Design of a normalized grey value correlator

Kessels, L.G.M.

Award date:
1996

Design of a normalized grey value correlator

Philips Industrial Vision

L.G.M. KESSELS

CTR554-96-0038 1996-06-11

Philips
Centre for
Manufacturing Technology

PHILIPS

Published by: **Nederlandse Philips Bedrijven B.V.**
Philips Centre for Manufacturing Technology
Eindhoven, The Netherlands

Title: Design of a normalized grey value correlator
Author: L.G.M. Kessels
Document Code: CTR554-96-0038
Date: 1996-06-11
Distribution: Internal Philips

If you have any comment to make, please feel free to contact the author:

Philips CFT
Philips Industrial Vision
L.G.M. Kessels
PO Box 218 / SAN3235
5600 MD EINDHOVEN Tel.: (+31 40 27)37872
The Netherlands Fax: (+31 40 27)37337

Copies of this document can be ordered from the distributor:

Philips CFT
Management Services
PO Box 218 / SAQp119
5600 MD EINDHOVEN Tel.: (+31 40 27)34184
The Netherlands Fax: (+31 40 27)32250

Abstract

Philips Industrial Vision (PIV), a department within Philips' Centre for manufacturing Technology (CFT), operates in the area of industrial image processing. This department uses its Single Board Image Processor (SBIP) in various industrial image processing applications like identification-, inspection- and position measurement- systems. The normalized correlation can be used for position measurements of arbitrary objects, independent of contrast variations in images. However, it is very computation intensive: even though the usage of subsample techniques, the normalized correlation is rather slow on the SBIP. Therefor PIV needed an extension board with a digital signal processor that accelerates computation intensive algorithms like the normalized grey value correlation. Since the correlator uses subsample techniques, an investigation was carried out on the effects of subsampling and pre-filtering on the correlator performance.

The design of the extension board resulted in a small image processing module that can operate in parallel with the SBIP. The board offers the possibility to transfer images to and from its image memory at transfer rates up to 20 Mhz. The extension board, based on Analog Devices' ADSP 21062 DSP, improves the processing performance of the SBIP due to the high instruction rate of the DSP, the zero-waitstated image memory and the DSP's calculation capabilities.

The implementation of the normalized grey value correlation on the extension board, reduces the execution time for a search of a 200 x 200 template in a 512 x 512 window from 15 seconds on an SBIP, to 125 ms on the extension board.

The architecture of the extension board allows porting code from the SBIP fairly easy, thus vision functions that are currently running on the SBIP can be implemented on the extension board. Especially vision functions that perform intensive calculations and/or intensive memory usage, are expected to run significantly faster on the extension board. However, here is pointed out that in certain algorithms, assembly language programming might be required to optimally use the DSP and achieve a high performance improvement.

In the analysis of the subsample effects and pre-filtering, where the image was assumed to be free of noise and free of distortions, was shown that a filter can be designed that improves the correlator performance after this filter is applied to the template.

This analysis is to be completed in future by analysing the optimization of the filter. Furthermore, the analysis assumed rather ideal conditions, so it is to be extended in future by taking noise and possibly distortions of the object in the image also into account during the analysis.

Preface

This report was written for my graduation project at the department of Electrical Engineering at the Eindhoven University of Technology. Carrying out this graduation project was the final assignment to complete my study at the university. As i was also working as an electrical engineer with Philips, i tried to combine the graduation project with my daily work at Philips. This implied shifting to the department Philips Industrial Vision, where they offered me this graduation assignment.

At this point i would like to take the opportunity to thank everyone who supported me in one or another way during the graduation work in the past year.
Special thanks go to Ing. L. Leistra, who offered me the opportunity to perform my graduation work in his department, and Prof. Dr. Ir. W.M.G. van Bokhoven for his permission. Furthermore to my coach from the university Dr. Ir. P.C.W. Sommen, the coaches at Philips Dr. M. Brok and Ir. J.A.C. Bernsen. And last but not least, Ing. E.F.J. Claessens for coaching me during the hardware design.

Contents

APPENDIXES

List of abbreviations

ADC	Analog to Digital Convertor
ALU	Arithmetic and Logic Unit
CFT	Centre for Manufacturing Technology (Dutch: Centrum voor Fabricage Technieken)
CPU	Central Processing Unit
DAC	Digital to Analog Convertor
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
DRAM	Dynamic Random Access Memory
DPRAM	Dual Ported Random Access Memory
EPLD	Erasable Programmable Logic Device
EPROM	Erasable Programmable Read Only Memory
FIFO	First In First Out memory
FLOPS	Floating Point Operations Per Second
GSP	Graphics System Processor
ID	Identifier
IDFT	Inverse Discrete Fourier transform
IOP	Input Output Processor
kb	kilo bytes
LUT	Look Up Table
MAC	Multiply ACcumulate
Mb	Mega bytes
MFLOPS	Million Floating Point Operations Per Second
MIPS	Million Instructions Per Second
ms	milli seconds
ns	nano seconds
PCB	Printed Circuit Board
PDF	Probability Density Function
PIV	Philips Industrial Vision
PLD	Programmable Logic Device
RAM	Random Access Memory
ROC	Receiver Operating Characteristics
SBIP	Single Board Image Processor
SRAM	Static Random Access Memory
us	micro seconds
VRAM	Video Random Access Memory

1 Introduction

Philips Industrial Vision (PIV) is a department within Philips' Centre for manufacturing Technology (CFT), that operates in the area of industrial image processing. It offers complete image processing systems tailored to demands of their customer's machine vision problems. These systems include specific application software, optics, dedicated hardware and also standard products like image processing modules with software libraries.

One of these standard products, the Single Board Image Processor (SBIP), is an image processor module that is used in many industrial image processing applications like identification, inspection and position measurement systems. As these applications tend to demand higher and higher computational performance from image processors, the execution time of the SBIP can be a bottleneck in certain applications.

One application requiring such a high computational performance is the normalized grey value correlation, an image processing algorithm that uses correlation techniques and a systematic search method to find a detail (called a template) in an image. The execution time of this application on the SBIP is too long and should be decreased drastically to meet the customers demands. The processor of the SBIP, dedicated to high performance graphical applications, is rather slow in mathematical computations. To achieve a better mathematical performance with the SBIP, PIV needs an extension board for the SBIP that will accelerate algorithms like the normalized grey value correlation.

The object is to design an extension board, that will perform the grey value correlation such that the execution time meets the demands of the customer. The extension board should not be dedicated to the grey value correlation only, and shall therefor be equipped with a digital signal processor (DSP). In that way the extension board can also be used to accelerate image processing applications that are currently being run on the SBIP. The extension board has to interface with the SBIP through the standard extension connector and the manufacturing price can be about Dfl. 1500,-.

The grey value correlators search algorithm uses subsample techniques to keep the execution time as short as possible. This implies that the usability of the grey value correlator is restricted to templates that don't contain too many fine details. Since there is no knowledge on the correlators performance for certain templates and whether pre-filtering the template and image will improve the correlators performance, the effects of subsampling and pre-filtering on the correlators performance are to be investigated.

To make the reader more familiar with the subject of the grey value correlation and the SBIP, the subsequent sections will discuss the following subjects in more detail: Section 1.1 will give general information about the SBIP and it's operation, section 1.2 describes the grey value correlation. After the expected performance of

the extension board is covered in section 1.3, section 1.4 explains the method of approach of the project.

1.1 THE SINGLE BOARD IMAGE PROCESSOR.

The most widely used product of PIV is the SBIP with it's vision library. The SBIP includes all the required hardware to perform image processing applications. From four camera inputs, images can be digitized and stored in the SBIP's memory. There they can be processed by the Graphics System Processor (GSP) or by dedicated image processing hardware. The SBIP can be controlled by a host via a VME or PC interface or operate in stand alone mode.
The vision library contains many useful image processing algorithms that run on the GSP.

Figure 1.1 shows the block diagram of the SBIP.

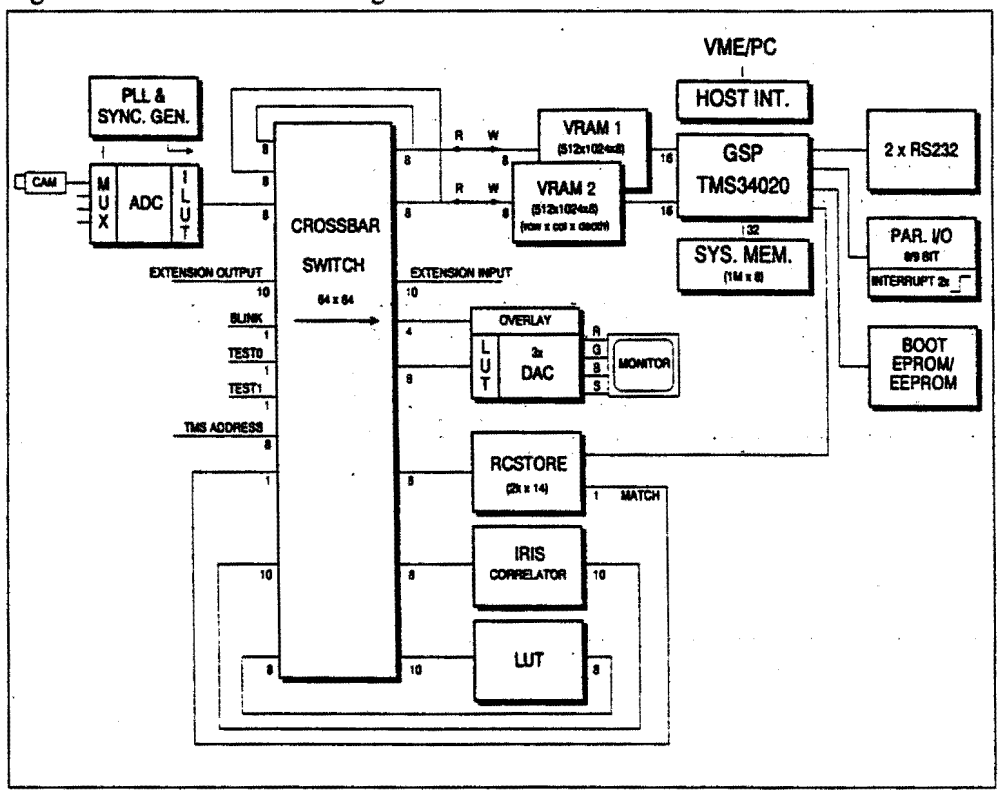


Figure 1.1 : Block diagram of the Single Board Image Processor.

The SBIP is built up of the following blocks:

a. A crossbar switch.
b. An Analog to Digital Convertor (ADC) with multiplexer and input look up table.
c. Two Video Random Access Memories (VRAM1 and VRAM2).
d. The GSP with system memory and host interface.
e. A Digital to Analog Convertor (DAC).
f. An Image Recognition Integrated System (IRIS).
g. A look up table.
h. An extension connector.

The crossbar switch provides a flexible way to create any video path between the devices connected to it. It is in fact a device with 64 inputs and 64 outputs, it can be programmed (through software) to make connections between any input and output, or from an input to multiple outputs. In this way, the user is free to route any path from eg. the ADC to both the DAC and video memory, or from the ADC via an extension board to video memory 1 and a connection from video memory 2 to the DAC etc.

The ADC digitizes the video signal that comes from one of maximum 4 cameras, selected by a multiplexer. A commonly used video standard is the CCIR (european) or RS170 (american) standard. The ADC converts the analogue video signal into 8 bit grey levels at sample rates up 20 MHz. The samples are translated by an Input Look-Up Table (ILUT) and are then fed to the video memory. The ILUT performs actions like inverting the image, enhancing dark levels, etc. This operation can make the image processing task for the graphics processor in certain cases a lot easier.

The image being digitized is stored into one of the two video memories. One video memory can store an image as large as 512 x 1024 pixels (8 bit grey values). It is built up of Video RAM's, which are dual ported Dynamic Random Access Memories. These VRAM's have besides the parallel random access port, also a serial port through which data samples can be shifted into a line buffer. (note that the data stream is 8 bits wide, thus 8 bits are shifted into the line buffer per clock cycle). Loading the data stream into the line buffer only requires a clock pulse synchronised with it. When one video line is written into the line buffer, the buffer contents can then be stored into one of the rows of the memory array by generating the row/column address via the row/column pins.
In the same manner, it is also possible to shift out a data stream from the memory.

The Graphics System Processor, a TMS 34020, is the heart of the SBIP. It has all the required logic to control the video memories and generate all the necessary synchronisation signals to take in an image from the camera. It is a bit cell oriented processor, thus one single address contains one bit of data. Physically, the data word size is 16 or 32 bits wide (depending on the processor generation). The GSP can access the images in VRAMS via their parallel port and has also access to system memory where program's, data or images are stored.

A host (VME or PC based) can manipulate all memories on the SBIP through the host interface. In that way software can be downloaded and the host can communicate with the TMS 34020. The host interface does not have to be used necessarily, the application software can be programmed in EPROM from which the SBIP can boot.

The presence of the DAC makes it possible to show images and application information on a monitor. It is connected to the crossbar, in that way it is possible to show live video from the camera, or images stored in the video memories.

The Image Recognition Integrated System (IRIS) is a real time (binary) template matcher. By transferring an image from eg. the ADC to this IRIS, the IRIS can search predefined binary templates in the video stream. The coordinates where the

template matches the image, will be recorded in the Row/Column-store (R/C-store), and can be read from there by the GSP.

Finally is noted that there exist in fact 2 generations of the SBIP, they are called SBIP1 and SBIP2. The DSP extension board is to be designed for using it with the SBIP2.

1.2 THE NORMALIZED GREY VALUE CORRELATION

A common machine vision problem is to locate and report the position of a predefined object in an image that was taken from a camera. As stated in [1], the normalized correlation can be used to locate arbitrary objects, independent of contrast variations in images. This section will describe the normalized grey value correlation and it's properties.

Figure 1.2 shows the template, the image and a detail of the image with the same size as the template. The template specifies the object to be found in the camera image. The normalized correlation coefficient is calculated between the template and the detail of the image to find out whether the template matches the particular image detail. By repeating this with various details from the image and keeping the coordinates where the correlation was maximum (and exceeds a certain threshold), the template can be located in the image. The normalized grey value correlation will thus report the position where the template and image match the best.

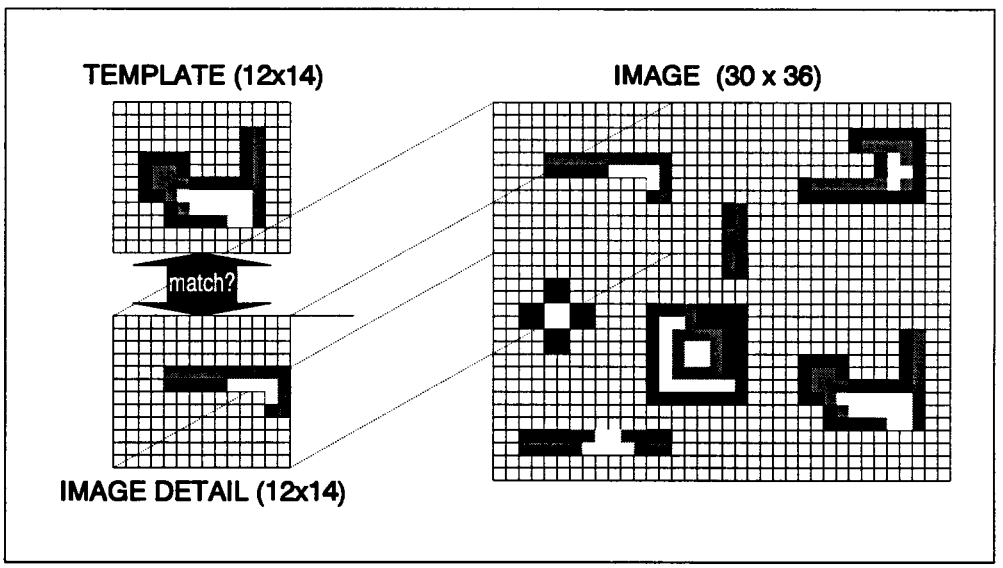


Figure 1.2 : Grey value correlation.

The grey value correlation algorithm actually keeps calculating normalized correlation coefficients with details from various positions in the image. Such a normalized correlation coefficient ranges from -1 to 1: a coefficient of 1 represents a perfect match, a coefficient of -1 represents a perfect mismatch (which means that the grey values of the image detail and the template are each others inverse).

Formula (1.1) shows how the normalized correlation coefficient can be calculated between a template *tpl* and an image detail $img_{u,v}$ (with the same size of template t) with it's upper left corner at position u,v within the image.

$$\rho(img_{u,v},tpl)=\frac{cov(img_{u,v},tpl)}{\sigma_{img_{u,v}}\cdot\sigma_{tpl}} \qquad (1.1)$$

with: cov(.) covariance
$img_{u,v}$,*tpl* (continuous) 2-dimensional light intensity functions of the image detail and the template.

The formula is built up of three terms: one numerator - and 2 denominator - terms:

The numerator term represents the covariance of the image detail and template. From this covariance only it is not possible to tell how well the image and template match, because it depends on the power contents of the image- and template- information.

The two denominator terms, representing the square root of the power contents (σ_{img}^2 and σ_{tpl}^2) of the image and template, normalise the covariance result to a value ranging from -1 to 1.

If the template *tpl* and the image detail $img_{u,v}$ from the image are uncorrelated, the covariance term equals 0, and hence, the result is 0. If they are strongly correlated, the numerator term approaches the denominator term and the result will approach 1.
If the two are inversely correlated (image detail and template are inverse video) the result will be -1.

The σ values in the denominator of (1.1) can also be expressed in (auto) covariance terms which yields the following expression for the correlation coefficient:

$$\rho(img_{i,j},tpl)=\frac{cov(img_{i,j},tpl)}{\sqrt{cov(img_{i,j},img_{i,j})\cdot cov(tpl,tpl)}} \qquad (1.2)$$

In digital image processing, only the sampled versions (spatially and in amplitude) of the template and the image are available. The samples of the image and template intensity functions are referred to as pixels in this report.
Since only the sampled versions of the image and the template are available the correlation coefficient can only be estimated.
According to [2] p.44 :

$$cov(x,y)=E\{x\cdot y\}-E\{x\}\cdot E\{y\} \qquad (1.3)$$

with E{.} the mathematical expectation operator.

Furthermore, if N samples are taken from a certain variable p, the expected value of p can be estimated from the N samples p_i ($0 \le i < N$) as follows:

$$E\{p\} \triangleq \frac{1}{N}\sum_{i=0}^{N-1} p_i \tag{1.4}$$

The correlation coefficient can then in fact be estimated from the three covariances in (1.2). The covariance between the image detail and template can be estimated as follows:

$$c(I_{i,j},T)=\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\cdot T_{r,c} - \left(\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\right)\cdot\left(\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\right) \quad \triangleq cov(I_{i,j},T) \tag{1.5}$$

$T_{r,c}$ represents the grey values of the template pixel at row r, column c.
$I_{i+r,j+c}$ represents the grey value of the image pixel from an image detail taken from row i, column j, at an offset r,c within the image detail.
R The number of pixel-rows in the template.
C The number of pixel-columns in the template.
N represents the total number of pixels in the template ($N = R \times C$).

The auto-covariance of the template can be estimated as:

$$c(T,T)=\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}^2 - \left(\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\right)^2 \quad \triangleq cov(T,T) \tag{1.6}$$

and the auto-covariance of the image detail can be estimated by:

$$c(I_{i,j},I_{i,j})=\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}^2 - \left(\frac{1}{N}\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\right)^2 \quad \triangleq cov(I_{i,j},I_{i,j}) \tag{1.7}$$

The correlation coefficient for a 2 dimensional image and template can then be estimated as follows:

$$r(I_{i,j},T)=\frac{N\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\cdot T_{r,c} - \left(\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\right)\cdot\left(\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\right)}{\sqrt{\left[N\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}^{\,2} - \left(\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}\right)^2\right]\cdot\left[N\cdot\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}^{\,2} - \left(\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\right)^2\right]}} \tag{1.8}$$

So far is only shown how to calculate whether the template and an image detail match. The object, however, is to find the position (row and column) of the image detail that matches the template. Generally one has a search window, a rectangular area within the image, in which one wants to find the template.
To find the position of the template, the correlation coefficient between the template and various image details throughout the search window are calculated. The position where the correlation coefficient is maximum will be at that position where the image and template match the best.

The exhaustive search is a possible solution to find this desired position. The principle of this algorithm is to place the template at the upper left corner of the search window, slide it over the search window and calculate the correlation coefficient between the template and the particular image detail every time the template is shifted by one pixel. The position were the template and the image detail match, is found by keeping the position where the correlation coefficient was maximum and exceeds a certain threshold. This method is known as the exhaustive search.

It will probably be clear that an exhaustive search requires quite some calculations. The amount of calculations for an exhaustive search with a template of R x C pixels in a search window of I rows x J columns will be derived by first analysing the required calculations for computing one correlation coefficient. Then, the total amount of calculations will be found by multiplying this with the number of correlation coefficients that is to be calculated.

Calculating one correlation coefficient as in (1.8) at an offset i,j within the search window requires basically 5 summation loops to be done:

- 1 (2-dimensional) summation loop to calculate the template-image cross correlation properties:

$$x_{i,j}=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\cdot I_{i+r,j+c} \tag{1.9}$$

- 2 (2-dimensional) summation loops to calculate the following image properties:

$$\begin{aligned} y_{i,j}&=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}^2 \\ z_{i,j}&=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c} \end{aligned} \tag{1.10}$$

- 2 (2-dimensional) summation loops to calculate the following template properties:

$$p=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}T_{r,c}^2$$
$$q=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}T_{r,c} \qquad (1.11)$$

After completing these calculations, the estimated correlation coefficient between image detail $I_{i,j}$ and template T is calculated as follows:

$$r(I_{i,j},T)=\frac{N\cdot x_{i,j}-z_{i,j}\cdot q}{\sqrt{\left[N\cdot y_{i,j}-z_{i,j}^2\right]\left[N\cdot p-q^2\right]}} \qquad (1.12)$$

For the exhaustive search the terms p and q only have to be calculated once while the terms $x_{i,j}$,$y_{i,j}$ and $z_{i,j}$ are to be recalculated for every correlation coefficient.
The calculations of terms $y_{i,j}$ and $z_{i,j}$ can be reduced if an exhaustive search is done, since the results from the adjacent correlation coefficient calculation can be used for the current one (this will be explained in more detail in section 2.3). For large templates, the amount of calculations that have to be done can then be approximated by the required amount of calculations for calculating the term $x_{i,j}$.

If the template with size R x C (rows x columns) is to be searched in a window of I x J (rows x columns), this requires calculating (I-R+1) x (J-C+1) correlation coefficients. (Note that the search algorithm is limited to that area where the entire template is within the search window).If the calculation of one correlation coefficient is approximated by R x C Multiply ACcumulates (MAC's), the number of MAC's to perform an exhaustive search can then be approximated by:

$$(I-R+1)\cdot(J-C+1)\cdot R\cdot C \qquad (1.13)$$

The amount of MAC's can thus be approximated by the number of pixels in the template, multiplied by the number of correlation coefficients that is to be calculated. For a search of a 200 x 200 template in a 512 x 512 search window this equals 200^2 x 313^2 and is approximately 4 * 10^9 multiplications for only calculating the term summation loops. (besides calculating these summation loops, also the 313^2 correlation coefficients are to be calculated from the results of the summation loops according to formula (1.12)).

It will be clear that this amount of multiplications is much too high to implement the algorithm like this on a single DSP, because the execution time won't even be close to the desired 200 msec. The calculations for the exhaustive search can not be optimised. Term $x_{i,j}$ represents the fact that each pixel of the template has to be multiplied by each pixel of the image. It can therefor not be optimised by

eliminating multiple identical operations, each multiplication in term $x_{i,j}$ is unique! Optimisation can thus only be made if the amount of data can be reduced.

There are different ways to reduce the amount of data that is to be processed:

a. It is possible to perform a search with only a detail from the template. The amount of data is then reduced in the sense that the search is done with a very small template. If this small template correlates strongly at a certain position in the image, one could expand to a larger detail of the template or to the whole template and see whether that position is the searched position.

b. Another approach that can be used is to subsample the image and template and thus reduce the amount of data, since both the image and template sizes are reduced.

c. One can also choose an alternative search method, eg. shifting the template by 2 pixels instead of 1, every time a correlation coefficient is calculated. This will reduce the amount of calculations a factor 2^2. The fact whether this is allowed depends on the (spatial) auto correlation function of template.

d. Another alternative search method, is to use a gradient based search algorithm that searches for the maximum correlation coefficient in the direction where the correlation coefficient rises the most.

e. Combinations of the methods mentioned above.

Some work has been done already in the past, which resulted in an algorithm that combines options *b* and *c*. This algorithm will be discussed in more detail in section 2.3.

This chapter is concluded by naming the properties of the normalized grey value correlation:

a. The normalized grey value correlation is not sensitive to offsets in grey values and differences in contrast between image and template pixels. Thus the result of the normalized grey value correlation does not depend on overall illumination variations. (Note that local illumination differences within the image detail that is used for calculating the correlation coefficient, do influence the result).

b. The normalized grey value correlation can not be used if objects in the image may have angular offsets. The allowed angular offset depends on geometry of the object. In general round objects will not be sensitive to angular offsets, while long narrow objects on the other hand, are very sensitive.

c. The normalized grey value correlation is also sensitive to variations in the size of the object in the image. If this object is larger or smaller than the object in the template, the grey value correlation might fail.

The influence of the two last properties, the allowed angular offset and scale factor also depends a great deal on the fact how much the template correlates throughout the rest of the image. If the template is completely uncorrelated throughout the rest of the image, then the algorithm will allow larger angular offsets or scale factors than when the template correlates more with details throughout the image. (note that even though the algorithm won't fail in finding the object, the angular offsets and scale factors might influence the accuracy of the reported position).

1.3 EXTENSION BOARD SPECIFICATIONS

Philips Industrial vision requires the following performance of the DSP extension board:

a. The extension board should be usable with the SBIP without making hardware modifications to the SBIP.

b. The extension board is to be equipped with a DSP, the selection of the DSP is part of the project

c. The extension board has to be able to perform general image processing applications that are currently done on the SBIP.

d. The computation performance of the extension board should be such that a normalized grey value correlation of a 200 x 200 template in a 512 x 512 image with the existing algorithm will be performed within 100 to 200 ms. Transferring the image from the SBIP to the extension board may not consume additional time.
The maximum template size for a grey value correlation is 64k pixels.

e. Since PIV intends to implement more algorithms on the extension board in future, the hardware design should be such that it does not require complex software algorithms to implement a certain application on the board.

f. The architecture of the extension board should be such that application software can be downloaded from the SBIP or preferably from the SBIP's host (via the host interface of the SBIP).

g. The price of the extension board is allowed to be as high as Dfl 1500,-

1.4 METHOD OF APPROACH

The project consists of the realisation of the extension board and of the investigation of the effects of subsampling and pre-filtering on the correlator performance.

The realisation of the prototype of the grey value correlator was mainly divided in three phases: an investigation-, a design- and a test & debug-phase.

The object of the design investigation phase is to straighten out more accurately which demands there are on the extension board, to define a global architecture and to select a DSP and a memory type that is to be used on the extension board. The demands with respect to the hardware interface to the SBIP, the performance of the grey value correlation, and the price of the extension board were used to define a global architecture and select a DSP. To make a proper selection for a DSP and memory type, also some time is spent on gathering more knowledge on properties of DSP's and memories. The design investigation will be covered in chapter 3.

During the design phase the global hardware architecture is to be worked out into functional blocks for the extension board, and the software is to be developed. The hardware design includes choosing the correct parts and analysing their operation and requirements. The hardware design shall be verified with simulations to verify that all timing requirements of the used parts are met. After simulations have shown that the design will be functioning correctly, a Printed Circuit Board (PCB) layout and some prototypes will be made to perform hard- and soft-ware tests afterwards.

In the software design phase, the required software for the extension board is to be developed. The DSP extension board requires system- and application software. The system software manages the operation of the DSP extension board, this includes booting the DSP and implementation of the host interface and interrupt handlers, if required.
The application software exists of the available C-source for the grey value correlator. It is eventually to be altered for optimal usage of the DSP.
The hard- and soft-ware design will be explained in chapter 4.

During the test and debug phase should be verified that all the work of the previous months meets the expectations and requirements. These tests are to be done with the prototype boards. The tests and verifications include hard- and software operation, and verification whether the correlator algorithm is working correctly and meets the timing requirements. Chapter 5 shows the test results.

To investigate the effect of subsampling and pre-filtering on the correlator performance, first a literature search was done to find out whether the subject had been investigated already in the past. Since this was not the case, a start was made with the investigation. Section 6 covers the investigation of the effects of subsampling and pre-filtering on the correlator performance.

Finally, chapter 7 concludes this report with conclusions and subjects for future research.

2 System requirements

A proper basis to start the project is to analyze the requirements from the requested performance carefully. The specifications in section 1.3 mentioned various requirements with respect to the extension board.
The investigation stage of the project has been used to analyze these requirements. The following information was found to be useful for selecting a DSP and defining a global architecture for the extension board:

a. The interface to the SBIP.
Knowledge on this interface will help to determine a global architecture for the hard- and soft-ware design. The SBIP's hardware interface will be described in section 2.1, and the software interface will be discussed in section 2.2.

b. The computation requirements of the normalized grey value correlation.
This will be of importance for selecting a DSP, section 2.3 shows these computation requirements and how they have been analyzed.

c. The selection of a DSP.
Selecting a DSP for the extension board requires knowledge about DSP properties. To get an idea about the possibilities that DSP's offer, some time is spent on analysing characteristics of DSP's that are currently available. Section 2.4 gives a brief description on the properties of DSP's. (Appendix B gives a more detailed explanation about DSP's).

d. Typical vision applications for the extension board.
Since more applications are going to be implemented, this information might be necessary if a DSP is to be selected.Section 2.5 describes these applications.

e. The maximum price for the required parts.
The manufacturing price of the extension board includes various costs. Section 2.6 explains the manufacturing price budget.

2.1 THE HARDWARE INTERFACE

Since the extension board has to communicate through the existing extension connector interface with the SBIP, this interface has been analyzed. Since the extension board is going to be used in combination with the SBIP2, only the SBIP2 interface is taken into account. Figure 2.1 shows the extension connector interface of the SBIP2.

The following signals are available on the extension connector:

a. 10 address lines of the TMS 34020

b. The data bus of the TMS 34020 processor (32 bits wide)

c. TMS 34020 Control signals : read, write and ready (a line to hold the TMS 34020 processor during access of slow memories)

d. 5 outputs of an address decoder of the SBIP, that become active if the TMS 34020 accesses a certain memory area.

e. A video bus consisting of:
10 bits video input
10 bits video output
a pixel clock defining when the samples on the video bus can be read.
a blanking signal, specifying the area of interest.

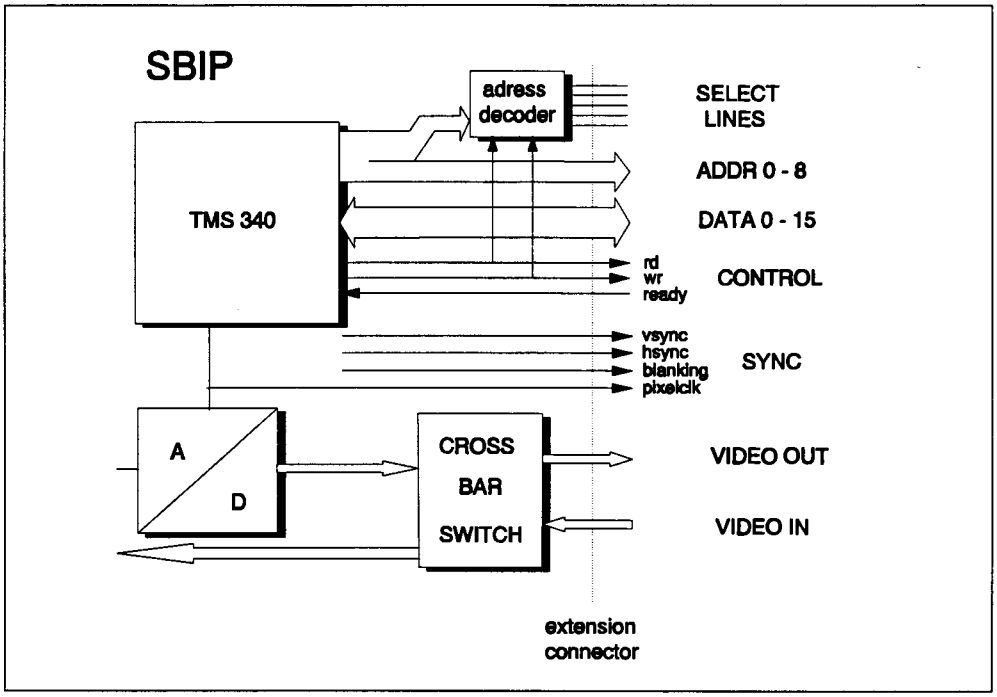


Figure 2.1 : The extension connector interface.

a vertical sync defining the start of a frame.
a horizontal sync defining the start of a video line.

f. Power supply.

The fact that only 10 address lines of the TMS 34020 are available on the interface connector and no modification can be made to the SBIP, implies that a DSP will not be able to address the images that reside in the video memory on the SBIP. However, in combination of the select lines from the address decoder the TMS 34020 is able to address a small memory space of 512 words that can reside on the extension board. Through this small memory space, the TMS and DSP can communicate.

With the presence of the video bus it is possible to transfer images to or from the extension board at video speed. For the video transfers, the pixel clock and the blanking line play an important role, The pixel clock is synchronized with the data stream from the ADC. Every rising clock edge, a data sample is valid on the video input (or the data should be present at the video output).
The data samples taken from the video output during synchronisation pulses contain no image information. Since the pixel clock is continuously present (also during synchronisation pulses), it is accompanied with the blanking signal. This is to be used as an enable line for the pixel clock. If one video line contains 1024 pixels, the blanking line will become inactive during 1024 pixel clock periods.

2.2 SOFTWARE INTERFACE

The implementation of the software interface between the SBIP and the extension board will depend on the hardware interface. The software interface should make the hardware interface invisible to the application programmer. (With an application programmer is meant, a software engineer who uses the SBIP with the vision library to design programs for image processing applications). The software interface between the host computer and the SBIP provides such an interface. To make the usage of the extension board as easy as possible for the application programmer, a similar interface between the extension board and the SBIP is to be designed. This section will give a short description on the implementation of the software interface between the SBIP and it's host computer.

In a configuration of an SBIP with a host computer (PC or VME based CPU) the SBIP operates as a slave of the host computer. The software on the SBIP contains a large library with vision functions with a software interface to the host. Via this software interface, the SBIP receives commands from the host to execute one of the vision functions from the vision library, it then executes this function and returns the results to the host. In this way the SBIP performs the actual image processing and reports results to the host software. The host software performs the actual interpretation of these results. Thus the vision application is in fact partially running on the SBIP and partially on the host.

The implementation of the interface is as follows: The host and the SBIP communicate through a (shared) communication buffer. This buffer consists of an entry for the function identifier (ID) followed by a parameter list. Each function in the vision library has a unique function ID assigned to it. When the host has to invoke a certain vision function on the SBIP, it first writes the parameters to be passed to the function into the communication buffer. Then it writes the appropriate function ID in the function entry. The command handler that runs on the TMS 34020 is continuously polling the function entry, and detects that the host wrote the function ID. It reads in turn the function ID and passes the parameter list from the communication buffer to the appropriate function. After the SBIP executed the function, it writes the return parameters into the communication buffer and sets the function ID in the function entry to 0, to notify the host that the command is finished.

Towards the host software, the software interface provides a transparent interface between that software and the SBIP. The functions in the vision library on the SBIP are also callable on host level with exactly the same syntax. Thus, there is also a host library (that resides on the host computer) that contains exactly the same functions as in the vision library. Only the function implementation on the host just copies the function arguments that are passed to it, to the communication buffer. The command handler on the SBIP passes these parameters that it receives via the communication buffer to the appropriate function. The software interface between the host and the SBIP consists thus in fact of the host library and the command handler on the SBIP.

Since the syntax of the vision functions on the SBIP and the host are exactly the same, it is possible to develop and debug application software for the SBIP on the host computer. Once the application is found to be working satisfactorily on the host, the source code can be compiled with the TMS 34020 cross compiler, and

ran on the TMS 34020 itself. Of course there are restrictions on having the capability of compiling one and the same source for both the host and the TMS 34020 environments. The programmer should use predefined functions and macros for operations on images.
Let's take the example that he likes to get the grey value of a pixel: This requires to load a pointer variable with the corresponding memory location where the grey value of that pixel is stored. Since the image is mapped in the SBIP's memory space, and not in the host's memory space, this pointer implementation will only work in the TMS 34020 environment, not on the host environment.
Therefor, the programmer should use the getpixel() function/macro, rather then manipulating a pointer though the image memory. In the host environment, the getpixel() function delegates the action to the SBIP via the communication buffer, the getpixel()-macro in the TMS 34020 environment is in fact the pointer operation.

The software environment for writing vision applications consists among other things, a number of predefined data structures, macro's and functions.
One of the requirements of the extension board is that the software interface should be very easy, such that other algorithms can be implemented on the extension board without much difficulties. It is therefor decided to adapt as many as possible from the SBIP's software environment to the extension board. This has the advantage that programmers with SBIP experience can implement algorithms easier on the extension board.

2.3 EXECUTION TIME OF THE NORMALIZED CORRELATION

An important requirement of the extension board is that the execution time of the normalized correlation meets the requirements: the search of a 200 x 200 template in a 512 x 512 search window should be performed within 100 - 200 msec.

During the investigation, the current correlator algorithm was analyzed, to find a measure of the complexity (an estimate of the number of instructions to complete the algorithm). This measure was found to be useful in selecting a DSP. This section will describe how the grey value correlation algorithm is implemented and how the amount of instructions was estimated.

As described in section 1.2, there are various ways to reduce the huge amount of calculations for a normalized grey value correlation. The principle of the current grey value correlator algorithm is that it performs exhaustive searches with subsampled template and image. It starts with a global search throughout the entire search window and keeps coordinates where the calculated correlation coefficient are maximum. These are estimated more accurately with some local searches around these maxima with lower subsample factors.

The implementation is a C-function that accepts parameters like:

a. A window size, defining an area within an image, where the template has to be searched, and the position of that window within the image.
b. The global subsample factor.
c. The number of maxima the algorithm should keep.
d. The size of the template.

The required execution time as specified by the customer holds for the parameter set like he specified:
a. A search window of 512 x 512.
b. A global subsample factor 8.
c. Three maxima to be kept.
d. A template size of 200 x 200.

The algorithm performs first the global search throughout the entire search window. This global search is an exhaustive search where template and image are subsampled with a factor 8. During this global search it keeps the three positions were the calculated correlation coefficient was maximum.
A parabola fit at these three positions will give a more accurate estimate where the maximum is situated. Thus after the global search the algorithm has a raw estimate of three positions where the template matches the image.
While evaluating the maximums during the search, they are also compared against a certain threshold. If the correlation coefficient is below the threshold, it will not be further evaluated, since there probably won't be a match. If no threshold would be used and there is no object in the image, the algorithm will still respond with a coordinate which suggests that there is an object in the image. The threshold will thus also serve as a protection, in case an object is not in the image the algorithm will be able to detect that.

Around the three maximums that were found during the global search, it will then perform a local raw search with a factor 6 subsampled template and window. This is also an exhaustive search where the window is reduced such, that 5 x 5 correlation coefficients are calculated. Also a parabola fit is done here. This local raw search modifies the three maxima from the global search thus into more accurate estimates of the positions where the image and template match the best.

