example below the computational complexity is approximately twenty times larger than in the LPD and MPD algorithms studied in chapter 3 and chapter 4, and the added antenna dimension makes the CPI image format three-dimensional rather than one- or two-dimensional. This sets new demands on inter-PE communication. Furthermore, the necessary space-time adaptive filtering requires the processing of large, complex, matrices.

This chapter is a preliminary study of how a future VIP architecture can be designed to meet these new requirements. Since the aim is an architecture that will be possible to implement by EMW within a few years rather than today we try to extrapolate the VLSI technology available within that time frame. We also assume that we can increase the clock frequency to 100 MHz. Since the large amount of matrix operations is the main difference between the standard radar signal processing and the Space-Time Adaptive Processing (STAP) we call the modified architecture Matrix-VIP, MVIP. Since the system will be airborne there are restrictions on the size of the system. It is estimated that the system must fit in a 25 liter box.

6.2 Space-time LPD radar algorithm

CPI Format

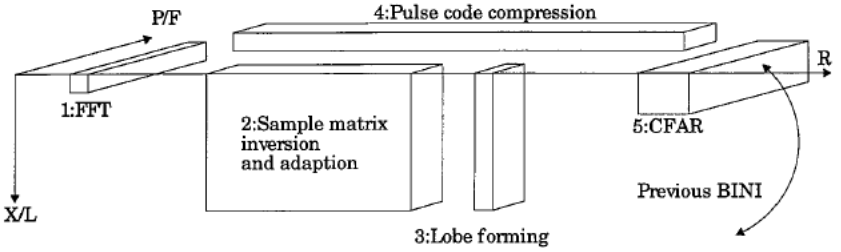
The radar has X parallel antenna channels which samples R range bins for each transmitted pulse, and processes P pulses per CPI. Typical values are given in Table 6.1. With 16-bit I and Q precision and 31 parallel channels the input data rate is almost 1.5 Gbit/s. The total amount of data in each CPI is 48 Mbit. Results from three consecutive CPIs, from the same bearing but with different prfs, are integrated into a Binary INtegration Interval, BINI, after the CFAR processing, however we only present the processing up to the multi-CPI BINI integration.

An overview of the 3D data structure and the inter-sample dependencies of the different parts of the algorithm is presented in Figure 6.1. In analogy with the work in the previous chapters we try to approximate the algorithm complexity to find an appropriate parallelization. We count the number of complex MAC operations required in each step of the algorithm as a measure of computation complexity.

Fourier transforms

The first operation is a pre-weighting window followed by a Fourier transform in the pulse direction. Each Fourier transform has an approximate complexity of $PlogP$ complex MAC operations if an FFT algorithm is used.

FIGURE 6.1.



The main parts of the space-time algorithm. The numbering indicates the sequence of the parts of the algorithm and the shapes of the boxes corresponds to the communication directions.

TABLE 6.1.

Space-time LPD radar specification	Value
Sampling frequency, f_s	1.5 MHz
Range resolution, Δr	100 m
PRF, f_p	~450 Hz
Number of prfs	3
Receiver duty factor, x	90%
Processed range bins, R	3000
Unambiguous range	< 333 km
Number of pulses, P	16
Velocity resolution, Δv	~0.5 m/s
Unambiguous velocity	~3.75 m/s
CPI time	~35 ms
Samples/CPI = RP	48000
Antenna channels, X	31
Lobe directions, L	15
Suppression integration, N	64

Adaptive clutter suppression

The next step is the adaptive clutter suppression, which is performed by multiplying the FFT output data with an approximation of the inverse of the covariance matrix of the antenna channel data [51], [52]. Each local covariance matrix is approximated by taking the mean of N neighboring covariance matrices $X_i^H \cdot X_i$, where N is at least twice as large as X (The superscript H denotes the conjugate-transponate of a vector or matrix.). Only one covariance matrix estimate per N samples is computed, i.e. $P \cdot R/N$ estimates are made per CPI.

Instead of inverting the covariance matrix the operation can be performed by QR-factorization of the sample matrix $X_i .. X_{i+N}$ followed by an inversion of the resulting upper-triangular R matrix. This is possible since we can write the covariance estimate as [30]

$$C = \begin{bmatrix} X_i \\ \dots \\ X_{i+N} \end{bmatrix}^H \cdot \begin{bmatrix} X_i \\ \dots \\ X_{i+N} \end{bmatrix} = \left(Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix} \right)^H \cdot \left(Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix} \right) = \tag{6.1}$$

$$= \begin{bmatrix} R \\ 0 \end{bmatrix}^H \cdot Q^H \cdot Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}^H \cdot \begin{bmatrix} R \\ 0 \end{bmatrix} = R^H \cdot R \tag{6.2}$$

which makes it possible to compute the inverse covariance estimate using

$$C^{-1} = R^{-1} \cdot R^{-H} \tag{6.3}$$

This inversion scheme is both less complex to compute and more numerically stable than the inversion of the mean of the N covariance matrices. The numerical stability decreases when the dynamic range increases, and C has a power of two larger dynamic range than R.

The QR-factorization scheme we have found most suited for factorization of an NxX matrix on an N-PE linear SIMD array is a Modified Gram-Schmidt, MGS, algorithm [30]. QR-factorization using Householder transformations, which theoretically has lower computational complexity, can be implemented similarly but the theoretical complexity gain is lost in the parallelization. Further discussions on parallel matrix factorizations can be found in for instance [11] and [16].

In pseudo code the MGS algorithm, where the data matrix A is overwritten by the orthogonal matrix Q, can be written [30]

```
for (k=1;k<=X;k++)          /* For all rows in R/cols in A */
  r_{k,k} = sqrt(a^H_k × a_k)   /* Diag el r_k, norm of column a_k */
  a_k = a_k / r_{k,k}           /* Normalize column k in A */
  for (j=k+1;j<=X;j++)      /* Columns above diagonal k */
    r_{k,j} = a^H_k × a_j       /* Compute r_{k,j} */
    a_j = a_j - (a_k · r_{k,j}) /* Update column in A */
```

All elements in R are computed by a complex inner product

$$a^H{}_k \times a_j = \sum_{i=1}^{N} a^*{}_{i,k} \cdot a_{i,j} \tag{6.4}$$

The complexity of the MGS algorithm is approximately NX^2 MAC operations. Note that the algorithm contains X square roots and divisions and that the diagonal elements in R are real-valued, which simplifies the implementation of the summation and division. When the elements of R are computed we store each element $r_{i,j}$ in memory position $f(j\text{-}i)$. That is, all diagonal elements are stored at position "zero", elements one position above the diagonal in position "one" and so on. This facilitates the inversion of R.

We call the inverse of R, which also is an upper-triangular matrix, P and it can be obtained by solving the equation system

$$p_{k,k} = \frac{1}{r_{k,k}} \tag{6.5}$$

$$p_{k,j} = p_{k,k} \cdot \sum_{i=k+1}^{j} r_{k,i} \cdot p_{i,j} \qquad j > k \tag{6.6}$$

and the complexity is approximately X divisions and $X^3/4$ MAC operations. If we reformulate this using the memory locations defined above we get

$$p_{k,0} = \frac{1}{r_{k,0}} \tag{6.7}$$

$$p_{k,j} = p_{k,0} \cdot \sum_{i=1}^{j} r_{k,i} \cdot p_{k+i,j-i} \qquad j > 1 \tag{6.8}$$

where the memory locations are independent of the row index k, and thus the expression is easy to implement in a SIMD array with one row of data in each PE. Once again the dividend, which is the diagonal element, is real-valued.

The last step to achieve C^{-1} is then to multiply P with P^H, i.e. the complex conjugate and transponate. Since C^{-1} is complex conjugate symmetric only half of the matrix need be computed. In our notation the inner products are

$$c_{k,j} = \sum_{i=j}^{X-k} p_{k,i} \cdot p^*_{k+j,i-j} \tag{6.9}$$

and again the memory locations are row index independent and PE-dependent addressing can be avoided. Each inner product requires approximately $X/2$ MAC operations, giving a total complexity of $X^3/4$ MAC operations.

Adaptive clutter suppression

The adaptive clutter suppression is matrix-vector multiplications between the computed C^{-1}matrices and the data vectors X. The complexity is X^2 MAC operations.

Lobe forming

From the X antenna channels we form L lobes (bearings). Each lobe is formed by a vector-vector multiplication between the antenna data and a lobe direction vector. For each lobe we then need to perform X MAC operations, giving a total of $X \cdot L$ MAC operations.

Pulse code compression

The pulse code length is approximately 10% of R and each compression will then require $R/10$ MAC operations.

Envelop detection and logarithm

After the pulse code compression the data is envelop detected and logarithmed. We approximate the complexity with five real-valued operations.

CFAR detector

There are four criteria which must be fulfilled for a bin to give a target detection:

```
Local max in R
Local max in F
Local max in L (6 neighboring lobes)
R is at least 3 dB larger than 12 of 16 closest range neighbors
```

Up to four of the six neighboring lobes are from the previous BINI. Sometimes the local max in frequency is exchanged for a global max in frequency to reduce the number of target indications. This reduction is performed since a target often returns echoes with a variety of Doppler frequencies within the same range bin. Twenty-six comparisons are required to implement the CFAR detector, making the complexity approximately 26 real-valued MAC operations.