Around the three modified maximums, a local medium search is done with a factor 2 subsampled template and image. This time 7 x 7 coefficients are calculated. Again a parabola fit is done at the three positions. This medium search modifies the three positions from the local raw search to more accurate estimates.

At last a local fine search at full resolution is performed around the one position from the local medium search, where the correlation coefficient was maximum. Here an exhaustive search consisting of 3 x 3 coefficients is done to estimate the position. A parabola fit will enhance the result to sub-pixel accuracy.

When selecting a DSP, it will be useful to have an expression for the number of multiplies or instructions that have to be performed. Some time is spent on analysis of the algorithm.

The procedure basically performs four exhaustive searches:
a. The global search with a factor 8 subsampled window:
    nett template size 25 x 25,
    calculation of 40 x 40 coefficients
b. The local coarse search with a factor 6 subsampled window:
    nett template size 34 x 34,
    calculation of 5 x 5 coefficients

c. The local medium search with a factor 2 subsampled window:
nett template size 100 x 100,
calculation of 7 x 7 coefficients
d. The local fine search at full resolution:
template size 200 x 200,
calculation of 3 x 3 coefficients

To get an idea of the complexity (number of computations) of the algorithm, an estimate is made about the complexity of one exhaustive search.

Performing one exhaustive search consists of the following actions:
a. perform the summation loops. (formulas (1.9),(1.10) and (1.11))
b. calculate correlation coefficient (formula (1.12))
c. perform the parabola fit.
d. overhead due to initialisation of variables, additional C-code, cache misses etc.

The number of instructions for each of these actions equals:
a. The amount of multiplies for the summation loops are derived in appendix A
The number of MAC's on each of the searches are as follows:

global search :25 x 25 template, 40 x 40 coeff's	1.100k
loc. raw search 3 exh. searches 34 x 34 template, 5 x 5 coef's	100k
loc. medium search 3 exh. searches 100 x 100 template, 7 x 7 coef's:	1.600k
loc. fine search 200 x 200 template, 3 x 3 coef's:	525k
Total	3.325k

This is a total of approx 3.3 * 10^6 cycles (assuming that a MAC is performed in a single cycle)

b. To estimate the required amount of time for calculating formula (1.12) some simulations have been done by compiling the piece of code and simulate it with simulators for the TI's TMS 320c3x and TMS 320c5x DSP's and for AD's ADSP 210xx DSP's. The results were respectively 125, 640 and 80 cycles. This results in another 200k cycles (floating point) or 1,2M cycles (fixed point).

c. After the coefficient is calculated, a parabola fit is done with four adjacent correlation coefficients (east,west, north, south). This gives a more accurate result with sub-pixel accuracy. Also this piece of code was simulated, it took approx 100 cycles per coefficient. Totally 170k cycles.

d. The previous items together are some 3.700 k cycles (floating point). An estimation for the overhead is hard to tell, since it will also depend on the performance of the processor. The operations as explained under *a*, *b* and *c* were believed to be quite accurate and represent the majority of the code that is to be executed for a normalized greyvalue correlation. Taking a 10-20% margin for overhead in account seems a safe estimation, thus the total amount of cycles is estimated to be 4.1 .. 4.5 X 10^6 cycles.

If an amount of 4.1 .. 4.5 X 10^6 instructions is to be performed within 100-200 msec's, this will require a DSP with an instruction rate of at least 25 Million

Instructions Per Second (MIPS). Since certain instructions may take more than one machine cycle, rather an even faster DSP should be selected.

2.4 DIGITAL SIGNAL PROCESSORS

One of the subjects of this project is to select a DSP for the extension board. Nowadays there are many DSP's available, each with it's own characteristics. Choosing a DSP for a certain application requires thorough understanding of the properties of DSP's. To be able making the best possible choice, the properties of various DSP's have been analyzed. Appendix B shows an overview of the various properties of digital signal processors, this paragraph highlights only the issues that are important for the design of the extension board.

One of the most important requirements for the DSP is its performance. DSP-manufacturers show performance specifications in their data sheets. One of the most frequently used is the sustained instruction rate in Million Instruction Per Second (MIPS). However, when reading these figures, one must not jump to conclusions, since the sustained instruction rate is only achieved under the most optimum conditions.

In the previous section became clear that the extension board requires a single cycle MAC. Every DSP has the ability to perform a MAC instruction in a single machine cycle. However, also every DSP has it's restrictions in performing this single cycle MAC. A MAC requires two operands and in general, one or even both operands have to be stored in on chip SRAM, this differs per DSP. Virtually every DSP requires that the operands are stored in a certain memory type (eg. on chip memory, external memory, data memory, program memory etc.) in order to achieve the best performance. If these conditions are not met, the processor performance might just decrease to 50 or even 33 percent.

Since the amount of on-chip SRAM is limited, programs have to transfer frequently used data into the on chip SRAM, in order to achieve a high performance. Most of the newer DSP's are equipped with on chip DMA controllers that perform these data transfers, so the DSP's "CPU"-time is not waisted for data transfers.

From these examples becomes clear that the performance of a certain DSP depends except the actual computation units also on the resources like on chip memory and DMA facilities that a DSP has integrated on the chip.

If a DSP has to access external memory intensively, the external memory access time will also play an important role, especially where it concerns typical DSP applications like digital filters. In such application, the DSP generally has to perform one external memory access per machine cycle (assuming that the data is in external memory). So if data is stored in slow external memory, the DSP will be continuously waiting for the memory access. In such case, one memory wait state will reduce the performance by 50%.

Furthermore, the instruction rate of the DSP will be influenced by pipeline conflicts, bus conflicts cache conflicts etc. Also if the DSP has many instructions that require more than one machine cycle, the actual instruction rate can be significantly less that the sustained instruction rate.

Finally is concluded that if the data is stored in the right places (eg. on chip memory, fast external memory), and the program is well constructed (optimised for bus and pipeline conflicts), the performance can be close to the sustained performance. How close the sustained performance can be met will depend on the hardware and software design.

2.5 OTHER APPLICATIONS FOR THE DSP EXTENSION BOARD

The main reason for the design of a DSP extension board is to accelerate the grey value correlation. However, it can also improve the processing speed for certain other image processing algorithms. Therefor the design of the grey value correlator should not be dedicated to the normalized grey value correlation, it should be able to perform image processing applications on the extension board that are currently running on the SBIP.

To give the reader an idea about the type of applications there are, this section describes typical applications that could be done by the DSP:
a. Projections
b. Rulers
c. Histogram's
d. Filter operations

A projection simply adds the grey values row and column wise. The result of a projection in an images with size *r* x *c* (rows x columns), are two arrays of size *r* and *c*. Projections can be used for :
- raw detection of position
- detection centre of gravity
- recognition of shapes
- intensity observation

A ruler is used to calculate the position of a light dark transition along specified line. At the position where a transition is detected, a 2^{nd} order polynomial is fit around the grey values that are measured along the line. The position where the polynomial passes a certain threshold is the position (with sub-pixel accuracy) of the light dark transition.

The aim of a histogram is to provide a grey value distribution function. This can be useful for adjustment of illumination or to define a threshold to convert a grey level image to a binary one. The algorithm for the histogram is quite simple, the grey value of each pixel is to be read and used to index an array element that will be incremented.

Filter algorithms are used to enhance the quality of images. The filter operations are 2 dimensional, the following algorithms are used:
- A convolution filter is a linear operation that performs a 2-D convolution.
- minimum/median/maximum filters are examples of non linear operations. They substitute a grey value at a position with the minimum, median or maximum grey value within a kernel around that pixel.

This section just showed a few examples of applications that are currently run on the SBIP. The next question that rises is how this information is to be used in the

selection of a DSP, moreover, there are no requirements with respect to the performance of these algorithms.

With a DSP, MAC-alike structures (digital filters, vector array processing) are implemented very efficiently. For applications without these structures a DSP will probably not really have an advantage over a general purpose processor, he may even have disadvantages! Some of the applications shown in the previous paragraph have, like the grey value correlation, this MAC - alike structure and can thus be implemented efficiently on a DSP. Other applications don't have that structure. The fact how efficient these general purpose applications can be implemented on the DSP will depend on how powerful the DSP is (eg. instruction set, register count, addressing modes etc.).

The strategy for the selection of a DSP will be to find a DSP that meets the timing requirements with respect to the grey value correlation. If then a choice has to be made from various DSP's, the properties just mentioned could help to make a final decision.

2.6 MANUFACTURING PRICE.

Another requirement from the performance specification is, that the manufacturing price for the module is allowed to be Dfl 1000,- to 1500,-. This price includes except purchasing cost for the components also costs for assembling and testing and a margin for indirect costs. During the design phase it will be more interesting to know the allowed costs for purchasing components, this section shows the how is the budget for these costs.

The manufacturing price of the DSP extension board can be divided as follows:
a. Purchasing costs for the components
b. Assembling and testing the extension board.
c. A margin of 23 percent calculated over the two previous items, to account for administrations and storage costs.

If the total should not exceed Dfl. 1500,- the price for purchasing the components, assembling and testing of the board is maximum Dfl 1220,-

The costs for a board depend on the batch size that is being assembled:
a. The price of components is usually lower if larger amounts are being ordered.
b. Assembling and testing usually involves some preparations, thus if a larger batch is assembled, the preparation time spread over more boards.

In the following calculations is assumed that the board will be produced in a batch sizes of 25 pieces.

The costs for testing the board depends on the required time to test the board. Testing the board will take some time to prepare the test environment, followed by testing and debugging the modules. If the 25 boards can be tested within 2,5 days, and eventual failures can be debugged within 2,5 days, the costs will be approximately 4500,- for 25 boards which is approx. Dfl 180,- per board. The price for assembling is estimated to be approximately 75,-

The price for the PCB is estimated to approximately Dfl. 100,-.
This leaves approx. Dfl 900,- to be spent for purchasing the parts for the extension board. During the hardware design, this budget has to be kept in mind when selecting parts for the design.

3 Hardware Design Considerations

During the design of the hardware, several considerations had to be made, these will be covered in this chapter. Section 3.1 describes the choice for an architecture for the extension board. Then, section 3.2 describes the selection of memory. For the choice of the DSP, first a raw selection was made which is covered in section 3.3. This selection led to two possible candidates which will be compared in section 3.4. Section 3.5 explains which of the DSP's on chip peripherals are used on the extension board. Section 3.6 shows the considerations that were made on how the image can be transferred from the SBIP to the extension board.

3.1 SELECTION OF AN ARCHITECTURE

The global architecture of the hardware design highly depends on how it has to interface with it's environment. Section 2.1 described the hardware interface with the SBIP. From that interface there were two architectures found to be possible. These will be covered in paragraphs 3.1.1 and 3.1.2. Paragraph 3.1.3 will motivate the choice for the architecture that was chosen for the extension board.

3.1.1 DSP as a coprocessor

Probably the most simple architecture for implementing the extension board is to use the DSP as a coprocessor of the TMS 34020. Figure 3.1 shows this architecture. The extension board has hardly any intelligence in this concept, it requires a minimum amount of memory and does not use the video interface.

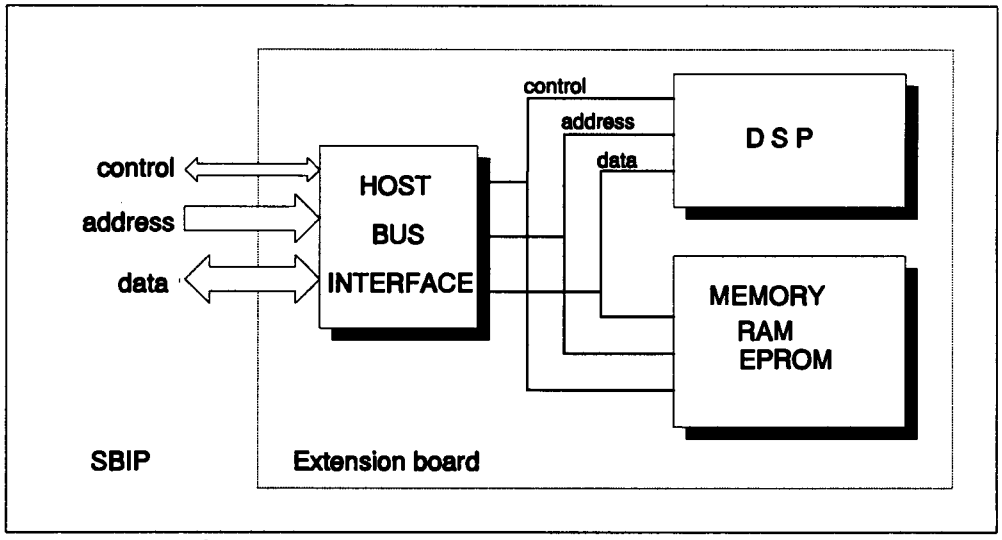


Figure 3.1 : Architecture 1: DSP as coprocessor.

Communication between SBIP and DSP takes place via a host interface (integrated on the DSP or possibly through DPRAM).

Because of the limited amount of external memory, the DSP can be a fixed point type with some on chip data memory. The advantage of using a fixed point DSP is that these DSP's are available with a high instruction rate: during the investigation phase, 40 MIPS fixed point DSP's were available and 50 MIPS were about to come out. Another advantage is that the price of fixed point DSP's is low with respect to floating point DSP's.

The performance of this concept depends besides the performance of the processor and the access time of the memories, also on the required communication time to transfer the required data from the SBIP to the extension board. Since the amount of memory on the extension board is limited, it can not store a complete image in it's memory. So the SBIP has to provide the extension board with the required data while the DSP is performing calculations. In certain cases it can happen that the DSP completed the calculations already, while a next data transfer is still in progress, the DSP will thus be idle until the transfer has completed. The fact wether communication will keep the processor idle, depends on the amount of operations that the DSP has to perform on the received data. Suppose the communication bandwidth is 1.5 Mb
s/sec., then one word will be transferred in 700 ns. A 40 MIPS DSP has to perform at least 35 machine cycles for each pixel to prevent him from being idle. Thus the fact whether communication will be a bottleneck in the execution time for a certain application is determined by the number of operations that the DSP has to perform on every pixel. This will be clear from the two following applications:

a. The execution times of eg. projections (where only the row and/or column - sums of grey values is to be calculated) will be limited by the communication time since the DSP only has to perform only 6 operations (two reads, two summations and two writes) for each pixel that it receives from the SBIP, which will take approx. 200 ns. In this type of application, the DSP is thus continuously waiting for data from the SBIP, and the TMS is busy transferring the pixels to the extension board.

b. The execution time of eg. convolution filter operations with a large enough kernel (eg. 7 x 7) will not be limited by the transfer of the image. The DSP has to perform 60-70 operations per pixel (for multiply accumulating 49 values, scaling and adding an offset value) which takes approx. 1.5 us. (note that every time one grey value is to be sent to the extension board, and the resulting grey value after the filter operation is to be read back to the sbip, one write and one read action takes then 1.4 us).

Note that if communication is a bottleneck in the execution time, it only means that the DSP is not used optimally. However, the performance might still be much better than when the algorithm would be run on the TMS processor.

These examples are relatively simple algorithms that will be easy to implement in software on the described hardware architecture. If more complex algorithms have to be implemented, the software will become more complex: The fixed point DSP has only a limited amount of memory (varying from 32 - 128 k words, depending on the DSP). In that memory program code, image data and intermediate variables, stack etc. have to be stored. Thus the program has to use this memory very efficiently. Furthermore, frequently used data is to be moved into internal

DSP memory to achieve a high performance, this all will make the software very complex.

A possible implementation for the grey value correlator would be that the DSP on the extension board performs exhaustive searches: First the SBIP downloads the template and a subsampled image, the DSP performs the global search. After this global search is completed, the DSP reports the found maximums to the SBIP and waits for the SBIP to transfer the required data for performing the local coarse searches, then the same procedure follows for the local medium - and the local fine searches.

Figure 3.2 shows a schedule for all the actions that are to be taken. According to this schedule the DSP will be occupied most of the time. However, it can not be avoided that the DSP will be idle at certain moments, because after the global search, the DSP returns the positions of the found maximums, and has to wait for the TMS processor to send the window for the local coarse search. The same happens after the third medium search.

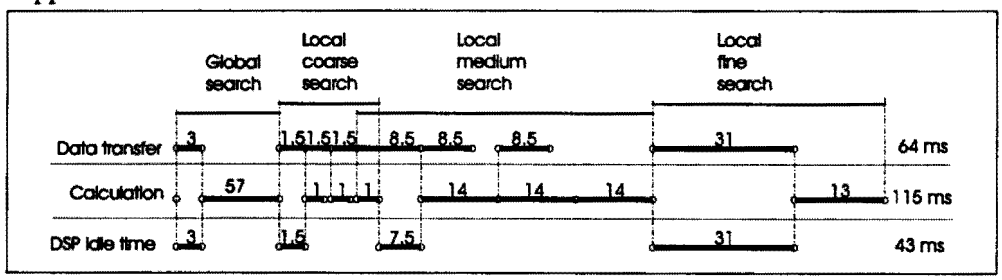


Figure 3.2 : Schedule for a normalised greyvalue correlation.

The calculation times for the DSP shown in the diagram are based on the derivation of section 2.3. Besides the communication times also transfer times for moving the data from the Dual Ported Ram into the RAM memory. Calculations of the required execution times are shown in appendix C.

If the grey value correlation can be implemented as shown in figure 3.2, the DSP will be idle for approximately 42 msec's, waiting for the data to be transferred by the SBIP. This is approximately 40 percent of the actual calculation time of 115 ms. that the DSP requires for the algorithm. The total throughput time is thus 157 ms.
Note that still a margin has to be added to the execution time as shown in figure 3.2. In this calculation time is no overhead included. Generally, there will be overhead for initializing variables, cache misses etc.

The advantage of this implementation is that it is cheap. The DSP can be a relatively cheap fixed point DSP and furthermore this implementation only requires a small amount of memory.
The disadvantage is that this hardware architecture might be simple, but in contrast, the software architecture can be very complex because of the various data transfers that follow each other: data that the DSP frequently uses, is to be transferred to on chip memory to obtain single cycle multiply accumulates.

The execution time of the normalized grey value correlation with this architecture includes some idle time for the DSP due to the data transfers, the following

paragraph explains an architecture where this communication time is eliminated by keeping a copy of the entire image in memory on the extension board.

3.1.2 DSP as an image processor

A more advanced implementation of the extension board will be the one where the extension board can perform the entire algorithm without intervention of the TMS 34020 processor on the SBIP. In this case, the extension board is in fact a small image processing module. This architecture requires that an entire image and template can be stored in the memory of the extension board, and will thus require significantly more memory with respect to the implementation as described in paragraph 3.1.1. Since the addressing space of fixed point DSP's is too small, this implementation will also require a floating point DSP.

Figure 3.3 shows the architecture.

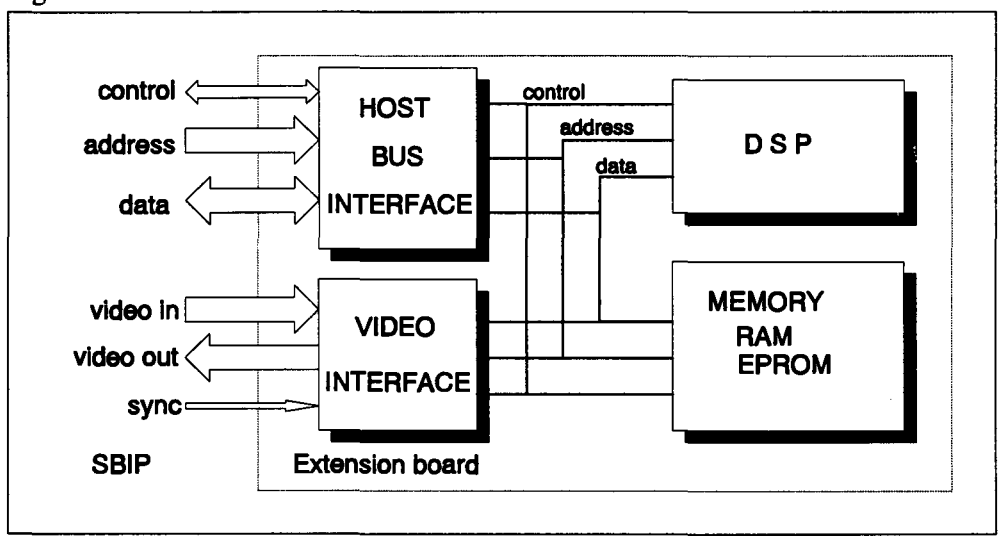


Figure 3.3 : Architecture II : DSP as image processor.

In this architecture the entire image and template and application software are stored in RAM on the extension board. The application software and template can be downloaded via the host interface. The TMS 34020 can access the memory on the extension board via a host interface. Through this host interface the TMS 34020 communicates with the DSP and downloads templates and software. The video bus on the extension connector provides an efficient way to transfer the image to the extension board, while it is being digitized. The transfer overhead with respect to the architecture described in paragraph 3.1.1 will thus be eliminated.

This concept differs from the previous one with respect to the amount of memory and the presence of the video interface. It will be more expensive for the following reasons:

a. It requires a larger amount of memory (approx. 768 k as will be explained in section 3.2).

b. A floating point DSP is to be used to be able to address the entire memory directly. (no fixed point DSP was found that is able to address this amount of memory)

c. Extra hardware is required to transfer images to and from the SBIP via the video bus.

The execution time for the grey value correlator will depend on the processor and the memory that is used. The execution time can be found from paragraph 3.1.1 by subtracting the communication time. Thus if a 40 MIPS DSP will be used in this architecture, the execution time of the normalized grey value correlator will be approximately 115 ms. If eg. a 30 MIPS DSP will be used, the execution time will be 4/3 * 115 = 153 ms. Thus even though the floating point DSP might not have such a high instruction rate like fixed point DSP, the execution time of the grey value correlation in this architecture is somewhat shorter since there is no data transfer overhead.

This implementation will be more expensive with respect to the coprocessor implementation. But in return, it will be much simpler to write application software on the DSP: every pixel is directly accessible by the DSP-core since the entire image resides on the extension board.

3.1.3 Choice of the architecture

Even though the global block diagram as shown in the previous two paragraphs look very similar, there are significant differences.

The only advantage of the first architecture is it's lower price. On the contrary, it requires a very complex software architecture with respect to the second architecture. Any algorithm to be implemented on the extension board will have a relatively complex software architecture, since it is to be divided over the two processors (every algorithm will contain inter-processor communication routines). Achieving short execution times for other algorithms thus requires thorough understanding of the operation of the extension board.

The analysis of the grey value correlation showed that the execution time of the coprocessor solution should be within the 200 ms. However, the software algorithm is complex, therefor there might be a drawback in the software design that leads to a longer execution time then the required 200 msec. Thus besides the disadvantage of complex software algorithms, there is also a potential risk involved.

The second, somewhat more expensive, implementation provides a simple software architecture. It might even be possible to port C code that is currently running on the TMS 34020 processor to the DSP without having to make significant changes to the software, since the DSP can access the entire image. Thus implementing an existing SBIP algorithm on the DSP might require only small- or even no modifications in the existing software sources.

The second architecture was chosen for the extension board. It may be somewhat more expensive, but in return it offers the possibility to implement existing SBIP algorithms with considerable lesser effort.
Furthermore the risks of drawbacks during the software design for the grey value correlator for the second architecture is expected to be significantly smaller then those of the first architecture.

3.2 SELECTION OF MEMORY

From section 2.4 it became already clear that only a fast processor will not guarantee that the required performance is achieved. Especially if the application has to access large amounts of memory, the performance will also depend on the access time of the external memory. The extension board requires a considerable amount of memory. First, in paragraph 3.2.1 is evaluated how much memory is required. The next question that rises is which type of memory should be used for the design, this will be covered in paragraph 3.2.2.
Since single cycle MAC is required to meet the calculation requirements, it is unavoidable to implement a certain amount of memory in Static RAM (SRAM). Since the SRAM is very expensive, some time was spent on analysing the possibility of using cheaper memories in combination with SRAM, this will be covered in paragraph 3.2.3. Paragraph 3.2.4 describes the chosen memory configuration.

3.2.1 Memory requirements

Since the DSP can not access the system memory on the SBIP, a large amount of memory has to be installed on the extension board. The question is now how much RAM will be required.
The RAM should be able to store the following data:

Image data
: One image can consist of up to 512 x 1024 pixels, this requires 512 kb of RAM. For the correlator one image will be sufficient. It will probably be wise to keep in mind that customers might want to expand the image sizes to 1024 x 1024 pixels in future. Thus the image-memory should be expandable to 1 Mb.

Template data
: One template can have the size of 64 k pixels. This will consume 64 kB of memory per template. The fact wether more templates could reside in the memory of the extension board depends on the fact how fast a template can be downloaded to it. If download time is short with respect to the execution time of the algorithm, it is generally no problem to transfer the template before the algorithm is executed.

System memory
: This memory is required for storing variables, stack, etc. The correlator algorithm requires approx. 8 kb for variables.
Let's reserve 64 kb for this area.

Program
: The memory space of 64 kb of code memory will probably be more than enough. (The compiled source code for a TMS 320C30 DSP was approx. 8 kb of assembly language code.)

If these amounts of memory are counted together, it seems that at least 650 to 700 kb of memory is required, expandable with another 512 kb. The following paragraph will explain what possible memory types can be used.

3.2.2 Memory types

There are different memory types available like Static RAM's (SRAM), Dynamic RAM (DRAM), Video RAM's (VRAM), synchronous RAM's etc. This paragraph will explain the applicability of the various memory types on the extension board.

Memories can mainly be divided into static and dynamic memories.
Static memories are very easy to use since they don't need additional logic. They are much faster than dynamic memories, therefor the price is also significantly higher. Dynamic memories require additional hardware to generate the refresh cycles and row column address strobes. Video memories are in fact dynamic memories, they have serial data I/O lines which offers the possibility to transfer data serially at a very high speed.

Memories can also be divided in synchronous or asynchronous types.
Asynchronous memories are most widely used. Except the address and data lines and a chip select line, they only require the read and write signals.
Synchronous memories require a clock pulse and use pipelining to reduce the access time. Synchronous memories can only be used if the processor supports them. Synchronous memories add in fact an (external) pipeline stage to the processor pipeline stages.

From the memory types just mentioned, a choice has to be made for the extension board. Since none of the DSP's support synchronous RAM's, they are not applicable for the extension board, hence, a memory selection has to be made from the asynchronous memories. Possible choices are DRAM's VRAM's or SRAM.

DRAM's are neither applicable for the extension board because their access time is too high to transfer the image into them at video speed. The remaining choice is thus between Video RAMS (VRAM) and SRAM.

VRAM's are dual ported DRAM's, they have a random access port, and a serial port. The data samples from the video bus can be shifted into the memory via the serial port at a very high transfer rate. Simultaneously, the DSP can access the memory through the random access port.
Using video memories would have the following advantages:

a. the DSP can also access memory during the image transfer, thus the execution time may be reduced, since the DSP can start processing while the image transfer is still in progress.
b. The required hardware for the video transfer is relatively simple.

and the disadvantages:

a. VRAM's require a controller for generating refresh cycles and special write cycles during image transfers.
b. the access time through the random access port is rather slow (equal to the access time of DRAM's).

SRAM has the following advantages:

a. it is very fast.
b. simple to interface with DSP.

and one disadvantage: it is very expensive.

For the design, the most convenient way would be to implement the entire memory in SRAM, especially since most DSP's provide all the required signals to control SRAM's without requiring additional logic.

Since a single cycle MAC is required to meet the calculation requirements and the SRAM on the DSP-chip is limited, it seems unavoidable to implement at least a certain amount of memory in SRAM. Since the SRAM is very expensive, some time was spent on analysing alternatives. These will be described in the following paragraph.

3.2.3 Memory configurations

The previous paragraph explained the various memory types that can be used on the extension board. DSP's generally contain on chip SRAM, so the memory on the extension board will consist of this on chip memory and external memory. On can then imagine various memory configurations, this paragraph analyses 4 memory configurations:

a. minimally 64k words of on chip RAM and 512k external SRAM.
b. a limited amount of on chip RAM and 512k external SRAM.
c. external VRAM.
d. a combination of external VRAM and SRAM.

64K INTERNAL MEMORY / 512K EXTERNAL SRAM

In case there are 64k words available in the on chip SRAM, and the external memory is completely implemented in SRAM, the shortest execution time will be achieved with minimum software complexity.
Since the entire template would fit within the internal memory and the external memory is zero waitstated, all the MAC's will be executed single cycle (provided that the DSP contains a program cache or a program memory block).
Since all data is accessible without wait states, the program implementation will be straight forward, since one does not need to worry about moving data from slow to fast memories.

There was only one DSP available that had the required amount of on chip memory, of course this does not mean that other DSP's can not be used. The following sub-paragraph will show that also a limited amount of on chip memory will be sufficient to achieve fast execution times.

LIMITED INTERNAL MEMORY /512K EXTERNAL SRAM.

If the on chip memory is limited and data of large templates is to be processed, it implies that data is to swapped into internal memory. With the external zero waitstated memory, it will be obvious to swap the template into internal memory, since this concerns less data than transferring the image.

The correlator algorithm can be changed such that the template is divided into n small pieces. The summation loops (formula's (1.9) and (1.10)) of an exhaustive search with the template will then be done by adding the results of n of these summation loops with each of the small pieces of the template. After all the results of the n separate loops are counted together the various coefficients can be

calculated with the formula (1.12). This implementation requires a large array to store all the intermediate results.

This implementation of the correlator adds some overhead to the previous implementation, consisting of the required additions of the *n* intermediate results of the summation loops and the required transfer time of the template data into on chip memory.

The impact on the execution time will depend on the amount of on chip memory, the smaller the on chip memory the larger the number *n*, the more extra additions have to be done.
The required time to transfer the template into internal memory will not really depend on the amount of on chip memory, since each template pixel is to be loaded once into internal memory independent of the size of the image.

Let's take the example that the DSP has 1024 words of on chip memory (which is the case on most of the DSP's).
The overhead due to the additions equals:

a. The template size for the global exhaustive search is 625 and thus fits completely in the internal memory, thus there is no penalty.
b. The local coarse search has a template size of approx. 1100 bytes, thus the template will be divided in two sub-templates. Since every time 3 additions (the 3 different summation loops as in (1.9) and (1.10)) have to be done, 3 local searches are done, and 7 x 7 coefficients are to be calculated, it requires 3 x 3 x 7 x 7 = 341 additions.
c. In the same way the local medium search (template size 10k) will take 3 x 10 x 5 x 5 = 750 additions.
d. The local fine search (template size 40k) 3 x 40 x 3 x 3 = 1080 additions

Together this overhead is less then 2500 additions. this is approx. 0.1 % of the number of multiplications to be done and is thus negligible.

Furthermore the templates have to be transferred into internal memory which takes 625 + 3 x 1100 + 3 x 10000 + 40000 = approx. 74000 pixel transfers.
The total overhead can then probably be done within 80000 instructions.
Which is less than 2% of the total amount of instructions.

This example showed that the required amount of on chip SRAM doesn't have to be so very large. The reduced amount of on chip SRAM does increase the software complexity, but not drastically.

The large amount of external SRAM might be too expensive, therefor the following sub-paragraph shows the possibilities of implementing the external memory entirely in VRAM.

EXTERNAL VRAM

With respect to the cost price for the extension board, it would be preferred if the external memory is not zero waitstated, a possible option is to use VRAM as external memory. Since a single cycle MAC is a requirement, both template and image data are to be read into on chip SRAM before the MAC takes place. Over here also a differentiation can be made to the amount of on chip memory.

As the amount of on chip memory decreases, the overhead will increase significantly since certain data will have to be transferred more than once into the on chip SRAM. The overhead in execution time will be determined for an important part by the DMA capabilities of the DSP. So it is hard to give a general estimate of the overhead. It is important that the DSP-core remains active wile the data transfers from external to internal memory are performed. This will require a significant amount of on chip memory. The configuration of an extension board with only VRAM seems thus only feasible if a DSP with large on chip memory and powerful DMA capabilities is being used.

COMBINATIONS OF SRAM WITH VRAM

The conclusion from the previous sub-paragraphs was that it seems likely that the external memory is to be equipped entirely in SRAM. Since this very expensive, it was checked whether the SRAM can be used in combination with the less expensive memories. Possible options are the usage of VRAM with a cache memory or combining SRAM with VRAM.

A cache memory is oftenly used in computer systems with a large amount of slow memory, to reduce the average access time of the memory. The principle of a cache memory is based on the locality of the data that is being processed. A cache memory is a relatively small SRAM (with respect to the main memory). The main memory consists of slow memory devices. When data is read from the main memory, it will be written into the fast cache memory. In case the same data is to be used again, it can then be read from the fast SRAM rather then the slow main memory. Of course there will occur conflicts if locations in the cache are occupied.
The usage of a cache memory is not found to be applicable in this design since this still requires VRAM (plus a VRAM controller) for the image transfer. Furthermore, the size of the cache needs to be rather large to obtain an acceptable cache-efficiency, since the memory accesses during a normalized grey value correlation are usually not local.
Furthermore, one might consider to use both SRAM and VRAM, however, this does not seem sensible for the following reasons:

a. In the first place, this combination will increase software complexity since data is to be transferred from the VRAM to the SRAM.
b. Since none of the DSP's supports external memory block moves, the external block moves will have to be done by the DSP-core, which gives a considerable amount of overhead. Accessing VRAM's will take some 60-70 ns. With a machine cycle time of approx 30 ns. at least two wait states will be required for a fetch from these memories, a word move from VRAM to SRAM would then take 4 machine cycles.
c. The usage VRAM's requires additional control hardware.

Since the program complexity increases, transferring data from VRAM to SRAM memories will result in significant overhead and a VRAM controller is to be implemented on the extension board this memory configuration is not considered applicable on the extension board.

3.2.4 Choice of the memory configuration

A choice for the memory configuration will mainly depend on the price versus program complexity. In the previous paragraph, four different possibilities were shown. The conclusion was that either the entire external memory was implemented in SRAM, or that VRAM could be used in combination with a DSP with a large on chip memory.

SRAM's are more expensive but very easy to interface to the DSP. On the other hand, SRAM's require an address generator to perform the image transfer from the SBIP. In section 3.6 will become clear that the I/O-Processor (IOP) of the DSP can be used for this purpose. And in that case, the additional required logic for the image transfer is limited to some First In First Out (FIFO) memories to buffer the data stream to and from the video bus, and a some logic to control the FIFO's (which will be implemented in an Erasable Programmable Logic Device). The SRAM's cost approx. Dfl 50,- per Mbit (4 pieces required). The two FIFO memories approx. Dfl 30,-. This implementation requires 512 kb external memory, the costs will be approximately Dfl. 230,-.

VRAM's are cheaper, very easy to use in certain respects, but also requiring a special controller, and since DSP's do not support VRAM control, a VRAM controller has to be made to generate the required signals to control the VRAMS. This controller would be implemented in an Erasable Programmable Logic Device (EPLD). The price for the EPLD was estimated to be at least Dfl 150,- more than that of a solution with SRAM. The VRAM devices cost approx. Dfl 20,- per Mbit (4 pieces required). For an extension board with 512 kb, the total costs for memory will also be at least Dfl. - 230,-.

Thus there is hardly a price difference between using SRAM and VRAM. Except if the board is to be equipped with an image memory of 1 Mb, in such case, using VRAM will be cheaper.

From the programming point of view, using SRAM's will be preferred since they don't require special programming techniques to achieve short execution times (like moving data from external VRAM's into fast internal SRAM).
The execution time for the grey value correlator might be somewhat lower if VRAM's would be used.

Since there is hardly a price difference between using SRAM and VRAM, there was decided to use SRAM for the external memory.

3.3 Raw Selection of a DSP

To select a DSP, it is impossible to analyze the operation of every DSP in detail. To choose a DSP for the extension board, a list of selection criteria was made that describe the demands on the DSP. After this list was used to select possible DSP candidates, the amount of possible DSP's was limited. From these remaining candidate list was made a selection, by analysing the properties of the DSP-candidates more accurately. This section describes only the raw selection, which results in two candidates. The following section compares these two candidates and selects one of them.

Paragraph 3.3.1 will describe the selection criteria, paragraph 3.3.2 the candidates that were left after the first selection.

3.3.1 Selection Criteria for first selection

The goal of the first selection is to limit the amount of possible DSP candidates to a smaller amount. To make this selection from the DSP's, a list was made that states the requirements that the DSP should meet:

a. The DSP should have an addressing space of at least 2 Mb.
The architecture that was chosen requires a large amount of memory to store images with sizes as big as 256 k, 512 k or even 1 Mb.
One could decide to use a processor with a smaller memory space, and enlarge the space with a bank switching technique, but this will increase the complexity of the software and is therefore not used.

b. The instruction rate should be at least 30 MIPS
Due to the computation requirements (> 4 $*10^6$ instructions in 100 - 200 ms) processors with an instruction rate of 20 MIPS will not manage to get the job done within 200 ms. To leave a margin between expected and required execution time, a DSP with a rate of minimal 30 MIPS was searched for.

c. The DSP has to be able to perform single cycle multiply accumulates.
This is obvious: most of the instructions will be MAC instructions, if these are not performed in a single cycle, the desired execution time will not be reachable.
This requirement generally means that the DSP should have one memory space with an on chip program memory or program cache and at least 512 bytes on chip data memory.

Analysis of the fixed point processors showed that none of them could address large memory amounts directly. Some devices like Motorola's 56004 and AD's ADSP 2781 can address some megabytes of memory via DMA, but the DSP-core can not access this memory space directly, data from this memory space has to be transferred by a DMA controller to the main memory before the DSP-core can actually access it. Since this will result in more complex software design, the DSP candidate was chosen from floating point types.

There were basically 4 manufacturers of floating point DSP's: Analog Devices, AT & T, Motorola and Texas Instruments. Three of them had a DSP that met the requirements with respect to the instruction rate. The DSP from Motorola had an instruction rate of 20 MIPS only and was therefore no longer evaluated.
Analog Devices, AT & T and Texas Instruments have floating point DSP's with instruction rates of respectively 40, 33 and 30 MIPS. These will be evaluated in the following chapter.

3.3.2 Candidates after first selection

The first selection as described in paragraph 3.3.1, limited the number of DSP candidates to the DSP's of 3 manufacturers. Each of the three manufacturers provides different types of DSP's that mainly differ in the on chip peripherals, they are based on the same CPU core. This paragraph will describe the characteristics of the DSP's and the fact wether they are favourable to use for the extension board.

AT&T

AT&T has one floating point DSP family, from which the DSP 3207 and the DSP 3210 (generally mentioned as 32xx) met the requirements that were stated in paragraph 3.3.1. The DSP 32xx DSP's are 32 bit's floating point DSP's, available with instruction rates of 27.5 and 33 MIPS. They have two 1 k memory banks and has a Von Neumann architecture. Appendix D.1 shows the characteristics of these DSP's.

A more detailed analysis of this DSP learned that the performance of this DSP would not be sufficient for the grey value correlator, because it's multiplier can not perform integer multiplications. Thus to multiply two integer values, they have to be converted to floating point numbers and can then be multiplied. This implies that the MAC can not be done in a single cycle. The DSP can only perform single cycle MAC on floating point numbers.
One might remark that all the grey values can be converted to floating point numbers and can thus perform single cycle MAC. Since this will require 4 times so much memory (to store the 32 bit floats instead of 8-bit grey values) this DSP was rejected from the candidate list.