Target reduction

Up to one hundred detected target positions and their respective envelops are output. This can be implemented using GLU logic analogous to the FVIP implementation discussed in section 4.2.

Algorithm complexity summary

The complexity of the windowing and FFT operation is approximately $R{\cdot}P{\cdot}X{\cdot}(1+logP)$ complex MAC operations. The QR-factorization requires $(P{\cdot}R/N){\cdot}N{\cdot}X^2$ operations. Computing R^{-1} and C^{-1} requires approximately $(P{\cdot}R/N){\cdot}X^3/2$ operations. The clutter suppression and lobe forming are matrix-vector multiplications requires approximately $R{\cdot}P{\cdot}X{\cdot}(X+L)$ operations. The pulse code compression requires approximately $(R{\cdot}P{\cdot}L){\cdot}R/10$ operations, and each envelop detection and logarithm around 5 real-valued operations. The CFAR complexity is around 26 real-valued operations. The total complexity is then approximately

$$C = RPX(1+\log P+X+L)+\frac{RP}{N}\left(X^2N+\frac{X^3}{2}\right)+RPL\left(\frac{R}{10}+\frac{31}{4}\right) \qquad (6.10)$$

complex MAC operations. This shall be computed for each CPI, i.e. in 35 ms, which gives a total complexity of approximately 10 G complex MAC operations per second, or approximately 40 G real-valued MAC operations.

MVIP parallelization

In the space-time algorithm outlined above the two most computationally demanding tasks are the matrix inversion/adaption and the pulse code compression. Both these steps require good communication in the range and channel directions. There is also a need for communication in the pulse/frequency direction in the FFT and the CFAR. Thus, an initial parallelization attempt would be to use one PE for all channels and pulses in one range bin. This would give $R=3000$ PEs, and the processing demand on each PE is around 15 M operations. With 100 MHz clock frequency a word operation must then be performed in 7 cycles, which is impossible in a bit-serial architecture but possible with a bit-parallel ditto.

However, since there is more communication in the channel/lobe direction than in the pulse/frequency direction, we then attempt to use one PE to process all channels in one range and pulse bin. This gives $R{\cdot}P$ PEs, and the processing demands on each PE is around 1 M operations. This indicates that a word operation must be performed in 100 cycles, which is possible with a bit-serial VIP processor.

With $R{\cdot}P$ PEs in a linear array we can either align the array so that neighboring PEs contains neighboring pulse or neighboring range values. With neighboring pulse values the FFT/CFAR communication is simple to implement, but the overhead in the matrix inversion and the pulse code compression is too large. Thus, we align the array so that neighboring PEs contain neighboring range values, making the matrix inversion and the pulse code compression possible to implement efficiently. However, now the communication overhead in the pulse direction makes the FFT and the CFAR impossible to implement. To overcome this problem we will introduce a new communication capability in the pulse direction. Now, in the matrix inversion N PE blocks work independent of each other. This makes it feasible to divide the array in N PE blocks in the range direction and P PE blocks in the pulse direction. Thus, a feasible MCM layout would contain $NxP = 1024$ PEs. Internally on the MCM the PEs can communicate in the pulse as well as in the range direction. This new communication scheme will be discussed in more detail below.

6.3 Floating-point arithmetics

Representation

One of the initial demands on the implementation of this application was that floating-point arithmetics were to be used. EMW considered the matrix inversion unsuitable for fix-point arithmetics due to the large dynamic range which would require very large word sizes in a fix-point implementation. Therefore we investigate a floating-point version of the VIP PE. A thorough investigation of the properties of floating-point arithmetics can be found in for instance [39]. However, we intend to use a non-standardized floating-point representation.

The m-bit mantissa and the e-bit exponent are stored as two-complement integers. Typically we intend to use 16-bit mantissa and 8-bit exponent. If we increase the mantissa to 24 bits the sizes of the multiplier and accumulator must be extended above their present sizes. We assume that the mantissa is normalized so that its absolute value is between

$$0.5 \le |m| < 1 \tag{6.11}$$

If the mantissa of a result cannot be normalized the number is considered as zero and the mantissa is set to zero and the exponent is set to the minimum value. The use of the minimum exponent value instead of zero facilitates arithmetic operations as will be evident below. The conversion to/from floating-point representation is made in the same way as the normalization step shown in the floating-point addition pseudo code example below and requires approximately $2m+2e$ cycles, i.e. around 50 clock cycles.

Arithmetic operations

Floating point addition/subtraction consists of a number of steps: First the mantissas must be aligned according to the difference of the exponents so that they have a common exponent. The mantissa from the number with the smallest exponent is down (right) shifted arithmetically while the exponent increases. After $m-1$ shifts the mantissa equals zero and no further

alignment shifts are useful. Note however, that if the number is negative the mantissa does not reach zero in our representation, only $-2^{-(m-1)}$. The aligned mantissas are then added/subtracted, and after that the resulting mantissa is normalized and the exponent is decremented accordingly. The normalization of the mantissa is made by arithmetic up (left) shifts of the mantissa fraction until the MSB differs from the sign bit. If the mantissa is not normalized after *m-2* shifts the number is considered as zero, and the exponent is reset to the minimum value to represent this. Before the normalization a check for mantissa overflow is needed, and if overflow occurred a down (right) shift is performed and the exponent is increased. Before the normalization starts we have an *m+1* bit mantissa.

Here we give two addition examples which shows both alignment, overflow and normalization.

```
   Example 1:

   0.1010·2^011          /* 10/16·2^3 */
+  1.0111·2^011          /* -9/16·2^3 */
--------------
=(0)0.0001·2^011         /* 1/16·2^3, m less than 0.5 */
=   0.1000·2^000         /* Normalized result */

   Example 2:

   0.1010·2^010          /* 10/16·2^2 */
+  0.1100·2^001          /* 12/16·2^1, Not aligned */
------------------
=  0.1010·2^010          /* 10/16·2^2 */
+  0.0110·2^010          /* 6/16·2^2, Aligned to largest e */
--------------
=(0)1.0000·2^010         /* 16/16·2^2, overflow */
=  0.1000·2^011          /* Normalized result 8/16·2^3 */
```

Multiplications and divisions are less complex to implement than additions and subtractions since no pre-alignment of the mantissas is necessary. In a multiplication the mantissas are multiplied and the exponents are added, and in a division the mantissas are divided and the exponents subtracted. The normalization after a multiplication is simplified since we know that the absolute value of the resulting mantissa is

$$0.25 \le |m| < 1 \tag{6.12}$$

and after a division between

$$0.5 < |m| < 2 \tag{6.13}$$

Thus, after a multiplication only a one position normalization test is required, and after a division only a mantissa overflow check. We note that the whole *2m-1* mantissa after a multiplication or division need not be saved since the last *m-2* bits always are discarded when the normalized mantissa is stored with *m*-bit precision.

Exception handling

One of the problems with floating-point arithmetics is how to handle exponent overflow and underflow and zeros in arithmetic operations. Exponent overflow or underflow occurs if the exponent value falls outside its range. In those cases we set a bit in a register to indicate that overflow or underflow has occurred, and the exponent is set to the maximum or minimum value.

The handling of zeros is a bit more complicated. During addition and subtraction we obtain the desired result automatically since the largest exponent always is from the non-zero number and the zero mantissa does not contribute to the resulting mantissa. If one of the operands in a multiplication is zero the resulting mantissa is zero, and this is detected during the normalization, giving a zero as result. The same holds for the dividend in divisions. However, if we divide by zero we must detect this, activate the overflow flag and set the exponent to the maximum value.

Complexity of bit-serial floating-point arithmetics

A bit-serial floating-point addition/subtraction requires that the two exponents are fetched from memory and subtracted, requiring at least $2e$ clock cycles. One mantissa must be fetched from memory and aligned, requiring at least m cycles. The mantissa addition requires at least m cycles as does the normalization/store process. The resulting exponent must also be stored. Thus, an optimal bit-serial add/sub operation requires $3m+3e$ clock cycles, which also is the number of memory access operations. In an SIMD array it is cumbersome to fetch the correct mantissa from memory and align it on the fly, it is easier to fetch the two mantissas to two registers and then align one of the registers conditionally. Likewise it is hard to make the normalization at the same time as the mantissa is stored back to memory. Therefore an add/sub in a bit-serial architecture is expected to require at least $6m+3e$ clock cycles. This can be decreased by pipelining if several consecutive additions are computed.

A bit-serial floating-point multiplication contains one addition and one multiplication. With a serial-parallel multiplier, a multiplication can be implemented in $3m$ cycles and the addition in $3e$ cycles. Thus, the optimum multiplication performance is around $3m+3e$ cycles. A floating-point MAC operation requires one floating-point multiplication, an alignment of the multiplication result with the accumulated sum, and an accumulation. Several of these steps can be pipelined and combined, e.g. the multiplication and the previous accumulation can easily be implemented concurrently. The theoretical limit is $3m$ clock cycles since a $3m$ multiplication must be performed. If a $2m$ mantissa is used in the accumulations the complexity increases to at least $4m$ since the $2m$ accumulation and $2m$ alignment cannot be parallelized.