ANALOG DEVICES

Analog Devices has two families of floating point processors: the ADSP 21020 and the ADSP 21060/62 (referred to as ADSP 2106x). The ADSP 2106x DSP's are in fact built up of the ADSP 21020 DSP-core, a huge on chip SRAM and an I/O-Processor. (Appendix D.2 shows the characteristics of Analog Devices' DSP's). The DSP's have the following characteristics:

The ADSP 21020 is a 32 bits floating point processor available with an instruction rate of 33 MIPS. It has a Harvard architecture. The performance of the DSP-core was expected to be very high for the following reasons:

a. Large register count:
   32 general purpose registers (40 bits)
   32 x 4 data addressing registers (32-bits)
b. A wide instruction word leading to more powerful parallel processing capabilities
c. A large instruction set, most instructions can be performed conditionally
d. Powerful address generating capabilities

Even though this DSP might perform very good it is not favourable since it has no on chip memory. If this DSP would be used it would be necessary to provide the extension board with three physical memory spaces: a 48 bit wide space for program words and storage of variables, and two 8-bit wide spaces for image and template data (two separate spaces required to perform single cycle MAC).

The ADSP 2106x DSP's are in fact the ADSP 21020 core, together with a large dual ported memory and an I/O-Processor (IOP) integrated on a singe chip.
The on chip memory can be configured almost randomly in 16/32/48-bit data or program space or combinations of those. The memory (512 kb and 256 kb for resp. the ADSP 21060 and the ADSP 21062) is divided into two memory banks, from each of them both the DSP-core and IOP can perform one fetch per machine cycle. The IOP handles memory requests from link ports, serial ports or external DMA requests. It contains peripherals like a DMA controller link ports and serial

ports. The DMA controller can handle data transfers from the 6 link ports, the 2 serial ports and 2 external DMA request lines, up to 33 (40) Mwords per second (depending on speed grade of the DSP). It can be used for the image transfer from the video bus.

Basically the memory requirements as described for the ADSP 21020 apply also for the ADSP 2106x DSP's since they are built around the 21020 DSP-core. Because the ADSP 2106x on chip memory is user configurable as program/data memory, it will only be necessary to connect one physical (8 bits wide) memory space to store the image. However, the fact that software programming still requires to store templates and images in different memory spaces still holds for these DSP's.

The CPU-performance characteristics of the 21020 also apply for the ADSP 2106x DSP's. Besides the fact that the DSP-core itself is powerful, it is also available in the highest speed grades among floating point DSP's. In the 3rd quarter of 1995, 33 and 40 MIPS versions were available.

Besides the fact that the software programmer has to store data in two different memory spaces, another disadvantage is that AD's DSP's are not used so frequently as TI DSP's. This means that it will be harder to find third party software vendors.

For the extension board an ADSP 21062 would be a good candidate. The price of this DSP is still very high because the DSP is still produced in rather small quantities. Once the DSP is in normal production, the estimated price (in small quantities) for a 33 MIPS version will be US$ 300,-.

TEXAS INSTRUMENTS

Texas Instruments has 2 families of floating point DSP's that have a very similar DSP-core, and differ mainly in the on chip peripherals. The TMS 320C3x family is the simplest of them with a small DMA controller. The TMS 320C4x family has extended I/O capabilities. (Appendix D.3 shows these characteristics).
The DSP-core of the C3x and the C4x families are very familiar. The C4x core is a more powerful due to his larger register count, more instructions, larger cache size and 32 bit integer arithmetic versus 24 bit integer arithmetic in the C3x. Both families are 32 bit floating point processors, available in instruction rates up to 30 MIPS.

Even though the performance of the TMS 320C4x DSP's is higher then the performance of the TMS 320C3x DSP's, the TMS 320C4x is not expected to perform significantly better for the normalized grey value correlator. The correlation algorithm will be mainly performing MAC's, in which both processors perform equally. The performance of the TMS 320C4x in general purpose applications is expected to be better than that of the TMS 320C3x.

The performance of the CPU core is expected to be lower than the performance of the DSP's of Analog Devices for the following reasons:
a. Lesser registers
b. Parallel processing capabilities are not so good due to smaller instruction word
c. Lesser instructions
d. Less powerful address generators

An advantage of TI's floating point DSP's in general is that they are very widely used and there is a lot of application software available for these DSP's. Besides that they have a "Von Neumann" architectures, thus programmers does not need to worry about data and program memory spaces. (of course they do have to keep in mind that data is to be stored in internal memory to perform single cycle MAC).

For the grey value correlation the TMS 320C31 or C32 would be good candidate (the 'C4x would not perform significantly better but will be significantly more expensive due to the many on chip peripherals).

There were two candidates found to be interesting for the extension board, the ADSP 21062 and the TMS 320C32, in the following section, these DSP's will be compared.

3.4 Comparison ADSP 21062 versus TMS 320C32

It became clear that there were two DSP candidates for the extension board. To make a final choice between the ADSP 21062 and TMS 320C32 DSP's, these DSP's were analyzed in more detail. Paragraph 3.4.1 will show an estimation of the execution time of the grey value correlation for both processors. Then paragraph 3.4.2 will show the expectation about the performance in general purpose algorithm. Paragraph 3.4.3 explains the grow path possibilities of both candidates. After the software development tools are analyzed in paragraph 3.4.4. Paragraph 3.4.5 will determine a cost price for the hardware configuration with the 2 processors. From this configuration will be estimated what price difference there will be for the two candidates. The final choice will be made in paragraph 3.4.6

3.4.1 Performance of the normalized grey value correlator

The expected performance for a normalized grey value correlation will be an important criterion for the selection of a DSP. The derivation of the amount of calculation as described in section 2.3 is used for that purpose. Appendix C shows how the execution times of the grey value correlation are estimated.

The expected execution times are respectively 132 ms for the ADSP 21062 and 179 ms for the TMS 320C32. Thus the execution time of the ADSP 21062 is approx. 15 - 20 % faster than the execution time of the TMS 320C3x. From the appendix follows that this differences consists mainly of the following contributions:
a. The instruction rate of the ADSP 21062 is 10% higher
b. The overhead in MAC loops (see appendix B.6) during the coarse search of the ADSP 21062 is significantly lower and accounts for the other 8%.

The procedure for performing a grey value correlation is generally that first an image is to be read from the SBIP, followed by the actual grey value correlation. Taking in an interlaced image takes 50 ms (average). Thus the total time will take 182 ms for the ADSP 21062 and 229 ms for the TMS 320C32. The time can be reduced for the ADSP 21062.

With the ADSP 21062 it is possible to perform the global search while the image transfer is still in progress. The image transfer of images with 512 lines is regularly in interlaced mode: The 512 lines are transferred in two frames, one containing the even image lines, the other containing the odd image lines. If an even subsample factor is chosen for the global search, this global search can start already after the first frame is transferred to the extension board. By taking the subsampled image in internal DSP memory. In this way, it is possible to gain another 20 msec.
Approx. another 10 ms can be gained by starting the global search when enough lines of the first frame are transferred to the extension board (approx. 50% of frame). This will give a somewhat more complicated software architecture.

Thus the execution time for reading in an image and performing the grey value correlation on the ADSP 21062 can be reduced to 162 (or 152) ms by enhancing the software of the grey value correlator.

3.4.2 Performance in general purpose applications

The previous paragraph showed how the two DSP's perform at the grey value correlation. Since also other applications have to be run, it also interesting to know about the difference between the performance of the candidates. The idea is to give a qualitative impression on how the DSP candidates differ.

In the previous section was discussed already that the DSP-core of the ADSP 21062 was expected to be more powerful than the TMS 320C3x core for the following reasons:

a. Large register count (32 general purpose registers and 4 x 32 data addressing registers)
b. Wider instruction word (leading to more powerful parallel processing capabilities)
c. Larger instruction set, most instructions can be performed conditionally
d. Powerful address generators
e. More on chip memory

It will be clear that it is hard to give a quantitative measure about how much the ADSP will perform better, this will also depend on the type of the algorithm being evaluated. With the arguments given above, is only tried to make clear that in general purpose algorithms the ADSP 21062 will probably be better.

3.4.3 Grow path

Since the demands on computation performance tend to increase when time passes, there will come a time that the computation performance of the extension board is be too low. In such a case it will be desired to upgrade the extension board with as little effort as possible. How well suite the two candidates?

The DSP's from Texas Instruments will be available with a performance of 30 MIPS at the end of 1995, the on chip memory of these DSP's is 512 words. The software that will be written for the TMS 320C32 can be ported easily to the TMS 320C4x. However, it is difficult to upgrade the TMS 320C32 DSP to a TMS 320C4x since they are not pin compatible. In such case the entire extension board would have to be redesigned.

The ADSP 21062 has a cycle time of 33 MIPS and a 256 kb internal memory and is available in the 3rd quarter of 1995. At that time there will also be 40 MIPS versions available (a 50 MIPS version is planned for future). Further, the same (pin compatible) DSP's are also available with 512 kb on chip memory (ADSP 21060). The software of the ADSP 21060 is compatible with that of the ADSP 21062.

The conclusion is that the ADSP 21062 offers better perspectives to upgrade the extension board to obtain a better computational performance.

3.4.4 Software Development

Writing applications for the DSP requires special development software and development tools. Both Analog Devices and Texas Instruments provide software development tools like an assembler, a C compiler, linker, simulator etc.
Since the available time was too little there was not done an investigation on the development tools (eg. compiler efficiency of the two compilers).

With both processors it is thus possible to use the C-sources that are available for algorithms that are currently running on the SBIP and implement them on the extension board. If the software environment that is currently used on the SBIP is adapted on the extension board, SBIP application software should be runable on the DSP with a minimum of changes to the algorithm.

An important disadvantage of the ADSP 21062 however, is that it requires to store data in two different memory spaces to facilitate single cycle MAC. The C language of AD's compiler has extensions to specify in which memory space certain data is to be stored.
The TMS 320C32 on the other hand, does also require that the data (for single cycle MAC) is stored in a certain memory area (on chip memory), but it is in the same memory space.

3.4.5 Hardware configuration

If a choice is to be made for either of the DSP's, one will also be interested in the costs of the complete hardware. For the total budget of the costs is referred to section 2.6. This paragraph will describe for both DSP's which additional hardware will be required for the extension board in order to make an estimation for the price.

HARDWARE CONFIGURATION FOR THE ADSP 21062

DSP

The price of the DSP is still expensive because the device is not being manufactured in large amounts at the moment. The current price for a 33 MIPS

21062 is approx. Dfl 700,- The expectation of the supplier is that the price will fall to approx Dfl 500,- by the end of 1996.

Memory

The internal 256 kb of on chip memory, can be used to store the template and the program. To store images, an external memory of 512 kb will be required. The 33 MIPS version requires 15 ns. memory devices for zero wait stated memory. These devices cost approx. Dfl 200,-

This design also requires some FIFO memories to buffer the asynchronous video stream to and from the video bus. These cost approx. Dfl 30,-

EPLD

It will be obvious that additional logic will be required to connect the various devices together, and to interface them to the extension connector and control the FIFO memories. A common practice is to use an EPLD for this purpose. The functionality that has to be done in the EPLD requires a relatively small device. The price for such a device is approx. Dfl 125,-.

General

Further additional parts like connectors, a crystal oscillator, bus drivers etc. will be required. These costs are estimated to Dfl 100,-

The total estimated price for the parts will be approx. Dfl 975,- (assuming that the price for the DSP will go down to Dfl 500,-). With this DSP the price for the extension board might exceed the Dfl 1500,- somewhat.

HARDWARE CONFIGURATION OF TMS 320C32

The solution with the TI processor requires the following hardware:

DSP

The price for the TMS 320C32 will be approx. Dfl 75,- (note that this price is significantly lower in comparison to the ADSP 21062 due to the fact that this chip has not so many on chip peripherals).

Memory

This configuration requires except the image memory also a memory to store code and template and variables, a total of 768 k external SRAM memory is planned to be used in this configuration. The DSP requires 12 ns. devices for zero wait stated memory. These devices cost approx. Dfl. 425,-

EPLD

In this configuration the programmable logic device provides the interface between the DSP and the TMS 34020. Further it has to contain the address generator for the image transfer. Some estimations about the required size for the address generator unit were done during the design phase. The results showed that a relatively large EPLD is required to store the address generator. The price for such a device is approx. 325,-

Logic

Further some small additional parts like connectors, a crystal oscillator, bus drivers etc. will be required. These costs are estimated to Dfl 100,-

The total estimated price for the parts is approximately Dfl 925,- for the solution with the TMS 320C32 DSP.

3.4.6 Choice of DSP

The previous paragraphs showed significant differences in the results between the two solutions. So the next step is to make a choice for one of them.

The ADSP 21062 scores well in it's performance, it has a better grow path while it is not significantly more expensive. On the other hand, the TMS 320C32 scores slightly better from the programming point of view, programmers won't be bothered with different memory spaces, they only have to use the on chip memory efficiently.

Due to the significantly better performance, it's ability to upgrade the extension board for a better performance and the marginal price difference was chosen for the ADSP 21062.

3.5 USAGE OF DSP PERIPHERALS

It became clear already that the DSP has peripherals on the chip that can be used on the extension board. This section explains which peripherals are available on the DSP, and which of them are used for the extension board, and which are not.

Paragraph 3.5.1 explains the I/O-Processor, paragraph 3.5.2 the host interface that is integrated on the chip and paragraph 3.5.3 the boot memory,.

3.5.1 I/O Processor

The I/O Processor (IOP) of the DSP contains 6 link ports, two serial ports and a DMA controller. The DMA controller manages requests from the link ports, serial ports and two external request lines, and is thus a 10 channel DMA controller.

The link ports each have 4 data lines and two handshake lines (all are bidirectional). The communication bandwidth for one DMA channel is 33 Mb per second. The data being received via a link port is packed to 32 or 48 bit words and stored into internal memory. The link ports are generally used for inter processor communication or communication with other devices with the same link port characteristics.

The link ports can not be used for image transfers from the SBIP. To explain this, the bandwidth of a link port is expressed in Mwords per second rather than Mb per second. This will give a better performance indication since every (8-bits) pixel is to be stored into one (32-bits) memory word.
The data that is being received through the link port is packed into 32 or 48 bit words. In case the data is packed into 32 bits, the bandwidth would be 8.33 Mwords per second per channel (4 bytes per word), which is too low for the video transfer.(which requires 20 Mwords/sec).

Furthermore, it is not possible to combine eg. 4 (4 bit) link ports into one 16 bit link port. In that case it would be possible to use 4 link ports to transfer the image into the memory. From these facts it follows that link ports are not usable for the image transfer.

One might argue whether these ports could be used to download templates from the TMS 34020 processor. This would require additional hardware to unpack the data from the TMS 34020 from 32 bits to 4 bits. The external request lines provide a better interface for this purpose, the link ports are thus not used on the extension board.

The (synchronous) serial ports can perform serial transfers up to 33 Mbit/s. The received data can be packed into word widths ranging from 3 to 32 bits and stored into internal memory. It will be clear these serial ports are not usable for image or data transfers because the bandwidth is much too low.

Two external DMA request lines provide the possibility to transfer up to 33 Mwords per second. The data can either be stored in internal DSP memory or external memory. In the last case, the DMA controller performs the address generation for the transfer, and is usable for the image transfer from the SBIP (which will be explained in more detail in section 3.6). This transfer method can also be used for the interface to the TMS 34020, to download templates.

3.5.2 Host interface

The host interface of the DSP only consists of bus request and bus grant lines. These are independent of the request lines for the DMA controller.
The host can request the bus from the DSP, and once the DSP has granted the bus, the host has control over the address, data and control busses of the DSP. It can the manipulate the external memory, the internal memory and the IOP's control register (not the boot memory). The host interface is to be used to boot the DSP.

3.5.3 Boot memory

The DSP has a special 8 bits wide memory space, called the boot memory. This is used for booting the DSP from a single (8 bits - wide) EPROM.
It is the intension to boot the DSP via the host interface, having an EPROM on the extension board is not desired. However, it might be handy to have an opportunity to boot the DSP from an EPROM. In case the SBIP is used stand alone, the code would have to be downloaded from the SBIP's EPROMs. With the EPROM boot option, the DSP can then boot from its own EPROM instead of storing the DSP code in the EPROM's of the SBIP. Using this boot memory requires no additional hardware, only a socket for the EPROM.
Furthermore it is possible to use a non volatile RAM instead of an EPROM, in that case it is possible for the DSP to store data in the non volatile memory.

3.6 VIDEO TRANSFER

The chosen architecture for the extension board requires that images can be transferred to the extension board at video speed. In the previous section was explained that the IOP of the DSP can perform this task. Another option would be to implement an address generator in hardware that performs this task. This section explains the advantages and disadvantages of both possibilities, paragraph 3.6.3 explains which option is chosen.

3.6.1 IOP as address generator

As explained in paragraph 3.5.1, the IOP of the DSP can perform (background) data transfers from the video bus to external SRAM.

A disadvantage of this solution is that there has to be an interrupt service routine in the DSP in case an image has to be de-interlaced. (Interlaced images contain two frames, the first contains the even lines, the second contains the odd lines). The image transfer would then be done by having the IOP performing one DMA transfer per image line. The entire image transfer would consist of 512 consecutive DMA transfers (assumed it is an image with 512 lines). The IOP can be programmed to generate an interrupt after a DMA transfer is completed (one image line is transferred into memory).
This interrupt should advance the DMA address generator one line and start the next DMA transfer. In this way there will be a gap (with the size of one image line) between two even lines, thus the odd line during the next frame can be inserted in between, and thus the image will be loaded into memory linearly.

To guarantee a successful image transfer, one has to be sure that the interrupts from the IOP are replied fast. Thus during the image transfer, the interrupts from the IOP should have the highest priority, and the response time of the interrupt should be very short.

Furthermore this solution requires FIFO memories to buffer the incoming video data (which will be explained in paragraph 4.1.3).

The advantage of this solution is that, except for the FIFO memories, does not require additional hardware.
Another advantage is that data might be transferred to the external SRAM via the internal memory. This offers more flexibility in the image transfer, the DSP can then only store a window of interest in SRAM rather than storing the entire image, or the incoming data might be processed already in the internal memory.

3.6.2 Hardware address generator

The main disadvantage of having the IOP performing the image transfer, is that it requires intervention from the DSP - core. A solution that is completely independent of the DSP would be preferred, since there is no risk at all that the image transfer is corrupted. By adding hardware, it is possible to achieve this. The additional hardware has to generate the memory addresses and write pulses to store the incoming video data (or read pulses in case an image is transferred to the SBIP).

A disadvantage of the hardware solution is that it requires extra hardware and will thus be more expensive. Appendix E shows a possible design of an address generator in an EPLD. This shows that a significantly larger EPLD has to be used to store the address generator. The price of this EPLD might be Dfl. 150,- to 200,- higher. On the contrary, the price for the FIFO's (Dfl 30,-) will be saved.
Usage of a hardware address generator will increase the costs of the extension board with probably more than Dfl. 100,-.

3.6.3 Choice of image transfer method

The two previous paragraphs showed two possibilities for the image transfer from the SBIP. The solution with the hardware address generator is the most secure solution, since it does not require intervention from the DSP-core.
The IOP offers more flexibility with the image transfers since the DSP can process data as it comes in, or only store a window of interest.

The involved risk in using the IOP is that the image transfer might fail if the DSP does not respond to an interrupt fast enough. The interrupt response time of the DSP is a few machine cycles only. If other interrupts are disabled during the image transfer there should not be a problem that the DSP-core does not respond fast enough to the interrupt. However, in cases where the response time is too large, there is an alternative solution (with chained DMA transfers) that does not require any processor intervention during the image transfer, so the risk that the image transfer will not work is rather small.

Basically, the solution with the hardware address generator is preferred, however, the price of the extension board roughly equals the maximum allowed value already. Since the price will become even higher with hardware address generator there was chosen for the solution where the IOP performs the image transfer.

4 Design of the extension board

While the previous chapter discussed the considerations that were made for the design, this chapter will explain the design itself. First the hardware design will be covered in section 4.1, then the software design will be explained in section 4.2.

4.1 HARDWARE DESIGN

This section will give a brief description of the block diagram and the operation of the extension board. First the block diagram and the global operation will be explained in paragraph 4.1.1, then the interface with the TMS 34020 will be described in paragraph 4.1.2 and paragraph 4.1.3 concludes this section by describing the video interface and the image transfer operation.

Here is noted that a detailed description on the hardware and its operation can be found in appendix F.

4.1.1 Block diagram of the extension board

Paragraph 3.1.2 showed a global architecture already for the extension board. The hardware was mainly divided into five functional blocks:
a. the processor
b. the memory
c. the TMS 34020 interface
d. the video interface
e. the controller

Figure 4.1 shows the block diagram of the extension board with these functional blocks and their interconnections. Within the functional blocks, a global internal architecture is shown to give an impression on the operation of the extension board. The global operation of the hardware will be described in this paragraph.

The heart of the extension board hardware is the processor and the memory. The DSP-core can access maximum 1 Mb of external SRAM (512 kb standard-, and 512 kb of optional - memory) and 256 kb internal DSP memory. The external SRAM is used to store images, the internal DSP memory is used to store the program, variables and the template. Furthermore the DSP has a boot memory device connected to it, this is an 8-bits wide memory device from which the DSP can boot or where it can store data.

The processor block contains besides the processor and the clock generator, also some hardware to create a Direct Memory Access (DMA) - bus. Via this DMA bus, the DSP communicates with the SBIP. The communication with the SBIP is handled through two blocks, the TMS 34020 interface and the video interface. On the SBIP's side, they are connected to the TMS 34020 busses and the SBIP's

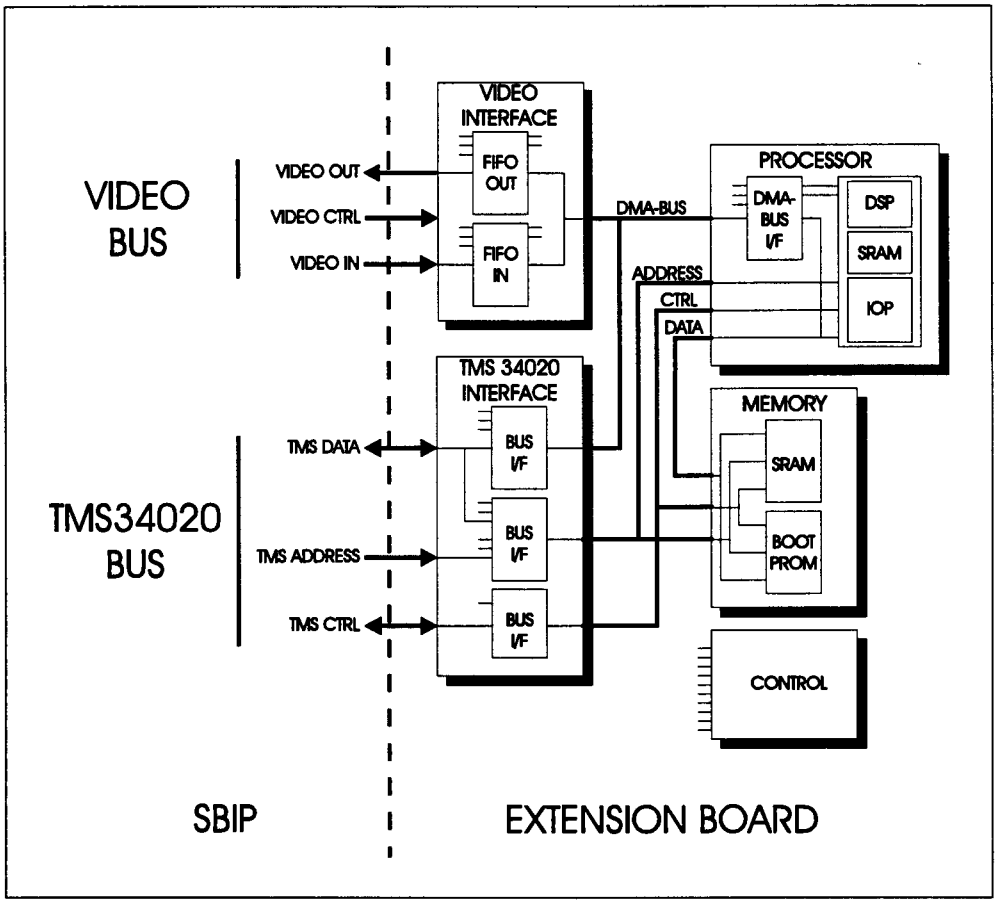


Figure 4.1 : Block diagram of the extension board

video bus respectively. On the DSP-side, these blocks are thus connected to the DMA bus.

The TMS 34020 interface provides the inter processor communication. Through this interface, the TMS 34020 has the ability to access the entire memory on the extension board. Furthermore the interface provides the possibility for both processors to assert each others interrupt line. This interface is used for giving commands, booting the DSP, transferring templates etc.

The video interface facilitates the transfer of images to and from the SBIP. Images can be transferred at video speed (10 - 20 Mb/s) to or from the extension board. The video stream coming from the SBIP's video bus, flows through a First In First Out memory (FIFO) to the DMA bus. The DSP's IOP manages the transfer from the DMA bus to the memory. The video stream from the extension board to the SBIP is performed in the same manner in reversed direction: the IOP transfers the data from the memory to the DMA bus, from where it flows through a FIFO memory to the SBIP's video bus.

The control block, implemented in an Erasable Programmable Logic Device (EPLD), is used to "glue" the various functional blocks together. It controls devices like the tri-state buffers, hardware registers and the FIFO memories. It also performs tasks like error checking during image transfers and data transfers from the host. The DSP communicates with the EPLD via three output lines and 3

input lines. Via the outputs, the DSP can set the EPLD in the desired operating mode. (image transfer to- or from- the SBIP).
Via the inputs, the DSP receives status information of the hardware, and the video synchronisation pulses that the DSP requires for the image transfer.
Furthermore the EPLD offers the possibility to set the extension board in a test mode, to test the various hardware functions while the extension board is connected to the SBIP.

4.1.2 TMS 34020 interface

The TMS 34020 interface provides the inter processor communication between the DSP and the TMS 34020, furthermore it allows the TMS 34020 to set the extension board in test mode and read hardware status information.

The inter processor communication is implemented such that the TMS 34020 can access the entire memory on the extension board via two different procedures Host DMA and I/O-Processor-DMA (IOP DMA):

HOST DMA

During Host DMA transfers, the DSP's address-, data- and control-busses are acquired by the TMS 34020. After the DSP has granted the busses, the TMS 34020 processor controls the DSP-busses, enabling him to access the external SRAM, the internal DSP memory and the IOP's control registers.
Since there is only a limited amount of TMS 34020 address lines available on the extension connector, it is not possible for the TMS 34020 to access the entire memory space of the DSP directly. The implementation is such that the entire main memory of the extension board is divided into 2^{24} memory pages of 256 words (32 bits wide). One of these memory pages of the extension board, is mapped in the memory space of the TMS 34020. The page is selected via a hardware register that is also mapped in the TMS 34020 memory space. Thus before the TMS 34020 can access a certain memory location, it first has to write the memory page number to the page register, followed by the actual memory access in the 256 word page that is mapped in the SBIP's memory space.

This transfer procedure is required for booting the DSP from the TMS 34020, furthermore it is used for transfers that do not require large data amounts, like giving commands to DSP and reading back the command results.
For transferring large amounts of data, the IOP-DMA transfer procedure was also implemented on the extension board.

IOP-DMA

Via IOP-DMA transfers, the TMS 34020 uses the DMA request/grant lines of the DSP, to perform the data transfer. This transfer method is intended for large data block transfers (like templates) between the SBIP and the extension board.
For an IOP-DMA transfer, parameters like the begin address where the data is to be stored and the data block size, have to be set in the IOP's control registers. Then the data is transferred by accessing one and the same SBIP-memory location. Every access to this memory location will generate a DMA request, the IOP will then generate the required addresses and control signals to access memory on the extension board.

The data transfer via IOP-DMA is faster, allowing a higher data transfer rate. The transfer is faster because the DMA operation is pipelined: the data word that is being transferred is written into a hardware register by the TMS 34020, then a DMA request is performed by the hardware. While the IOP then generates the address and control signals to store the current data word, the TMS 34020 can fetch the next data word already from it's memory. The TMS 34020 does not have to wait until the IOP has written the data into the memory on the extension board.

This transfer method, however, requires error checking, because the data transfer might be incomplete if the IOP does not answer a DMA request within a certain time limit. Such situation might occur if the DMA channel has lower priority than another DMA channel or the DSP-core. In such case, the TMS 34020 will be halted during the next access to the DMA port for the particular time limit. After this time is expired, the data in the hardware register is discarded, and an error bit is set.
The TMS 34020 has to check this error bit after the data transfer is completed, to verify that the data transfer was performed without errors.

DSP EXTENSION BOARD MODE AND STATUS

A four bits read/write register is used for reading status or setting the board in a specific mode. This register is located in the EPLD and does not concern accesses to the DSP's memory.

Reading this memory location returns four status bits:
a. 1 bit represents the image transfer status bit. It is set to '1' if the DSP uses it's external busses for performing an image transfer. Using this status bit, the TMS 34020 can check whether he is allowed to access the memory of the DSP extension board because such memory access will interrupt the image transfer.
b. 1 bit representing the error status during a IOP-DMA transfer.
c. 1 bit representing error status for image transfers between extension board and the SBIP.
d. 1 bit representing a warning during the image transfer between the extension board and the SBIP.

Writing to this memory location can be used for:
a. Resetting the four status bits as described before.
b. Setting the extension board into a test mode:
In this test mode the EPLD operates in a such a mode that incoming video data from the SBIP's crossbar is passed back to the cross bar via the FIFO's. This offers the possibility to test the video path, without having to boot the DSP. During this test mode, the DSP can also run a test program and pass status info via its three output flags. These can be read by the SBIP from bits 0 - 2 of this memory location.

The TMS 34020 interface contains another (8 bits -) register for module reset and identification.

Writing to this address can be done for:
a. Performing a reset on the extension board hardware.
b. Asserting an interrupt line to the DSP.

Reading from this address returns the following information:

a. A module identification (for the extension board it is always 0x6). It can be used by the SBIP software, to auto detect the presence of the DSP extension board.

b. The hardware revision code defining the revision number of the extension board hardware. In case hardware modifications are made to the extension board in future, it might be necessary that the software has to control the hardware differently. The software can detect the hardware revision and control the board accordingly.

c. Timer expired output of the DSP. This output is asserted by the DSP, when its hardware timer is expired. It could be used as a watch dog to check if the software in the DSP is still running.

4.1.3 Video interface

The interface between the SBIP and extension board consists besides the processor interface as described in the previous paragraph, also of a video interface. This interface consists of 10 inputs and 10 outputs of the crossbar switch of the SBIP, and 4 synchronisation signals. The video interface makes it possible to transfer images to/from the SBIP at video speed (up to 20 MHz data transfer). By setting the proper selections in the SBIP's crossbar switch, images can be transferred to the extension board while they are being digitized. This paragraph briefly explains the video interface operation, by first explaining the hardware operation, then the image transfer operation and at last the error checking that is done during the image transfer.

HARDWARE OPERATION

In paragraph 4.1.1 became already clear the video in and outputs of the SBIP's crossbar are connected to FIFO memories on the extension board. These memories buffer the video stream to and from the SBIP: the SBIP transfers the video information at a maximum transfer rate of 20 M words per sec. The DSP on the other hand, reads out the information at only 16.7 Mwords per second. However, during the sync pulses, no data is written into the FIFO, so the DSP can manage to read the FIFO until it is empty before the next image line starts. In appendix C.2 is shown that a FIFO size of 512 words is sufficiently large.

The EPLD plays an important role during the image transfers. It manages the data transfer from the SBIP's extension connector to the data bus of the DSP. (The transfer of the data from the data bus into the memory is managed by the IOP). In case the DSP orders the EPLD that an image is to be transferred, the EPLD performs the following 3 (asynchronous) processes:

a. it generates one FIFO write pulse for each pixel that is transferred.

b. it generates one DMA request to the IOP for each pixel that is written into the FIFO memory.

c. it generates a FIFO read pulse for each DMA grant pulse it receives from the IOP.

These processes make sure that each pixel will arrive on the extension board.

IMAGE TRANSFER OPERATION

During the image transfer operation, the software in the DSP has to make sure that DMA transfers are set up at the time that the hardware starts generating DMA request pulses. Since the operation of the software is strongly related to the hardware, it will be explained in this paragraph:

The general operation of the image transfer consists of software that initializes the DMA transfer on the DSP and software that becomes active once per image line. The initialisation software performs the following tasks when it is invoked:

1. It sets the output flags such that the EPLD puts the Vsync pulse from the SBIP on the DSP's input flag, and it awaits a rising Vsync edge (start of frame).
2. Then it sets the EPLD in "get image mode (via its output flags). In that case, the EPLD sets the frame ID (ODD or EVEN) on the DSP's input flag and furthermore performs the 3 processes as just mentioned. In case of an interlaced image transfer, the DSP checks whether the next frame will contain the even or odd image lines.
3. The DSP sets up a DMA transfer of 1 image line to the appropriate memory locations and enables interrupts.

The software (interrupt service routine) that becomes active once per image line performs the following tasks:

1. After the first line transfer is finished, the interrupt server becomes active, which will initiate a DMA transfer for the next line. This interrupt server will then become active after each line transfer.
2. After the image is transferred completely, the interrupt server will reset the get image command code and disable DMA interrupts so it will not become active any more. The image transfer is then completed.

Since the image transfer operation involves intervention of the DSP-core and it uses the DSP busses intensively, there are certain restrictions to make sure that the image transfer operates correctly:

a. Not too many other interrupts with higher priority should become active during the image transfer since they increase the interupt latency for the image transfer interrupt service routine.
The DMA interrupt becomes active once per line time (average), the interrupt latency for this interrupt should therefor be shorter than one line time, otherwise pixels of the next lines might be lost.

b. The TMS 34020 should not perform host DMA transfers. If the TMS does perform host DMA transfers, it will acquire the DSP busses, and hence the IOP will loose control over the DSP's busses and can not perform the image data transfers.

c. The IOP should have bus priority over the DSP-core.
The image transfer operation requires a considerable amount of the DSP's bus bandwidth. The remaining bandwidth may be used by the DSP-core to perform external memory accesses. As long the IOP has a higher priority than the DSP, there are no further restrictions on the software that is running on the DSP-core with respect to external memory accesses: it may perform external memory accesses as desired, the bus arbiter on the DSP chip will make sure that the IOP gets the bus when it needs it.

During the image transfer, the DSP core is only used for a small amount of time, and hence there might be running another program on the DSP-core. The consequences for software that is running on the DSP-core during image transfers are:

a. External memory accesses will be delayed, since the IOP has bus priority over the DSP-core.

b. The program will be interrupted once per line time, this will consume approx. 2% processor time during an image transfer.

ERROR CHECKING DURING IMAGE TRANSFER OPERATION

During the image transfer from the SBIP, the hardware also performs the following checks:

a. FIFO status.
If the video input FIFO ever becomes full during an image transfer to the SBIP, there is a potential risk that data samples will be lost. Therefor an error bit is set, every time the video input FIFO becomes full during an image transfer from the SBIP.

b. DMA request/Grant difference.
If for one or another reason, the DMA controller does not respond to DMA requests fast enough, and the hardware would keep generating DMA requests, the operation of the DMA controller would be undefined. Therefor, no more DMA requests will be performed if the difference between the amount of requests and grants reaches 7. In such situation, a warning bit will be set.

The TMS 34020 reads these status bit's after an image transfer completed. If the error bit is set, the image transfer was probably corrupted.
The warning bit only tells that the IOP is having problems keeping up with the transfer rate. It does not imply that the transfer was corrupted.

4.2 SOFTWARE DESIGN

The operation of the grey value correlator contains besides the hardware a considerable amount of software. This software consists of the application and system software on the DSP and of software for the TMS 34020 and the SBIP's host computer. The aim of this chapter is to briefly explain how the software for the extension board was set up. A more detailed description can be found in appendix G.

Paragraph 4.2.1 will describe how the software of the extension board interfaces to the SBIP software. Then paragraph 4.2.2 describes the DSP software, paragraph 4.2.3 the host and SBIP software and at last, paragraph 4.2.4 explains something about porting SBIP software to the extension board.

4.2.1 Software interfaces

As became clear in section 2.2 the software environment for the extension board had to be adapted from that of the SBIP. This resulted in a DSP library for the DSP, one for the SBIP and one for the host computer. The DSP-libraries on the host and the SBIP are an extension to the current vision library.

The DSP operates as a slave of the SBIP, it receives commands from it, executes them and returns results to the SBIP.

Figure 4.2 shows an overview of hierarchy of the host, the SBIP and the DSP extension board.

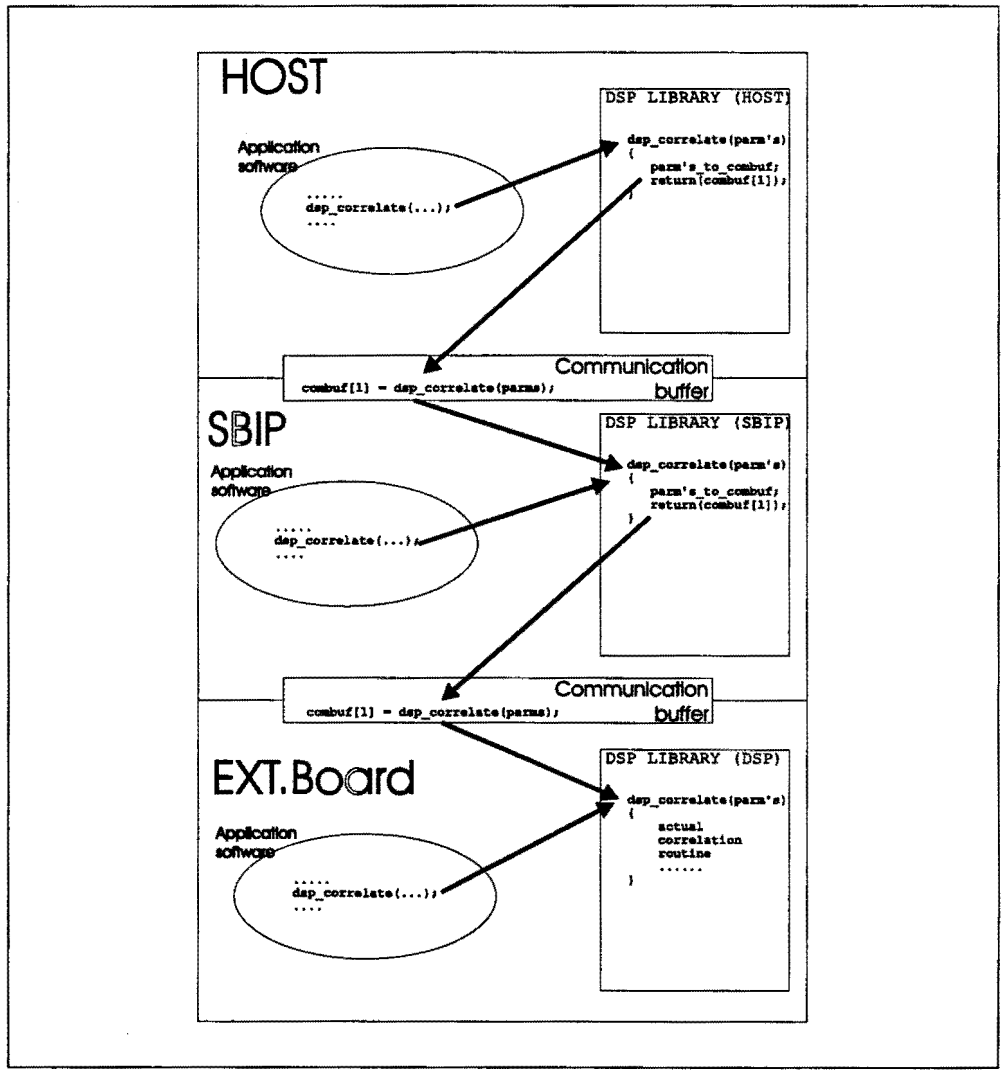


Figure 4.2: Software interfaces host - SBIP - extension board.

There are basically two software interfaces that play a role: one between the host and the SBIP and the other between the SBIP and the DSP.
The software interface between the host and the SBIP provides the possibility to call the various image processing functions of the vision library on the SBIP and also the DSP routines, from the host computer.
The command handler on the SBIP passes parameters for the DSP routines that it receives via the communication buffer, to the functions in the DSP library that is on the SBIP. (Here is pointed out that the same DSP function in this library can also be called by the application software on the SBIP as the arrow suggests). This function passes the parameters in turn to the communication buffer on between the SBIP and the DSP.
The command handler on the DSP passes parameters that it receives from the SBIP via the communication buffer, to the appropriate function. After that

function is executed, the results are passed back via the communication buffer to the SBIP.

Figure 4.2 shows that if the function `dsp_correlate(..)` is called from an application program on the host. The library function passed the parameters via the communication buffer to the SBIP. The command handler on the SBIP calls the function `dsp_correlate(..)` from the DSP library that retains on the SBIP. This function passes the parameters via the communication buffer to the DSP extension board via the communication buffer. The command handler on the DSP calls the `dsp_correlate(..)` function that performs the actual correlation. Furthermore, it is shown that the function `dsp_correlate(..)` can also be called from application software on the SBIP or the extension board. The software interfaces provide thus a transparent communication between the various processors.

As shown in figure 4.2, DSP-function calls by the host are passed to the DSP via the SBIP. In fact, every command could be issued straight from the host computer to the DSP via the address and data busses of the TMS 34020. For test purposes, it came out very handy to issue commands directly from the host to the extension board without intervention of the TMS 34020 CPU. For normal operation, however, it is a better practice to involve the TMS 34020 in passing commands to the extension board. In that case the DSP receives the tasks always from one and the same CPU, thus no conflicts like the host and the TMS 34020 calling a function simultaneously, can occur.

4.2.2 DSP software

As explained in the previous section, the extension board contains a library with functions that can be called by the SBIP. The three following sub-paragraphs explain the software that runs on the DSP: the DSP initialization, the image transfer and the correlator software.

DSP INITIALISATION

After booting the DSP, some initialization has to take place: the run-time environment has to be set up and some variables have to be set to defaults.

The C run-time environment controls dynamic memory allocation, system initialisation, stack usage etc. The initialisation of the C - run-time environment is derived from a system-architecture file describing the memory resources, and a header file with interrupt tables. The required routines for initialisation and memory allocation are linked with the user objects into an executable, during the link process.
After a chip reset, the processor jumps to the initialization procedure of the run-time environment. After the initialization of the run-time environment is completed, function `main()` is called.

After initialisation of the run-time environment, some DSP registers and variables have to be initialized. This is done by the `dsp_init()` routine which is called from function `main()`. This routine sets the default image transfer parameters and some of the DSP's control registers to default values. This routine can also be

called from the TMS 34020 and the host via the software interfaces to bring the DSP into a defined state.

IMAGE TRANSFER ROUTINES.

As became clear in the paragraph 4.1.3, some software is required to transfer images from the SBIP to the extension board. The software consists of the actual functions to perform the image transfer and of a function that sets the transfer parameters.

Transferring images to or from the SBIP requires knowledge of parameters that define the format in which the image is being transferred. After power up these transfer parameters are set to default values. The routine `dsp_settransferparam()` can be called to change these parameters. Through this routine the image size and transfer method (interlaced/non-interlaced) can be specified.

The image transfer functions consist of a routine `dsp_getimage()` that is called initially when the image is to be transferred to the extension board, and an interrupt service routine `vdma_in()` that becomes active once per transferred image line. (Similar functions exist for image transfers from the extension board to the SBIP). The `dsp_getimage()` function can be called by the application software on the extension board, the SBIP or the host.

The operation of the image transfer is basically as follows:

a. The `dsp_getimage()` routine initializes the various registers that are used for the image transfer, and sets up a DMA transfer for the first image line from the FIFO to an internal line buffer. Furthermore it prepares the first transfer from the internal line buffer to the image memory already.

b. Every time after one image line is transferred, the interrupt handler `vdma_in()` becomes active and initiates the required transfer for the next video lines (if not all image lines are transferred).

CORRELATOR ROUTINES

Even though there was a C-source available for the grey value correlator application, some time was spent on enhancing this code: A part of this C source has been rewritten in assembly language, to achieve the desired execution time. The search algorithm from the existing source was left unchanged.
The correlator application consists now of two C-routines that contain the search algorithms, and two assembly language routines that perform the calculation of the normalized correlation coefficient These will be described in this sub paragraph.

The main (C-) routine `dsp_correlatecoarsefine()` performs the actual search algorithm as described in section 1.2. The search method performs multiple exhaustive searches. The `dsp_exhaustsearch()` function (the second C-source) performs an exhaustive search within a specified search window with a certain subsample factor. It returns the coordinates where the best matches were found.

The two assembly language functions, the `correlate1()` and the `correlate_next()` functions, are invoked from the exhaustive search routine.

Both functions calculate the normalized correlation coefficient between the template and an image detail with a certain subsample factor.
The `correlate1()` function calculates all the terms as described in (1.9), (1.10), and (1.12) (note: calculation of (1.11) is only performed once per exhaustive search).
The `correlate_next()` function makes use of the calculations that were done for the previous correlation coefficient. Therefor it only has to calculate the terms as described in (1.9) and (1.12). For the calculation of the terms in (1.10), the result of a previous calculation of these terms is used as described in appendix A. The `correlate_next()` function is therefor significantly faster.

Most of the calculations in these assembly language routines are performed using integer arithmetic. For these calculations no numerical inaccuracies will occur as long there is no overflow on variables. Since the summation loops perform maximum 2^{16} MAC's on a product of two 8-bit numbers, the result can be represented by a 32 bit (unsigned) integer. Since the DSP can accumulates 32 bit's, no inaccuracies will occur in the summation loops (formula's (1.9), (1.10) and (1.11)).
When (1.12) is calculated, care is to be taken since 3 subtractions have to be performed. Appendix G.2.3 shows that the values to be subtracted are 48 bits values. The 3 subtractions can not be performed in 40 bits floating point format: If the two variables being subtracted from each other are large and almost equal in magnitude, the result of the subtraction will be very small in magnitude. With a 40 bit floating point format, the relative error will become large in such cases. Therefor the 48 bit subtractions are performed with integer arithmetic. The results of these subtractions are casted to floats, then the normalized correlation is calculated using floating point math.

4.2.3 HOST & SBIP software

The previous paragraph explained the software that is running on the DSP. Most of the routines that were explained (eg. `dsp_getimage()`, `dsp_correlatecoarsefine()` and `dsp_exhaustsearch()`) can be called from the SBIP and the host via the software interfaces. Furthermore, there is some specific software for the host computer and the SBIP.
This concerns functions for booting the DSP and downloading templates from the SBIP to the extension board.

BOOTING THE DSP

After powering up the SBIP and the DSP extension board, both the SBIP and the extension board are to be booted. First the software for the SBIP is to be downloaded from the host, then the software for the DSP is to be downloaded.

The DSP provides special facilities for downloading software. After power up, reset logic on the DSP chip sets up a DMA transfer of 256 words which is meant for a special boot loader program. The software development tools of the DSP produce a so called "loader file", that contains the user code for the DSP and the initialisation data for variables. Furthermore, this loader file contains a the small boot loader program of 256 words, that takes care that the user program code blocks are placed at the appropriate memory locations, and that variables are

being initialized. Thus to boot the DSP, the host computer just downloads this loader file to the DSP.

Alternatively, it is also possible to configure the extension board for EPROM booting, by installing a jumper on the appropriate location. In such case, the DSP will be booted automatically after a chip reset. The DSP detects the presence of the configuration jumper, and performs the DMA transfer from the boot memory device instead of from the host computer.

DOWNLOADING TEMPLATES

The grey value correlator application requires that a template can be downloaded from the TMS 34020 (or host computer). A routine was written to download templates to the DSP extension board. It downloads a template that is stored in the SBIP memory, to the extension board using a host IOP DMA transfer as explained in paragraph 4.1.2.

The `dsp_gettemplate()` routine is available on the host and SBIP only. It is not callable by an application that is running on the extension board, because it involves that the template is downloaded by the TMS 34020. As the DSP is a slave of the TMS 34020, it can not order the TMS 34020 to download the template.
The `dsp_gettemplate()` function being called on the host, passes the function parameters via the communication buffer to the SBIP, where the actual routine is invoked.

4.2.4 Porting SBIP software to the extension board

The software of the extension board is user-extendable in the same manner as the user routines on the SBIP. Certain function ID's are reserved for user routines. If the command handler on the extension board receives one of these function ID's, it passes it to function `dspuser()`. This function can be edited by the user. User functions are added by placing the call to the appropriate function in the switch() - case structure as figure 4.3 shows:

```
void dspuser(short functid)
{
     .
     .
   switch (functid)
   {
      case 100:
          userfunct1(...);
          break;
      case 101:
          my_function(dsp_parameter(0),dsp_parameter(1), ...);
          break;
          .
          .
   }
}
```

Figure 4.3 : Example of adding a user function to the DSP library.

The bold text shows how the user routine `my_function()` is added. By recompiling this source, and linking the object code of `my_function()` with the rest of the object code for the extension board, a new loader file will be obtained that contains the user function. After this file is downloaded to the

extension board, the function is available on the DSP extension board. It has function ID 101 assigned to it.

Furthermore, the host and TMS library contain a function `invokedsp()` that accepts a function ID that is to be called on the extension board.
Thus `myfunction()` on the DSP extension board can be called from the host computer (or the SBIP) by setting the parameters in the parameter buffer and calling `invokedsp(101)`.

This method of adding user routines was adapted from the SBIP software. Therefor, it should be fairly easy for application programmers with SBIP experience, to add their user routines on the extension board. However, here is noted that still a lot of work needs to be done, since not every basic function of the SBIP is present in the DSP software package. This might result in the fact that the linker will report missing object modules.

5 Test results

A number of characterics of the extension board had been tested after some prototypes were built. The most important tests will be described in this chapter. These tests concern mainly the TMS 34020 interface, the video interface and verification of the grey value correlator algorithm.

The tests were done in the following order:

a. In the first place, DSP operation and the connection with the SRAM was tested. This tests concerned a small test program that writes data into the external SRAM, reads it back and verifies whether this value is the same. This test was done with the emulator, which offered the possibility to download programs via the JTAG interface. The object of this test was to verify that a the DSP operates properly and that the interface to the SRAM is OK.

b. Then the TMS 34020 interface was tested which will be explained in more detail in section 5.1.

c. The final test of the hardware was the image transfer operation. This will be explained in section 5.2.

d. After the hardware seemed to be working correctly, correlator algorithm was tested, this will be covered in section 5.3.

5.1 THE TMS 34020 INTERFACE

The object of the tests of the TMS 34020 interface were to verify it's correct operation and the transfer rate that can be achieved when data is downloaded from the SBIP.

The correct operation was tested by writing data to the extension board memory, reading it back and verify whether the written- and read- data is the same.

After the interface was found to be working properly, some transfer rates were measured. Since the transfer rate depends in certain cases on the abilities of the host computer, some theoretical rates will be given and some measured transfer times.
The theoretical rates are derived from the required time for the extension board to complete a host access.

The transfer rate is to be measured for both the host-DMA and IOP-DMA transfers. The following tables shows various properties of the host DMA transfers.

Operation	Host DMA	IOP-DMA	
Read/Write transfer time (1)	375	250	ns
Max.transfer rate from SBIP2@32MHz (2)	approx. 1M	approx. 1M	words/s
Download rate of template (3)	N/A	400k	pixels/s
Extensionboard software download rate 386SX@20MHz (4)	55k	N/A	words/s
Extension board software download rate 486DX2@66MHz(4)	to be measured	N/A	Mwords/s

notes:

(1) The transfer time represents the required time when the TMS 34020 generates the memory address until the extension board generates the ready pulse.

(2) These transfer rates are achieved with an assembly language program on the TMS 34020 that writes data to the extension board.

(3) Download with `dsp_gettemplate()` function of (software version 0.9)

(4) Download with `dsp_boot()` function (software version 0.9)

The conclusion of this test is that the transfer rate is mainly determined by the SBIP or the host. The read/write transfer time shows that the extension board can complete data word transfers within 250/375 ns representing data rates of 4/2.6 M words per second.

The maximum transfer rate of 1 Mwords/s from the SBIP2 (32MHz) is significantly lower. The reason for this is probably that the SBIP also has to fetch its program from memory. Perhaps an optimisation might increase this performance, however, since this was not of major importance for the operation of the greyvalue correlator, it was not yet carried out during the graduation project.

The next performance measure, the download rate of a template, is somewhat worse again. The reason for this is that program on the SBIP that performs the transfer, has to pack two pixels together for each word transfer. The download rate is rather slow: 163 ms to download a template of 64k pixels. However, by optimizing the transfer program on the TMS 34020 a transfer rate of 1M pixels/s should be reachable.

The download rate of the extension board software from a host (PC 386SX 20MHz) is about 50kwords/s This transfer rate is this low due to the slow host. Anyway, for most applications, this rate will be acceptable since this is only of importance during power up.

5.2 THE VIDEO INTERFACE

The image transfer operation on the extension board is partially performed by the DSP's DMA controller and partially by the hardware on the extension board. It is of major importance that image transfers are being performed correctly in the various transfer modes. The tests were mainly done to verify the correct operation.

To verify the image transfer operation, a special image was defined in the SBIP's video memory. This image was defined such that potential errors (like missing pixels, horizontal or vertical image shift, swapped even and odd frame etc.) could be detected.

The test was performed such that the DSP extension board grabs an image from the SBIP and that it verifies the entire contents of the image.
During the graduation project, only the interlaced transfer operation was tested for image sizes of 512 x 512, 512 x 768 and 512 x 1024 pixels (representing transfer rates of 10, 15 and 20 MHz respectively).

The required transfer time for the image is basically 40 msec. for a CCIR image that is transferred according to the CCIR standard, and 36 ms for the RS 170 standard. Since the image transfer actually starts after the first occurrence of a Vsync pulse, and a transfer is generally initiated at an arbitrary time, the execution time in practical situations ranges from 40-60 ms (CCIR) or 36-54 ms (RS170) since the software awaits the first occurrence of a Vsync pulse before it starts the actual image transfer.

After endurance tests proved that over 10000 images were transferred without errors were transferred, the grey value correlator algorithm was tested.
This will be explained in the following section.

5.3 THE CORRELATOR ALGORITHM

The normalized correlation algorithm was tested for correctness and the execution time for various function parameters.

The correctness was tested by performing the normalized correlation on the SBIP and on the extension board on the same image. After each correlation, the reported results from the DSP extension board and those of the SBIP were verified to make sure that they are equal.

The execution times were measured with the cycle counter in the DSP's emulator and verified with a logic analyzer. The execution times of the SBIP were measured via software with the real-time clock in the PC.
In paragraph 3.4.1 some estimations were made about the execution times of the grey value correlation.

The following table shows these estimated times, the measured execution time on the DSP extension board, the measured execution time on the SBIP (32 MHz) and the acceleration of the extension board with respect to the SBIP :

Operation:	Estimated [ms]	Measured [ms]	SBIP [ms]	Acc.
Normalized correlation: 200 x 200 template in 512 x 512 image (1 local search)	90	88	10100	115
Normalized correlation: 200 x 200 template in 512 x 512 image (2 local searches)	111	106	12430	117
Normalized correlation: 200 x 200 template in 512 x 512 image (3 local searches)	132	124	14800	119
Normalized correlation: 256 x 256 template in 512 x 1024 image (3 local searches)	275	261	31760	121
normalized correlation: 150 x 150 template in 512 x 768 image (3 local searches)	116	110	12810	116

Notes:
- All global searches are performed with a factor 8 subsampled templates and images.
- With 1 local search is meant that is searched with the three subsample factors 6 2 and 1 consecutively

This table shows that the estimated execution times on the DSP extension board approach the actual execution time rather well. (note the estimation of the execution times for the various image and template sizes were estimated with a spreadsheet that was also used for the estimation of the execution times in appendix C.2).

In comparison with the execution times of the algorithms on the SBIP, the extension board achieves a considerable acceleration of the algorithm. This acceleration is achieved mainly in the summation loops.

In the summation loops. the following influences play the most important role:

a. a multiply accumulate on the TMS 34020 takes 10 machine cycles, versus 1 machine cycle on the DSP.

b. The DSP has no loop overhead in the most inner loop: It performs indexing through the images, loading the pixels into registers in parallel with the computations. On the TMS 34020 this happens sequentially and analysis of the assembly listing showed that this introduces 17 ! machine cycles overhead per multiplication.

c. a machine cycle of the DSP is more than 4 times as short (30 ns for the DSP versus 125 ns of the TMS 34020).

Since mainly summation loops are done during a greyvalue correlation, a factor 4.17 x 27 = 112.5 would be expected to be won.

Perhaps one might remark that this comparison is not really fair, since the correlator algorithm on the extension board contains a few programs that were coded in assembly language, while the algorithm on the SBIP was coded entirely in C.

To perform a fair comparison, one can choose in fact two different ways define the performance improvement.

a. One can compare the capabilities of the TMS 34020 and the ADSP 21062 chips. This comparison is independent of the programming language.

b. One can compare the execution time when one and the same C source is compiled and ran on both processors. This comparison compares the compiler efficiency and the capabilities of both chips (for the particular algorithm in this C source).

On one hand, the generated assembler file from the TMS 34020 compiler showed that no improvements could be made to the assembler code to make it more efficient. Thus the performance improvement as shown in the table is in fact a measure for the performance of the processors for the greyvalue correlator algorithm.

On the other hand, the C source for the greyvalue correlator that was used on the SBIP was compiled with the DSP compiler. This resulted in the fact that the execution time of the greyvalue correlator on the extension board dropped by a factor 10 !
The code that was generated by the compiler performed various actions sequentially even though the DSP could perform it in parallel, and is therefor significantly slower.

The conclusion from this comparison is that the extension board does give a significant performance improvement for the greyvalue correlator.
To achieve the best performance improvement for an arbitrary vision algorithm, it will in certain cases be required to write time-critical programs in assembly language. Especially if the parallel processing capabilities of the DSP are to be exploited, assembly language will improve the performance significantly.

Finally is noted that the performance improvement will depend on the application. Many applications will not achieve a performance improvement of a factor 115 like in the greyvalue correlator. This depends on the fact how well the application suits the typical DSP operations.

6 Effects of subsampling and pre-filtering on correlator performance

As concluded in section 1.2, the grey value correlator algorithm has to limit the amount of data in order to execute the algorithm within an acceptable time. To achieve this, the search algorithm subsamples the image and template, that are spatially sampled with frequency f_s, by a factor s. This will result in the fact that spatial frequencies larger than $f_s / 2s$ will be folded into the primary frequency spectrum, leading to distortions. The question is what consequences these distortions might have on the search algorithm. If the template and image are not pre-filtered, essential information might be lost during subsampling, which might cause the search algorithm to fail: the object as specified in the template will not be found, even though it is present in the image.

Other questions are whether filtering the template and image is required (this is the case if filtering improves the performance of the search algorithm), and how one has to filter. Or is it perhaps possible to divide images into certain classes that require filtering and classes that do not.

The available time for the investigation was limited and therefor the questions as stated before could not all be answered. Therefor the investigation was limited to the following subjects:
Section 6.1 will give an introduction on the subsample effects and the correlator performance measures. Then section 6.2 describes the effects that occur during subsampling the template and image.
Section 6.3 gives in accordance with section 6.1 a brief description on the probability of false alarm, this is to be investigated in more detail in the future.
The probability of detection will be covered in sections 6.4 and 6.5: section 6.4 describes some approaches that did not result in useful conclusions and section 6.5 shows how cross correlation products can be used to design a filter that can be used to increase the probability of detecting the template.

6.1 INTRODUCTION

In the grey value correlation algorithm, a global search is performed for raw estimation of potential template matches. Since these potential matches are based on a search with the subsampled template which contains only a small amount of information from the actual template, the outcome of the local searches can result in either of the following possibilities:

a. A more accurate estimation of the location of the object in the image, in case the actual image detail matches the object in the template.
b. A very low correlation coefficient in case the actual image detail turns out to be different from the template as the algorithm searches with lower subsample factor.

Since the number of calculations can be decreased by a factor s^4, when a (2 dimensional) image is subsampled by a factor s, it will be clear that an as high as

possible subsample factor is desired to obtain a short execution time. The subsample factor of the exhaustive search is of major importance since this search covers the entire image. (The subsample factor of the local searches are of less importance since on one hand they are performed over a relatively small area and on the other hand they need to yield accurate position information, thus they just can't be subsampled too much).

Since subsampling is applied to reduce the execution time of the grey value correlation, it will be obvious that filtering is not really desired since this will increase the total execution time of the filter operation plus the grey value correlation. (Some estimations of an 8 x 8 convolution filter in a 512 x 512 window on the extension board showed an execution time of approx. 750 ms. which is 4-5 times as long as the grey value correlation). On the other hand, one might decide to filter only the template, and perform the exhaustive search with the filtered template. Since the template is smaller than the image, filtering will take lesser time. And besides that, one can filter the template already at the time that it is being defined, and store both the filtered and the non-filtered template. Thus during run-time of the grey value correlation, only some time might be lost to download the filtered template (in addition to the unfiltered template) to the extension board.

In the case that filtering is not really desired, another analysis has to be done in future, in order to determine the maximum subsample factor that can be applied to a certain template during the exhaustive search (and in cases where the maximum subsample factor is smaller, one might decide to filter). The conclusion is that the effects that occur during subsampling need to be analyzed. This chapter makes a start of the analysis of subsampling effects and the effect of filtering.

This section introduces the reader into the subject by first explaining the correlation procedure where no subsampling is applied in paragraph 6.1.1. Then paragraph 6.1.2 briefly describes a search that was done in literature for publications on the subsample effects. At last paragraph 6.1.3 will describe the correlator performance.

6.1.1 The correlation procedure

Before the subsampling effects are being analyzed, this paragraph will first describe the correlation procedure for a signal that is not subsampled, to make the reader more familiar with the subject. For simplicity, the analysis of the subsample effects is done one dimensional.

The template t consists of N pixels t_u,

$$\begin{aligned} &\textit{with} \quad (u \in Z \cap 0 \le u < N) \\ &\qquad t_u \in \mathbf{R} \end{aligned} \tag{6.1}$$

the image i consists of Ni pixels i_u,

$$\begin{aligned} &\textit{with} \quad (u \in Z \cap 0 \le u < Ni) \\ &\qquad (Ni > N) \end{aligned} \tag{6.2}$$

furthermore for the image holds:

$$i_u = \begin{cases} x_u & (0 \le u < m) \cup (m+N \le u < N_{img}) \\ t_{u-m} & (m \le u < m+N) \end{cases} \qquad x_u \in \mathbb{R} \tag{6.3}$$

the image contains "random" information x_k, at location m it contains the actual template t.

During the exhaustive search, the correlation coefficients r_j are calculated, where j runs from 0 to Ni-N:

$$r_j = \frac{N \cdot \sum_{k=0}^{N-1} t_k i_{j+k} - \sum_{k=0}^{N-1} t_k \sum_{k=0}^{N-1} i_{j+k}}{\sqrt{\left(N \cdot \sum_{k=0}^{N-1} t_k^2 - \left(\sum_{k=0}^{N-1} t_k\right)^2\right)\left(N \cdot \sum_{k=1}^{N-1} i_{j+k}^2 - \left(\sum_{k=0}^{N-1} i_{j+k}\right)^2\right)}} \tag{6.4}$$

Figure 6.1, shows a graphical representation of the correlation coefficient r_j as function of the position j.

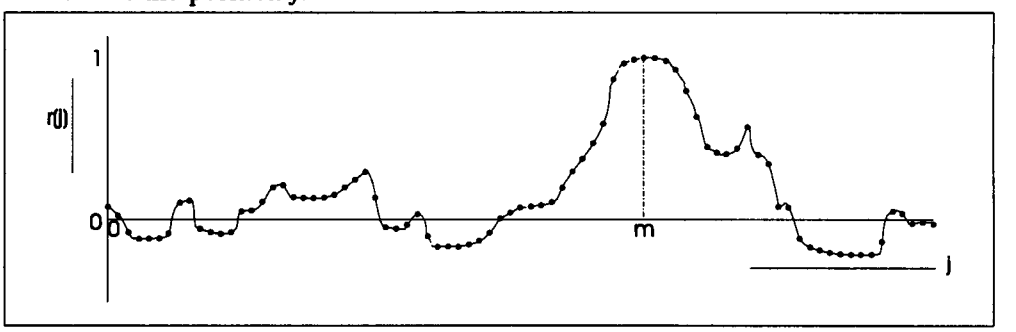


Figure 6.1: Correlation coefficient versus position.

At $j = m$ the correlation result will equal:

$$r_j = \frac{N \cdot \sum_{k=0}^{N-1} t_k t_k - \sum_{k=0}^{N-1} t_k \sum_{k=0}^{N-1} t_k}{\sqrt{\left(N \cdot \sum_{k=0}^{N-1} t_k^2 - \left(\sum_{k=0}^{N-1} t_k\right)^2\right)\left(N \cdot \sum_{k=1}^{N-1} t_k^2 - \left(\sum_{k=0}^{N-1} t_k\right)^2\right)}} = 1 \tag{6.5}$$

Since the template matches the image at position m, there will occur a correlation peak at that location. In the ideal case as described above, the correlation peak will equal 1, since the image matches the template perfectly.

In general, the image will not contain an exact copy of the template as specified in (6.2) and (6.3), but holds a distorted copy. The fact that the image contains a distorted template causes the correlation coefficient to be lower than 1.

The height of the peak depends on various parameters, in the list below they are divided into four main groups:

a. Geometrical distortion of the object in the image.
The object in the image might be different in geometry due to rotations, difference in size, small deviations in the shape of the object etc.

b. Presence of noise.
Noise will be introduced in the sensor, electronics, and during quantisation of the intensity, especially if the image has lesser contrast, the noise can play an important role in the calculation of the correlation coefficient.

c. Sampling effects.
The actual image consists of continuous information. In order to process the image, it is being sampled in the space domain with sample frequency f_s. If the continuous image contains spatial frequencies larger than $f_s/2$, the discrete image will be distorted.

d. Subsampling effects.
The correlators exhaustive search algorithm subsamples this discrete image to reduce the amount of calculations. The same effects as described before occur also in this case if the discrete image contains spatial frequencies larger than $f_s/2s$.

As the normalized grey value correlation is a nonlinear operation, it is not possible to analyze the influences separately, and add the results together afterwards. Only the effects that occur during subsampling are analyzed in this chapter. In all cases is considered that the object that is defined by the template is present in the image, and thus that it is free of noise, geometrical distortions and sample distortions. Thus an exhaustive search without subsampling the image would yield a correlation coefficient of 1 at the location m where the template is located.

6.1.2 Previous research

Before the investigation of the subsample effects was actually started, a literature search had been done to find out whether the subsampling effects in correlators had been subjected to investigation before.

A search through the INSPEC, Comendex, NTIS and Pascal data bases did not result in articles on subsample effects in correlators. However, some papers were found on the investigation of error analysis in correlation filters, and correlator performance both under most general assumptions. The articles were published in the period '89 - 94 and came from the area of optical correlators.

The papers on error analysis in correlators were straight forward for the reasons that not a normalized correlation but the covariance operation was analyzed under the assumptions that it concerned images containing gaussian data with additive gaussian noise. This approach was not usable for the normalized correlation coefficient on the extension board, since the normalized correlation is a nonlinear function and the input signals are most probably not gaussian.

The articles on correlator performance showed methods on defining the measure of the correlator performance.

The conclusion from the library search was thus that the found papers could only be used to get more familiar with the theory of error analysis in correlators. The actual analysis of the subsample effects in the normalized correlation had to be started from the beginning.

6.1.3 Performance definition

The aim of the investigation is to find out whether subsample effects influence the correlator performance and whether pre-filtering will improve this performance. Therefor performance needs to be defined first.

In machine vision there are basically two aspects involved in the performance: the robustness of the search and the accuracy of the found position.
With robustness is meant that the algorithm has to be reliable and should function correctly under each condition: In case the object as specified in the template is present in the image the algorithm should report it at the correct position. The algorithm should not report positions where the object is not located.
With accuracy is meant that the algorithm should report the exact location where the object is located without position errors.
In terms of the grey value correlator, the following remarks can be made to the robustness and the accuracy:

a. Robustness.
During the global search, both the image and the template are possibly distorted (eg. due to subsampling). These possible distortions might lead to the fact that the correlation coefficient at the location where the object is located is lower as expected. If the correlation coefficient is too low, the algorithm will not detect the object even though the object is in the image.
On the other hand, the correlation coefficient at a position where the object is not located, can perhaps become very high, and the search algorithm will interpret this as a potential match.

b. Accuracy.
Due to the distortions in the template and the image, the highest correlation coefficient will perhaps be found with a position error. However, the fine searches, will compensate this error, provided that the position error during the global search is not larger than the search windows of the local searches. If the position error during the coarse search exceeds the window of the fine searches, there will remain a position error after the final local search.
(Changing the local search to a gradient based algorithm would not introduce such problem, since such algorithm will search in the direction of the correlation peak until this peak is found, however, here only remains the risk that a local minimum will be found).

The subsample effects will play the most important role during the global search, since the subsample factor is the highest during this search. Since the object of the global search is to locate potential positions where the template matches the image, rather than yielding an accurate position, the robustness of the search is of major importance for the global search. Moreover, after the global search the local searches with lower subsample factors should compensate the position error that occurred during the global search. Therefor this chapter will focus on the robustness of the exhaustive search algorithm.

Various performance measures for correlators are described in [6]. From these measures, the robustness of the algorithm can be expressed the best in terms of probability of detection, and the probability of false alarm which is explained in more detail in [7]. (Since these papers concern optical correlators, here is also referred to [4] and [5], papers explaining optical correlators). The probability of detection specifies the probability that an object will be detected in case the object is present in the image detail where the correlation coefficient is calculated. The probability of false alarm specifies the probability that the algorithm assumes that it found the object while the image detail does not contain the object.

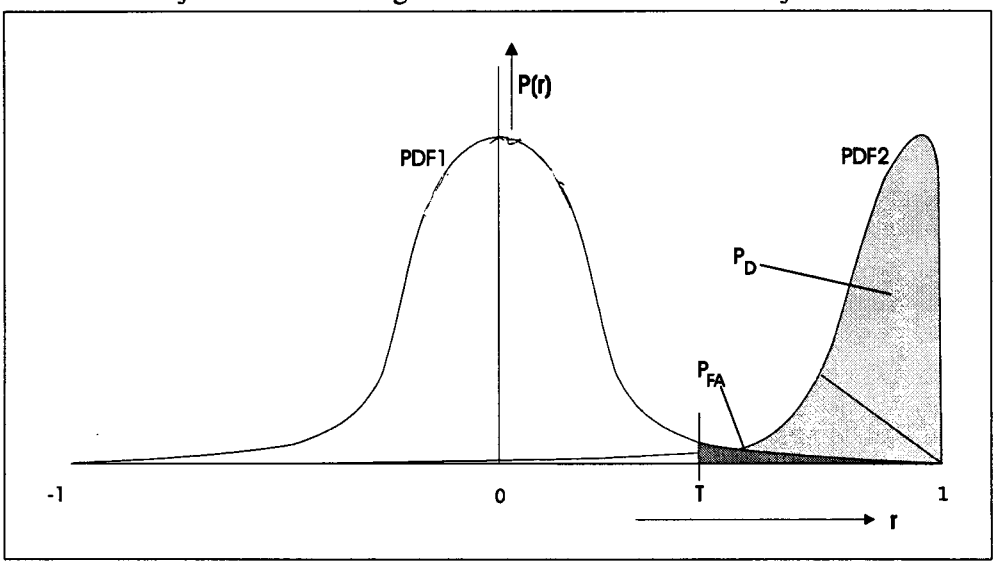


Figure 6.2 : Probability density functions for detection and false alarms.

Figure 6.2 shows two Probability Density Functions (PDF). In this figure, $r \in [-1,1]$ is the correlation coefficient as in (1.8). PDF1 represents the probability that a certain correlation coefficient will be computed at a location where the object is not present. PDF2 represents the probability that a certain correlation coefficient is computed at a location where the object is present. Generally a threshold T is used to decide whether a calculated correlation coefficient represents a match or not. The hatched areas above this threshold T specify the probability of detection P_D and false alarm P_{FA}.

The exhaustive search algorithm as described in section 2.2 does not use a fixed threshold, but instead keeps the 3 highest correlation coefficients found during the exhaustive search. Analysis of the probability of detecting the object in the image would actually require a different approach. For simplicity, the problem will be generalized for now by using a threshold to decide whether a computed correlation coefficient concerns a match or not.

Another point of discussion is the definition of PDF2: Correlation coefficients computed in the neighbourhood of the exact location of the object tend to be higher than those calculated at a location where the image and template are uncorrelated. The question is whether PDF2 is only valid at the exact position where the object is located, or it also includes the correlation coefficients computed within a certain region around that position. In this report only the exact position of the object is taken into account for PDF2.

It is obvious that the probabilities P_D and P_{FA} depend on the threshold that is used during the exhaustive search, and hence the correlator performance depends on this threshold. Therefor, the performance is commonly expressed in the Receiver Operating Characteristics (ROC).

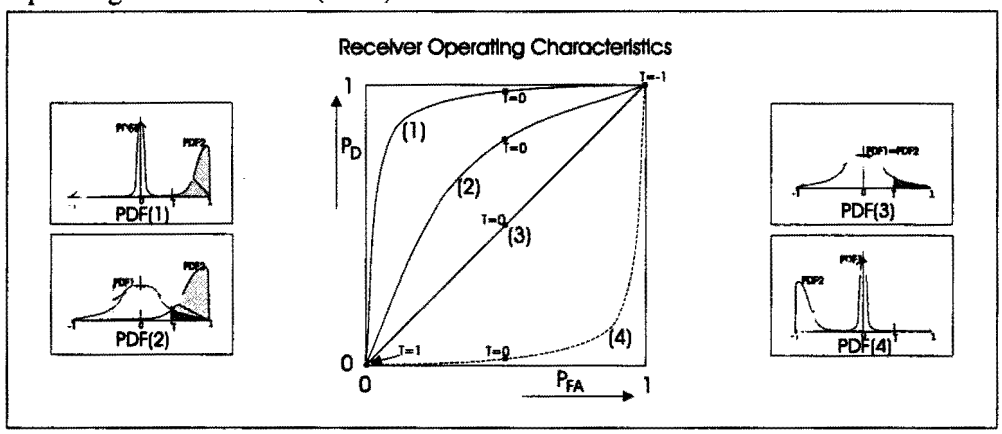


Figure 6.3: Receiver Operating Characteristics:
(1) = desired (2) = undesired.

Figure 6.3 shows an example of these characteristics. The ROC plots P_D versus P_{FA}, where the threshold T is varied from -1 to 1. In the ROC are drawn 4 different curves and next to it, the corresponding PDF's for each curve. It will be clear that curve (1) shows the desired characteristic: a large probability of detection with a small probability of false alarm for the threshold $T = 0.7$. Curve (2) is somewhat worse since it has a higher probability of false alarm for a certain probability of detection. Curve (3) is the worst possible situation since the probability of detection equals the probability of false alarm, independent of the threshold. Finally is noted that curve (4) is in fact also an example of a well performing "detector". A large probability of false alarms with a small probability of detection which is in fact the inverse of the desired operation: if eg. the estimated correlation coefficient is negated, image details with inverse grey values of those of the template will be found.

6.2 SUBSAMPLE EFFECTS IN THE CORRELATOR

To start the analysis of the problem, this section explains the effects that occur during subsampling.
First paragraph 6.2.1 discusses the effects that occur when the image is subsampled. In this section will become clear that there are more ways to specify the probability of detection, therefor, paragraph 6.2.2 will define the probability of detection that is used throughout this report. Finally paragraph 6.2.3 describes why the analysis was performed in the space domain rather than the frequency domain.

6.2.1 The subsample effects

Now the case is considered where both the template and the image are being subsampled. Subsampling involves a subsample factor s that denotes by which factor the template and image are subsampled, and a sampling phase p ($0 \leq p < s$) denoting which samples of the of the original image and template are taken.

Calculation of the correlation coefficient is done with the template t and a detail of the image i with the same length as the template.

Analysis of the probability that the template at location m will be detected, involves calculating the correlation coefficient between the template t and the image detail at position m.

The subsampled template is denoted as t' and consists of N_s samples t_u':

$$t_u'=t_{s\cdot u+p} \quad (u\in Z \cap 0\le u<N_s,\ 0\le p<s) \tag{6.6}$$

with:

$$s=sub\ sample\ factor$$
$$p=sampling\ phase\ (0\le p<s)$$
$$N_s=int\left(\frac{N-p+s-1}{s}\right)$$

(int(...) represents the integer value).

The subsampled image is denoted as i' and consists of Ni_s samples i_u':

$$i_u'=i_{s\cdot u+p} \quad (u\in Z \cap 0\le u<Ni_s,\ 0\le p<s) \tag{6.8}$$

with:

$$s=sub\ sample\ factor$$
$$Ni_s=int\left(\frac{Ni-p+s-1}{s}\right)$$

Here is assumed that the sampling phase for both the template and the image are 0.

For simplicity the normalisation terms are left out from the formula's for now: instead of denoting the correlation coefficient r, we denote the not normalized correlation (= covariance) with c. The calculation of the covariance c looks like:

$$c_{s,j}=\frac{1}{N_s}\cdot\sum_{k=0}^{N_s-1} t_{s\cdot k}\cdot i_{s\cdot(j+k)}-\left(\frac{1}{N_s}\right)^2\cdot\sum_{k=0}^{N_s-1} t_{s\cdot k}\cdot\sum_{l=0}^{N_s-1} i_{s\cdot(j+l)} \tag{6.10}$$

In this equation, the sample phases for the template and image are assumed 0. Since the probability of detection is being analyzed, the particular image information at location m in the image is of interest since that is where the template is located. Thus the image samples in (6.10) are to be substituted by the template samples. And when this is being done, the sampling phase p is to be taken into account: The position m at which the template is located in the image is arbitrary, thus if the image is subsampled with a sample phase 0, the template that is in the image will be subsampled with a certain (unknown) sample phase p. The sampling phase p, depends thus on the location m where the template is found in the image.

Assuming that the template (without noise and geometrical distortions) is present in the image, the following value for the covariance will be found during an exhaustive search with subsample factor s:

$$c_{s,p}=\frac{1}{N_s}\cdot\sum_{k=0}^{N_s-1} t_{s\cdot k}\cdot t_{s\cdot k+p}-\left(\frac{1}{N_s}\right)^2\cdot\sum_{k=0}^{N_s-1} t_{s\cdot k}\cdot\sum_{l=0}^{N_s-1} t_{s\cdot l+p} \qquad (6.11)$$

with $0\leq p<s$.

There are in fact s correlation coefficients that are possibly found. Which of these s correlation coefficients will be found depends thus on the position of the object within the image. Generally, the position of the object in the image is random and unknown. In case the offset variations of the template within the image are large with respect to the subsample factor, the probability that either of the s correlation coefficients will be found is equal.