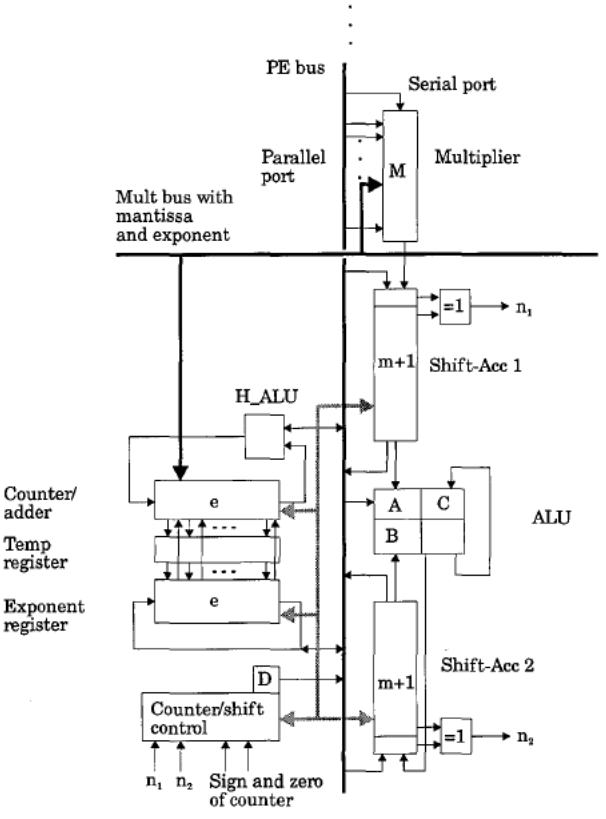
In our modified floating-point MVIP PE we aim to get close to these performance figures.

6.3.1 Floating-point PE

As was described above the main difference between floating-point and fix-point arithmetics is the need of data dependent arithmetic shift operations. To be able to shift data we redesign the accumulator to become a bi-directional shiftregister, see Figure 6.2. For efficient alignments and MAC operations we need two such accumulator shiftregisters, called shift-accs. Both of these registers are $m+1$ bits long, to allow storage of an m-bit mantissa and an overflow bit. The extra shift-acc is connected between the multiplier and the ALU, and therefore an $m+1$ clock cycle pipeline is introduced between a multiplication and an accumulation. In order to

perform exponent operations concurrently with mantissa operations we have added an e-bit parallel up/down counter and adder. The counter also has an ALU for serial exponent addition and subtraction. A parallel eight-bit up/down counter and adder working at 100 MHz should not be any problem to implement. To make MAC operations more efficient we also include two extra parallel exponent registers, called temp and exponent register.

FIGURE 6.2.



The floating-point MVIP ALU. The grey arrows depict control signals.

Conditional operations

The counter/adder is used to control the data dependent shift-acc shifts during alignment and normalization operations. During additions and subtractions the counter and the H_ALU is used to compute the exponent difference of the operands, and the mantissas are stored in the shift-accs. The sign of the result of the exponent subtraction indicates which of the two mantissas that should be shifted in the alignment, and the value the number of times. To perform this

a conditional count and shift operation is performed *m-1* times. If the counter value is positive it is decremented and shift-acc2 shifted, if the counter value is negative it is incremented and shift-acc1 is shifted. When the counter reaches zero it stops and no more shifts are executed.

After an arithmetic operation the resulting mantissa in shift-acc2 should be normalized. Initially the n2 signal indicates overflow, and if it is *one* the counter is incremented, otherwise shift-acc2 is shifted up so that n2 indicates if the register contents are normalized. Then a conditional count and shift operation is performed *m-2* times. If n2 is *zero* the register is shifted up and counter1 is decremented, and if n2 is *one* the shifts and decrements stop. If the mantissa remains non-normalized after *m-2* shifts it is zero and the exponent is reset to the minimum when it is stored.

During the alignment in MAC operations we need both the counter and the two exponent registers. We align the *m*-bit multiplication result in shift-acc1 with the so far accumulated, non-normalized, sum in shift-acc2. Initially the counter contains the exponent of the multiplication result and the exponent register the exponent of the accumulation result. We store the contents of the counter in the temporary register and then we compute the exponent difference. After a mantissa alignment implemented as above we use the sign from the exponent subtraction, and if it is *zero* we load the result in the temporary register to the exponent register to update the accumulation exponent.

Associated with the counters we also have a register *D* indicating if overflow or underflow of the exponent has occurred. If the contents of the *D* registers are output to the GLU we can obtain an error indicator from the array by reading the GOR. It might be worthwhile to have separate overflow and underflow registers since underflow might be tolerated but probably not overflow.

Floating-point addition/subtraction

In MVIP we propose to implement floating-point addition/subtraction according to the pseudo code below

```
SA1 = m1                  /* Load mantissa 1, m */
SA2 = m2                  /* Load mantissa 2, m */
C1 = e1                   /* Load exponent1 to counter e */
C1 -= e2                  /* Subtract exponent2, e */
Align                     /* Align according to counter, m */
{                         /* Concurrent processing */
   Add_M                  /* Add M-bit mantissas, m */
   C1 = sign*e1+sign*e2   /* mux new exp, 2e */
}
Norm_M_SA2/C1             /* Normalize mantissa/exp m */
Save(SA2)                 /* Store new mantissa, m */
Save(C1)                  /* Store new exponent, e */
```

In the code we annotate operations performed in parallel within brackets. If we count the number of cycles we see that the addition/subtraction requires approximately *6m+3e*, around 120 cycles. Without the extra hardware we expect that a floating point addition requires at least 400 clock cycles. If several sequential additions are performed the load of one mantissa can be implemented concurrently with the normalization and the number of cycles decreases to *5m+3e* clock cycles. Furthermore, if several consecutive accumulations are computed the

number of cycles decreases to *3m+e*. Also, if we know that both operands have the same sign, e.g. after a square-root or an envelop detection, the normalization can be skipped, only an overflow correction is needed, giving *5m+3e* cycles.

Multiplication and division

The pseudo code for a floating-point multiplication is

```
Load_coeff(m1)          /* Load multiplier, m/1 cycles */
{                       /* Parallel execution */
   Mult(m2)             /* Mult mantissas 2m */
   C1 = e1              /* Load counter with e1, e */
   C1 += e2             /* Add with e2, e */
}
Norm_M_SA1/C1           /* Only one step/cycle */
Save(SA2)               /* Save new mantissa, m */
Save(C1)                /* Save new exponent, e */
```

The multiplication operation requires approximately *4m+e*, around 75 clock cycles. With parallel coefficient load the number of cycles decreases with *m* cycles. If several multiplications are pipelined and *m* is at least three times as large as *e* the number of cycles decreases to *4m*. In a division the mantissas are divided and the exponents are subtracted and if the multiplier is redesigned to a combined multiplier/divider as in [35] the number of cycles is approximately *15m+e*, around 140 clock cycles.

Multiply and accumulate

The easiest way to compute a floating-point MAC is to use a multiplication followed by an addition. Then the number of cycles is then approximately 10m+4e, around 200 clock cycles. Nestling the two operations, and using the double shift-accs in a pipeline gives an operation as in the pseudo code

```
(Load_coeff(m1)         /* Load next coefficient mantissa, m/1 */)
Mult(m2)                /* Multiply and discard LSBs m */
{
   Mult(m2)             /* Multiply to shift-acc1, m */
   Add_M                /* Accumulate previous, m */
   C1 = em = e1+e2      /* Mult-exponent, 2e */
}
Norm_M_C1               /* Normalize mult-res */
Tmp = C1                /* Save mult-exponent in Tmp-reg */
C1 -= Exp_reg           /* Subtract accumulated exp */
{
   Align_M_C1           /* Align, use Counter m */
   Load_coeff(m1)       /* Load next coefficient mantissa, m/1 */
}
If (sign == 1)          /* Conditionally store new exp */
   Exp_reg = Tmp
```

where the previously accumulated mantissa and exponent are found in shift-acc2 and the exponent register respectively. We see that the number of cycles is approximately *3m*, around 50, and the save from the straight-forward implementation is around 75%. In a MAC operation parallel coefficient load does not give any performance gain. If the shift-accs are expanded to *2m* a MAC operation will require *4m* clock cycles.

Floating-point performance

The cycle counts for real and complex floating-point operations using the modified floating-point MVIP PE can be seen in Table 6.2. We note that with 16-bit mantissa and 8-bit exponent a complex MAC requires around 225 clock cycles and a complex division around 950 cycles. If we transform the figures in Table 6.2 to be proportional to the data size $b=m+e=1.5m$ we get *5b* for an addition, *2b/3b* for multiplication and less than *3b* for a MAC operation, and we see that multiplications and MAC operations are performed as efficient as fix-point ditto in the RVIP/FVIP PE, but that additions have lower performance.

The performance figures given for the 32-bit precision floating-point ALU in [69] state that pipelined addition/subtraction, multiplication and MAC-operations can be performed in *4m* cycles and non-pipelined in approximately *8m*. Compared with our figures we note that the VIP MAC performance is better, but that pipelined additions, subtractions and multiplications are slightly slower in VIP. If pipelining cannot be used the VIP performance is better. The largest architecture difference between the two architectures is that they use three shift-accs, one *2m* and two *m* long, where the two *m* long registers can be connected in series, and that *2m* mantissas are used in the accumulations.