6.2.2 Definition of the probability of detection

From the previous section, the definition of the probability of detection may seem obvious. However, by taking a close look at the correlation peak, one might remark that the probability of detection can be expressed in different manners.

Having the knowledge that one of the correlation coefficients as specified by (6.11) the probability of detection could be defined as a (weighted) sum of $c_{s,p}$ with $0\leq p<s$:

$$P_D(s)=\sum_{i=0}^{s-1} P(occurence\ of\ c_{s,i})\cdot P(c_{s,i}\geq T) \qquad (6.12)$$

In case the probability P(occurrence of $c_{s,i}$) is equal for each $0\leq i<s$, the probability of detection equals:

$$P_D(s)=\frac{1}{s}\sum_{i=0}^{s-1} P(c_{s,i}\geq T) \qquad (6.13)$$

One might remark that the probability of detection might be higher if i runs from $-s/2+1$ to $s/2$ for the reason that the absolute value of the sampling phase is smaller. This depends on the contents of the template and will be covered in paragraph 6.5.3.

The probability of detection as specified by (6.12) represents the probability that the template will be found with a position offset of maximum s pixels. This definition will be used throughout the report.
Note that the probability of detection can possibly be enhanced by allowing a larger position error during the global search as described in 6.1.3.

6.2.3 Frequency- versus Time- domain analysis

So far the computation of the correlation coefficient was performed in the spatial domain. The correlation coefficient can also be calculated in the (spatial) frequency domain. Especially for the purpose of analysis, it was considered whether it is convenient to do so, since both subsampling and filtering effects can be analyzed very well in the frequency domain. When the template is being subsampled, this will result in the fact that spatial frequencies $> f_s/2s$, will be folded into the fundamental frequency range which lead to distortions. In case the template is being filtered with eg. an ideal Low pass filter, one can guarantee that distortions due to folding will not happen, and hence analyze the improvement of the performance. However, this method was found to be disadvantageous for the following reasons:

a. There was no knowledge about frequency spectrums of templates, thus the only effects that can be analyzed is to take a certain template, calculate it's DFT and analyze the effects of filtering. Since the approach can only be used for templates from which the contents are defined and thus no general statements can be made about the effect of filtering, this was not really desired.
b. As became clear from paragraph 6.2.1, subsampling the image involves a sampling phase p, and the correlation error that occurs depends on this sampling phase. In frequency domain analysis, this sampling phase will result in a variation in the argument of the complex frequency spectrum. Taking these into account in calculations seemed to be more complicated: Analysis of sampling effects on a template would imply that first an DFT is to be performed to get the frequency spectrum. Then the complex spectrum is to be processed with a filter, then an IDFT is to be performed, this method of analysis seemed more complex than just working the data in the space domain.

For these reasons, the analysis was not done in the spatial frequency domain.

6.3 PROBABILITY OF FALSE ALARM

During the exhaustive search, a large number of correlation coefficients is being calculated at the various positions in the image. When analysing the probability of false alarm, the correlation coefficients at locations in the image where the template is not located are of interest.
One can take two approaches: the first one assumes that the data at those locations is uncorrelated with the template, the second one assumes that the data at those locations might be correlated with the template.
The two following paragraphs will give a brief description on these approaches.

6.3.1 Probability of false alarm with uncorrelated data

If one assumes that the information throughout the rest of the image is uncorrelated with the template itself, one will find a zero valued correlation coefficient in the ideal case, if it is computed at these locations. However, since calculation of the correlation coefficients involves a finite amount of data, the calculated correlation coefficients will generally not be zero and hence there is a probability that such value is higher than the threshold which results in a false alarm.

According to [3] p.509, the case where the cross correlation coefficient is 0, the sample product moment correlation coefficient r based on n pairs of observations will be related to the Student's - T distribution with n-2 degrees of freedom (which in turn approaches the gaussian distribution for large n). Figure 6.4 shows the Student-t distribution for two values of n, from this it becomes clear that the

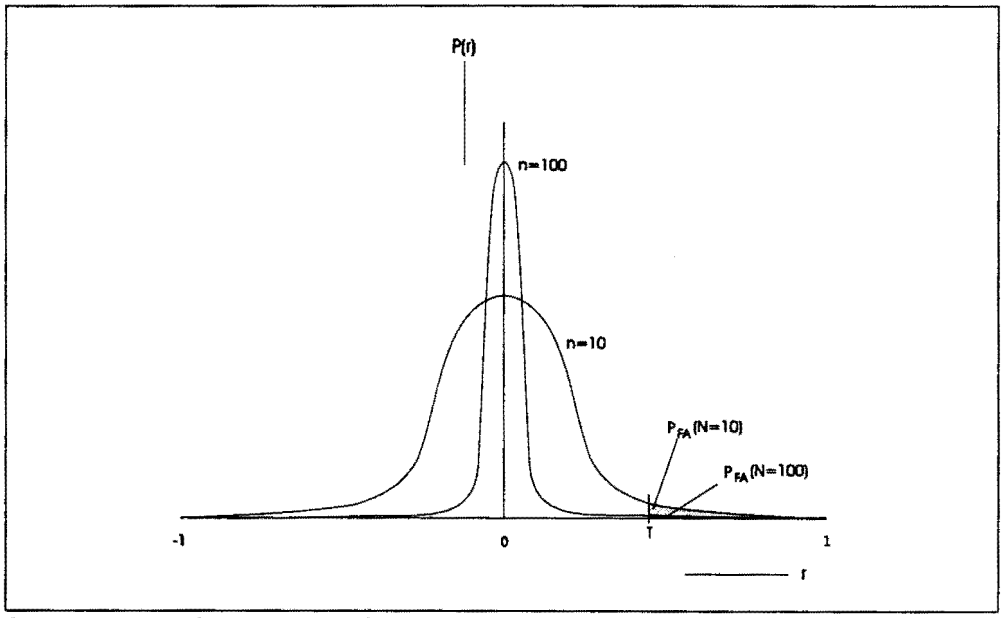


Figure 6.4: Student's - t distribution for two values of n.

probability of false alarm depends in such case thus on the sample count. In case the subsample factor s is chosen larger, the sample count will decrease and hence, the probability of false alarm will increase.

6.3.2 Probability of false alarms with correlated data

In the previous paragraph false alarms were assumed to be caused by uncorrelated information. However, in many machine vision applications, this will hardly be the case since certain patterns in the template will sometimes be found throughout the image. Just suppose the example where one tries to locate the letter **O** in an image that contains the entire alphabet. It will be obvious that the template will also be significantly correlated with eg. the letters **Q** and **G**. It will be clear that this greatly increases the probability of false alarm.

No attention has been payed to this problem since this requires a considerable amount of investigation, here is just pointed out that in many practical situations this will play an important role and should be taken into account if one wants to find quantitative values for the ROC of the grey value correlator.
Furthermore is referred to [8] and [9], two papers which may give some ideas to tackle this problem.

6.4 APPROACHES TO THE ANALYSIS OF PROBABILITY OF DETECTION.

To investigate the sampling effects and the effect of filtering, various approaches have been tried in order to draw conclusions. This section describes two approaches that did not result in useful information about the subsample effects, they are given just for reference.

First paragraph 6.4.1 describes the usage of statistical signal models, then paragraph 6.4.2 describes an attempt to reduce data during a correlation with a deterministic template.

6.4.1 Using statistical signal parameters for the template

In certain signal processing area's, various statistical signal parameters are known, which make it possible to perform statistical analysis of processing algorithms. These parameters concern the auto correlation function of the signal:

$$\rho_{xx}[p]=E\{x_k \cdot x_{k+p}\} \tag{6.14}$$

In which E{.} represents the mathematical expectation operator.
The question is whether these parameters are known for templates. To obtain this parameter, one needs ensemble averages of templates. Further the auto correlation function as described in (6.14) requires the data in the templates to be ergodic. The conclusion is that the template has to meet these conditions in order to use these signal models.

With the properties as described in (6.14), it can be shown that the correlation error and error variance depend on the sampling phase. In case the sampling phase p is 0, both the error and variance will be zero. In case of a non-zero sampling phase p, there will be an average error and error variance that can be expressed in terms of the auto correlation function $\rho_{xx}[p]$.

Since this approach requires knowledge of the statistical parameters about the template and this information was not available, this analysis was not continued. The investigation was continued instead by assuming that all data samples of the template were represented as variables, the following paragraph shows this approach.

6.4.2 Using a deterministic template

As the previous paragraph concluded that there are no statistical parameters of the template available, there was decided to model the template as a deterministic sequence of numbers (representing the grey values). Then an attempt was made to simplify the correlation error formulae for the following cases:
a. neither template and image are filtered
b. only the template is filtered
c. both the template and image are filtered

The correlation error is defined as the difference between the expected correlation coefficient (auto correlation of the template) and the actual correlation coefficient that is being calculated between the template and an image detail.

This paragraph shows the attempts that were made by using a deterministic template where the correlation error has been analyzed. For simplicity, the normalisation terms have been left out as before.

First a general description for the correlation error is defined. Here is assumed that the correlation coefficient is being calculated between a subsampled template T and the subsampled detail I_m' in the image. Both have length N_s. in the sub-paragraphs, T and I_m' will be substituted by the appropriate templates and images. (Here is noted that T and I_m' are not per definition the template and images as represented in paragraph 6.1.1)

In the ideal case, the covariance c_{exp} will be found:

$$c_{exp} = \frac{1}{N_s} \cdot \sum_{k=0}^{N_s-1} T_k \cdot T_k - \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} T_k \cdot \sum_{l=0}^{N_s-1} T_l \qquad (6.15)$$

The correlation coefficient c_{calc} will be calculated at location m in the image:

$$c_{calc} = \frac{1}{N_s} \cdot \sum_{k=0}^{N_s-1} T_k \cdot I'_{m_k} - \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} T_k \cdot \sum_{l=0}^{N_s-1} I'_{m_l} \qquad (6.16)$$

The correlation error equals c_{exp} -c_{calc}:

$$\epsilon_c = \left(\frac{1}{N_s} \cdot \sum_{k=0}^{N_s-1} T_k \cdot T_k - \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} T_k \cdot \sum_{l=0}^{N_s-1} T_l\right) - \left(\frac{1}{N_s} \cdot \sum_{k=0}^{N_s-1} T_k \cdot I'_{m_k} - \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} T_k \cdot \sum_{l=0}^{N_s-1} I'_{m_l}\right) \qquad (6.17)$$

This can be written as:

$$\epsilon_c = \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} \sum_{l=0}^{N_s-1} \left(T_{s,k} \cdot (T_{s,k} - I'_{m_{s,k}}) - T_{s,k} \cdot (T_{s,l} - I'_{m_{s,l}})\right) \qquad (6.18)$$

In the following sub-paragraphs the image details $I_m'_{sk}$ and templates T_{sk} in (6.18) will be substituted by the expressions of the filtered and non filtered versions of the image detail and template. In these cases is analyzed whether filtering can reduce the correlation error. The first sub-paragraph analyses the case where no filtering is applied.

NON-FILTERED TEMPLATE AND IMAGE

In case neither of the template and image are filtered, the analysis is based on the subsampled template (6.6), and the subsampled image around the location m as in (6.8).

Substituting (6.6) for T and (6.8) for I_m' results in:

$$I'_{m_{s\cdot k}} = t_{s\cdot k+p} \qquad T_k = t_{s\cdot k} \tag{6.19}$$

Substituting these in (6.18) yields the following results for the correlation error ε_c:

$$\epsilon_c = \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} \sum_{l=0}^{N_s-1} \left(t_{s\cdot k}\cdot(t_{s\cdot k} - t_{s\cdot k+p}) - t_{s\cdot k}\cdot(t_{s\cdot l} - t_{s\cdot l+p})\right) \tag{6.20}$$

It becomes clear that only for the case that p=0, the error ε_c is zero. In all other cases, there are no identical terms that cancel each other out (since $0 \le p < s$).

Thus in case the object is present in the image such that it will be subsampled with the same phase as the template is being subsampled, it will be found during an exhaustive search, regardless the spatial frequencies that are present in the template.

From this can also be concluded that the object will also be found within a maximum of s exhaustive searches each with a different sampling phase p. This is in fact one of the methods as described in section 1.2 to reduce the amount of data: up till now the combination of subsampling and enlarging the step size between two calculation coefficients, both with a factor s, were used to reduce the amount of computations (items b and c respectively in section 1.2). The example of s exhaustive searches as described above is in fact the situation were the image and template are subsampled by a factor s, but the step size between the calculations of two correlation coefficients remains 1 pixel. One can thus conclude that if the image and template are being subsampled by a factor s, and leaving the step size 1, this will reduce the amount of computation time by a factor s (for 1 dimensional case), while the probability of detection remains preserved.

In practical cases, however, the correlator performance will probably degrade as the subsample factor becomes higher. Since the image will contain noise, there will always remain a correlation error, and the larger the subsample factor, the fewer data samples are used to compute the correlation coefficient, the larger the uncertainty on the calculated correlation coefficient.

The conclusion is that only for the case were the sampling phase is 0, the correlation formulae can be reduced to a trivial solution. If the sampling phase differs from 0, the formulae do not contain identical terms and can thus not be reduced to more simpler values. Only if assumptions are made about the signal properties, eg. $E\{t_{sk}\} \approx E\{t_{s(k+1)}\}$, one can perhaps conclude that the error will be very small. Since this assumption is a rather extreme case and the aim of the investigation is more to draw conclusions without assumptions about signal properties, the effect of filtering was investigated next. The aim was to compare the correlation error in such case with the expression for the correlation error that is described in this sub-paragraph. The next sub-paragraph analyses the case where only the template is filtered.

FILTERED TEMPLATE, NON-FILTERED IMAGE

This sub-paragraph considers the case where only the template is filtered with a convolution filter of length L. Also in this case, the correlation error (6.18) was analyzed, the following values are substituted in (6.18):

$$\begin{aligned} T_{s\cdot k} &= \sum_{i=0}^{L-1} T_{s\cdot k+i} \cdot h_i \\ I'_{m_{s\cdot k}} &= T_{s\cdot k+p} \end{aligned} \tag{6.21}$$

the correlation error equals then:

$$\epsilon_c = \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} \sum_{l=0}^{N_s-1} \left(\sum_{i=0}^{L-1} t_{s\cdot k+i} \cdot h_i\right) \cdot \left(t_{s\cdot k} - t_{s\cdot k-p} - \left(t_{s\cdot l} - t_{s\cdot l-p}\right)\right) \tag{6.22}$$

The conclusion after simplifying (6.22) was that under the conditions that a uniform filter is chosen, the expression contains some duplicate terms. However, it was not possible to draw a conclusion on whether the correlation error will be larger then the error in expression (6.20).

At last also the case, where both the template and image were filtered, was analyzed.

FILTERED TEMPLATE AND IMAGE

In case both the template and the image are filtered with a convolution filter of length L. The following values are substituted in (6.18):

$$\begin{aligned} T_{s\cdot k} &= \sum_{i=0}^{L-1} t_{s\cdot k+i} \cdot h_i \\ I'_{m_{s\cdot k}} &= \sum_{i=0}^{L-1} t_{s\cdot k+i+p} \cdot g_i \end{aligned} \tag{6.23}$$

the correlation error term equals now:

$$\epsilon_c = \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} \sum_{l=0}^{N_s-1} \left(\sum_{i=0}^{L-1} t_{s\cdot k+i} \cdot h_i\right) \cdot \left(\sum_{i=0}^{L-1} t_{s\cdot k+i} \cdot h_i - t_{s\cdot k-p} - \left(\sum_{i=0}^{L-1} t_{s\cdot l+i} \cdot h_i - t_{s\cdot l-p}\right)\right) \tag{6.24}$$

When this expression was written out, it showed quite a number of duplicate terms if:
a. The filters G $(=(g_0, g_1 \ldots g_{L-1})^T)$ and H $(=(h_0, h_1 \ldots h_{L-1})^T)$ were equal.
b. The filters are uniform FIR filters of length s.
But also in this case it was not possible to draw a conclusion whether the resulting error will be larger then the error that was calculated in (6.20).

The conclusion from the procedures as described in this paragraph was that it didn't seem possible to reduce the expression for the correlation error. Therefor another approach was chosen, which will be explained in the following section.

6.5 CROSS CORRELATIONS FOR EXPRESSING THE PROBABILITY OF DETECTION.

From the previous section it became clear that a deterministic template is to be used during the analysis of the problem. The approach of paragraph 6.4.2 yielded large expressions for the correlation error that made it difficult to draw conclusions. The problem that is being analyzed is actually the one where the correlation coefficient is being calculated between two subsampled versions of a template, which are shifted by a few samples with respect to each other. This operation is very similar to the calculation of the auto correlation function of the template itself. Therefor the question was whether the auto correlation function could be used to obtain a measure of improvement, for cases were filtering is applied with respect to the case were no filtering is applied.

Figure 6.5 shows two plots of templates with their corresponding auto correlation functions. The plot of the template shows the grey value (intensity) versus the position. The template in figure 6.5a contains no fine details since there are hardly fluctuation in the grey values. The auto correlation function of such template has a large side lobe.

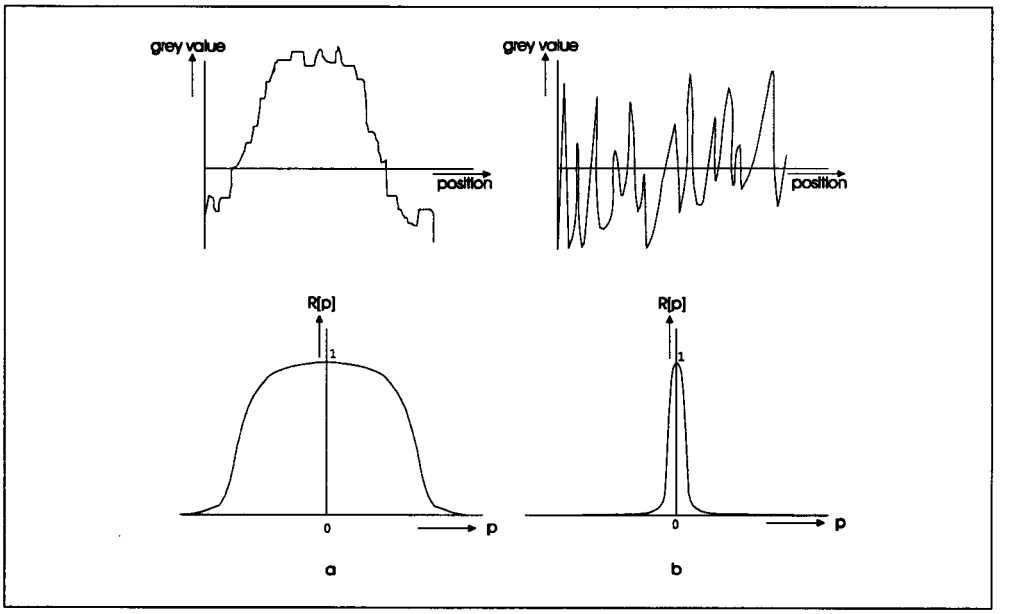


Figure 6.5: Grey value fluctuations with the corresponding autocorrelation function:

a: image contains no fine details b: image contains fine details

The template in figure 6.5b has many fine details because the grey values fluctuate heavily, the auto correlation function of such template has a very small side lobe.

If the two templates of figure 6.5 are being subsampled, it seems obvious that the shape of template in figure 6.5a remains preserved for higher subsample factors than the one in figure 6.5b because the template in figure 6.5b contains higher spatial frequencies.
Therefor it also seems that the subsampled template of figure 6.5a will have larger probability of detection than the one in figure 6.5b.

If one can proof that for templates with an auto correlation function as in figure 6.5a, this auto correlation function is a measure for the probability of detection, the analysis might be easier since it would involve a smaller amount of data.

Since the template is deterministic, its auto correlation function can be calculated and one can speak in terms of auto correlation functions rather than the various grey values of the pixels. This makes the amount of data significantly smaller, and perhaps this might make it easier to draw conclusions about a filtered template. Paragraph 6.5.1 discusses whether the auto correlation functions can be used to analyze the subsample effects of the template. The conclusion, however, is that the auto correlation function can only be used to estimate the correlation coefficient of the subsampled template that is found at location *m* during the exhaustive search. Therefor paragraph 6.5.2 will discuss the calculation of cross correlation functions between various subsampled versions of the template. The paragraph will show how one can design a filter to optimize the found correlation coefficients. Paragraph 6.5.3 will give some explanation on how one could optimise that filter. At last paragraph 6.5.4 will give a physical interpretations of the filter procedure.

6.5.1 Auto correlation function

To find out whether the auto correlation function is useful to analyze the effects of filtering the template, the auto correlation function of the template is being analyzed in this paragraph.
For simplicity, the normalisation terms will be left out again. An estimation of the auto-covariance of the template is defined as:

$$Ct[p]=\frac{1}{N}\sum_{k=0}^{N-1} t_k \cdot t_{k+p} - \left(\frac{1}{N}\right)^2 \cdot \sum_{k=0}^{N-1} t_k \sum_{k=0}^{N-1} t_{k+p} \qquad (6.25)$$

The estimated auto-covariance function for a subsampled template equals:

$$Ct_s[p]=\frac{1}{N_s}\sum_{k=0}^{N_s-1} t_{s\cdot k} \cdot t_{s\cdot(k+p)} - \left(\frac{1}{N_s}\right)^2 \cdot \sum_{k=0}^{N_s-1} t_{s\cdot k} \sum_{k=0}^{N_s-1} t_{s\cdot(k+p)} \qquad (6.26)$$

If the results *Ct[s k]* of (6.25) are being compared those of $Ct_s[k]$ in (6.26) (with $k = 0 .. N/s-1$), it will be clear that they might differ, since (6.26) contains not all data samples. Under the conditions that the template does not contain too many spatial frequencies larger then f_s, these results might be approximately the same and (6.25) could in such case be used to estimate (6.26):

$$Ct[s\cdot p] \hat{=} Ct_s[p] \qquad (6.27)$$

Note that the auto-covariance function estimated with (6.26) differs a factor *s* from the one that is estimated by (6.25).

Considering the subsampling effects that are being analyzed as in (6.11), we are basically interested in terms like:

$$C_{s,p}=\frac{1}{N_s}\sum_{k=0}^{N_s-1} t_{s\cdot k}\cdot t_{s\cdot k+p}-\left(\frac{1}{N_s}\right)^2\sum_{k=0}^{N_s-1} t_{s\cdot k}\sum_{k=0}^{N_s-1} t_{s\cdot k+p} \qquad (6.28)$$

These terms express the correlation coefficient of subsampled template with a phase difference smaller than s like it is possible in $Ct[p]$ in (6.25). Furthermore $Ct[p]$ might be used to estimate the correlation function $Ct_s[p]$ of the subsampled template under the mentioned conditions. So perhaps it is possible that the auto correlation function (6.25) can be used to estimate (6.28) under the conditions as mentioned before.

A closer look at (6.28), however, learns that we are actually calculating a covariance product: the two series t_{sk} and t_{sk+p} $(0\leq p<s)$ do not have any data samples in common for the cases that $p=0$ and are two different templates. On performs in fact a cross correlation between two different subsampled versions of the template.
A more accurate analysis can thus be done by defining the s possible subsampled versions of the template as being sub-templates T_p $(0\leq p<s)$ and using the covariance products between them, rather than the auto-covariance function (6.25) to estimate these covariance products. The following paragraph will use these considerations.

6.5.2 Cross correlation functions of sub-templates

In the previous paragraph was concluded that the cross correlation between so called sub-templates should be investigated. A sub-template is defined as a subsampled version of the template t as Ts_i and consists of N_s pixels:

$$Ts_{i_u}=t_{s\cdot u+i} \quad (u\in Z \cap 0\leq u<N_s) \qquad (6.29)$$

The problem being investigated can be formulated as follows:

If a template that is to be searched in an image is present in that image, and both the image and template are being subsampled by a factor s, one will find one of the following covariances during an exhaustive search with sub-template Ts_0, at the position m where the template is located in the image:

$$c_j=cov(Ts_0,Ts_j) \qquad (6.30)$$

with $0\leq j<s$.

Note: Here is assumed that sub-template Ts_0 is being used to perform the exhaustive search. In paragraph 6.5.3, where filter optimisation will be discussed, will be pointed out that Ts_0 will be the worst choice in many cases. For now, however, is continued with Ts_0.

In the following sub-paragraphs will be analyzed how filtering will influence the result c_j. This is considered again for the case where only the template, and then both the template and the image are filtered. The first sub-paragraph considers the case where only the template is filtered.

FILTERED TEMPLATE AND NON-FILTERED IMAGE

In this sub-paragraph, the case is considered where an exhaustive search is performed with a subsampled filtered template in a non filtered image.
The subsampled filtered template Y_0 that is used during the search consists of the pixels:

$$Y_{0_u}=y_{su} \quad with \quad y_i=\sum_{j=0}^{L-1} t_{i+j}\cdot h_j \tag{6.31}$$

Performing an exhaustive search in the subsampled image, results in occurrence of one of the following covariances at location m:

$$\begin{aligned} c_0&=cov(Y_0,T_0)\\ c_1&=cov(Y_0,T_1)\\ .&=.\\ .&=.\\ .&=.\\ c_{s-1}&=cov(Y_0,T_{s-1}) \end{aligned} \tag{6.32}$$

The covariance product that is to be calculated at location m equals:

$$c_j=\frac{1}{N_s}\sum_{k=1}^{N_s-1} t_{sk+j}\cdot\sum_{i=0}^{L-1} t_{sk+i}\cdot h_i-\left(\frac{1}{N_s}\right)^2\cdot\sum_{k=1}^{N_s-1} t_{sk+j}\sum_{k=1}^{N_s-1}\sum_{i=0}^{L-1} t_{sk+i}\cdot h_i \tag{6.33}$$

This can be rewritten as:

$$c_j=\sum_{i=0}^{L-1} h_i\left(\frac{1}{N_s}\sum_{k=1}^{N_s-1} t_{sk+j}\cdot t_{sk+i}-\left(\frac{1}{N_s}\right)^2\cdot\sum_{k=1}^{N_s-1} t_{sk+j}\sum_{k=1}^{N_s-1} t_{sk+i}\right) \tag{6.34}$$

And this equals:

$$c_j=\sum_{i=0}^{L-1} h_i\cdot cov(T_j,T_i) \tag{6.35}$$

the equations as in (6.32) can be written as:

$$\begin{pmatrix} c_0\\ c_1\\ .\\ .\\ c_{s-1}\end{pmatrix}=\begin{pmatrix} cov(T_0,T_0) & cov(T_0,T_1) & . & cov(T_0,T_{L-1})\\ cov(T_1,T_0) & cov(T_1,T_1) & . & cov(T_1,T_{L-1})\\ . & . & . & .\\ . & . & . & .\\ cov(T_{s-1},T_0) & cov(T_{s-1},T_1) & . & cov(T_{L-1},T_{s-1})\end{pmatrix}\cdot\begin{pmatrix} h_0\\ h_1\\ .\\ .\\ h_{L-1}\end{pmatrix} \tag{6.36}$$

Defining a vector $\underline{H}$ containing the filter coefficients h_i, a vector $\underline{C}$ containing the correlation coefficients c_j and the matrix C_u with elements:

$$C_u(r,c)=cov(T_r,T_c) \tag{6.37}$$

This can be written as:

$$\underline{C}=C_u \cdot \underline{H} \tag{6.38}$$

The formula above shows that the various correlation coefficients c_i can be manipulated with the filter coefficients h_i.
As each of the the covariances c_j before filtering, as described in (6.30), will be different the threshold that is used during the exhaustive search has to be chosen lower than the lowest value c_j in order to guarantee that the object will be found during the exhaustive search. If filtering the template with filter $\underline{H}$ can result into a new template from which the lowest value of c_j is higher then the one of the unfiltered template, the performance will be better, since the threshold can then be chosen higher, and the false alarms will be lesser.

Now the normalized correlation is considered, also in the situation where only the subsample effects are analyzed. To obtain the normalized correlation, the covariance is to be divided by the estimated sigma values of the template and the image detail:

$$r_{s,i}=\frac{cov(Y_0,T_i)}{\sqrt{cov(T_i,T_i)\cdot cov(Y_0,Y_0)}}=\frac{cov(Y_0,T_i)}{St_i \cdot Sy_0} \tag{6.39}$$

with: St_i the estimated sigma for sub-template i.
Sy_0 the estimated sigma for the subsampled filtered template
the equations as described in (6.35) are in such case:

$$r_j=\frac{1}{Sy_0 \cdot St_j}\cdot\sum_{i=0}^{L-1} h_i \cdot cov(T_j,T_i) \tag{6.40}$$

The following sub-paragraph will show that the estimated sigma value for the subsampled filtered template Sy_0 equals:

$$Sy_0=\sqrt{H^t \cdot C_u \cdot H} \tag{6.41}$$

The equations that are to be solved equal then:

$$\sqrt{H^t \cdot C_u \cdot H}\cdot St_j \cdot r_j=\sum_{i=0}^{L-1} h_i \cdot cov(T_j,T_i) \tag{6.42}$$

with j =0 .. s-1;

The following step is that the filter is to be optimized. The following paragraph will briefly discuss the optimisation of the filter. This paragraph is concluded with some examples.

Examples

First a one dimensional example will be considered where the template is being subsampled by a factor 4. A test was done with a template that consists of white noise. It will be clear that the cross correlation between two different sub-templates will be 0, and between the same sub-templates it will be 1.
The correlation coefficient at the position m, in fact the cross correlation between sub-template 0 and one of the four sub-templates, depends on the sampling phase. If the sampling phase is 0, the correlation coefficient will be 1 and in the other cases it will be 0.
Next a signal was defined just as described in (6.2) where the terms x_u contain white noise: it contains the template surrounded by white noise at the right and left end side.

If then an exhaustive search is done as described in (6.4) for the different possible sampling phases, one will obtain one of the four results as figure 6.6 shows (depending on the sampling phase): only in the case where the sampling phase is 0, a correlation peak occurs, in the three other cases the template will not be found. Thus one can not choose a threshold such that the probability of detection is 1.

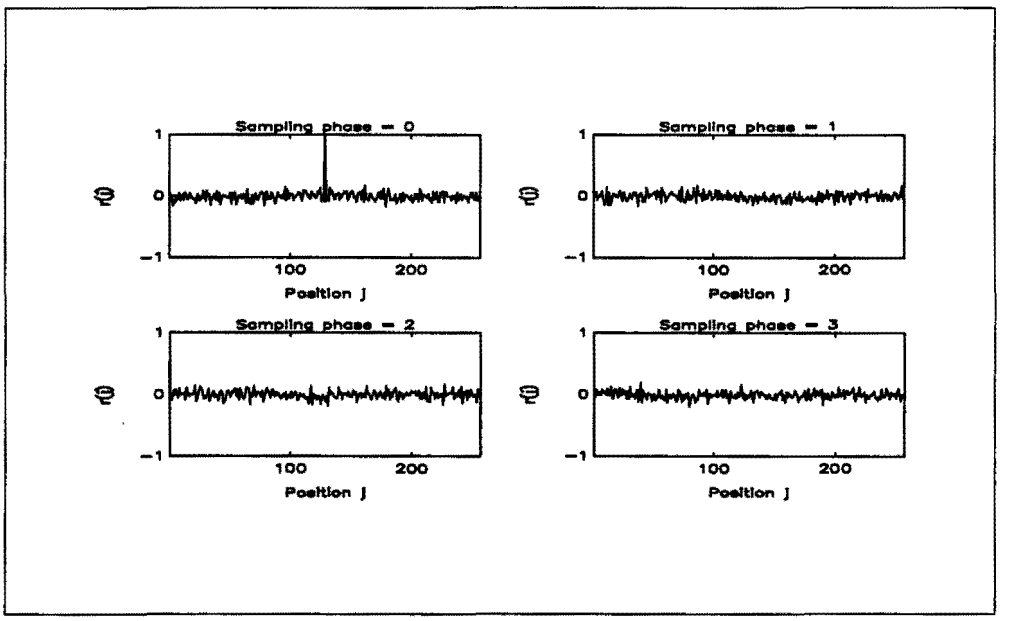


Figure 6.6: Exhaustive search with a factor 4 sub sampled, non filtered template for the 4 different sampling phases.

Then equation (6.40) was used to design a filter such that all correlation coefficients were equal:
This was done by assigning the sigma term Sy_0 to 1: and assuming the vector $\underline{R} = (r_0, \ldots, r_{L-1})^T$ to $(1,1,\ldots,1)^T$.

If one assumes the filter length L to be equal to s, one has to solve the following equations:

$$\begin{pmatrix} r_0 \\ r_1 \\ . \\ . \\ r_{s-1} \end{pmatrix} = \begin{pmatrix} cov(T_0,T_0)/St_0 & cov(T_0,T_1)/St_1 & . & cov(T_0,T_{s-1})/St_L \\ cov(T_1,T_0)/St_0 & cov(T_1,T_1)/St_1 & . & cov(T_1,T_{s-1})/St_L \\ . & . & . & . \\ . & . & . & . \\ cov(T_{s-1},T_0)/St_0 & cov(T_{s-1},T_1)/St_1 & . & cov(T_{s-1},T_{s-1})/St_L \end{pmatrix} \cdot \begin{pmatrix} h_0 \\ h_1 \\ . \\ . \\ h_{L-1} \end{pmatrix} \quad (6.43)$$

defining a matrix R_{tt} with elements:

$$R_{tt}(r,c)=cov(T_r,T_c)/St_r \quad (6.44)$$

This can be written as:

$$R=R_{tt}\cdot H \quad (6.45)$$

Solving these equations will result in equal values for r_i, however, they will be scaled by Sy_0.

The filter $\underline{H}$ that was obtained after the equations were solved was then used to filter the template. An exhaustive search with the subsampled filtered template resulted in the correlation functions in figure 6.7.

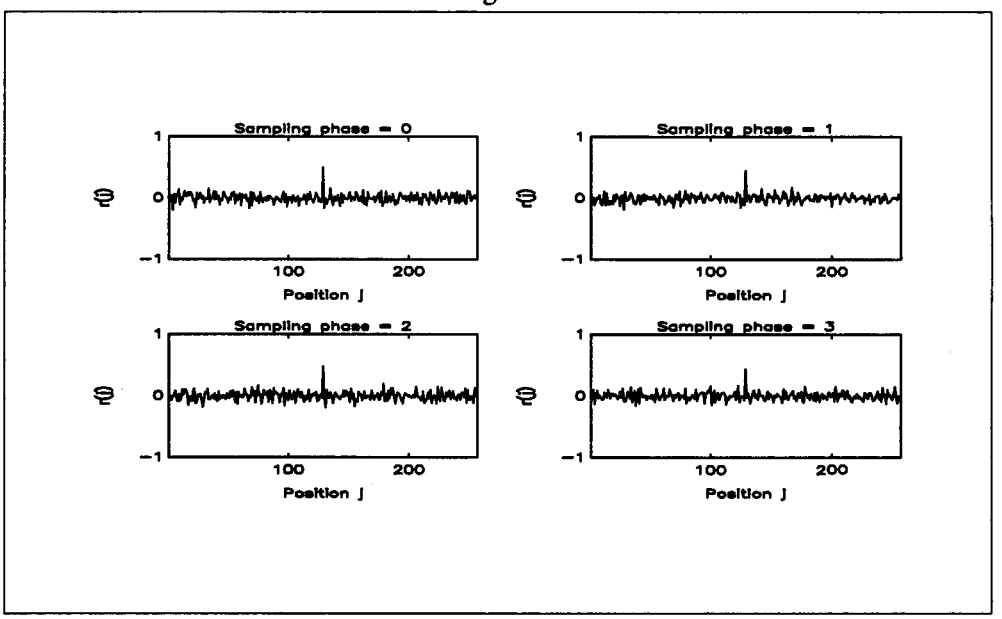
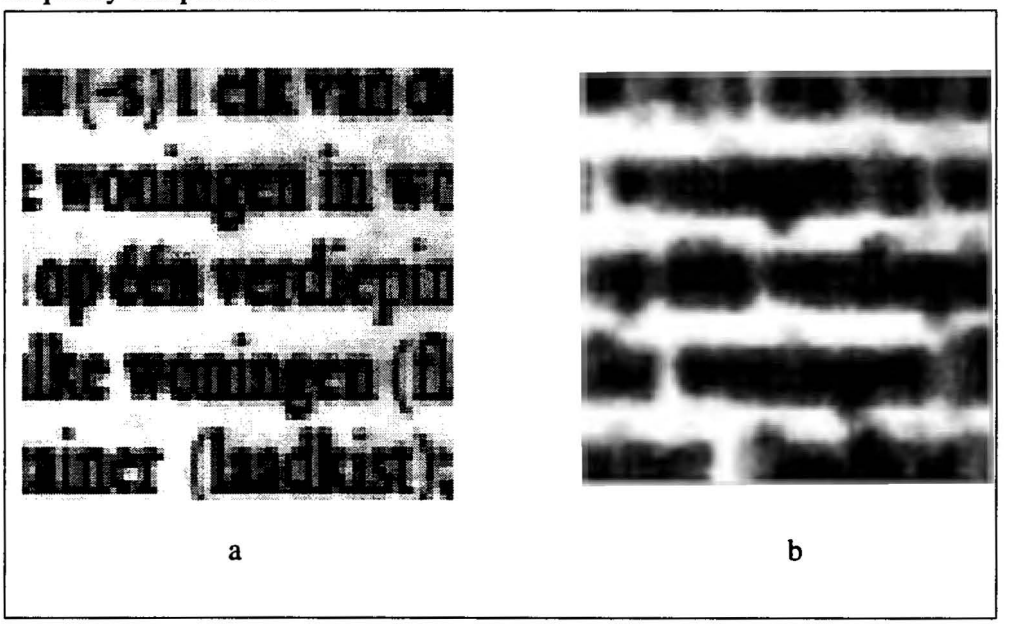


Figure 6.7: Exhaustive search with a factor 4 sub sampled, filtered template for the 4 different sampling phases.

The result of the filtering is that independent of the sampling phase there will always occur a correlation peak equal in height. However, the correlation peak is smaller than 1. By choosing the threshold low enough, one can achieve that the probability of detection is 1.
It is straight forward to prove that the filter $\underline{H}$ is a uniform FIR filter in case the template contains white noise. (In that case, matrix ***Rtt*** is an identity matrix).

It is also straight forward to extend the theory that was described so far to two dimensional situations, so similar test was done with an image.
Figure 6.8a shows a template that contains a considerable amount of high frequency components.



a b

Figure 6.8: Two dimensional example
a: original template b: filtered template

In case the template is subsampled by a factor 4, and the exhaustive search is done without filtering the image, one of the correlation coefficients as shown in the following table will be found: (note: since this concerns a 2 dimensional example so the sampling phase is to be specified in 2 directions)

sampling phase	0	1	2	3
0	1	0.74	0.64	0.60
1	0.67	0.47	0.39	0.39
2	0.45	0.34	0.23	0.26
3	0.12	0.02	0.00	0.02

Figure 6.8b shows the filtered template. When this is used for the exhaustive search, one of the following correlation coefficients is found.

sampling phase	0	1	2	3
0	0.49	0.50	0.49	0.51
1	0.60	0.62	0.63	0.60
2	0.52	0.59	0.60	0.56
3	0.54	0.60	0.60	0.55

Note: The differences between the various correlation coefficients are probably due to the limited dynamic range that could be used for the filter coefficients.

From this example can be concluded that the filtering can yield a better performance since the threshold can be chosen higher, resulting in a lower probability of false alarms.
Generally, one will choose the threshold during the exhaustive search lower as the minimum correlation coefficients between the sub-templates. Some of the sub-templates (sampling phase 3.1, 3.2 and 3.3) will not be detected when the template is not filtered (unless one chooses a negative threshold, resulting in a large probability of false alarms). By filtering the template, the minimum correlation coefficient between the sub-templates becomes 0.47. Thus by choosing a threshold of eg. 0.45, the template will be found, regardless the sampling phase. Here is noted that in practical situations where also noise and distortions play a role, there remains a probability that the template will not be detected !