TABLE 6.2.

Operation	Approximate cycle count	
	Real	**Complex**
Add/sub	6m+3e	12m+6e
Accumulate	3m+e	6m+2e
Multiplication	4m+e	18m+6e
MAC	3m	12m
Division	15m+e	57m+11e

6.3.2 Floating-point function approximations

With floating-point arithmetics several function approximations can be implemented easier than with fix-point arithmetics. Here we present two functions that are required in our STAP application, square root and logarithm computations.

Floating-point square root computation

For high-accuracy square root approximations we have chosen an iterative Newton-Raphson algorithm taken from [17]. We assume that we have a real, positive, floating-point number

$$X = m \cdot 2^e \tag{6.14}$$

which we normalize so that m is larger than one and the exponent is an even number. This requires a one or two position normalization from our normally used normalization. Thus we have

$$X = a \cdot 2^{2 \cdot k} \tag{6.15}$$

and then the square root is

$$\sqrt{X} = \sqrt{a} \cdot 2^{k} \tag{6.16}$$

Thus, we shift the new exponent one step down, and approximate the square root of a where we know that

$$1 \le a < 4 \tag{6.17}$$

The Newton-Raphson iteration formula gives

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{6.18}$$

which converges fast and monotonously towards the square root of a if we select a good starting approximation x_0. We implement a small table-lookup to find a good starting approximation. The table-lookup is implemented as follows: We store the b first bits of a in the counter. Then we decrement the counter whilst conditionally loading the parallel multiplier port with start approximations. In each PE the value loaded to the mult port the last cycle when the counter contains a positive value is saved. Thus, the b-bit table lookup can be implemented in approximately $b+2^b$ cycles. We propose to use a four-bit table-lookup and the initial approximation error will then satisfy

$$\left|x_0 - \sqrt{a}\right| \le 2^{-b} \le 2^{-4} \tag{6.19}$$

and after two iterations the error satisfies

$$x_2 - \sqrt{a} \le \frac{1}{2^3}\left(x_0 - \sqrt{a}\right)^4 \le \frac{1}{2} \cdot 2^{-18} \tag{6.20}$$

Therefore a 4-bit table lookup and two iterations gives the desired 16-bit mantissa precision. Each iteration contains one division using positive fix-point values and one addition, requiring around $8m+2m$, i.e. around 160 clock cycles. The total square root computation will then require around 340 clock cycles. If we use integer arithmetics we must perform a conversion to floating point numbers first, and then convert back afterwards. The cycle count then increases to around 1340 clock cycles with a fix-point PE with division hardware, and to 5340 with the RVIP/FVIP PE without special division and floating-point hardware.

Floating-point logarithm computation

Almost all radar algorithms we have studied contains a logarithm step, and with floating point arithmetics it is feasible to compute an accurate logarithm approximation. It is desirable to compute the $^{10}\log(x)$ but the $^{2}\log(x)$ is easier to approximate, and a conversion can then be made using

$$^{10}\log(x) = {}^{2}\log(x) \cdot {}^{10}\log(2) \tag{6.21}$$

That is, a conversion is made by a multiplication with a known constant. If we have a floating point representation of a positive number normalized so that the mantissa is between 1 and 2 we have

$$^2\log(m \cdot 2^e) = {}^2\log(m) + {}^2\log(2^e) = {}^2\log(m) + e \qquad (6.22)$$

This means that we must find an approximation for $^2\log(x)$ for x between 1 and 2. With the limited range of x a low order polynomial approximation is very accurate, and the error decrease with approximately one order of magnitude for every increment in the degree of the approximation polynomial. For a least-square adapted third order approximation, for instance, the absolute error is less than 0.0013, which is better than 8-bit fraction precision. The evaluation of a third-degree approximation polynomial and a final fix-point constant multiplication requires approximately $25m$ clock cycles, and for higher degree approximations, $5m$ more clock cycles for each degree. Thus, an approximation with 8-bit integer and 8-bit fraction precision can be performed in approximately 400 clock cycles. In the RVIP/FVIP PEs the computations can be made in approximately the same number of clock cycles, but the initial conversion to floating-point representation requires at least 500 clock cycles. This would give a total complexity of around 900 clock cycles. The logarithm approximation result is in fix-point representation.

6.4 The MVIP PE

Given the parallelization sketched above, where each PE is assigned to process all antenna samples in one range/pulse bin, we can find the PE modifications necessary to implement the space-time algorithm. We intend to use either the floating-point ALU presented above or up to 32-bit fix-point arithmetics with the complex MULT/ALU/ACC designs presented below. We assume that we can use a PE clock frequency of 100 MHz, and we will show later that we can implement 128 PEs on each chip. Since we want the PEs to work in blocks of 64 the layout is 64x2 PEs rather than 128x1 PEs. The chip layout is discussed in more detail later in this chapter.

The IO register

The input to each PE is data from $X=31$ channels, which is almost 1000 bits, in each CPI. Therefore we extend the IO register to 32 32-bit register banks per PE. However, since we only intend to use the register for data input the register cell layout can be simplified compared with the layout in RVIP/FVIP. To reduce the number of external connections on the chip we remove the random addressability, and instead data is written in sequential order to the registers. With a sampling frequency of 1.5 MHz and 31 parallel channels the input word data rate to each chip and PE is 1.5x31 = 46.5 MHz.

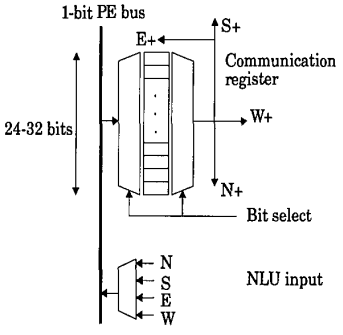
Inter PE communication

During the algorithm mapping above we found that two-dimensional communication was required to implement the STAP algorithm. The addition of a vertical, parallel, shiftregister is not feasible since the number of chip interconnections would be too large. For instance, with 64 PEs in a row on each chip a 16-bit parallel vertical shiftregister gives 2048 connections.

However, if only consecutive one-PE word shifts are required a serial communication channel is just as effective as a parallel ditto in a bit-serial architecture. This can be understood by noting the following: Assume that we wish to perform a convolution, i.e. a series of MAC operations where data is collected from neighboring PEs. For each *b*-bit sample in the convolution *b* cycles are required to move the word to the multiplier. With a serial neighbor communication channel a sample can be fetched from a neighboring PE instead of from the internal PE RAM without overhead. Using the broadcast function of the MVIP PE it is also possible to simultaneously store the word in the PE. Thus, at the next iteration each PE can access a word from a PE one step further away, effectively achieving the effect of a parallel shiftregister. To implement the word storage function a parallel communication register is still required in each PE, see Figure 6.3. Analogous with the name used for similar designs in LAPP/MAPP [20], [21] and NSIP [73] designs we call this design Neighborhood Logic Unit, NLU.

If longer shifts are required, as was the case at several points in the MPD algorithm described in chapter 4, there will however be a large overhead if serial NLU communication is used.

FIGURE 6.3.



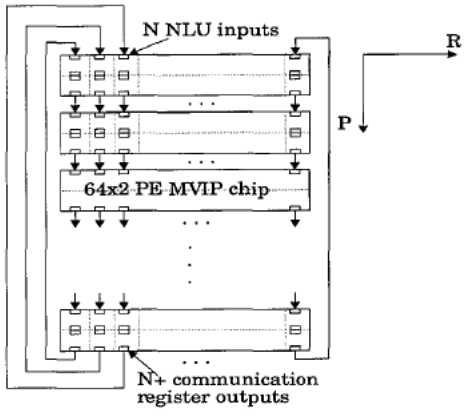
The new communication register design. The Neighborhood Logic Unit, NLU, is used to access data from one of the four nearest neighbors, North, South, East and West.

The N+ NLU outputs on one chip are connected to the corresponding NLU N inputs on the chip containing data from the next pulse, see Figure 6.4. The N+ output register in the chip with data from the last pulse is connected to the chip with data from the first pulse. The S+ and S connections are not required in our implementation and are not connected between chips. Thus we obtain a one-way vertical 64-bit communication between chips and with all the thus connected chips on a common MCM we believe that the communication can operate synchronous with the PE processing. The one-way communication makes it hard to implement FFT algorithms, but since the transforms are very short the gain of using an FFT algorithm compared to a DFT algorithm is low.

The horizontal NLU connections, which are connected between MCMs cannot operate synchronous with the PE processing however. One possible way to reduce the overhead we introduce high-speed asynchronous serial interfaces [63]. Thus, before a horizontal NLU operation the contents of the communication registers at the edge of a chip is transferred to a temporary register at the next chip. A horizontal NLU operation can then be performed on a word at the PE clock frequency.

We assume that each serial link, using asynchronous transfer, can use a transfer frequency of 1 GHz. With a communication register width which is up to 32 bits, a word transfer between two chips require up to 3.2 MVIP clock cycles. Including synchronization overhead we expect that a one-step shift requires ten MVIP clock cycles. When only consecutive shifts to the next PE are used this overhead can often be hidden in the processing. Also, when no data need to be transferred between the chips, as is the case with the communication in the matrix inversion, the horizontal NLU communication can be clocked at the PE frequency. The use of fast serial links also makes it easier to make the system reconfigurable, i.e. Asynchronous Transfer Mode type switches can be used to rout data to different MCMs.