FILTERED TEMPLATE AND IMAGE

Next, a case will be considered where both the image and the template are subsampled. Also this time, the covariance is being analyzed.
When both the image and template are filtered, the covariance looks as follows:

$$c_p=\sum_{k=1}^{N_s-1}\left(\sum_{i=0}^{L-1} t_{sk+p+i}\cdot g_i\right)\cdot\left(\sum_{j=0}^{L-1} t_{sk+j}\cdot h_j\right)-\left(\sum_{k=0}^{N_s-1}\sum_{i=0}^{L-1} t_{sk+p+i}\cdot g_i\right)\cdot\left(\sum_{k=0}^{N_s-1}\sum_{j=0}^{L-1} t_{sk+j}\cdot h_j\right) \quad (6.46)$$

This can be rewritten as:

$$c_p=\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} g_i\cdot h_j\cdot\left(\sum_{k=0}^{N_s-1} t_{sk+p+i}\cdot t_{sk+j}-\sum_{k=0}^{N_s-1} t_{sk+p+i}\cdot\sum_{k=0}^{N_s-1} t_{sk+j}\right) \quad (6.47)$$

and:

$$c_p=\sum_{i=0}^{L-1} g_i\sum_{j=0}^{L-1} h_j\cdot cov(T_{p+i},T_j) \quad (6.48)$$

This can be rewritten as

$$c_p=\sum_{i=0}^{L-1} g_i\cdot x_i \quad (6.49)$$

with:

$$x_i=\sum_{j=0}^{L-1} h_j\cdot cov(T_{p+i},T_j) \quad (6.50)$$

By defining the matrix $Ctt_{p,q}$ containing the elements:

$$Rtt_{p,q}(r,c)=cov(T_{r+p},T_{c+q}) \tag{6.51}$$

$\underline{X}$ can be expressed as :

$$\underline{X}=Ctt_{p,0}\cdot H=\begin{pmatrix} x_0 \\ x_1 \\ . \\ . \\ x_{s-1} \end{pmatrix}=\begin{pmatrix} cov(T_p,T_0) & cov(T_p,T_1) & . & cov(T_p,T_{s-1}) \\ cov(T_{p+1},T_0) & cov(T_{p+1},T_1) & . & cov(T_{p+1},T_{s-1}) \\ . & . & . & . \\ . & . & . & . \\ cov(T_{p+s-1}T_0) & cov(T_{p+s-1},T_1) & . & cov(T_{p+s-1},T_{s-1}) \end{pmatrix}\cdot\begin{pmatrix} h_0 \\ h_1 \\ . \\ . \\ h_{s-1} \end{pmatrix} \tag{6.52}$$

and the covariance between the filtered template and image (6.48) can be rewritten as:

$$c_p=\underline{G}^t\cdot Ctt_{p,0}\cdot H \tag{6.53}$$

For the s different covariances this yields the following system of equations

$$\begin{aligned} c_0&=\underline{G}^t\cdot Ctt_{0,0}\cdot H \\ c_1&=\underline{G}^t\cdot Ctt_{1,0}\cdot H \\ &\ .\quad\ . \\ &\ .\quad\ . \\ c_{s-1}&=\underline{G}^t\cdot Ctt_{s-1,0}\cdot H \end{aligned} \tag{6.54}$$

If the normalisation terms are also taken into account, the following system of equations is found for the normalized correlation:

$$r_{s,i}=\frac{cov(Y_0,Y_i)}{\sqrt{cov(Y_0,Y_0)\cdot cov(Y_i,Y_i)}}=\frac{cov(Y_0,Y_i)}{Sy_0\cdot Sy_i} \tag{6.55}$$

with Sy_0 and Sy_i the estimated sigma values for the filtered sub-templates 0 and i respectively.
each of these equations changes into:

$$r_i=\frac{1}{\sqrt{H^t\cdot Rtt_{i,i}\cdot H\ \cdot\ H^t\cdot Rtt_{0,0}\cdot H}}\cdot H^t\cdot Rtt_{i,0}\cdot H \tag{6.56}$$

Solving these equations is more difficult since they contain quadratic terms in the filter coefficients.

Since filtering the image was not really desired since this would take too much time in practical situations, this approach was not yet analyzed in further detail.

6.5.3 Filter optimisation

The previous paragraph showed that filtering can improve the probability of detection. Some simple experiments showed that the performance can be increased. However, there was not proven that it was the optimum solution. Neither was proven that filtering the template will always improve the performance. Therefor the filter is to be optimized first. After this is done, it is not unlikely that for templates that hardly need to be filtered (because the cross correlation coefficients are nearly 1), the resulting filter $\underline{H}$ will approach a delta pulse.
Another point of discussion is the length of the filter and which sampling phase is to be used for the template in the exhaustive search.

This paragraph only gives some ideas that can be used in future investigations.

FILTER LENGTH

In an example in the previous paragraph was assumed that the filter length L equals the subsample factor s. One might also decide to choose a shorter or a longer filter.

When a shorter filter is selected, one can generally not obtain the optimum solution. The problem that is to be solved in such case is in fact a best fit problem, since there are lesser equations than variables.

If a longer filter is used in combination with the s sub-templates, it will result in an over estimated system of equations. In such case, there are more solutions for the filter $\underline{H}$.

SAMPLING PHASE FOR TEMPLATE

In all the cases where subsampling effects were analyzed, the sampling phase for the template was assumed to be 0. A better practice is to choose that sampling phase for the template that the corresponding sub-template correlates the best with each of the other sub-templates. This information is present in the ***Ctt*** matrix (see (6.37)). The covariances in row n describes how sub-template n correlates with each of the other sub-templates.
An example of such a matrix is:

$$\boldsymbol{Ctt}=\begin{pmatrix} 1 & 0.8 & 0.8 & 0.6 \\ 0.8 & 1 & 0.8 & 0.8 \\ 0.8 & 0.8 & 1 & 0.8 \\ 0.6 & 0.8 & 0.8 & 1 \end{pmatrix} \qquad (6.57)$$

From this example it becomes clear that sub-template T_0 is not a good choice for the given template, since the correlation coefficient between sub-templates T_0 and T_3 is only 0.6.
Thus the threshold during an exhaustive search is to be set at a maximum of 0.6 to guarantee that the template will be detected.. Sub-template T_2 or T_3 would be a better choice since the minimum correlation coefficient for these templates equals 0.8.

Generally, the optimal sub-template to be used for the exhaustive search will be nearby $T_{s/2}$ since the maximum phase difference between that sub-template and one of the other sub-templates is then the smallest. However, this does not necessarily have to be the case!

The performance in two dimensional situations might further be enhanced by choosing a different subsample geometry. Eg. a different subsample factor in row and column direction in case the spatial frequencies in these directions are different.

FILTER OPTIMISATION

When optimizing the filter, one has to decide for which property is to be optimized. One of the most important properties is of course the probability of detection. If one uses expression (6.12) for the probability of detection, one should try to maximise the sum of the values $c_{s,i}$ for $0 \leq i < s$, which results in an expression with the various filter coefficients, the filter coefficients should be chosen such that this sum is maximum. However, only this property is not sufficient.

Another issue that is to be kept in mind is the minimum correlation coefficient between the sub-template that is used for the exhaustive search and the different sub-templates that can possibly be found in the image (this is the minimum value of the covariances in (6.40)). As mentioned already in the previous paragraph it seems likely that the threshold is chosen such that at least each of the sub-templates will be detected. Thus the lowest correlation coefficient dictates at which level one can set the threshold. The result of the optimisation that uses the probability of false alarms only, might be that s-1 correlation coefficients (after filtering) are very high, and the other one is significantly lower. In order to detect all sub-templates, the threshold has to be set lower than that lowest correlation coefficient. This will result in a high probability of false alarm. Thus in order to reduce the probability of false alarm this threshold is to be maximised. This can only be done by enhancing the minimum cross correlation between the various templates.

Thus both the probabilities of detection and false alarm are taken into account in optimising the filter, this should result in the fact that the minimum correlation coefficient will be higher, yielding a low probability of false alarms and a high probability of detection.

6.5.4 Physical interpretation

The previous paragraphs showed an approach on how the correlator performance could be enhanced. We can ask ourselves how this can be physically interpreted.

The case where only the template is filtered, is in fact the design of a matched filter. To explain this, the matched filter theory is briefly described:

The aim of a matched filter is generally to detect a certain signal that is corrupted with eg. noise. If the signal and the noise characteristics are known, one can design a filter such that the signal to noise ratio at the filter output is maximized, if that particular signal is at the input of the matched filter. One can prove that in case the signal to be detected is corrupted with gaussian

noise, this matched filter has an impulse response that is in fact the reversed version of the signal. And the implementation of such filter is in fact a correlator ! Thus the correlation that is done on the extension board is in fact a matched filter operation for an image that contains the template that is corrupted with white noise.

Let's now take the case where only the template is filtered as described in paragraph 6.5.2:
The filter operation with $\underline{H}$ on the template, changes the template and hence the sub-template that is used for the exhaustive search. This sub-template can in fact be considered as a matched filter, since it is optimized that it correlates such with each of the s sub-templates, that the correlator performance is maximum.
Thus where the unfiltered template represents a matched filter for an image that contains the template with additive gaussian noise and where the output signal to noise ratio is maximised, the subsampled filtered template represents a matched filter where one of the s sub-templates is to be detected in an image and where the correlator performance is maximized.

One can also consider the s sub-templates to be s different objects. In fact, the operation of filter $\underline{H}$ on these sub-templates results in a sub-template that is able to detect these s different objects. (more on this topic can be found in [6]). This results thus in a matched filter to detect different objects.

Finally is noted that this interpretation can also be exploited in another way:
Where the objects in the s sub-templates will probably look similar, one might also use this principle to detect completely different objects in a single exhaustive search. If the s sub-templates are substituted by s (not subsampled) images of different objects and a filter $\underline{H}$ is designed as described in 6.5.2, this will result in a template that contains features of the s different objects. If this is used for the exhaustive search, it is possible to detect these different objects in one exhaustive search. (In fact, the example with the white noise in paragraph 6.5.2 was such an operation since the 4 sub-templates were uncorrelated and hence can be considered as 4 completely different objects.)

A variant on this is of course the case were one specifically wants to detect an object A, but one specifically does not want to detect an object B. Considering these 2 objects as two sub-templates and generating a filter $\underline{H}$, one obtains a new template such that it correlates maximally with object A and minimally with object B. This variant can perhaps be used to suppress the probability of detecting objects that look very similar to the object that is actually being searched in an image. It will reduce the probability of false alarm in images that contain information that is more or less correlated with the template as explained in paragraph 6.3.2.

7 Conclusions and future investigation

7.1 CONCLUSIONS

The design of the extension board resulted in a small image processing module that can operate in parallel with the SBIP. The board offers the possibility to transfer images to and from the 1Mb image memory on the extension board at a transfer rate up to 20 Mhz.
The extension board, based on Analog Devices' ADSP 21062 DSP, improves the processing performance of the SBIP due to the high instruction rate of the DSP, the zero-waitstated image memory and the DSP's calculation capabilities.

The architecture of the extension board allows porting code from the SBIP fairly easy, thus vision functions that are currently running on the SBIP can be implemented on the extension board. Especially vision functions that perform intensive calculations and/or intensive memory usage, are expected to run significantly faster on the extension board. However, here is pointed out that in certain algorithms, assembly programming might be required to optimally use the DSP and achieve a significant performance improvement.

The normalized grey value correlation on the extension board, reduces the execution time for a search of a 200 x 200 template in a 512 x 512 window from 10 seconds on an SBIP, to 125 ms.

In the analysis of the subsample effects and pre-filtering, where the image was assumed to be free of noise and free of distortions, was shown that a filter can be designed that improves the correlator performance after this filter is applied to the template.
The principle of the filter design is that the subsampled template that is being used for an exhaustive search, is composed of the s^2 possible subsampled templates that can be obtained by subsampling the template by a factor s. This can in fact be interpreted as the generation of a matched filter for finding each of the s^2 different subsampled versions in the image in a single exhaustive search.

Since rather ideal conditions were assumed during the analysis, there is still a lot of work to be done for quantative measures on the correlator performance as will be described in the following section.

7.2 FUTURE INVESTIGATIONS

The analysis of the subsample effects and pre-filtering resulted in a method to design a filter that improves the correlator performance. However, still a lot of work needs to be done in future:

In order to determine quantitative measures for the performance, the probability of false alarms needs perhaps more investigation. In practical situations, the information throughout the image will be correlated more or less with the template as explained in paragraph 6.3.2. It will be clear that this approach needs significantly more investigation.

Then the filter optimisation is to be analyzed in more detail. Paragraph 6.5.3 showed an approach on how the filter could be optimised. This approach needs to be worked out. In the case where one analyses the effects of filtering on the correlator performance, the simpler approach for the probability of detection can be used as mentioned in 6.3.1. In practical cases of the synthesis of a filter for pre-filtering the template, the approach of paragraph 6.3.2 is perhaps better to use, since this might result in a filter such that the filtered template discriminates the possible unwanted objects and enhances the desired objects.

After these two topics have been analyzed, one should be able to prove whether filtering the template will always yield an improvement in correlator performance, or to specify which subsample factor can be used for a certain template in order to guarantee a certain performance.

When this work is done, one analyzed in fact only the subsample effects. As was pointed out in paragraph 6.1.3, the object in the image will generally also contain noise and geometrical distortions, which means that the results that will be found so far can not be used for practical situations. A next step would be, to take these noise and geometrical distortions also into account, which should yield a matched filter for the template with subsample distortion, noise and geometrical distortions.

Another subject of investigation might be the detection of multiple templates in an image, by composing a template from each of the templates to be found in the image as was pointed out in paragraph 6.5.4. And a variant to this is the optimal detection of one template and discrimination of an other template.

Referenced literature

[1] Bernsen, J.A.C.
NEW ALGORITHMS FOR OBJECT LOCALISATION AND INSPECTION.
Eindhoven : Nederlandse Philips bedrijven, 1996
Doc. nr.CTR596-96-0004

[2] WAARSCHIJNLIJKHEIDSREKENING VOOR E.
Faculteit Wiskunde en Informatica
Technische Universiteit Eindhoven 1992
Collegedictaat nr 2380

[3] Owen, J.
HANDBOOK OF STATISTICAL TABLES,
London : Addison-Wesly, 1962

[4] Vijaya Kumar, B.V.K. and Z. Bahri, L. Hassebrook.
CORRELATION FILTERS FOR OPTICAL PATTERN RECOGNITION,
Journal of the institution of Electronics and Telecommunication Engineers
Vol. 35 (1989), no.2, p.105-113.

[5] Vanderlugt, A.
SIGNAL DETECTION BY COMPLEX SPATIAL FILTERING,
IEEE Transactions on Information Theory, Vol. IT-10 (1964), p139-145.

[6] Vijaya Kumar, B.V.K. and L. Hassebrook
PERFORMANCE MEASURES FOR CORRELATION FILTERS,
Applied Optics, Vol. 29 (1990), no. 20, p. 2997-3006.

[7] Vijaya Kumar, B.V.K. and C.L. Chen, J.D. Brasher,
CORRELATION FILTERS ASSESSED IN TERMS OF PROBABILITY OF DETECTION AND PROBABILITY OF FALSE ALARM.
In: Proceedings of Optical Pattern Recognition IV. Orlando, Fl, USA. 13-14 April 1993.
Proceedings of the SPIE
(The international Society for Optical Engineering)
Vol. 1959 (1993), p. 12-22.

[8] Downie J.D.
REDUCED CLASSIFICATION ERROR PROBABILITY WITH TERNARY CORRELATION FILTERS.
In: Proceedings of Photonics for Processors, Neural Networks and Memories.
San Diego, CA, USA. 12 - 15 july 1993
Proceedings of the SPIE
(The international Society for Optical Engineering)
Vol. 2026 (1993), p 17-28.

[9] Downie J.D.
DESIGN OF TERNARY CORRELATION FILTERS TO REDUCE PROBABILITY OF ERROR,
Optical Engineering Vol. 33 (1994), no 6, p1777 - 1784.

A Complexity of an exhaustive search

This appendix describes the amount of calculations for an exhaustive search with a template T of R x C pixels in a window of L x M pixels.

The summation loops for the estimation power contents of the template pixels are only calculated once.

$$p=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}^2$$
$$q=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c} \qquad \text{(A.1)}$$

The operation takes R x C Accumulates (ACC's) and R x C Multiply ACcumulates (MAC's).

The summation loops for estimation of the template-image cross correlation properties:

$$x_{i,j}=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} T_{r,c}\cdot I_{i+r,j+c} \qquad \text{(A.2)}$$

require the most calculations. For one correlation coefficient, it takes R x C MAC's thus for the entire exhaustive search, this factor takes $(L - R + 1)$ x $(M - R + 1)$ x R x C MAC's. (Note that no calculations are done at the positions where the template exceeds the window).

The summation loops for estimating of the power contents of the image pixels:

$$y_{i,j}=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c}^2$$
$$z_{i,j}=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1} I_{i+r,j+c} \qquad \text{(A.3)}$$

have to be recalculated for every correlation coefficient. The amount of calculations can be reduced during an exhaustive search. Since the template is shifted by one pixel for every next correlation coefficient, there is an overlap between the area with image pixels $y_{i,j-1}$ and $y_{i,j}$ resp. $z_{i,j-1}$ and $z_{i,j}$) the summation values $y_{i,j}$ can be calculated from $y_{i,j-1}$ by subtracting the left most column (of $y_{i,j-1}$) from $y_{i,j-1}$, and then add the right most column (of $y_{i,j}$) to it. The same holds for $z_{i,j}$.

Something similar can be done when the template reaches at the left- or right-end of the window and it is shifted down to the next row.

In the most optimum solution, the minimum amount of MAC's (equals the number ACC's) equals L x M, however there will be some overhead.
Per correlation coefficient there is overhead for calculating the values $y_{i,j}$ from $y_{i,j-1}$ respectively $z_{i,j}$ from $z_{i,j-1}$. For each correlation coefficient this requires R MAC's.
Per row the template is shifted (M-C) times, resulting in an overhead of R x (M-C) MAC's and ACC's for 1 row.
For the exhaustive search, a total of (L-R+1) rows have to be done. Resulting in an overhead of R x (M-C) x (L-R+1) MAC's and ACC's for subtracting the columns.

Every time when the template is shifted down one row, an overhead of C MAC's and ACC's exists for calculating the values $y_{i,j}$ from $y_{i-1,j}$ respectively $z_{i,j}$ from $z_{i-1,j}$.
Since the template is shifted down (L-R) times, this results in an overhead of (L-R) x C MAC's and ACC's.

The total amount of MAC's and ACC's for calculating (A.3) equals :
L x M + R x (M-C) x (L-R+1) + C x (L-R) MAC's and ACC's.

Since the execution of a MAC and an ACC generally takes the same amount of time, the number is expressed in MAC's only. The total number of MAC's for calculating (A.1), (A.2) and (A.3) equals then:

2 x (RxC) + (L-R) x (M-C)xRxC + 2 x (LxM) + R x (M-C) x (L-R+1) + C x (L-R)

B DSP - properties

One of the subjects of this project is to select a DSP for the extension board. Nowadays there are many different DSP's available, each with it's own characteristics. Choosing a DSP for a certain application requires thorough understanding of the properties of DSP's. To be able making the best possible choice, the properties of various DSP's have been analyzed.

Digital Signal Processors are micro processors dedicated to signal processing applications. Their architecture is such that typical signal processing applications like digital filters, FFT's etc. can be implemented very efficiently. The two following examples show the most important features of DSP's.

Digital filter applications usually consist of program loops in which multiply accumulate operations are to be done. DSP's are very efficient in the sense that most of them can perform the multiply accumulate operation in a single machine cycle. Besides that, loop counters and data pointers are de-/in-cremented while the MAC is executed. A program address generator will generate addresses without introducing pipeline conflicts. Another feature in these filter applications are circular buffers: by specifying the base address and the length of a array (eg. filter coefficients), the DSP will wrap the index pointer around to the first array element after the whole array is indexed through. Thus the overhead operations are done by dedicated hardware.

Performing FFT's requires an addressing technique called reversed bit addressing. Many DSP's provide this addressing mode, to increase the efficiency of FFT calculations. Another operation in FFT's is that the sum and difference of two operands are to be calculated. Some DSP's are equipped with ALU's that can calculate the sum and difference of the input operands simultaneously in a single machine cycle.

The consecutive sections describe the DSP properties and the differences among various DSP's. Differences exist in the operation of the DSP-core but also in the on chip peripherals like SRAM, serial ports, DMA capabilities, host interfaces etc.

B.1 Fixed / floating - point DSP's

DSP's can be characterized based on their numerical format: fixed- or floating-point. The main difference is of course the presence of respectively fixed- or floating- point computation units. However, there are more characteristics associated with fixed and floating point DSP's, which will be described in this section.

Floating point DSP's usually have a 32 bits wide data bus width. Virtually every floating point DSP can multiply two floating point numbers in a single machine

cycle. Numerical formats are usually 24 bit's mantissa (defining the precision) and an 8 bits exponent (defining the dynamic range). Some DSP's offer the possibility of a representations up to 32 bits mantissa and 11 bits exponent.

Fixed point processors are available with 16 to 24 bits data bus width. They perform multiplies of two 16/24 bits operands in a single cycle. The width of the term that is being accumulated equals usually at least the twice the width of the multiplication operands.

There is a significant difference in the addressing space of floating and fixed point DSP's. The address space of a fixed point DSP is limited to 64 to 128 kwords. Floating point DSP's have address spaces ranging from 2M to 4G words.
Some fixed point processors offer an additional address space which can be as large as few mega words. The data that is stored in this memory space is not accessible by the DSP-core. A DMA controller has to transfer the data into the DSP's main memory before it can access the data.

Floating point DSP's are more powerful CPU's then fixed point CPU's due to their architecture:
their wider instruction words allow more instructions to be coded in parallel, leading to more powerful parallel processing capabilities (see section B.2).
All floating point DSP's operate from a multi ported register file while most fixed point DSP's have an accumulator based architecture (see section B.2).

Fixed point processors are available with higher instruction rates. At the moment of this investigation, fixed point processors with instruction rates as high as 40 Million Instructions Per Second (MIPS) were available, and 50 MIPS versions were about to come. Floating point processors were available at rates up to 33 MIPS and 40 MIPS were about to come.

B.2 DSP-CORE ARCHITECTURE

The architecture of the DSP can usually divided into the DSP-core that actually executes the instructions and on chip peripherals like SRAM, serial ports, DMA controllers etc.

The DSP-core consists of address generator units and computation units, which operate in parallel. The address generators handle the instruction- and data fetches and the modification of index pointers, while the computation units perform the actual calculations. The computation units are an Arithmetic and Logic Unit (ALU), a multiplier and sometimes a barrel shifter. They also can operate in parallel

The instruction address generators feature capabilities like zero overhead looping (this topic will be covered in section B.6) the data address generators support circular buffers and reversed bit addressing as described in the introduction of this section.

On can make a main differences between DSP's on how the operands are fed to the arithmetic units: an accumulator based architecture an architecture operating from a multi ported register file.

The accumulator based architecture assigns certain registers to the in- and out-puts of computation units (accumulator, multiplier and shifter). The computation unit can receive/send the operand only from/to that particular register or from/to a data bus. This architecture is thus not so flexible in moving data around between the computation units.

In DSP's with a multi ported register file, each computation unit can get it's operands from any register in the register file, which offers more programming flexibility and increases the performance.

B.3 BUS INTERFACES

The DSP bus architecture can basically be divided into two variations, the "Harvard"- and the "Von Neumann"- architecture.

The Harvard architecture, most commonly used in DSP's, has separate address spaces for program and data. This has the advantage that the CPU can fetch instructions and operands simultaneously. Certain DSP's even have two data spaces, so the DSP can fetch an instruction, and two operands simultaneously. Other DSP's allow the user to store data in the program memory space, so the core can then still fetch two operands in one machine cycle if the instruction can be executed from an instruction cache.(see also section B.4).

The "Von Neumann" architecture has a single address space in which both program and data are stored. The advantage is that it gives the user flexibility in mixing data and program in the same memory space.

A feature that is oftenly seen on general purpose processors to maintain software compatibility between new processors and their predecessors, is dynamic bus-sizing. If eg. 8 bit data values are to be stored in 32 bit wide memory, dynamic bus sizing provides the possibility to store 4 byte values in a memory that is physically 32 bits wide. The processor is thus byte oriented, but can also access 2 or 4 bytes simultaneously.
No DSP was found that supports dynamic bus-sizing, which results in the disadvantage that 8 bit's values will be stored in 16/32 bits wide memory locations, thus 100/300% of the memory would be left unused. Equipping the memory physically with 8 bits wide memory solves this problem. The disadvantage that still remains is that only 8 bit values can be stored in that memory area: this limits the programming flexibility, since it is not possible to store 32 bits data (eg. programs) in that memory space.

Another common feature on general purpose processors is Dynamic RAMS's support. Especially if large memory spaces are needed, DRAMS are usually preferred due to their low price. DRAMS require special control signals and refresh logic. Many general purpose processors have the required logic integrated on the chip, DSP's are seldomly equipped with the required logic for DRAMS.

B.4 MULTIPLY ACCUMULATE INSTRUCTION

A multiply accumulate is one of the most typical instruction for a DSP. Especially since this is also a crucial instruction for the greyvalue correlation, a section is dedicated to this subject, and the restrictions there are on it.
Virtually every DSP performs the MAC in a singe machine cycle and most of them perform it pipelined. This means that in the current machine cycle, two operands are being multiplied while the product from the previous cycle is being accumulated. To implement filter structures efficiently, a DSP should be able to perform a MAC in a single machine cycle. Every DSP has the ability of performing single cycle multiply accumulate, however, there are restrictions on it.

Performing single cycle MAC's usually require more resources like on chip memory. Performing a multiply accumulate requires 3 words of data, the instruction word plus the two operands. As the available external bus bandwidth is usually limited to 1 access per machine cycle, the DSP-core needs other resources to complete a MAC in a single cycle. The three following solutions are commonly used to facilitate a single cycle MAC:
- on chip data memory
- on chip program memory or on chip instruction cache
- an external harvard architecture

If two of them are available, then the DSP can perform a single cycle MAC. It can be concluded that each of the three solutions provide one additional operand or the program word.
Thus if there is on chip data- and program-memory, the program word and one operand can be fetched from the on chip memory while the second operand is being fetched from external memory. Or if there is an external harvard architecture plus an on chip instruction cache, the two operands can be fetched simultaneously from external memory, while the instruction is read from the instruction cache. Only the first instruction of a loop there will be a conflict due to the cache miss, but the consecutive iterations will be performed single cycle.

From the mentioned resources, the external harvard architecture and instruction cache are least preferred. The external harvard architecture introduces separate memory spaces for which decreases the flexibility in storing data.
Usage of an instruction cache will lead to penalties due to cache misses, which might decrease the performance.

B.5 PIPELINING

Pipelining in processors is introduced to increase the CPU performance by shortening the instruction cycle time. Since CPU performance is usually specified under the most optimal conditions, special care is to be taken since pipelining can decrease this specified CPU performance.

The principle of pipelining is that the fetch- decode- and execution-units of a CPU are used simultaneously. A processor cycle can mainly be divided in three stages, the instruction-fetch, - decode and -execution stage. In early processors these three

stages were performed sequentially in one machine cycle. To further decrease the machine cycle time, these operations are performed in 3 consecutive machine cycles. This means that if instruction *n* is executed, instruction *n+1* is being decoded and instruction *n+2* is being fetched already.
Some DSP's have four pipeline stages, the other stage is used to read or write an operand from memory.

The specified CPU throughput can be influenced by so called pipeline conflicts: the fact that certain CPU activities will be delayed, due to penalties in fetching instructions or data.
An example of a pipeline conflict can be an ordinary jump instruction: during the execution of instruction *n* (which is the jump instruction), the program counter will be altered to *m*. However, the instruction pipe still contains instruction *n+1* and *n+2*. It will take two extra machine cycles to refill the pipe with instructions *m*, *m+1* and *m+2*, thus the CPU's execution unit is idle for two machine cycles.

To keep the loss due to pipeline conflicts as low as possible, DSP's provide special instructions like delayed branches and repeat instructions:

When a delayed branch is executed, the instructions that are currently in the pipe will be executed before the branch instructions actually takes place. (thus the instruction at the destination address of the jump will simply be placed at the end in the instruction pipe).

The repeat- or the "do until"-construct performs a certain block of instructions a certain amount of times without any pipeline conflicts for the branch at the end of the loop. This feature is being mentioned in literature as zero overhead looping.

Virtually every DSP supports delayed branches and zero overhead looping. Differences between DSP's exist in penalties due to nesting of loops, some DSP's don't allow nesting, others allow even 4 to 6 levels deep nesting without overhead.
From the previous discussion can be concluded that the number of pipeline conflicts can be limited by using the dedicated instructions and proper software design (avoiding unnecessary jumps), however, not every conflict can be eliminated.

Pipelining is also being mentioned in the context of multiply accumulate. This means that in one processing cycle, a multiply is executed on two operands, and that the product term of the previous cycle is being accumulated in the current cycle.

B.6 ZERO OVERHEAD LOOPING

Another typical DSP feature is zero overhead looping. In eg. FIR filter applications loops only consist of a single multiply accumulate instruction that is to be repeated a number of times (and so is the exhaustive search for the greyvalue correlation). A loop construct consists beside the MAC instruction of some overhead: a counter that is decremented every mac-cycle, compared to 0, and while it is not 0, the processor has to jump back to the MAC instruction. Especially if the loop consists of only one instruction, it is important that the overhead is as less as possible, since every lost cycle due to overhead will be a 100 % time penalty.

Every DSP has a special instruction to perform the overhead, as mentioned above, on the background. As explained in section B.2, the address generator performs these activities.
Possible implementations are repeat- , repeat block - instructions and do-while constructs. The repeat construct, fetches an instruction once, and executes it the specified number of times. The repeat block and do while constructs perform a range of instructions, the program address generators update the program counter and keep track whether the number of repeats is reached.

In practical situations there is still a bit overhead, which will be limited to a few cycles to execute the repeat instruction.

Even though every DSP features zero overhead looping, there are differences between DSP's in the nestability, and interruptibility of the repeat instructions. Certain processors do not allow the user to nest the repeat block instructions while others allow nesting up till 6 levels deep. In some processors, these loops are not interruptible, the interrupt response time can in such case depend on the size of the loop.

If an algorithm requires nesting (eg. a two dimensional FIR filter) but the DSP doesn't support it, the overhead as mentioned before has to be done by the computation units and thus is to be done at the expense of actual computation performance. The impact on the performance depends on the number of loop cycles that are to be repeated. If the repeat count is very large, the overhead will probably be acceptable, in eg. 2 dimensional filter algorithms a small kernel (eg. 5 * 5) the overhead can result in significant loss.

B.7 DMA CONTROLLERS

As DSP's process data very fast, it implies that it must be able to access the data fast enough. Most DSP's are equipped with some on chip zero wait stated SRAM which can be used to store frequently used data like eg. filter coefficients. Since the amount of data that can be stored in this on chip SRAM is limited, also off chip memory will be necessary in many cases.

In case the on chip SRAM is to small, it is possible to swap in data from external to internal memory. Many DSP's offer DMA facilities that allow background data

transfers without processor intervention, rather that the DSP-core performing the data transfers. All the processor has to do is specifying the source and destination addresses and the word count. While the DSP continues execution of the program, the DMA controller takes care of the block transfer.

There are differences between DSP's with respect to DMA functionalities.

- Performance:
  Some DSP's are equipped with dual ported memory, in that case DMA transfers do not influence DSP performance at all. Other DMA controllers keep the DSP-core idle for one machine cycle per word transfer, to write the data word into the internal memory.

- Subsampling
  If a block move is being performed, two pointers of the DMA controller are to be incremented or decremented after a word is moved. Some DMA controllers allow a pointer to be incremented by +1, 0 or -1. Others have modifier registers, in which any increment value can be specified. These DMA controllers can thus perform subsampling, by modifying the source address by the subsample factor after every word move.

- Chained DMA transfers.
  Some DSP's provide chained DMA transfers to transfer multiple blocks of memory without processor intervention. The consecutive block moves are specified in a memory area in a linked list, a hardware pointer is loaded with the base address of first transfer to follow. A as soon as the first block transfer is completed, the next is loaded from the memory en executed etc.

- 2 dimensional DMA
  This feature offers the possibility to extract a sub array from a larger two dimensional array.

B.8 COMMUNICATION PORTS

In the previous sections some on chip peripherals were discussed already. This concerned peripherals that are likely to be used on the extension board. In this section, some other peripherals that can be found on DSP's are discussed.

The more sophisticated DSP contain link ports to communicate with a high speed with other DSP's. These link ports are mainly used in multi processor systems and is thus not applicable for the extension board.

Also many DSP's have serial ports, to communicate with serial devices. This also does not seem to be useful in the extension board. Perhaps for debugging applications they might be useful.

Some DSP's have some parallel I/O lines. These lines might be useful on the extension board. In certain processors. Virtually every DSP has one or more interrupt lines. This might also be useful in the extension board, for communication purposes.

B.9 WAIT STATES

If slower memories are connected to a DSP, wait states are required to hold the DSP for a certain amount of cycles during external memory accesses. Since most of the DSP instructions are performed in a single machine cycle, wait states can decrease the performance drastically if many memory accesses have to be done from external memory. Wait states can be inserted through hardware, software programmable or, like in most DSP's, via a combination of both.

For hardware wait state insertion, the processor has an input that if asserted, holds the processor. The input is to be asserted by additional hardware if the processor accesses slow memories (or I/O devices).

Software wait state insertion requires no additional hardware, the number of wait states can be specified in a register, the processor will insert the wait states every time it accesses external memory. Some DSP's have multiple wait state registers for different memory blocks, to allow usage of slow and fast memories in the same design.

Most DSP's have both the hardware input and programmable wait state insertion capability. The two can be combined: the processor will insert wait states until either one, or both conditions (de-assertion of input or/and software wait states completed) are met.

B.10 DSP PERFORMANCE SPECIFICATIONS

DSP manufacturers also show performance specifications in their data sheets. One of the most frequently used is the sustained instruction rate (MIPS) and the number of Floating Point Operations Per Second (FLOPS). Some DSP vendors also show performance figures of eg. FFT's.

An important measure for DSP computation throughput is the instruction cycle time or it's inverse: Million Instructions Per Second (MIPS). However, one must not jump to conclusions too fast, since the sustained instruction rate is only achieved under the most optimum conditions. In practical situations, the instruction rate will be influenced by pipeline conflicts, bus conflicts cache conflicts etc.
The actual instruction rate can reduce to 50 percent easily if the data is not stored in the memory properly: In filter algorithms where all data is stored in one 0-wait stated external memory block, the instruction rate will approximately be 50% of the sustained instruction rate. This is because every MAC cycle, there is a bus conflict since the DSP has to fetch the two operands from external memory.
Also if the DSP has many instructions that require more than one machine cycle, the actual instruction rate can be significantly less that the sustained instruction rate.
Generally spoken, if the data is stored in the right places (eg. on chip memory), and the program is well constructed (optimised for bus and pipeline conflicts), the performance can be close to the sustained performance.

For floating point processors the number of floating point operations is also specified with the number of FLOPS. This figure equals the number of MIPS multiplied by the number of floating point operations that can be done per machine cycle. As every DSP can perform a single cycle MAC, the number of MFLOPS equals twice the number of MIPS. In case the ALU calculates the sum and difference of the two input operands, it can be three times the number of MIPS.

Certain DSP vendors specify the number of cycles, that the DSP requires for a FIR filter or for a 1024 points FFT's. One has to be very careful in evaluating these figures if the DSP is going to be used in other applications. As shown in the introduction of this section, DSP's are often optimized for FFT calculations, performance of other algorithms can be completely different.

Only performance specifications like they were discussed in this section don't tell very much (except if the vendor happens to specify bench marks for the same type of application that has to be done).
In typical DSP applications (MAC-alike structures like filter algorithms), the performance (eg. instruction rate) can help to determine how a DSP will perform if all restriction are kept in mind. In general purpose algorithms, probably the best procedure to analyze the performance is to run it on an evaluation board or simulate it with a simulator.

B.11 DEVELOPMENT TOOLS

During the development, software development and debugging tools are required. Every manufacturer provides the common tools for his DSP like a C-compiler, assembler, linker etc.

Most DSP manufacturers also provide a simulator for software debugging. This is a software package for eg. PC systems that simulates the operation of the DSP. It's input is an object file from the linker, that will be simulated. In this way, a program can be tested / debugged / evaluated without using the target system, or even before the target system is even built.

Especially during the debug phase of the hardware, an emulator can be a very useful debugging tool. This tool makes it possible to step through software, set breakpoints, examine or alter register contents etc.
Before, this tool existed of a specific processor board (called a POD) that could be plugged into the target processor socket. A special software package and board for eg. a PC takes care of communication between the POD and PC.
Most modern DSP's do not require the POD any more for emulation purposes, they have the emulation logic integrated on the chip, communication between PC and target DSP is oftenly done via the JTAG port. In this way, debugging is easier for high speed designs since they are debugged with the target processor, thus no influences due to the extension cable between POD and target processor socket play a role during debugging.

C Execution time estimation

In this appendix the execution times for the grey value correlation are estimated for the search of a 200 x 200 template in a 512 x 512 window, for various DSP's.

The calculation time consists of the following elements:
- the required time for the MAC loops
- loop overhead
- the required time to calculate the correlation coefficient from the sum terms
- the required time to perform the parabola fit.
- a margin to account for overhead occurring in pieces of C-code.

Appendix A gave an overview of the complexity of an exhaustive search in terms of Multiply Accumulates. Assuming every MAC takes one cycle, the required time for running the MAC loops can be derived from the results of this appendix.

For the normalized grey value correlation the following exhaustive searches are to be done:

Search	count	template size	Search window
Global	1	25 x 25	64 x 64
Local coarse	3	34 x 34	38 x 38
Local medium	3	100 x 100	106 x 106
Local fine	1	200 x 200	202 x 202

With loop overhead is meant the overhead (number of cycles) to process one row of the template. If loops are nested, the second nesting level will introduce overhead if the DSP does not support nested loops. This overhead is due to saving register contents (see section B.6). Especially in the coarse search, where the template size is rather small, this may lead to significant overhead.
The loop overhead was derived by writing an assembly language program with some nested loops. For the TMS 320C32 this overhead was significantly more than for the TMS 320C50 and the ADSP 21062. The large overhead for the TMS 320C32 processor consisted of:
- the repeat instruction : 4 cycles
- saving and restoring of outer loop counters : 6 cycles
- pipeline conflict : 3 cycles

The two other DSP's support nested loops, and have only an overhead of 1 or 2 cycles.

The required time for calculation of the correlation coefficient and performing the parabola fit were derived by compiling the pieces of C-code and running them on the simulator. This information was found, by performing some simulations with

simulators for the ADSP 21062 and for the TMS 320C32 and the TMS 320C50 simulators.

The results are shown in the table below:

Operation:	TMS 320C3x	TMS 320c5x	ADSP 21062
Loop overhead	13	2	2
Calc R	120	640	80
Parabola fit	80	460	60

The total estimated execution time contains further of a margin of 20% to account for operations like initialisations of variables, parameter passing during function calls, additional C-code etc.

Appendix C.1 will show the execution time for the coprocessor architecture, and appendix C.2 for the image processor architecture.