FIGURE 6.4.



The vertical NLU communication on an MCM.

The RAM

The RAM must be extended to allow storage of all required data. The input data is almost 1000 bits, and with bit-expansion to 24 or 32 bit precision 1500 or 2000 bits are required. The Fourier coefficients require P complete words, and the storage of the Q and R matrices $2X$ complex words. To be able to implement the CFAR we must also store data from the three latest CPIs requiring $3L$ words. If we use 12-bit precision after the logarithm this sums up to 6 Kbit RAM with 24-bit precision and 8 Kbit with 32-bit precision. This is a much larger memory than we have previously used, but compared with the 32Kbit RAM/PE in the 64 PE IMAP chip it is still not very much [65].

Division in the multiplier

The multiplier has been redesigned analogously to the IVIP/WVIP architecture so that it is possible to perform division of positive fix-point integers in approximately $8b$ cycles. If both the divisor and dividend are two-complement integers the complexity increases to around $13b$. The design of the combined multiplier/divider is described in [35].

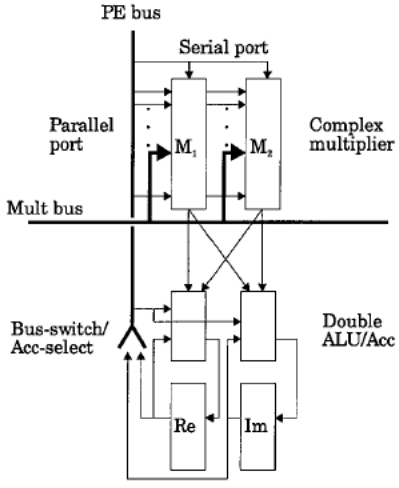
Complex Multiplier-Accumulator

To faciliate the processing of complex data we can extend the fix-point PE with a second multiplier and an ALU-Accumulator, see Figure 6.5. Using this structure the pseudo code for a multiplication of two complex-valued numbers will look like

```
/* Complex Mult operation Re = AC-BD, I = BC+AD */
/* A+jB is "data", C+jD is "coefficient" */
Load_M1(C)                          /* 1 or p cycles */
Load_M2(D)                          /* 1 or p cycles */
M1*= A(0),M2 *= A(0)
for (i=1;i<p;i++)                   /* loop over data, p-1 */
   M1*= A(i), M2*= A(i),
   R_Acc += M1(i-1),I_Acc += M2(i-1)
end
for (i=p;i<=2p;i++)                 /* Loop over mult, p */
   R_Acc += M1(i-1),I_Acc += M2(i-1)
end
Reset_mult                          /* Reset internal mult-state */
M1*= B(0), M2 *= B(0)
for (i=1;i<p;i++)                   /* loop over data, p-1 */
   M1*= B(i), M2*= B(i),            /* Crosswise add/sub */
   R_Acc -= M2(i-1),I_Acc += M1(i-1)
end
for (i=p;i<=2p;i++)                 /* Loop over mult, p */
   R_Acc -= M2(i-1),I_Acc += M1(i-1)
end
Save(R_Acc,RAM)          /* p cycles (2p for whole acc) */
Save(I_Acc,RAM)          /* p cycles (2p for whole acc) */
```

The complexity for the multiplication is approximately *6p* or *8p* depending on if the coefficient is loaded serially or via the external parallel load port. Performance figures for different complex operations are collected in Table 6.3. The performance of the RVIP/FVIP design from Table 3.3 is also given in the table and we see that the gain is approximately 50%. A complex MAC with 16-bit precision now requires approximately 100 clock cycles.

FIGURE 6.5.



The double-multiplier ALU.

TABLE 6.3.

Operation	Input 1	Input 2	Result	Clock cycles	
				RVIP/ FVIP	Complex mult
Complex mult	RAM	RAM	RAM	14p+4	8p+1
Complex mult	RAM	External	RAM	10p+8	6p+3
Complex MAC/ MSUB	RAM	RAM	Acc	12p+4	6p+1
Complex MAC/ MSUB	RAM	External	Acc	8p+8	4p+3
Butterfly operation	RAM	RAM	RAM	~24p	~18p

The active register

Analogous to the FVIP design we use an active register for PE-dependent operations. It is required at several places in the matrix inversion, where often only one single PE is allowed to store a result.

The GLU

A GLU, similar in function to the FVIP design described in section 4.2, is needed for local data distribution in the matrix inversion and for extracted data output. The GLU is not connected serially between chips as in the FVIP case, instead each *N*-PE block works independent. This increases the GLU speed since the ripple length is less than *N*, and we believe that a GLU oper-

ation requires 2 PE clock cycles, i.e. 20 ns to ripple through the N-PE array. We also have a new bit-serial COUNT design in the GLU, see Figure 6.6. Note that the output of the bit-serial COUNT is fed back into the array via the Global Or.

At several places in the matrix inversion we need to compute inner products where we sum data from N neighboring PEs on one chip. This can either be made in a tree structure using $logN$ additions and intermediate shift operations, using the COUNT logic as in [72] or using the new bit-serial COUNT logic. When using any of the two COUNT logic implementations we use the fact that the desired sum of N B-bit words D can be written

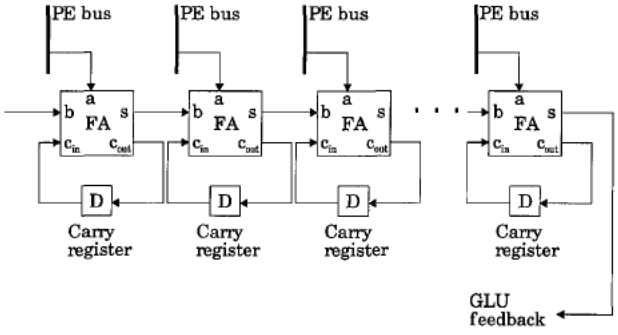
$$S = \sum_{i=1}^{N} D(i) = \sum_{i=1}^{N} \sum_{b=0}^{B-1} D(i,b) \cdot 2^b = \sum_{b=0}^{B-1} 2^b \cdot \sum_{i=1}^{N} D(i,b) \qquad (6.23)$$

Thus, the desired sum can be computed using B weighted summations of COUNT values. When using the new bit-serial COUNT logic all carry signals generated while summing the bit-plane with the weight 2^i are stored locally in the carry registers. The carry signals have the weight 2^{i+1} and are fed back into the summation at the next iteration together with the data whose weight is 2^{i+1}. To acquire all bits of the sum log$N$ extra iterations are required after all bit-planes have been added to output all carry information giving bit expansion and which is saved in the carry registers. The log$N$ steps are also required when using this design to compute a single bit-plane COUNT sum.

If we have floating-point data we must align the mantissas before the operation. The maximum exponent can be found using a successive approximation loop similar to the algorithm in section 7.4, part I, and each PE can then find the appropriate alignment shift.

The delay from input of one bit-plane until the GOR result can be polled by the PEs is expected to be 20 ns, i.e. 2 MVIP clock cycles.

FIGURE 6.6.



The bit-serial COUNT logic. FA denotes a full adder.

6.5 MVIP chip layout

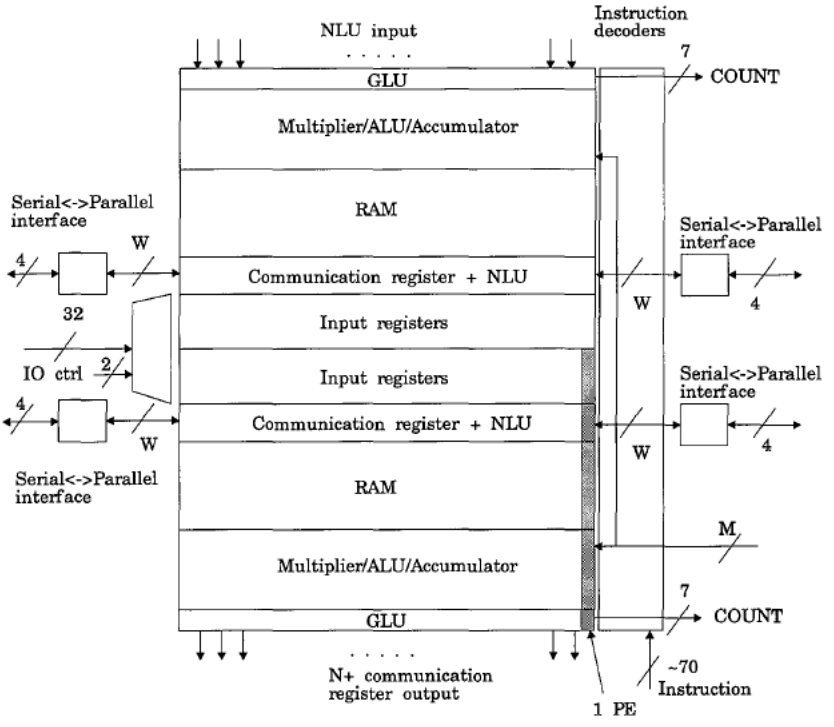
In our chip layout estimates we have assumed that we can use a 0.55 µm 2-metal CMOS memory process, and that the maximum chip size is around 15x15 mm. The expected areas for the PE-core of the two versions of the MVIP chip are presented in Table 6.4. We expect that a 128 PE chip with floating-point PEs and 6 Kbit RAM or a 128 PE chip with 32-bit complex fixpoint PEs and 8 Kbit RAM can be implemented on a 13x13 mm chip. The number of transis-

tors per chip is approximately 4 million for the floating-point version and 5 million for the fix-point version. If a non-SRAM process is used the number of transistors increases to 6 and 8 million respectively. Since we want 64 PEs to work in a group during the matrix inversion we use a 64x2 PE layout instead of a 128x1 ditto. This reduces the number of external connections on each chip and the number of chips required in a 64x16 PE MCM. The layout of such an MVIP chip can be seen in Figure 6.7. The number of external connections on each chip is around 250. If the pin-count is found to be too high serial interfaces of the same kind as on the shiftregisters can be implemented on the IO register and possibly also on the vertical NLU communication.

TABLE 6.4.

	Area in mm^2 for 128 PE core	
PE part	**24-bit float**	**32-bit fixed**
IO	26	26
SREG	5	5
RAM	42	58
MUL-ALU-ACC	41	28
Sum	**114**	**117**

FIGURE 6.7.



The layout of a 64x2 PE MVIP chip.

6.6 MVIP system layout

Each MCM consists of eight MVIP chips and one micro controller. This gives 64x16 = 1024 PEs on each MCM. The number of required MCMs is then 47, giving a total of 3008x16 = 48128 PEs. If we assume that each module requires 10x100x100 mm the modules requires 5 l. This leaves 15 l for interconnection, power supply, cooling and global system control.

6.7 MVIP results

The expected floating-point and fixed-point computation times for the different parts of the LPD STAP algorithm can be seen in Table 6.5. Without the bit-serial COUNT logic with feedback the execution times increase with 5-10 ms, proportional to the used precision. The large difference in time between the adaption and the lobe forming is due to the internal GLU distribution of C^{-1} in the adaption. The results indicate that the MVIP can solve the problem within the given time limit. If we increase the shift-accs to $2m$ bits and accumulate $2m$ bits in all MAC operations the execution time using the floating-point PE increases with around 5 ms. With 24-bit mantissa the execution time for the floating-point PE increases to 33.5 ms.

TABLE 6.5.

Algorithm part	Time (ms)		
	24-bit float	16-bit fixed	32-bit fixed
DFT	1.2	0.6	1.3
QR	7.1	4.3	8.3
Adaption	3.7	2.1	4.3
Lobe forming	1	0.4	0.9
PCC	9.1	4.8	9.1
CFAR + extract	1	0.8	1.6
Sum	**23.2**	**13.3**	**25.6**

6.8 Results and comparisons

In the earlier application studies we have compared the number of VIP chips required for a particular radar application with the number of required ideal DSPs operating at the same clock frequency. The complexity of the STAP algorithm was approximated with 40 Gops and an ideal DSP solution, where we assume that the DSPs work at 100 MHz, would then require at least 400 DSPs, which is slightly more than the 376 required MVIP chips. We believe however, that if we consider the additional problems of algorithm partitioning and the large memory requirements the MVIP solution is clearly favorable.

Another design of a PE array and a system solving this radar problem has been proposed by a research group from Halmstad/Gothenburg [2], [7], [70]. Their approach is to use 2D PE arrays with external memories. Each PE in a 32x32 array has a 16-bit ALU, 8-neighbor direct communication and row and column broadcast, but no internal RAM. All IO and memory communication is handled by the broadcast buses. It is expected that 64 PEs can be integrated on one chip, and that a 16-chip 32x32 PE MCM module the size of a palm can be fabricated. The expected PE clock frequency is 20 MHz [71]. Five such modules connected in a pipeline are stated to be capable of solving the task. Due to the abstract level of the architecture and system descriptions in the papers, and no knowledge of the used VLSI process parameter estimates, an accurate comparison between the solutions is impossible to make. We note however that their estimate has only 10% as many MCM modules and PEs, which indicates a much smaller system than the described MVIP system, and that they use 16-bit precision.

Using the same PE array layout but a bit-serial architecture and 16+8 bit floating-point precision it is expected that 30 modules clocked at 125 MHz are required [71].

Chapter 7

Discussion

7.1 Results

In this part of the thesis we have shown how three generations of the VIP architecture efficiently can solve three demanding real-time radar signal processing tasks.

The RVIP, which is currently manufactured by EMW, is aimed at ground-based LPD radar signal processing.

The FVIP, where the linear VIP array successfully emulates 2D arrays of varying size, is aimed at airborne MPD radar signal processing.

The MVIP, where the inter-PE interconnections has been extended to allow limited 2D array communication, is aimed at airborne phased array antenna LPD processing.

7.2 PE performance comparison

In chapter 1 we showed two somewhat VIP-similar linear SIMD architectures, the Serial Video Processor, SVP, and the Integrated Memory Array Processor, IMAP. Throughout this part of the thesis we have used MAC operations as a complexity measurement, and in Table 7.1 we show the MAC performances of the respective architectures. For larger word sizes than eight we have tried to extrapolate the MAC performance. In the table it is clear that the SVP architecture, which has no hardware support for multiplications, scales very poorly with increasing precision. The 8-bit IMAP architecture has outstanding performance at 8-bit precision, but already at 16-bit precision the VIP performance is better. If we study the IO capacity we see that the SVP has very high IO capacity, and that VIP has better IO capacity than IMAP. Furthermore, if we increase the IO frequency in VIP, as we did in FVIP and MVIP, the capacity can increase from the values in the table with a factor larger than two. The relative size of an IMAP and an RVIP PE can be hinted by noting that an IMAP PE contains around 15000 transistors and an RVIP PE around 5000. No transistor information is available for the SVP.

TABLE 7.1.

Operation		SVP	IMAP	VIP
K MAC operations per PE, scaled to 50 MHz	8-bit	340	4100	2100
	16-bit	95	930	1000
	24-bit	36	420	690
External communication in Mbit/s	Input Capacity	1320	320	640
	Output Capacity	792	320	640

If we use the performance figures from Table 7.1 above we can find out if the architectures can be used for the radar applications studied here. If we approximate the data precision to be 16-bit in the LPD and 24-bit in the MPD algorithm, and we use one PE to process each range/pulse bin we find that an SVP PE only has 10% of the required performance, but that IMAP has 25-60% more capacity than needed. Thus it would be possible to implement the algorithms

with the IMAP but not with the SVP. If we study the IO requirements the IMAP architecture has enough IO capacity, but since the width of the IO channel is much smaller than the width of the data temporary buffering is probably needed. Furthermore, the data dependent global operations in the MPD algorithm cannot be implemented efficiently in the IMAP or the SVP arrays.

7.3 Future work

The ongoing VLSI evolution makes it possible to integrate more features on each chip, and the smaller feature sizes also makes it possible to increase the on-chip clock frequency. This gives design problems that we have not encountered before. It will become increasingly difficult to distribute instructions to the PE array even with a low level micro controller unrolling bit-serial loops. It will be impossible to ensure complete synchronism in large systems, making it necessary to find new strategies for inter MCM communication. It will be increasingly difficult to implement large PE RAMs with access times that matches the PE clock frequency.

We have not studied resolving implementations very thoroughly. Once we found an implementation that was feasible to implement we used that, but by studying simulations using real data it is probably possible to find a more effective implementation.

The use of serial communication links and on-chip serial-to-parallel and parallel-to-serial conversions introduced in chapter 6 will be subject to further investigations. Apart from studying the maximum transfer frequency, we want to find suitable parallel buffer layouts for different reconfigurable 2D and 3D array emulations. This would for instance be needed if the space-time radar algorithm would use changing CPI matrix sizes in the same way as the MPD algorithm.

The floating-point PE should be investigated further to verify the performance and area estimates, and to find possible improvements. One detail to study is if the use of an m long accumulator gives significantly poorer accuracy than using a $2m$ ditto. It would also be interesting to find the precision requirements that makes it favorable to use a floating-point PE. Different QR factorization algorithms should be studied to find the best trade off between numerical stability and implementation efficiency. The used modified Gram-Schmidt algorithm was selected only because it could be implemented efficient and more numerical stable methods are known. It might also be worthwhile to study square-root and division free QR factorization algorithms [23], which makes a fix-point implementation much easier.

The extension of the bit-serial PE to handle two to eight bits in parallel has been studied in [35], and should probably be studied further. In that thesis there is also a study of different ways to code the instruction word to possibly decrease the bit-rate of the instruction flow. There is also an ongoing cooperation with the computer science department to design efficient high-level compilers capable of generating VIP code.

Abbreviations used in part II

BINI Binary INtegration Interval, The time during which binary data from several CPIs is integrated to increase the detection probability, remove ghost echoes, and possibly resolve velocity and range data beyond the ambiguity limits.

CFAR Constant False Alarm Ratio, Processing to reduce the number of false/multiple target indications from a CPI.