C.1 EXECUTION TIME FOR THE COPROCESSOR ARCHITECTURE

This appendix shows how the execution time is being calculated for the coprocessor architecture in paragraph 3.1.1.

The calculation times are as follows:

Search:	Operation:	Count	t [ms]	tot [ms]
Global search	MAC	$1.1*10^6$	27	57
	Calc.	1600	27	
	Loop	$40*10^3$	3	
Local Coarse	MAC	$100*10^3$	3	3
	Calc.	25	-	
	Loop	833	-	
Local Medium	MAC	$1.6*10^6$	41	42
	Calc.	49	1	
	Loop	4900	-	
Local Fine	MAC	$525*10^3$	13	13
	Calc.	9	-	
	Loop	1800	-	
Total calculation time:				115

Operations:
MAC: Required time for multiply accumulate loops
Calc.: Required time for calculation of correlation coefficients from the sum terms and parabola fit.
Loop: Loop overhead

Data transfers.

The required transfer time to send the data from the SBIP to the extension board is as follows: (Assumed transfer rate is approx. 750 ns per word)

Search	Window size	Comm. time [ms]
Global	64 x 64	3
Local coarse	39 x 39	3 x 1.5
Local medium	107 x 107	3 x 8.5
Local fine	202 x 202	31

C.2 EXECUTION TIME FOR IMAGE PROCESSOR ARCHITECTURE

The following table shows the characteristics for the floating point processors: a 30 MIPS TMS 320C32 and a 33 MIPS ADSP 21062.

Search:	Operation:	Count	TMS 320C32 t [ms]	ADSP 21062 t [ms]
Global search	MAC	$1.1*10^6$	36	33
	Calc.	1600	11	4
	Loop	$40*10^3$	17	3
Local Coarse	MAC	$100*10^3$	4	3
	Calc.	75	-	-
	Loop	2500	1	-
Local Medium	MAC	$1.6*10^6$	54	49
	Calc	147	1	1
	Loop	14700	6	1
Local Fine	MAC	$525*10^3$	18	16
	Calc.	9	-	-
	Loop	1800	1	-
Calculation time			149	110
Margin (20 %)			30	22
Estimate time grey value correlation:			179	132

Operations:
MAC: Required time for multiply accumulate loops
Calc.: Required time for calculation of correlation coefficients and parabola fit.
Loop: Loop overhead

D DSP candidate overview

This appendix shows characteristics of the DSP's that remained after the raw selection.

D.1 AT&T

AT & T	DSP 3207/3210
Type: Fixed point Floating point	 - 32,40 bits
Floating point formats: mantissa,exponent	24,8 32,8
Instruction rate [MIPS]	27.5, 33
On Chip memory	2 x 4kb
Cache: size [program words] algorithm	 - NA
On chip peripherals: Host interface DMA Link ports Serial Ports Parallel I/O Timer	 yes 2 channels - 1 8/16 1 x 32 bits
Register Count: Gen.Purp. registers Index registers	 4 x 40 bits 22 x 24 bits
Zero overhead looping: Nestable levels Interruptible	 - yes
Pipeline stages	3
Emulator Interface	-
Boot options	none
Tools	
Price	US$ 70,-
Specials	DRAM controller

D.2 ANALOG DEVICES

Analog Devices	ADSP 21020	ADSP 21060/21062
Type:		
Fixed point	32 bits	32 bits
Floating point	32,40 bits	32,40 bits
Floating point formats: (mantissa,exponent)	24,8 (IEEE) 32,8 (IEEE)	24,8 (IEEE) 32,8 (IEEE)
Instruction rate [MIPS]	25,33	33,40
On Chip memory	-	2 x 128 kb/2 x 256 kb
Cache:		
size [program words]	32	32
algorithm	Set Associative *	Set Associative *
On chip peripherals:		
Host interface	-	16 or 32 bits
DMA	-	10 channels
Link ports	-	6 x 4 bits
Serial Ports	-	2
Parallel IO	4	4
Timer	1 x 32 bits	1 x 32 bits
Register Count		
Work registers	2 x 16	2 x 16
Index registers	2 x 16	2 x 16
Zero overhead looping		
Nestable levels	6	6
Interruptible	yes	yes
Pipeline stages	3	3
Emulator Interface	JTAG	JTAG
Boot options	eprom	host,link port,eprom
Tools		
Price	US$ 130,-	US$300,-
Specials		

*: Set associate cache algorithm stores only instructions in the cache if there is a conflict with an operand fetch from program memory. (Thus only in the cases where two program memory accesses are performed in a machine cycle, the program word is saved in the cache.)

D.3 TEXAS INSTRUMENTS

Texas Instruments	TMS 32C3x	TMS 320C4x
Type: Fixed point Floating point	 24 bits 32,40 bits	 32 bits 32,40 bits
Numerical formats (mantissa/exponent)	24,8 32,8	24,8 32,8
Instruction rate [MIPS]	15,20,30	15,20,30
On Chip memory	2 x 4 kb	2 x 4 kb
Cache: size [program words] algorithm	 64 least recently used *	 128 least recently used *
On chip peripherals: Host interface DMA Link ports Serial Ports Parallel I/O Timer	 - 2 - 2 - 2 x 32 bits	 yes 10 channels 6 x 8 bits 2 - 2 x 32 bits
Register Count Work registers Index registers	 8 x 40 8 x 32 12 x 32	 12 x 40 8 x 32 14 x 32
Zero overhead looping Nestable levels Interruptible	 - yes	 - yes
Pipeline stages	4	4
Emulator Interface	JTAG	JTAG
Boot options	eprom	eprom,link port
Tools		
Price	US$ 30,- - 60,-	US$150 - 200,-
Specials		

*: The cache is divided into two blocks of 32 words. A program word that is to be stored in the cache will be stored in either of the two blocks. It will be stored in that block that holds the instruction that is least recently used.

E Hardware address generator

This appendix gives an example of an address generator to transfer an (interlaced) image from the SBIP.

To transfer an image to the SBIP via the video bus, there are basically 4 control signals involved:
- The Vsync to define the start of a frame
- The Hsync to define the start of an image line
- The pixel clock to define the video-sample timing of the ADC on the SBIP
- The blanking signal to specify which part of the video line contains valid video information. The blanking signal will become inactive for exactly as many pixel clock periods as there are pixels on a line, thus a logical AND of the pixel clock with the blank signal results in a (discontinuous) clock pulse, that defines when there is valid video information on the video lines. (note: in a practical situation, the blank signal is to be synchronised with the pixel clock before the logical and can be performed, in order to prevent race conditions).

From the time difference between the Hsync and Vsync pulses can be derived if the following frame contains the even- or odd- image lines.
Furthermore it is required to know at which memory address the image is to be stored, and the memory pitch between two column pixels on adjacent rows (in this appendix is assumed that the memory pitch represents the number of pixels on a line).
Figure E.1 shows a block diagram of the hardware address generator.

The databus of the TMS is connected to the base and pitch registers: these specify the base address of the image, and the pitch. They have to be initialized by the TMS 34020 before the image transfer is performed. The two control signals "wr_base" and "wr_pitch" are the clock signals to write this info into the latches.

To transfer an image, the get_img line has to be asserted, the "frame/line detection" awaits the following Vsync. After this occurs, it performs a parallel load on the counter with the base address. Then at the first Hsync it detects whether the frame will contain even or odd lines. In case the frame contains the odd lines, it will advance the counter one image line and does this as follows: it controls the Multiplexer to select the output of the adder, which holds the memory address of the first pixel on the next image line. This will be parallel loaded into the counter.

After the first line (say line n) is transferred to memory and the next Hsync pulse appears, the counter points to memory location of the first pixel of line $n+1$. The output of the adder will contain the memory address of the first pixel from line $n+2$. By performing a parallel load again on the counter, there will be left a gap in the memory with the size of one image line. This procedure happens every time a Hsync appears.

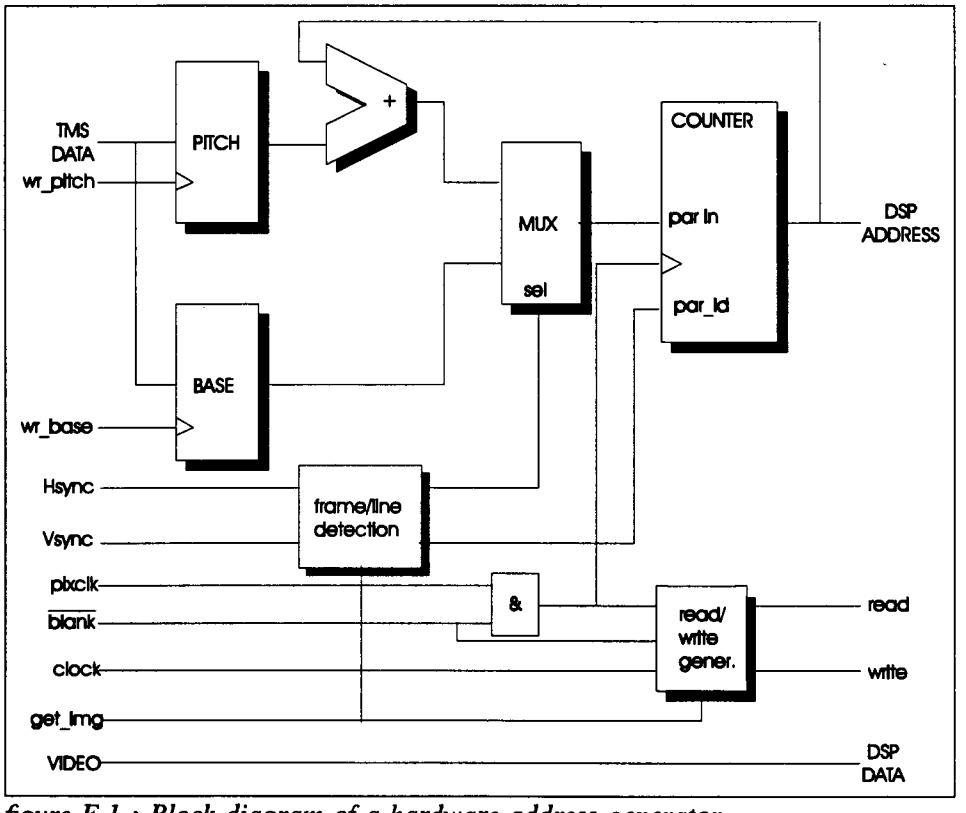


figure E.1 : Block diagram of a hardware address generator

After the first frame is read, the whole procedure is repeated one more time to read in the next frame.

F Hardware design

This Appendix explains the hardware design of the extension board. First the block diagram and the global operation of the extension board will be described in section 4.1.1. Then the operations of the various functional blocks will be explained in the sections F.1 to F.6.

F.1 BLOCK DIAGRAM OF THE EXTENSION BOARD

The hardware was mainly divided into five functional blocks:
- the processor
- the memory
- the TMS 34020 interface
- the video interface
- the controller

Figure 4.1 shows the block diagram of the extension board with these functional blocks and their interconnections. Within the functional blocks, a global internal architecture is shown to give an impression on the operation of the extension board.

The global operation of the hardware is as follows:

The heart of the extension board hardware is the processor and the memory. The DSP-core can access maximum 1 Mb of external SRAM (512 kb standard-, and 512 kb of optional - memory) and 256 kb internal DSP memory. The external SRAM is used to store images, the internal DSP memory is used to store the program, variables and the template. Further the DSP has a boot memory device connected to it, this is an 8-bits wide memory device from which the DSP can boot or where it can store data. Section F.3 explains the memory block.

The processor block contains besides the processor and the clock generator, also some hardware to create a DMA bus. Via this DMA bus, the DSP communicates with the SBIP. The communication with the SBIP is handled through two blocks, the TMS 34020 interface and the video interface. On the SBIP's side, they are connected to the TMS 34020 busses and the SBIP's video bus respectively. On the DSP - side, these blocks are thus connected to the DMA bus. The processor block will be explained in section F.2.

The TMS 34020 interface provides the inter processor communication. Through this interface, the TMS 34020 has the ability to access the entire memory on the extension board. The interface further provides the possibility for both processors to assert each others interrupt line. This interface is used for giving commands, booting the DSP, transferring templates etc. The TMS 34020 will be explained in section F.4.

The video interface facilitates the transfer of images to and from the SBIP. Images can be transferred at video speed (10 - 20 Mb/s) to the extension board. The video

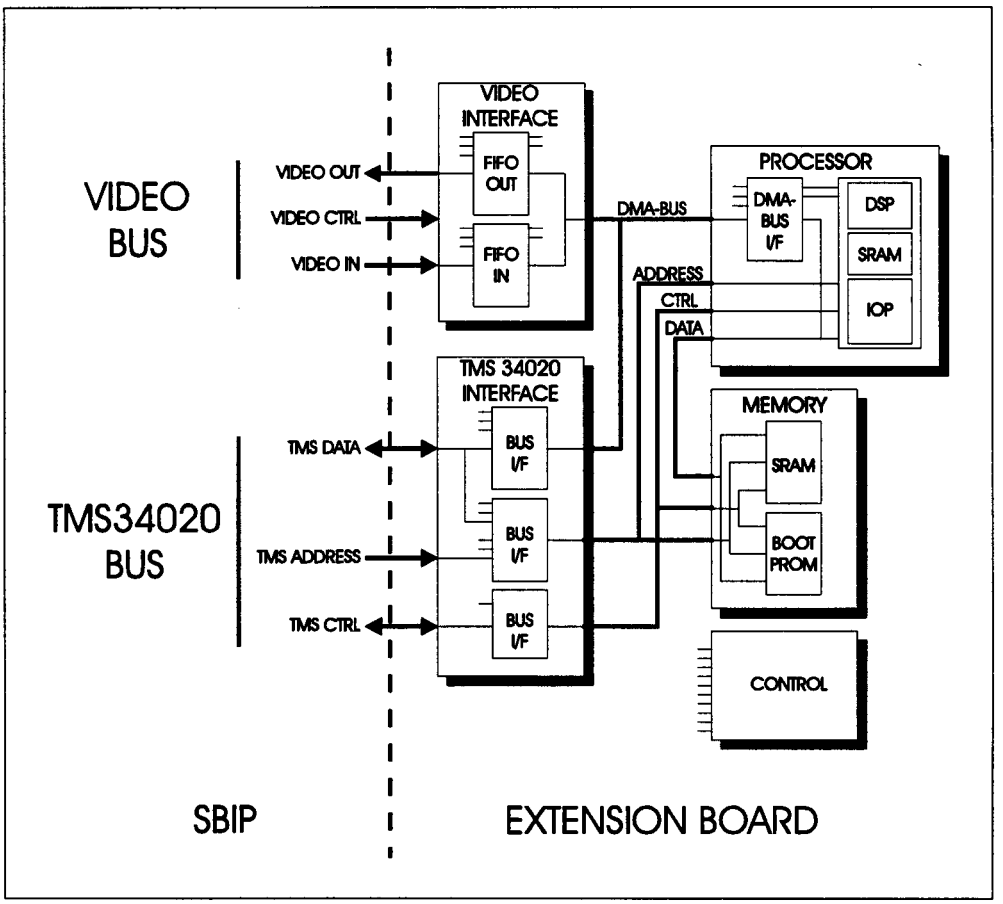


Figure F.1 : Block diagram of the extension board

stream coming from the SBIP's video bus, flows through a FIFO memory to the DMA bus. The DSP's I/O-Processor (IOP) manages the transfer from the DMA bus to the memory. The video stream from the extension board to the SBIP is performed in the same manner in reversed direction: the IOP transfers the data from the memory to the DMA bus, from where it flows through a FIFO memory to the SBIP's video bus. This interface will be explained in section F.5

The control block, implemented in an Erasable Programmable Logic Device (EPLD), is used to "glue" the various functional blocks together. It controls devices like the tri-state buffers, hardware registers and the FIFO memories. It also performs tasks like error checking during image transfers and data transfers from the host. Further it offers the possibility to set the extension board in a test mode, to test the various hardware functions while the extension board is connected to the SBIP. The controller can actually be seen as part of all the other functional blocks that were mentioned just before, it's operation will therefor be described in the various section about the functional blocks. Section F.6 will give a brief description on how the design of the controller and the test mode operation.

F.2 THE PROCESSOR

The processor block contains the DSP, a clock generator and the required hardware to create the DMA bus. paragraph F.2.1 will give a brief explanation about the DSP and it's operation, paragraph F.2.2 explains the operation on the processor block.

F.2.1 DSP operation

The DSP contains a considerable amount of peripherals. To make the reader more familiar with the DSP, this paragraph will give a short description on the used features of the DSP. Figure F.2 shows the block diagram of the DSP. The various blocks: the DSP-core, the dual ported memory, the external port, the I/O Processor (IOP), and a JTAG port will be described in this paragraph. At last, a short description is given on how the DSP communicates with the EPLD.

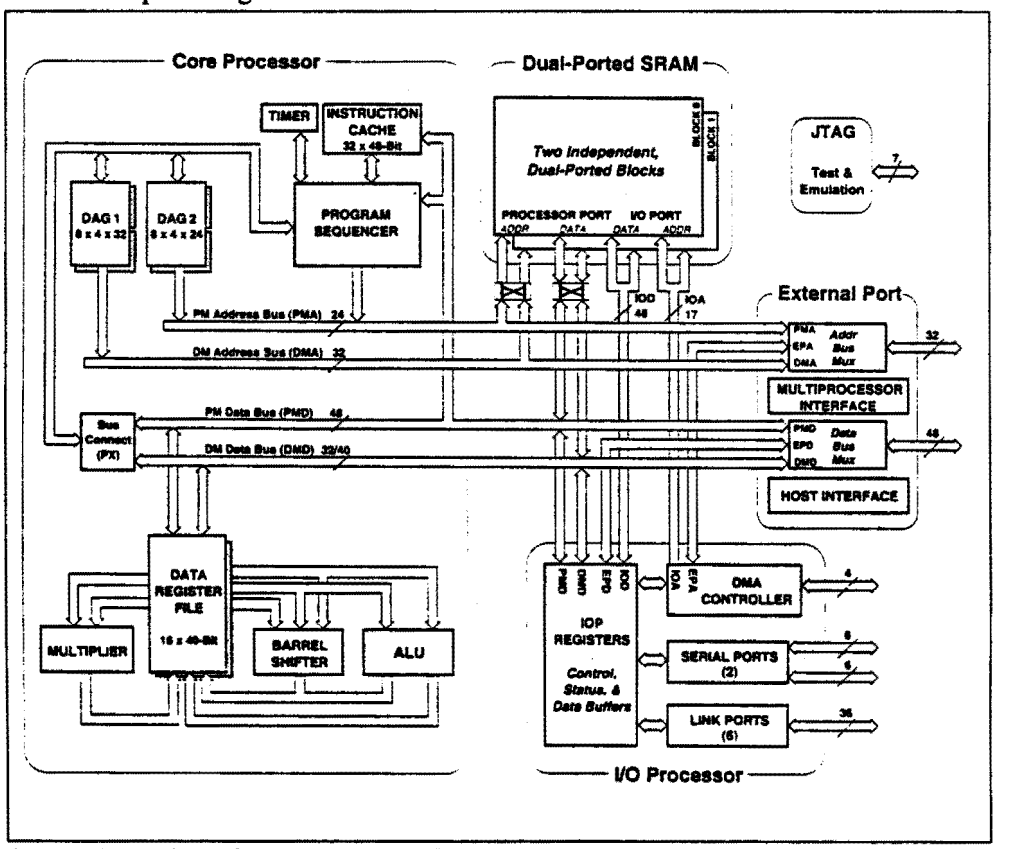


Figure F.2: Block diagram of the DSP

DSP-CORE

The DSP-core is in fact the ADSP 21020 DSP. The computation units, a multiplier, a barrel shifter and an ALU operate from a 16 x 40 bit register file. The DSP-core has two data address generators to exchange data between memory and the register file: one to access the program memory space, the other to access the data memory space. A program sequencer takes care of the fetches of the program words from the program memory. It uses an instruction cache that stores instructions, which fetch conflict with data fetches from program memory.

ON CHIP MEMORY

The DSP contains two dual ported memory blocks for storage of programs and data. The DSP accesses the memory via one port, the IOP accesses it via the other port. The DSP-core is connected to the memory blocks via a cross bar switch. In that way it is possible for the DSP-core to access both memory blocks via its program memory- and data memory- busses. Section F.3 will explain the internal DSP memory in more detail.

EXTERNAL PORT

The external port provides a path to exchange data between the DSP and external SRAM or the host. This concerns data transfers to external memory by both the DSP-core and the IOP, or internal DSP memory accesses by the TMS 34020.
For accesses from the DSP-core or IOP, the internal data memory busses, the internal program memory busses or the IOP's busses can be connected to the external address and data busses.
For host accesses to internal DSP memory, the host interface offers the possibility for the TMS 34020 to acquire the DSP's external busses and access the internal DSP memory or the external SRAM.
The bus priority during simultaneous requests for the external bus can be defined by the user. In a control register can be set whether the DSP-core, or the IOP has higher bus priority. However, in both cases the host bus request has higher priority.

The host interface, consists of a bus request (HBR) and a bus grant (HBG) line. If the host asserts the HBR line, the DSP completes the current bus cycle, tri states its busses, and grants the bus request by asserting the HBG line. The host may then occupy the bus until it de-asserts the HBR line.
For internal DSP memory accesses, the host interface provides two additional lines: A chip select (CS) input which has to be asserted if the internal DSP memory or IOP registers are to be accessed and a ready (REDY) line to acknowledge that an internal DSP memory access is completed.
The host interface is used on the extension board, to boot the DSP and to perform memory accesses. The interface will be discussed in section F.4.

I/O PROCESSOR OPERATION

The I/O Processor (IOP), consisting of a DMA controller, 2 serial ports and 6 link ports, offers the possibility to perform data transfers to and from the internal DSP memory. The transfer can be to external memory or the DMA bus. The IOP has a large amount of control registers, that can be accessed by the DSP-core and the TMS 34020.

The DMA controller has 10 channels: 6 channels for the 6 link ports (channels 0 - 5) , and 4 so called external port channels (channels 6-9).
Since the link ports are not used on the extension board, channels 0-5 are not used on the extension board.
The 4 external port DMA channels have access to the external SRAM, the DMA bus and the boot memory of the extension board, they are used for the image transfers, DMA transfers from the host and booting the DSP.

Each DMA channel has a dedicated interrupt, the priority is fixed and descending with the channel number (channel 0 has the highest priority, channel 9 the lowest priority).

Multiple (in fact all) DMA channels can be active simultaneously. The controller can perform per machine cycle one word transfers to each memory block and allows 7 pending DMA requests. While the DMA interrupt priority for the 10 DMA channels is fixed, the interrupts from the external port DMA channels can be set to a rotating priority scheme: initially, the priority scheme is 6 7 8 9, after a request on eg. channel 7, the priority order will be 8 9 6 7 etc.

On the extension board, external port DMA channels perform the following (background) data transfers:

a. transfers between internal and external memory,
b. transfers between the DMA bus and either internal or external memory
c. transfers between the host and internal memory.

These block transfers can be set up by either the DSP-core or the host, and once set up, the DMA controller will handle the data transfer.

Internal to external memory transfers simply perform a 'background' block move from one memory area to the other. On the extension board they are used during the image transfer to and from the SBIP. During this DMA transfer, data can be packed from 8 to 48, 16 to 32, 16 to 48 or 32 to 48 bit words (or vice versa) by the IOP.
Since the DSP has dual ported SRAM (1 port to DSP-core, 1 port to the IOP), these block transfers will generally not influence the DSP-core operation. Only if the DSP wants to access external memory the performance of the DSP-core might be influenced, because only either of the DSP-core or IOP can use the external bus during one machine cycle. The fact whether the DSP core will be idle during external memory accesses depends on the fact whether the IOP or the DSP-core has the highest external port priority.

External port transfers are transfers from the DMA bus to internal DSP memory or external SRAM. Channel 7 and 8 can perform DMA transfers that are asynchronous with respect to the DSP's clock frequency. On the extension board they are used for image transfers to and from the SBIP, and for host DMA transfers.
For asynchronous operation, the DMA request and DMA grant lines are to be used. After the DMA transfer is set up in the control registers, the DMA request line is to be asserted once for each word transfer. The DMA controller will reply every request by asserting the DMA grant line. During asynchronous transfers to the DSP (or external memory), the external hardware has to make sure that the data to be transferred is set ready in an external hardware register, because the DMA controller expects data to be set on the DSP's data lines, right after it asserts the DMA grant line.
For asynchronous transfers from the DSP, the DMA controller will set the data on the DSP's data bus, and then assert the DMA grant line.
In case the asynchronous transfer is performed to or from external SRAM, the DMA controller will also set the DSP's address- and memory select- lines and the write/read control signals to control the external SRAM.

Another possibility for transferring data under supervision of the IOP is to write data to a so called "external port buffer" that belongs to the corresponding external port DMA channel. An external port buffer is a (6 deep) fifo memory which is part of the IOP-register set.
If data is written to this memory location (via a host bus request), the data is written into the FIFO memory and a DMA transfer is requested internally. The DMA controller will read the data from the FIFO memory when granting the request. In case the data transfer is in reverse direction, the IOP fills the fifo memory and the data is read from the fifo memory location.
This DMA transfer can also perform data packing as explained under the internal to external DMA transfers. It is used when the DSP is booted by the host or EPROM.

The following table lists the registers, associated with the external port DMA channels, to give an impression about required data to setup a DMA transfer.

II, EI	Pointer to the start of the internal and external memory blocks respectively
IM, EM	Pointer-Modifier (for internal and external memory) to specify how much the pointer is to be increased after each word transfer.
C, EC	Internal/external memory word count to be moved
DMAC	Control register to specify the mode of operation for the DMA channel
EPB	External port FIFO buffer
CP	Chain pointer which can be used to initiate a new DMA transfer after the current one is finished. It points to a memory location where a set of parameters for the next DMA transfer is stored. Once the current transfer is finished the IOP loads these parameters in the appropriate registers and initiates the next transfer.

JTAG PORT

The JTAG port is used for boundary scan testing, and emulation purposes. The extension board is equipped with two connectors that are connected to the DSP's JTAG port: one is to be used for boundary scan testing, the other to connect the emulator for the DSP.

INTERFACE TO EPLD

The DSP has 4 I/O lines and two interrupt lines which are used to communicate to the EPLD. Three I/O-lines are configured as output to the EPLD, to set the EPLD in the desired operating mode. The other (FLAG 0) is configured as an input line from the EPLD to obtain status information.
The following table shows the various output codes and the operation mode for the EPLD and the data that the EPLD will set on the input flag and the interrupt line.

Flags 1..3	Operation	Flag 0	IRQ 2
000	-	error	-
001		Vsync	Hsync
010	Image from SBIP	frame	transfer_interrupted
011	Image to SBIP	frame	transfer_interrupted
100	-	HDMA error	Vsync
101	-	VDMA error	-
110	Reset error flags	VDMA warning	-
111	Interrupt TMS 34020	frame	-

Interrupt line IRQ0 is asserted by the EPLD during test mode operation.

F.2.2 Hardware description processor block

Figure F.3 shows a block diagram of the processor block. It contains components for the clock generation and the DMA bus.

The clock is generated by a 66.67 MHz crystal oscillator. The output of this oscillator is divided to a 33.33 Mhz clock pulse. The DSP requires this 33.33 Mhz clock. The 66.67 MHz is used by the controller for edge detection of the pixel clock.

DMA - BUS

The DMA bus is used for communication with the SBIP. The DMA bus, 32 bits wide, is connected with the DSP's data bus via transparent/registered tri state buffers. The data bus width of the DSP is 48 bits, since data (32 bits) is being transferred via the data lines 16 to 47, the DMA bus is connected to these data lines via these buffers. Data bus bits 0 - 15 of the DSP are left unconnected on the extension board, since the program is stored entirely in internal memory.

The DMA bus is connected to the DSP's data bus via 2 32 fold transparent / registered tri state buffers. These devices are tri state buffers which can be used for registered or transparent operation. A detailed description on these devices will be given in sections F.4 and F.5. This section only will give a brief description.

The registered operation of the devices is used for asynchronous DMA transfers by the IOP: For transfers to the extension board, the data will be set ready in device 1, and the DMAR1 line will be asserted by the EPLD. When the DMA controller grants this request, the data will be set on the DSP's data bus by device 1.

During asynchronous DMA transfers by the IOP from the extension board, the EPLD asserts the DMAR2 line for every word to be transferred. The DSP then sets the data on the data bus and asserts the DMAG2 line. The rising edge of this pulse latches the data in the register in device 2. Afterwards the data can be set on

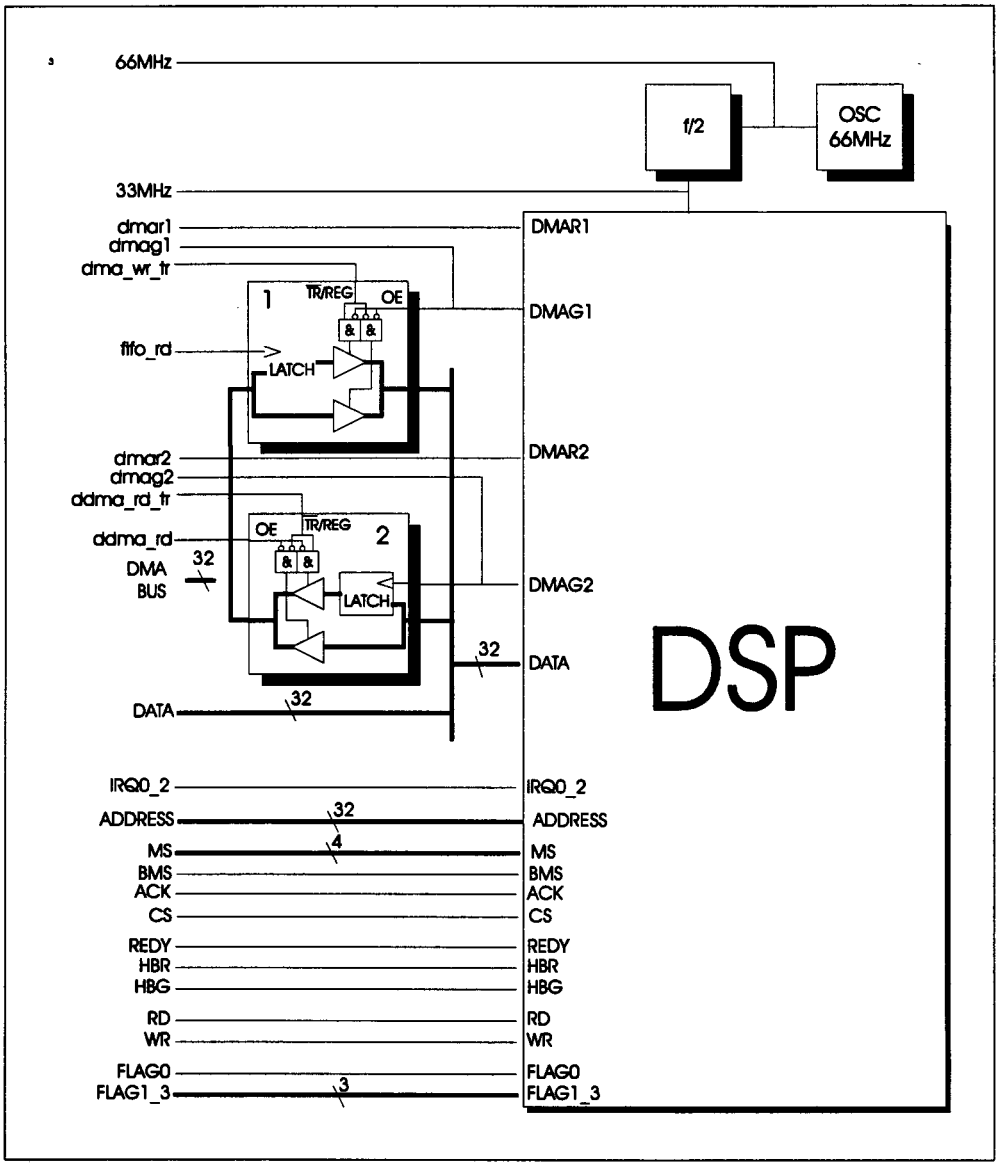


Figure F.3: The processor block

the DMA bus by the EPLD (by activating the output enable of the device).

The transparent operation of these devices is mainly used for the host DMA transfers that use the bus request/bus grant protocol.

The various signal names in the diagram have the following functions:

ADDRESS: The address bus of the DSP.

MS Four memory select lines, serve as a chip select for external memory devices.

BMS Boot memory select line, serves as a chip select for the boot memory device.

ACK Acknowledge line to the DSP to insert wait states during external memory accesses (currently not used)

CS Chip select line, to be asserted by the controller when the TMS 34020 accesses internal DSP memory.

REDY	Ready line to acknowledge to controller that an internal memory access has completed.
HBR	Host bus request line
HBG	Host bus grant line
RD	Read line
WR	Write line

F.3 MEMORY

The memory on the extension board consists of the internal DSP memory, the external SRAM to store images and a boot memory for booting the DSP (only used optionally). This section will first explain the available memory resources on the extension board in paragraph F.3.1, then the implementation of the external memory explained in paragraph F.3.2.

F.3.1 Memory resources on the extension board

The memory resources on the extension board can mainly be divided into of the main memory, consisting of the DSP's internal memory and the external SRAM, and the boot memory, consisting of a single boot memory device. The main memory is used to store programs, data and images and is accessible by the TMS 34020, the DSP-core and the IOP. The boot memory can be used to boot the DSP or to store data, and is only accessible by the IOP.

The DSP-core accesses the main memory via two busses: a program memory bus and a data memory bus. Via these busses the DSP-core can perform two memory accesses per machine cycle : either two memory accesses from the internal DSP memory, or one from internal and one from external memory. From the block diagram of the DSP (figure F.2) it can be seen easily that the DSP has a harvard architecture, however, the DSP has physically (apart from the boot memory) only a single memory space. The program- and data- memory spaces are physically mapped together into a single memory space: a certain memory address will only exist either data or program memory, never in both.

The following table shows a map of the main memory space:

From:	To:	Description:
0x00000000	0x000000FF	IOP control & status registers
0x00020000	0x0007FFFF	Internal DSP memory (program and data storage)
0x00400000	0x004FFFFF	External SRAM (image storage)

Note: The memory range from 0x00080000 to 0x003FFFFF is by convention reserved for multi processor operation, and could therefor not be used on the extension board.

The memory resources: the internal DSP memory, the external SRAM and the boot memory will be described in this paragraph.

INTERNAL DSP MEMORY

The internal DSP memory consists of two memory blocks of 128 kb each. Both memory blocks are dual ported, within one machine cycle both the DSP-core and the IOP can perform one access from each memory block. The two memory blocks are connected to the DSP-core via a crossbar, which makes it possible to configure the memory blocks as either program or data memory space. A memory block can also be configured partially as program memory and partially as data memory, however, in such case only one access from either of these two spaces can be performed per machine cycle, since they are located in the same memory block.

It is important to configure the two memory blocks properly to achieve the best performance with the DSP. For applications with multiply accumulate loops, one (program memory) operand should be stored in one internal memory block and the other (data memory) operand in the other block or eventually in external memory. In this way, these two memory fetches can be done simultaneously in one machine cycle. After the program word is stored in the program cache (in the first MAC - cycle), the rest of the MAC's in that loop will be performed single cycle. Storing two MAC operands in the same memory block would cause a bus conflict, the MAC would then take two machine cycles.

The data width of the memory blocks are user-configurable, as 16 or 32 bits wide data memory, or as 48 bits wide program memory. It is possible to mix these three different memory configurations within one memory block. As an example the memory configuration of the grey value correlator is shown in the following table. For the correlator application, memory block 1 was configured as 12 k words of program memory and 12 k words of 32 bits data and 4 k words of 16 bit data, memory block 2 was defined entirely as 16 bit data memory:

From:	To:	Block:	Description:
0x20000	0x22FFF	1	48 bits program memory
0x23000	0x23FFF	1	Gap
0x24000	0x27FFF	1	32 bits data memory
0x48000	0x49FFF	1	16 bits data memory
0x50000	0x5FFFF	2	16 bit's data memory

Configuring the main memory is done in a so called architecture file which describes the various memory spaces with their widths. The DSP offers the possibility to define almost any combination of memory areas with different sizes, for more information about these topics is referred to the "ADSP-2106x SHARC User's Manual".

EXTERNAL SRAM

The DSP-core can access external memory via its program memory, or data memory busses. External memory accesses by the DSP-core and the IOP are performed via the DSP's external port. This external port interfaces the internal busses to a single external bus, to which the SRAM devices (and also the boot memory device) are connected.

The external memory consists of 512k x 8 bits wide (zero wait stated) memory, and can be extended to 1Mb optionally. It is divided in 4 blocks of 256kb each. The reason for this division is that the DSP provides four memory selection signals that serve as chip select lines for the SRAM devices. This allows to connect the SRAM's to the DSP without additional logic.

The main advantage of using the four memory selection lines for the memories' chip select, is that they are set simultaneously with the address signals. An external address decoder to create chip select signals would introduce additional delay for the chip select signals, thus faster memory devices would have to be used for zero wait stated memory.

BOOT MEMORY

The DSP has a boot memory space, an 8 bits wide memory space in which an EPROM or non volatile-RAM can be mapped. This offers the possibility that after a reset, the DSP reads its code from this device, and starts up afterwards. This boot procedure, called EPROM-booting, is selected via a configuration input of the DSP. The extension board is equipped with a socket for a 1- or 4-Mbit (Non volatile) RAM or EPROM device. By default, the extension board is configured for host booting, installing a jumper will select EPROM-booting.

In case a non volatile RAM is used as boot memory device it is possible for the DSP to store data in this RAM. The boot memory socket has 4 jumpers accompanied with it, these are used to configure the pin allocation for either an (E)EPROM - or a (Non Volatile) RAM - device.

The boot memory space uses the same address-, data-, and read/write-lines as the main memory. To select the boot memory space, the DSP provides a special chip select line (BMS) that can serve as a chip select for the boot memory device.

F.3.2 Implementation of the external memory

As became clear in the previous paragraph, connecting external SRAM and the boot memory device is straight forward. Figure F.4 shows the memory configuration of the extension board.

The main memory is built up of four memory blocks of 256 kb (8 bits), each containing two 256k4 memory devices. These memory devices were chosen to use the four memory selection lines from the DSP optimally: there is no other logic required for decoding chip select signals for the memory devices. The number of wait states for the external SRAM is set to 0. In such case, a 33 MIPS ADSP 21062 requires memory devices with an access time of 17 ns.