CPI Coherent Processing Interval, The time during which radar signal data is processed phase coherent. Typical CPIs contains data received during the transmission of 10-1000 pulses.

DFT Discrete Fourier Transform, A discrete version of the continuous Fourier transform.

DSP Digital Signal Processor, A processor aimed at signal processing, typically with high IO-bandwidth and fast floating-point MAC hardware.

EMW Ericsson Microwave systems, Swedish radar manufacturer, previously ERE.

ERE Ericsson Radar Electronics, Swedish radar manufacturer, name changed to EMW 1995.

FFT Fast Fourier Transform, A class of efficient algorithms implementing the discrete Fourier Transform, DFT.

FVIP Flight-VIP, A VIP architecture aimed at airborne MPD radar signal processing. Two different versions, FVIPa and FVIPb have been designed.

GOPS Giga OPerations per Seconds, The number of word-operations, typically MAC equivalent, in an algorithm scaled with 10^9. Or the peak-performance in number of operations for an architecture.

HPD High Pulse repetition Doppler, A class of radars aimed at unambiguous velocity detection. Typical prfs range from 100-300 KHz.

IFM Integrated Functional Memory, An SIMD architecture presented by NEC, name later changed to IMAP.

IMAP Integrated Memory Array Processor, An SIMD architecture with 8-bit PEs presented by NEC.

IVIP Infra-red VIP, A VIP architecture aimed at high resolution IR signal processing

LP Low-Pass, A low-pass filter.

LPD Low Pulse repetition Doppler, A class of radars aimed at unambiguous long distance ranging. Typical prfs range from 250-4000 Hz.

MAC Multiply and ACcumulate, An operation which multiples two numbers and adds the results to an accumulated sum. Used in convolutions and matrix/vector multiplications.

MCM Multi Chip Module, A large (ceramic) substrate where several chip can be bonded together.

MGS Modified Gram Schmidt, A QR factorization algorithm.

MIMD Multiple Instruction stream Multiple Data stream, A class of parallel computer archi-

tectures where all processors can execute individual programs asynchronous.

MISD Multiple Instruction stream Single Data stream. A class of data architectures where several processes the same data concurrently. Pipelined architectures are often considered as MISD.

MOPS Mega OPerations per Seconds, The number of word-operations, typically MAC equivalent, in an algorithm scaled with 10^6. Or the peak-performance in number of operations for an architecture.

MPD Medium Pulse repetition Doppler, A class of radars aimed at simultaneous unambiguous range and velocity detection. Typical prfs range from 10-20 KHz.

MTI Moving Target Indicator, A filtering to remove all echoes from non-moving targets.

MVIP Matrix-VIP, A VIP architecture aimed at LPD STAP-processing.

NLU Neighborhood Logic Unit, Hardware for inter-PE communication used in LAPP/MAPP, NSIP and MVIP architectures.

PCC Pulse Code Compression, A filtering which compresses the energy of a long transmitted pulse into a single range bin.

PRF Pulse Repetition Frequency, The frequency of the pulse transmissions from the radar.

RVIP Radar-VIP, A VIP architecture aimed at ground-based LPD radar processing.

SIMD Single Instruction stream Multiple Data stream, A class of parallel computer architectures where all processors execute the same instruction synchronously.

STAP Space-Time Adaptive Processing, A class of algorithms used for adaptive clutter suppression in phased-array antenna radars.

SVP Serial Video Processor, An SIMD architecture with bit-serial PEs presented by Texas Instruments.

VIP Video Image Processor, An SIMD architecture aimed at real-time video-rate image processing.

WVIP Wood-VIP, A VIP architecture aimed at real-time wood inspection signal processing.

References

[1] Adams III G., Bronson E. C., Casavant T. L., Jamieson L. H., Kamin III R. A., *Experiments with Parallel Fast Fourier Transforms,* In *Parallel Algorithms and Applications for DSP Applications,* Ed. by Bayoumi A., Kluwer Academic Publishers, 1991.

[2] Andersson H., Gustafsson S., *A Simulator for a Class of Computer Architectures in Radar Systems,* Master thesis No CCA-9909, Halmstad, june 1995.

[3] *Alpha 21164 Microprocessor Product Brief,* Digital, 1995.

[4] Arvidsson R., *Massively parallel SIMD processor for search radar signal processing,* Proc of RADAR '94, Paris 1994.

[5] Barbarossa S., Farina A., *Space-Time-Frequency Processing of Synthetic Aperture Radar Signals,* IEEE Trans. on Aerospace and Electronic Systems, Vol. 30, No. 2, April 1994.

[6] Batcher K.E. *Design of a Massively Parallel Processor,* IEEE Computer, Vol C29, 1980.

[7] Bengtsson L., Nilsson K., Svensson B., *A High-Performance Embedded Massively Parallel Processing System,* Proc of Proc of IEEE and Euromicro, MPCS '94, Italy, 1994.

[8] Blevins D.W., Davis E.W., Heaton R.A., Reif J.H., *BLITZEN: A Highly Integrated Massively Parallel Machine,* journal of Parallel and distributed computing8, 1990.

[9] Bracewell R.N., *The Fourier transform and its applications,* McGraw-Hill, 1986.

[10] Choudhary A. N., Patel J.H., *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems.,* Kluwer Academic Publishers, 1990.

[11] Clint M., Weston J.S., Flannagan J.B., *Efficient Gram-Schmidt Orthogonalization on an array processor,* Proc of CONPAR 94-VAPP VI, LNCSS 854, Springer Verlag, 1994.

[12] Danielsson P.E., Emanuelsson P, Chen K, Ingelhag P, Svensson C, *Single-Chip High-Speed Computation of Optical Flow, Proc. of MVA '90 IAPR Workshop on Machine Vision Applications,* Tokyo 1990.

[13] Denyer P., Renshaw D., *VLSI Signal Processing: A Bit-serial approach,* Addison-Wesley, 1985.

[14] Edefors P.L., *A 900-Mbit/s 1.0-mm CMOS serial-parallel multiplier,* In Thesis 377, Liu-TEK-LIC-1993:19, Linköping 1993.

[15] Edman A., *The design and implementation of a global logic unit in the VIP SIMD architecture,* Technical report to be published, Department of Physics and Measurements, Linköping, 1995.

[16] Eldén L., Svensson G., *Matrix computations on an SIMD Parallel Computer,* Report LiTH-MAT-R-1990-19,Linköping 1990.

[17] Eldén L., Wittmeyer-Koch L., *Numerical analysis -an introduction (in swedish),* Studentlitteratur, Sweden, 1987.

[18] Elliot D.G., Snelgrove W.M., Stumm M., *Computational RAM: A Memory-SIMD Hybrid and its Application to DSP,* Proc of the IEEE Custom Integrated Circuits Conference, Boston, 1992.

[19] Elliot D., Snelgrove M., Cojocaru C., Stumm M., *A PetaOp/s is Currently Feasible by Computing in RAM.,* Proc of 5th Symposium of the frontiers of massively parallel computation, Washington DC, 1995.

[20] Forchheimer R., Chen K., Svensson C., Ödmark A., *Single-Chip Image Sensor With a Digital Processor Array,* Journal of VLSI Signal Processing 5, pp 121-131, 1993.

[21] Forchheimer R., Ödmark A., *Single Chip linear array processor,* Applications of digital image

processing, SPIE, Vol 397, 1983.

[22] Fountain T.J., *Processor Arrays Architecture and Applications,* Academic Press, 1987.

[23] Frantzeskakis E.N., Liu K.J., *A Class of Square Root and Division Free Algorithms and Architectures of QRD-Based Adaptive Signal Processing,* IEEE Trans. on signal processing, Vol 42, No 9, 1994.

[24] Fujita Y., Yamashita N., Okazaki S., *Image Functional Memory: Architecture and Performance,* Proc. of 1991 Workshop on Computer Architectures for Machine Perception, Paris, France 1991.

[25] Fujita Y., Yamashita N., Okazaki S., *A Real Time Vision System Using Integrated Memory Array Processor Prototype LSI,* Proc. of MVA '92 IAPR Workshop on Machine Vision Applications, Tokyo 1992.

[26] Fujita Y., Yamashita N., Okazaki S., *A real-time vision system using an integrated memory array processor prototype,* Machine Vision and Applications, Vol 7, Number 4, 1994.

[27] Fujita Y., Yamashita N., Okazaki S., *A 64 Parallel Integrated Memory Array Processor and a 30 GIPS Real-Time Vision System,* Proc. of CAMP '95, Como, Italy, 1995.

[28] Fujita Y., *Oral communication,* Como, Italy, 1995.

[29] *General Arithmetic Parallel Processor,* Data Sheet NCR45CG72, NCR Corporation, 1984.

[30] Golub G.H., Van Loan C.F., *Matrix Computations,* North Oxford Academic, Great Britain, 1986.

[31] Gupta A., Kumar V., *The scalability of FFT on parallel computers,* IEEE Trans. on Parallel and Distributed systems, 4, 1993.

[32] Gustavsson M., Mojtahedzadeh S., *MCM Technologies for a Massively Parallel Processor Array,* Masters Thesis, CCA-9512, Halmstad, 1995.

[33] Hall M., *Contributions to the VIP-project,* Report No. LiTH-ISY-R-1692, Linköping 1994.

[34] Hall M., *An implementation of a low pulse repetition frequency Doppler radar algorithm on the massively parallel computer MasPar,* Report No. LiTH-ISY-R-1780, Linköping, 1995.

[35] Hall M., *A study of Parallel Image Processing on the VIP Architecture,* Lic Thesis, LiTH-ISY-LIC-1995:41, Linköping, september 1995.

[36] Hall M., Åström A., *High speed wood inspection using a parallel VLSI architecture,* Proc of 3rd International Workshop on Algorithms and Parallel VLSI Architectures, Leuven, Belgium, Aug. 1994.

[37] Hockney R.W., Jesshope C.R., *Parallel Computers,* Adam Hilger Ltd., 1981.

[38] Hussain Z., *Digital Image Processing, practical applications of parallel processing techniques,* Ellis Horwood, 1991.

[39] Hwang K., *Computer Arithmetic Principles, Architecture, and Design,* John Wiley & sons, 1979.

[40] Hwang K., Briggs F. A., *Computer Architecture and Parallel Processing,* McGraw-Hill, 1984.

[41] Ingelhag P., Jonsson B., Sikström B., Wanhammar L., *A High-Speed Bit-Serial Processing Element,* Proc. of ECCTD '89, Brighton 1989.

[42] Ingelhag P., Svensson C., *A study of the RADAR-VIP implementation,* (In swedish), Linköping 1992.

[43] Ingelhag P., Åström A., Ehlersson T., *Radar Signal Processing Using A 512-Processor Array Chip,* Proc. of the International Conference on Digital Signal Processing, Florens 1991.

[44] Johannesson M., *FFTs on a linear SIMD array,* Proc. of EURO-PAR '95, LNCS 966, Springer

Verlag, 1995.

[45] Johannesson M., Jonasson K., Åström A., *PASIC/VIP System Design and Evaluation.* Report No. LiTH-ISY-I-1332, Linköping 1991.

[46] Johannesson M., Åström A., Edman A., *FVIP An implementation study of an MPD radar algorithm*, Report LiTH-ISY-R-1698, Linköping 1994.

[47] Johannesson M., Åström A., Ingelhag P., *RVIP The Final Report*, Report No. LiTH-ISY-R-1595, Linköping 1992.

[48] Johannesson M., Åström A., Ingelhag P., *The RVIP Image Processor Array,* Proc of CAMP '93, New Orleans, 1993.

[49] Jonker P.P., *Morphological Image Processing: Architecture and VLSI Design,* Kluwer Technische Boeken B.V., 1992.

[50] Kamin R.A., Adams, G.B., *Fast Fourier Transform Algorithm Design and Tradeoffs on the CM-2,* Proc of Conf. on Scientific Applications of the Connection Machine, Ed. by Simon H.D., World Scientific 1989.

[51] Klemm R., *Suboptimum clutter suppression for airborne phased array radars,* Proc of RADAR '82.

[52] Klemm R., *Adaptive clutter suppression for airborne phased array radars,* IEE PROC, Vol 130, Feb 1983.

[53] Lawson H. W., *Parallel Processing in Industrial Real-time Applications,* Prentice Hall, 1992.

[54] Linde A., Taveniku M., Vasell J., *Teknisk granskning av radarsignalbehandligns-processorn APN 502 01 (In swedish)*, Teknisk rapport no. 174, Gothenburg, 1993.

[55] Miyaguchi H., Krasawa H., Watanabe S., *Digital TV with Serial Video Processor,* Proc. of the 9th IEEE International Conference on Consumer Electronics, Illinois, USA 1990.

[56] Munthe-Kaas H., *Superparallel FFTS*, Siam J. Scientific Computing, Vol 14, No 2, pp. 349-367, March 1993.

[57] Nordhamn E., *Design of an Application Specific FFT Processor,* Report No. LiU-Tek-Lic-1992:16, Linköping, 1992.

[58] Oppenheim A.V.,Schafer R.W., *Digital Signal Processing,* Prentice-Hall International Editions, 1975.

[59] Owens R.M. Irwin M. J., Nagendra C., Bajwa R. S., *Computer Vision on the MGAP,* Proc of CAMP'93, New Orleans, 1993.

[60] Sköld S., *Implementation of Fast Fourier Transforms on a Transputer Network,* Masters thesis LiTH-ISY-EX-1171, Linköping, october 1992.

[61] Stimson G.W., *Introduction to airborne radar*, Hughes Aircraft company, El Segundo, California, 1983.

[62] Svensson B., *LUCAS Processor array - Design and Applications,* Ph.D. thesis, Department of Computer Engineering, University of Lund, 1983.

[63] Svensson C., Yuan J., *High Speed CMOS Chip to Chip Communication Circuit,* Proc. of ISCAS '91, vol 4, june 1991.

[64] Swartzlander E., *Systolic FFT Processors,* Systolic Arrays, Ed by Moore W., et al. Adam Hilger, 1987.

[65] Yamashita M. Kimura T., Fujita Y., Aimoto Y., Manabe T., Okazaki S., Nakamura K., Yamashina

M., *A 3.84 GIPS Integrated Memory Array Processor with 64 Processing Elements and a 2-Mb SRAM*, IEEE Journal of solid state circuits, Vol 29, No 11, 1994.

[66] *The LC3100 System,* Overhead copies, Litton Inc., California, 1995.

[67] Tucker L.W. Robertson G.G. *Architecture and Applications of the Connection Machine,* IEEE Computer, 1988.

[68] Wilson S., *One dimensional SIMD architectures - the AIS-5000, Multicomputer Vision,* S. Levialdi, Ed., Academic Press, London.

[69] Åhlander A., Svensson B., *Floating-Point Calculations in Bit-Serial SIMD Computers,* Proc of DSA -92: 4th Swedish Workshop on Computer System Architecture, Linköping, 1992.

[70] Åhlander A., Linde A., Lyckegård B., Taveniku M., Svensson B., *Computer Architectures for Radar Systems,* Proc of DSA -95: 6th Swedish Workshop on Computer System Architecture, Stockholm, 1995.

[71] Åhlander A., E-mail communication, october 1995.

[72] Åström A., *A Smart Image Sensor. Description and evaluation of PASIC.* LiU-TEK-LIC-1990:57, Linköping 1990.

[73] Åström A., *Smart Image Sensors*, Ph.D. thesis No. 319, Linköping 1993.

[74] Åström A., Danielsson P-E., Chen K., Ingelhag P., Svensson S., *Videorate signal processing with PASIC and VIP.* Proc. of Barnaimage '91, Barcelona, Spain, September 1991.

[75] Åström A., Isaksson F., *Real-time Geometric Distorsion Correction and Image Processing on the 1D SIMD architecture IVIP,* Proc of MVA 1994, Kawasaki 1994.

[76] Åström A., Johannesson M., Edman A., Ehlersson T., Näsström U., Lyckegård B., *An Implementation Study of Airborne Medium PRF Doppler Radar Signal Processing on a Massively Parallel SIMD processor architecture,* Proc of RADAR '95, Washington 1995.

Ph.D. Dissertations
Image Processing Laboratory, Department of Electrical Engineering
Linköping University, Sweden

Björn Kruse, *Design and implementation of a picture processor.*
Linköping Studies in Science and Technology. Dissertation No. 13, 1977.

Rao Kameswara, *Pattern recognition techniques applied to fingerprints.*
Linköping Studies in Science and Technology. Dissertation No. 19, 1977.

Björn Gudmundsson, *Contributions to the design of a picture processing system.*
Linköping Studies in Science and Technology. Dissertation No. 42, 1979.

Roger Cederberg, *On coding, processing and display of binary images.*
Linköping Studies in Science and Technology. Dissertation No. 57, 1981.

Olov Fahlander, *Development towards animation in computer graphics.*
Linköping Studies in Science and Technology. Dissertation No. 78, 1982.

Reiner Lenz, *Reconstruction, processing and display of 3D-images.*
Linköping Studies in Science and Technology. Dissertation No. 151, 1986.

Björn Lindskog, *An SIMD architecture for multi-dimensional signal processing.*
Linköping Studies in Science and Technology. Dissertation No. 176, 1988.

Qin-Zhong Ye, *Contributions to the development of machine vision algorithms.*
Linköping Studies in Science and Technology. Dissertation No. 201, 1989.

Olle Seger, *Model building and restoration with applications in confocal microscopy.*
Linköping Studies in Science and Technology. Dissertation No. 301, 1993.

Ingemar Ragnemalm, *The Euclidean distance transform.*
Linköping Studies in Science and Technology. Dissertation No. 304, 1993.

Anders Åström, *Smart Image Sensors.*
Linköping Studies in Science and Technology. Dissertation No. 319, 1993.

Maria Magnusson, *Linogram and other direct Fourier methods for tomographic reconstruction.*
Linköping Studies in Science and Technology. Dissertation No. 320, 1993.

Mats Österberg, *Quality functions for parallel selective principal component analysis.*
Linköping Studies in Science and Technology. Dissertation No. 345, 1994.

Mats Gökstorp, *Depth Computation in Robot Vision,*
Linköping Studies in Science and Technology. Dissertation No. 378, 1995.

ISBN 91-7871-609-8

ISSN 0345-7524