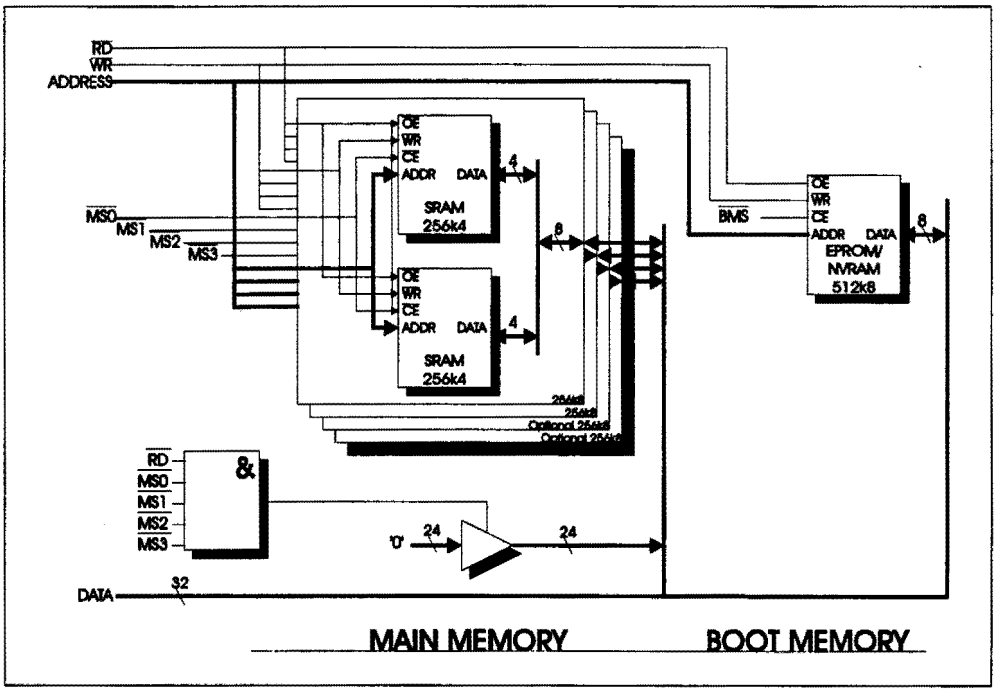


Figure F.4 : The memory configuration

A 24 bit's tri state buffer will drive the data lines 8 to 31 to a '0' in case an external memory read action is performed. The DSP reads 32 bits from the external memory and places this 32 bits data in eg. a data register. The external memory is physically only 8 bits wide, so if the 24 other data lines would remain undriven, the 24 most significant bits would contain undefined data. To prevent this from happening the tri state buffers were used. The buffers are enabled if one of the four memory banks is selected, and the read line is asserted.

The extension board is equipped with a socket to insert a boot memory device. Booting from this device is selected with a configuration input on the DSP. As became clear in the previous paragraph, the DSP provides the Boot Memory select line that serves as a chip select for the boot memory device. If EPROM booting is selected (or a DMA transfer is performed to boot memory), this line will be asserted by the DSP. The number of wait states for the boot memory is set to six at chip reset, in that way it possible to connect an EPROM with an access time of 200 ns for a 33 MIPS DSP.
Since the boot memory space is 8 bits wide, accesses from this device do not require driving the 24 address lines, because the IOP will only access data lines 0-7 while addressing the boot memory space.

F.4 TMS 34020 INTERFACE

The TMS 34020 interface provides the communication path between the TMS 34020 and the ADSP 21062 processors, further it allows the TMS 34020 to read status from the extension board, set it in a test mode and perform a hardware reset on the extension board.

Paragraph F.4.1 will describe the operation of the interface, paragraph F.4.2 the hardware and paragraph F.4.3 describes some error checks that the hardware performs on DMA transfers.

F.4.1 Operation

The interface between the TMS 34020 and the ADSP 21062 processors provides the inter processor communication and allows the TMS 34020 to read status information from the extension board and set it into a test mode.

Through the interface the TMS 34020 can access the entire memory on the extension board via two different procedures:
- via Host DMA transfers, the TMS 34020 directly accesses the memory on the extension board using the bus request protocol.
- via IOP-DMA transfers, the TMS 34020 uses of the DMA request/grant protocol, and thus the IOP to perform the transfer.

Besides data transfers, the TMS 34020 can perform actions like reading the boards status, setting the board in test mode, or even performing a hardware reset. This paragraph will describe the operation of the extension board.

HOST DMA

During Host DMA, the ADSP 21062 address-, data- and control-busses are acquired by the host. After the DSP has granted the busses, the TMS 34020 processor controls the DSP-busses, enabling him to access the external SRAM, the internal DSP memory and the IOP's control registers.
Since there is only a limited amount of TMS 34020 address lines available on the extension connector, it is not possible for the TMS 34020 to access the entire memory space of the DSP directly. The implementation is such that the TMS can access a 256 word memory page (32 bits wide) of the extension board directly through SBIP addresses 0x90000000 - 0x90001FE0. The page is selected via a hardware register on the extension board at SBIP memory address 0x9002020. (note: Since the TMS 34020 is a bit oriented processor, 32-bit memory addresses have an offset of 0x20).

The entire main memory of the extension board is divided in 2^{24} address pages of 256 words each. Via a 24 bits hardware register, that is mapped in the TMS 34020 memory space, one of these pages can be selected. When the TMS 34020 performs the actual memory accesses, the (32 bits) DSP - address consists of the 24 bits from the hardware page register (most significant value of the address) and the 8 address lines (least significant value of address) from the TMS 34020. This makes it possible for the TMS 34020 to access the entire main memory of the extension board.

This transfer procedure is required for booting the DSP from the TMS 34020, further it is used for transfers that do not require large data amounts, like giving commands to DSP and reading back the command results.
For transferring large amounts of data, the IOP-DMA transfer procedure was also implemented on the extension board.

IOP-DMA

For IOP-DMA accesses, the DSP's IOP is used to generate addresses to access memory on the extension board. It is intended to be used during large data block transfers (like templates) from the SBIP. The advantage of these accesses with respect to the Host-DMA transfer is that the TMS 34020 does not have to keep rewriting the page number every 256 memory accesses, and does not have to wait until the DSP grants it's busses. The transfer procedure during IOP-DMA is thus faster, allowing a higher transfer rate.

For an IOP-DMA transfer, parameters like the begin address and the block size have to be set in the IOP's control registers. Then the data is transferred by accessing one and the same SBIP-memory location. Every access to this memory location will generate a DMA request, and the IOP will handle the data transfer to the desired DSP memory location.

In fact, there are two memory addresses in the SBIP memory map that can be used for the IOP-DMA transfer:
1. writing to the external port FIFO buffer in the IOP's control register (control register is to be accessed via a host DMA)
2. a hardware DMA register, independent of the DSP, that is mapped in the SBIP's memory space at address 0x90002000

Accesses to the external port fifo buffer are to be done via host DMA. The external port fifo buffer is located in the IOP-register set and thus located in the memory range 0 - 0xFF of the DSP memory space.
Assuming that the DMA transfer is set up, the data is transferred by first writing the address page number of the IOP registers (page 0), followed by the actual accesses to the appropriate external port buffer memory-location. For the operation of the external port DMA is referred to paragraph F.2.1.

Accesses to the hardware DMA register in the SBIP's memory space do not require host DMA accesses and are therefor faster. The data will be set in a hardware register, the EPLD will generate the DMA requests to the IOP, which in turn will generate the appropriate memory addresses and write pulses to store the data in memory.

Note that the two IOP-DMA transfer methods require different setups of the DMA control register, since the 2nd concerns an asynchronous transfer, and the first one does not.

DSP EXTENSION BOARD MODE AND STATUS

A four bits read/write register at SBIP address 0x90002060 is used for reading status or setting the board in a specific mode. This register is located in the EPLD and does not concern accesses to the DSP's memory.

Reading this memory location returns four status bits:
- bit 0 represents an image transfer status bit. It is set to '1' if the DSP is using it's busses for the image transfer. Using this status bit, the TMS 34020 can check whether he is allowed to access the memory of the DSP extension board because such memory access will interrupt the image transfer.
- Bit 1 is set if an error occurred during a Host - IOP-DMA transfer

- Bit 2 is set if there was a FIFO full occurrence during a transfer from the SBIP, or a FIFO empty occurrence during a transfer to the SBIP
- Bit 3 is set if the DSP's IOP can not keep up with the rate at which DMA transfers are requested.

Writing to memory location 0x90002060 can be used for the following purposes:
- setting bit 0 to '1' reset's the status bis that are read from this memory location.
- setting bit 1 to '1' brings the extension board in a test mode:
  In this test mode the EPLD operates in a mode that incoming video from the SBIP's crossbar is passed back to the cross bar via the video input and video output FIFO's. This offers the possibility to test the video path, without needing to boot the DSP.
  During this test, interrupt line 2 of the DSP is pulled. If an EPROM with test software is inserted in the boot memory socket, this program may test the DSP and it's connection with the external SRAM to the memory.
  The three output flags of the DSP that are connected to the EPLD can be read by the SBIP from bits 0-2 of the status address (0x90002060).
- bits 2 and 3 are reserved

SOFTWARE RESET AND MODULE ID

The TMS 34020 interface contains an 8 bits register for module reset and identification. This register is located at SBIP address 0x70040000.

For writes to this address the allocation of the various bits is as follows:
bit 0: The hardware reset for the extension board.
By writing consecutively a '0' and a '1' the EPLD, the DSP and the FIFO memories on the extension board will be reset.
bit 1: Writing a '1' to bit 1 asserts interrupt line IRQ2 of the DSP.
bits 2-7: Reserved

Reading from address 0x70040000 returns the following information:
bit 0-2: Hardware revision code defining the revision number of the extension board hardware. In case hardware modifications are to be made in future to the board that involve software modifications as well, the software can detect the hardware revision and control the board accordingly.
bit 3: Timer expired output of the DSP.
This output is asserted by the DSP, when it's hardware timer is expired. It could be used as a watch dog to check if the software in the DSP is still running.
bit 4-7: These bits contain a module identification. For the extension board it is always 0x6, It can be used by the SBIP software, to auto detect the presence of the DSP extension board.

The following table summarises the address map of the extension board interface in the SBIP's memory space.

TMS 34020 address	direction	Description
0x70040000	read	Module&revision ID
0x70040000	write	SW reset & interrupt DSP
0x90000000 - 0x90001FE0	read/write	Host DMA space (256 words)
0x90002000	read/write	hardware IOP-DMA gate
0x90002020	write	Host-DMA page number
0x90002040	-	Spare
0x90002060	read	Extension board status/DSP flags
0x90002060	write	Reset error/set test mode
0x90002080 - 0x90002E0	-	Spare

F.4.2 The TMS interface hardware

The hardware implementation of the TMS 34020 interface, is quite simple. Figure F.5 shows the block diagram.

Host DMA transfers require that the DSP's address and read/write lines are set by the TMS interface. This requires the address page register, a tri state buffer to connect 8 TMS 34020 address lines to the 8 least significant DSP address lines and a tri state buffer to connect the read/write lines of the SBIP to those of the DSP.
Latch (4) is the address page register, it is loaded via the TMS 34020's data lines 8 to 32, when its clock input is asserted. (Which is done by the EPLD when the TMS 34020 writes to memory location 0x90002040).
Tri state buffer (3) connects SBIP's address lines 5 to 12 to address lines 0 - 7 of the DSP. Buffer (5) connects the read and write control signals of the SBIP to those of the DSP.
The output enable's of these tri state buffers and the address page register are activated by the DSP's bus grant line (HBG).
(The Bus request line is asserted by the EPLD when the TMS 34020 performs a host DMA transfer by accessing a memory location in the range 0x90000000 - 0x90001FE0)

The data bus connection of the TMS 34020's- and DSP's data busses during this host DMA access is done with transparent/registered tri state buffers (6) and (7). These devices connect the TMS 34020 bus to the DMA bus on the extension board, connecting the DMA bus to the DSP busses involves the bidirectional transparent/registered tri state buffers (1) and (2) in figure F.3 (processor block diagram).

During Host DMA transfers, all these devices are used in transparent mode:
For a write action to the extension board, device (6) in figure F.5 and (1) in figure F.3 are set transparent and enabled (devices (2) and (7) are disabled).

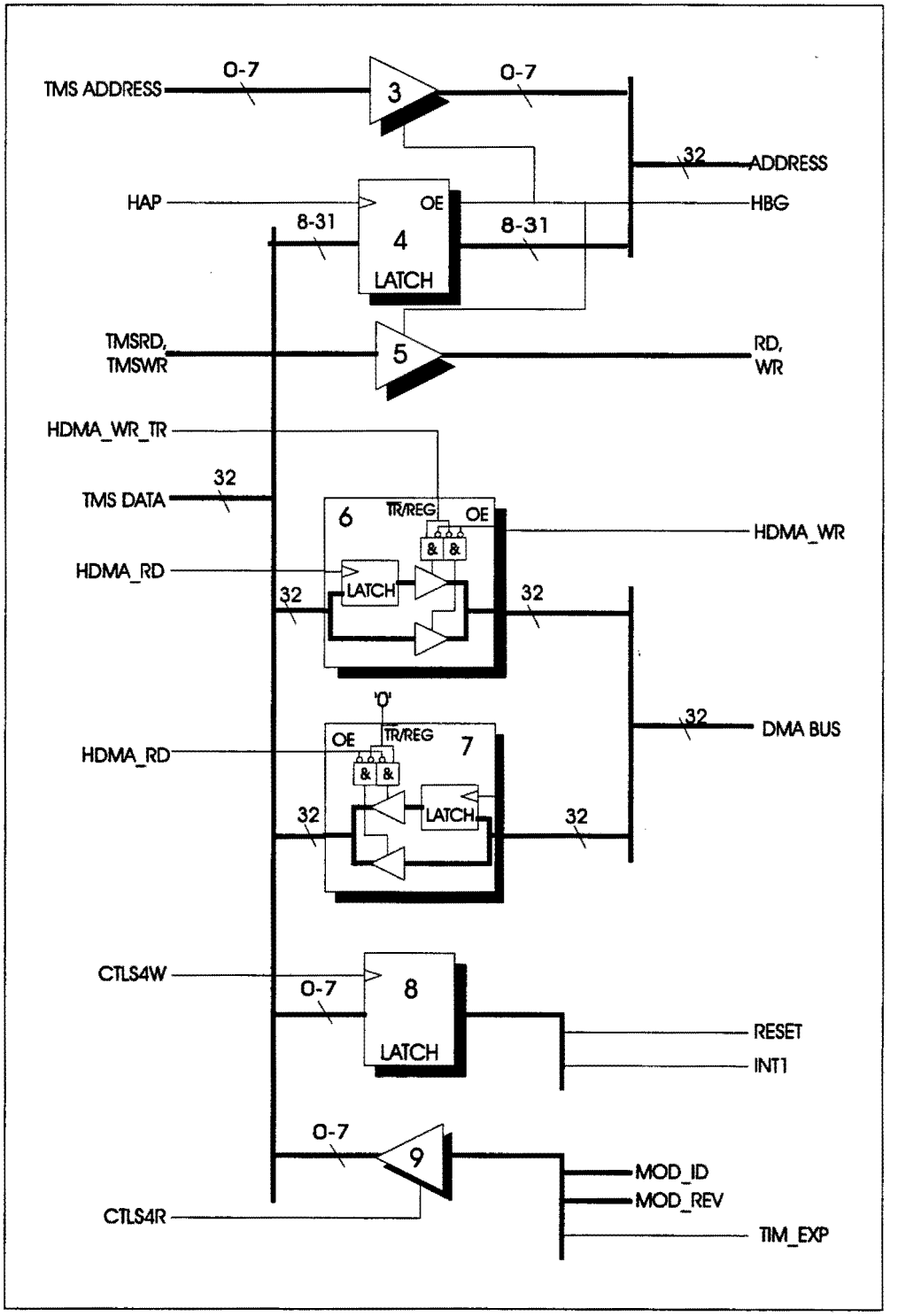


Figure F.5: The TMS 34020 interface.

In case of a read action, devices (2) and (7) are used in transparent mode and enabled (devices (1) and (6) are disabled).
The fact that the data is transferred through 2 transparent buffers (1 and 6 or 2 and 7) might seem strange, it is designed this way due to the hardware interface requirements from the DSP with respect to the asynchronous DMA.

During IOP-DMA transfers, the IOP generates the address and read/write signals, this transfer method only involves the DSP's data bus devices and not the address bus devices (3 4 and 5).
In case of an IOP DMA write cycle, device (6) is used as a registered buffer, and device (1) in figure F.3 is used transparent.
When the TMS 34020 writes the data value to the DMA port (0x90002000), the EPLD sets device (1) (figure F.5) in transparent mode and generates a write pulse for the register of device (6) (figure F.5) and enables it's output. The write cycle is then finished for the TMS 34020. The data that the TMS 34020 wrote into register (6) will appear on the DMA bus. Then the EPLD generates a DMA request to the DSP (via DMAR1). The DMA grant will enable the output of device (1) (figure F.3), the data that was present on the inputs of this device will appear on the DSP's data bus, and will be read by the DSP.
After the DSP de-asserts the DMA grant line, the EPLD disables the output enable of device (6) (figure F.5).

In case the TMS 34020 wants to perform the next IOP DMA write cycle, while the DSP did not grant the previous DMA request, the TMS 34020 will be halted by pulling it's ready line to '0' until the previous DMA cycle is finished. A watchdog timer in the EPLD will prevent hang up situations: In case the DMA grant does not occur within 7 DSP cycles after the DMA request, an error bit will be set, and the TMS 34020 will not be halted any longer.

For an IOP DMA read cycle, device (7) is used in transparent mode, and device (2) in figure F.3 is used in registered mode. The operation is as follows:
When the TMS 34020 reads from address 0x90002000, a DMA request is performed, the TMS 34020's ready pulse is pulled to '0' and the outputs of device (2) and (7) are enabled. When the DSP grants the DMA request, the data is written into device (2) and appears on the DMA bus, since device (7) is used in transparent mode, and it's output is enabled, this data will appear on the TMS 34020's data bus. By releasing the TMS 34020 ready pulse, this DMA cycle will be finished.
Also during this IOP DMA read transfer, a watchdog timer in the EPLD prevents a hang up situation, by releasing the TMS 34020's ready pulse if the DMA request is not granted by the DSP within 7 DSP cycles and setting an error bit in that case.

After this IOP DMA read transfer is completed, the EPLD will perform a next DMA request. So while the TMS 34020 is writing the previous data word into it's memory (or processing it), the IOP transfers the next data word to device (2) and (7) already. In such case, the next read transfer by the TMS 34020 will not require wait states.

All the required signals like output enables, clock signals for the registers, DMA requests etc. are generated by the controller.

F.4.3 Hardware checks on operation

In the previous paragraph it became clear already that the EPLD sets an error bit in certain cases. This is to be done to detect that the data was not accepted by the DSP, or that the DSP did not return data. This error condition only plays a role during IOP DMA transfers. Errors might occur in the following cases:

- the DMA transfer was not set up properly.
- the DMA channel was idle because it has no priority.

The consequence is that the DMA transfer requires a transfer protocol. Before the transfer is started, the error flags in the EPLD are to be reset. When the transfer is finished, the error bit should be checked to verify that no errors occurred.

Host DMA accesses do not need error checking because they will never fail. The DSP will always grant its busses when it's host requests them. Since the host bus request has priority over both the DSP-core and the IOP, no hang up condition can occur.

F.5 VIDEO INTERFACE

The interface between the SBIP and extension board consists besides the processor interface (section F.4) also of a video interface. This interface consists of 10 inputs and 10 outputs of the crossbar switch of the SBIP, and 4 synchronisation signals. The video interface makes it possible to transfer images to/from the SBIP at video speed (up to 20 MHz data transfer). By setting the proper selections in the SBIP's crossbar switch, images can be transferred to the extension board while they are being digitised.

This section is set up as follows: Paragraph F.5.1 will give a general description on image transfers. Then paragraph F.5.2 will describe the hardware for the image transfer. Since the operation of the image transfer is quite complicated, paragraph F.5.3 explains the image transfer procedure. At last paragraph F.5.4 describes a number of checks that are performed during the image transfer.

F.5.1 Image transfers

The video interface facilitates the transfer of images from the SBIP to the extension board. The image transfer requires except the 2 x 10 lines that carry the actual video information, the 4 following synchronisation signals:

a The pixel clock,
in fact the clock signal for to the ADC that digitizes the analogue camera input. Pixel data from the cross bar is available on the rising edge of this pixel clock.

b The Blanking line,
defining which portions of the image lines contain valid data

c. The Vertical synchronisation signal,
defining the start of a frame.

d. The Horizontal synchronisation signal,
defining the start of an image line.

The image can be transferred using different video standards. A main division can be made in interlaced and non interlaced images: An interlaced image consists of two frames, one containing the even image lines, the other containing the odd image lines. The two frames are transferred alternately by the video camera. Non interlaced images are transferred in one frame, that contains all video lines. Other differences between video standards exist in the image size and image frequency.

The SBIP can generate video signals for virtually any combination of image size and image frequency in interlaced or non interlaced mode. In practical situations, however, only a few standards are being used, since most video cameras have a standardized video output. With the SBIP, CCIR (625 lines, 25 images/sec.) or RS170 (525 lines, 30 images/sec.) are most frequently used.

The SBIP digitizes the incoming (analogue) video signal which results in a 2 dimensional array of pixels in an image memory. Common sizes of these digitized images are eg. 256 x 512, 512 x 512, 512 x 768 and 512 x 1024 pixels.
The number of rows in an image, 256 or 512, show that the SBIP does not sample each of the 525 or 625 image lines (for RS170 and CCIR video respectively). Thus for eg. a 512 x 512 image from a CCIR camera (625 lines), 113 lines are not used.
The number of columns in the image, 512, 768 or 1024, depend on pixel clock frequency (the sample frequency of the ADC). This frequency ranges from 10 to 20 Mhz on the SBIP2, resulting in 512 - 1024 pixels per row respectively.

The extension board should thus also be capable of transferring images with any of the video standards that the SBIP supports. The most important requirements for the image transfer procedure are that it should be able to:
a. handle transfer rates up to 20 MHz
b. de-interlace the incoming image, so the image will be stored in memory linearly.
c. transfer images with different sizes (number of rows, pixels per row)

In chapter 3.6 became already clear that the peripherals of the DSP contribute an important part in the image transfer, however, some additional hardware is required to perform the image transfer. This will be explained in the following paragraph.

F.5.2 The video interface hardware

To transfer images from the SBIP to the extension board via the SBIP's crossbar switch, some additional hardware was required. This paragraph will explain this hardware.

HARDWARE ARCHITECTURE

Figure F.6 shows an overview of the hardware for the image transfer.

It only consists of a tristate buffer, two latches and two FIFO memories. A part of the controller, also plays a very important contribution to the image transfer.

The Video input consists actually of 10 lines, 9 of these lines are fed via a latch and through the fifo memory to the DMA bus of the extension board. (only 9 lines are used since most FIFO memories are 9 bits wide). Also at the output side, 9 data lines of the extension boards DMA bus are fed via a FIFO memory and a latch to the SBIP's crossbar switch.

At the extension boards side, the video interface is connected with 16 lines to the DMA bus. Only 9 bits contain information from/to the SBIP's crossbar switch. Even though the image memory on the extension board is only 8 bits wide, the 9th

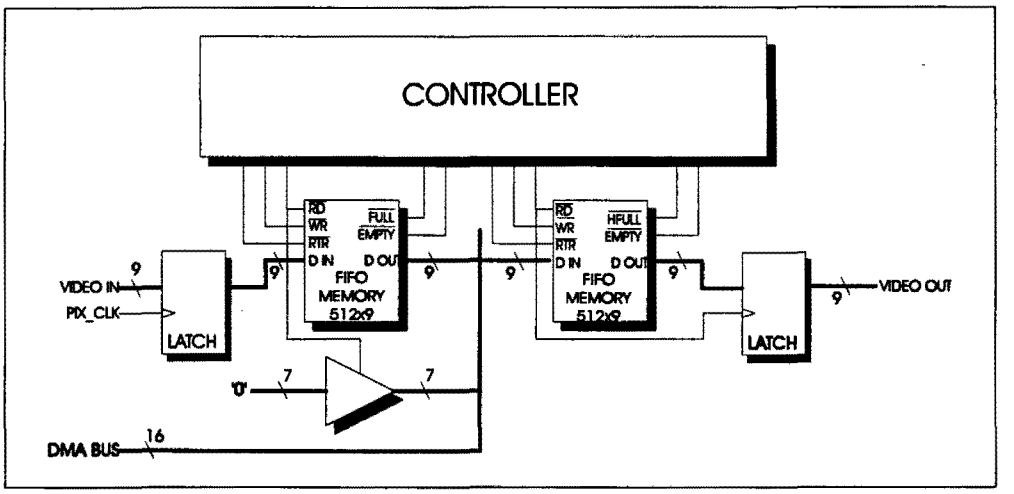


Figure F.6: The video interface hardware.

bit might be usable for transfers to the internal DSP memory.

The read input of the FIFO memory is connected to the clock input of the register in device (1) of the DSP's DMA interface (figure F.3). These two inputs were both connected to one output pin of the EPLD to eliminate time differences on the FIFO's read pulse and the latches clock input pulse.
If the two signals would be connected to two different output pins of the EPLD, there would be a chance that a delay time difference for the activation of these two output pins will lead to situations that the latch is triggered while the output data of the FIFO memory is not yet stable (the output is changing). Therefor the register in device (1) in figure F.3 was dedicated to the FIFO memory.

The tristate buffer drives the remaining 7 of the 16 output lines to the DMA bus '0'. If the transfer is performed to 16 bits internal memory of the DSP, all 16 data bits will be defined: the 9 least significant bits contain the information from the SBIP, the 7 most significant bits are '0'.

FUNCTION OF THE FIFO MEMORY AND IT'S SIZE

The FIFO memories are required to buffer the data stream to/from the SBIP.
There are two reasons for buffering the data stream:

a The IOP needs 2-3 machine cycles to respond to a DMA request.
When the video stream is coming in, the DMA controller has to start generating addresses and write pulses to write the data to memory. When the first pixel arrives, the EPLD generates a DMA request. Then the DMA controller requires some time to generate the appropriate address (because it's operation is pipelined). After 2-3 machine cycles (67 - 100 ns), the address is present on the address bus and the DMA request will be granted. In the mean time, one or two pixels were transferred by the SBIP already, these are buffered in the FIFO memory.

b Memory access times
Principally, the data from the FIFO memory could be written into the external SRAM at a rate of one word per machine cycle, thus at a transfer rate of 33 M

words / sec. However, this would require extremely fast SRAM to meet the data setup times.
Possible solutions for this problem were to insert one wait state for every DMA - memory write cycle, or to perform the transfer via internal DSP memory. Both solutions have the consequence that the transfer rate will be 16.7M words per second (with a 33 MIPS DSP). One might remark that this transfer rate is too low since the video data comes in at a 20 MHz rate. This is true, but during the blanking (approx. 12- 13 us per line time), the FIFO memory will not be filled. Transferring 1024 pixels at a rate of 20 MHz requires 51.2 us. Reading this data from the FIFO memory at a rate of 16.7 MHz takes 61 us. Since one line time equals at least 63.5 us. the contents of one image line can thus be read within one line time from the FIFO memory at 16.7 MHz.

Since the transfer from the FIFO memory will take two machine cycles per word, the required depth of the fifo memory can be calculated. At the start of an image line, the FIFO memory will be empty. The FIFO memory will then be filled at a rate of 21 MHz (worst case situation assumed), and read at a rate of 16 MHz. The required size of the FIFO memory can be calculated from the difference between these two rates. The difference in these frequencies is the rate at which the FIFO memory runs full, in this case a rate of 5 MHz.
Transferring the 1024 pixels at a rate of 21 MHz takes approx 49 us. In these 49 us the number of filled locations is $50 * 10^{-6} * 5 * 10^{6}$ = approx. 250. Thus a 512 deep fifo would be sufficient.

CONTROLLER OPERATION

One task of the controller is to derive the frame ID (odd or even) in interlaced mode, from the horizontal and vertical synchronisation pulses. The time difference between the rising edge of the Vsync and the Hsync pulses determine the frame. In case an even frame is transferred the rising edge of the Hsync is in the middle of an image line, thus right in the middle between two Hsync pulses.
In case an odd frame is transferred, the rising edge of the Hsync will occur simultaneously with the Hsync pulse. A part of the EPLD analyses where the rising Hsync edge occurs, concludes whether the frame is odd or even and sets the frame output accordingly.

Another task of the controller is to regenerate the blanking signal. The blanking signal defines which pixel clock periods of the image line contain valid image information that is to be stored in the image memory of the extension board. Since the edge of the blanking signal occurs simultaneously with the rising pixel clock edge in certain cases, it can be ambiguous to detect at which pixel clock edge the video data inputs from the crossbar carry valid video information. This might result in the fact that entire video lines might be shifted by one pixel. Therefor the blanking signal is regenerated by sampling it on the falling pixel clock edge, this results in the fact that the image on the extension board memory is shifted by one pixel with respect to the image.

The most important task of the controller is to support the DSP's DMA controller with the image transfer. This will be explained in the following paragraph.

F.5.3 Image transfer operation

The image transfer between the SBIP and the extension board, is partially handled by the hardware that was explained in the previous paragraph, and partially by the DSP's DMA controller. This paragraph will explain how the image transfer is actually done, it is divided into several sub-paragraphs:
First will be explained how the image transfer generally operates. Since an image transfer actually consists of 512 line transfers (assuming an image with 512 rows is to be transferred), the operation of the hardware is explained by discussing the transfer of one image line to the extension board divided in three sub-paragraphs: one explains the transfer from the SBIP to the FIFO memory, the second concerns the transfer from the FIFO memory to the DMA bus, and the third explains the transfer from the DMA bus to the external memory.
At last some restrictions on the application software in the DSP will be stated.

GENERAL IMAGE TRANSFER PROCEDURE

The general operation of the image transfer is as follows:
1. The DSP receives the command from the SBIP to perform an image transfer.
2. It sets the output flags such that the EPLD puts the Vsync pulse from the SBIP on the DSP's input flag, and it awaits a rising Vsync edge (start of frame).
3. Then it sets the output flags such that the EPLD performs a get image, or put image. In that case, the EPLD sets the frame ID on the DSP's flag input. In case of an interlaced image transfer, the DSP checks whether the next frame will contain the even or odd image lines.
4. The DSP starts a DMA transfer of 1024 data samples (1 image line) to the appropriate memory locations and enables interrupts. The interrupt service routine will then become active after every line transfer.
5. After the first line is read in, the interrupt server becomes active, which will initiate a DMA transfer for the next line.
6. After the second frame is transferred completely, the interrupt server will reset the get image command code and disable DMA interrupts so it will not become active any more.

The image transfer consists actually of a number of line transfers that are done consecutively. Since every line transfer procedure is identical, the operation of one transfer is explained. The transfer is divided in the transfer from the SBIP to the fifo, from the fifo to the DMA bus and from the DMA bus to the external SRAM.

TRANSFER SBIP TO FIFO

The FIFO control at the SBIP's side is fairly simple, figure F.7 shows the video timing for one image line.
The operation is as follows:
While the blanking signal is active ('0'), the information on the video input is not valid, and will thus not be written into the FIFO memory.
As soon as the blanking becomes inactive, meaning that the video input contains valid information, every sample on the video input is written into the FIFO memory. Since the blanking signal is sampled on the falling pixel clock edge, pixel 0 of the image line that is being transferred will not be written into the FIFO memory. Thus pixel 0 of a certain image line in the image memory of the DSP extension board equals pixel 1 of that image line that resides in the image memory on the SBIP.

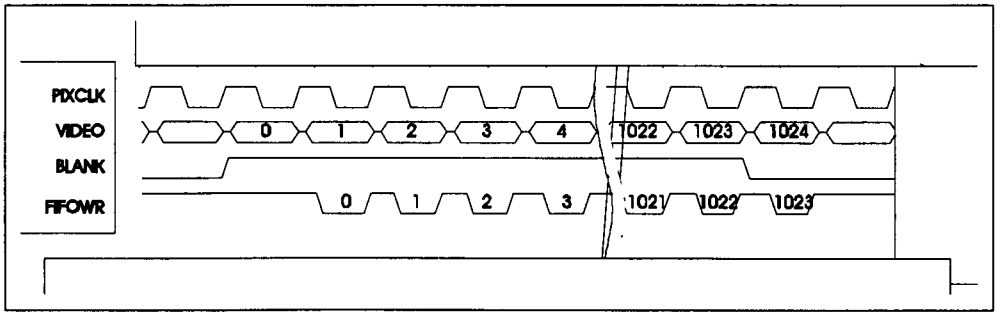


Figure F.7: FIFO memory write control

The blanking signal remains inactive for exactly as many pixel clock tick's as there are pixels in a row. Thus if the SBIP has to get an image with 1024 pixels per line, the blanking remains inactive for 1024 pixel clock ticks.

Another process in the EPLD takes care that the data in the FIFO is read out and transferred to the data bus. This will be explained in the next sub-paragraph.

TRANSFER FROM FIFO TO DMA BUS

The transfer from the FIFO to the IOP concerns actually two processes in the EPLD. One generates the DMA requests to the DMA controller, the other the FIFO read pulses.

Figure F.8 shows a timing diagram of transfer from the FIFO memory to the DSP.

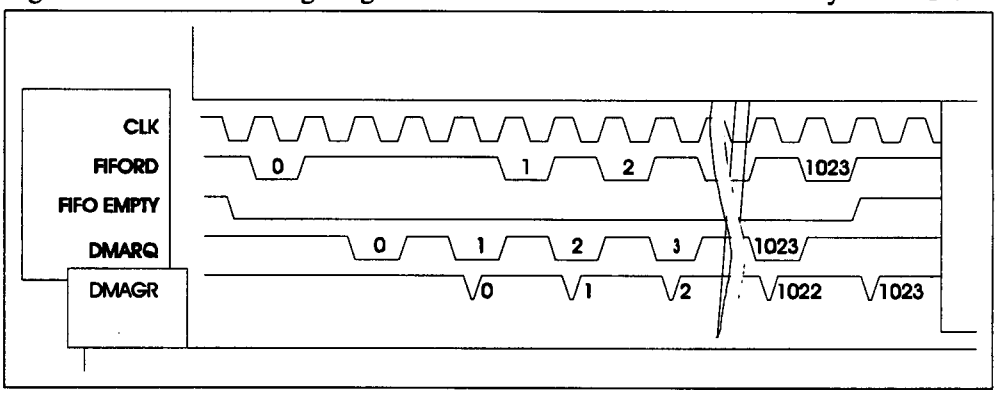


Figure F.8: Timing diagram of transfer from FIFO to the DSP.

The first process performs a DMA request for each write pulse to the FIFO memory. The process conditionally generates the DMA requests. Since the data transfer rate can be higher (20 MHz) than the DMA request rate (16.67 MHz) the process consists of a counter that is incremented every time a pixel is written into the FIFO memory.The counter is decremented every time a DMA request is performed. Initially, the process waits until the first data sample is available in the hardware register. This is after the first FIFO read pulse. Then it performs a DMA request once every two machine cycles, as long as this counter is non-zero and the difference between DMA requests and the DMA grants is smaller than 7 (to prevent a DMA request overrun on the DMA controller).

The second process monitors the FIFO's empty flag, and the DMA grant line. When data was written into the FIFO memory, it's empty flag will become inactive. On that occurrence, the EPLD performs a FIFO read. Since the FIFO read signal is hardwired with the clock input of the hardware register of the DMA bus, the first pixel value is also written into the hardware register. After the DMA is granted, the next read cycle from the FIFO memory is performed, provided that the FIFO memory is not empty. Because the DMA request are performed once per two machine cycles, the DMA grant pulses will also occur once every two machine cycles. The FIFO is read in the intermediate machine cycle (between the two DMA grant pulses, as shown in figure F.5.3.

TRANSFER FROM DMA BUS TO EXTERNAL MEMORY

The transfer from the DMA bus to the image memory can be done in two different manners as mentioned in the previous paragraph.
In case the data from the DMA bus is written straight into the SRAM, one has to insert a wait state during each memory write cycle, to comply with the required memory write timing of the memory devices.
Writing data to the internal DSP memory does not require wait states. The image can also be transferred from the DMA bus to the internal DSP memory, and from this memory to the external memory.

Transferring the image from the DMA bus to a small buffer in internal DSP memory, and from this buffer to the external SRAM has to be done as follows:

1. Per image line, a DMA transfer is set up to transfer the image line from the DMA bus (FIFO memory) to the buffer in internal DSP memory.
2. The interrupt server becomes active when that transfer is finished.
   This server initiates two (simultaneous) DMA transfers:
   - one to performs a transfer from the internal buffer to external SRAM (taking the de-interlacing procedure into account).
   - The other to perform the transfer of the next image line from the DMA bus to the internal buffer.

Here was chosen to transfer the image to the memory via an internal buffer memory for the following reasons:

- The interrupt server can perform transfers from the internal buffer conditionally or partially: in this way it is possible to extract only a certain (eventually sub-sampled) window from the image being transferred by the SBIP.
- The DSP-core could perform operations on the image data that resides in the internal memory buffer.

The two simultaneous DMA transfers (one from DMA bus to internal buffer, the other from internal buffer to external image memory) are possible:
the EPLD performs the DMA requests once every 2 machine cycles, the IOP grants these DMA requests also once every two machine cycles.
Thus in the interleaved machine cycle, the DMA controller will schedule the transfer from the internal buffer to the external SRAM.

An internal buffer size of one image line will be sufficient because the transfer from the internal buffer to external SRAM will always be ahead of the transfer from the DMA bus to the internal buffer:

the transfer from the buffer to the external memory is initiated right after the transfer from the DMA bus to the buffer was finished. After that time, there will be a time difference of approx 40-50 machine cycles before the next transfer from the DMA bus to the internal buffer will be started. During this time, only the transfer from the buffer to external memory is busy, and can thus transfer one word per machine cycle. The transfer from the DMA bus to the buffer becomes busy again when the first pixel from the next image line is written into the FIFO memory. From that moment, the data bus bandwidth of the DSP is equally divided over the two DMA channels, each of these two channels will alternately perform one word transfer per machine cycle: thus the transfer from the buffer to external memory (which is reading the line buffer) stays ahead of the transfer from the DMA bus to the internal buffer (that is writing to the line buffer).

RESTRICTIONS ON SOFTWARE DURING IMAGE TRANSFERS

From the previous sub-paragraph became already clear that the DMA procedure has a very tight timing schedule. There are certain software restrictions to make sure that the image transfer operates correctly:

a. no other interrupts should become active during Hsync pulses,
   The DMA interrupt will become active during that time, and has to start a new DMA transfer within a short time. If another interrupt keeps the DMA interrupt pending, pixels of the following line might be lost.
b. the TMS 34020 should not perform host DMA transfers,
   In such case, the IOP will loose control over the DSP's busses and can not perform any data transfers.
c. the IOP should have bus priority over the DSP-core.
   In that case there are no further restrictions on the software that is running on the DSP-core with respect to external memory accesses. External memory accesses by the DSP-core are allowed, but they will be delayed significantly.

The consequences for software that is running on the DSP-core during image transfers are:

- External memory accesses will be delayed, since the IOP has bus priority over the DSP-core.
- The program will be interrupted once per line time, this will consume approx. 2% processor time during an image transfer.

F.5.4 Hardware checks on transfer operation

The hardware also performs the following checks during the image transfer:

- Check on input fifo overrun
  If the input fifo ever becomes full, one is never sure whether the input fifo has been exactly full, or whether it overran. In the first case, there is no problem, in the latter, data samples will be lost. Therefor an error is set, every time the video input fifo becomes full.
- DMA request/Grant difference.
  If for one or another reason, the DMA controller does not respond to DMA requests and the hardware would keep generating DMA requests, the operation of DMA controller would be undefined. Therefor, no more DMA requests will be performed if the difference between the amount of requests and grants reaches 7. At that moment, also a warning bit will be set, that the DSP actually can not keep up with the DMA requests.

The TMS 34020 should read these bit's after an image transfer. If the error bit is set, the image transfer was corrupted most probably.
The warning bit only tells that the IOP is having problems keeping up with the transfer rate. It does not imply that the transfer was corrupted.

F.6 CONTROLLER

The controller is besides the DSP the most complicated part of the design, it forms an interface via some 50 -60 lines between TMS 34020, SBIP's video interface, FIFO memories and the ADSP 21062. Only a brief description is given here on its operation and architecture.
The controller was implemented in an Erasable Programable Logic Device (EPLD), an Altera EPM 7160. The operation of the EPLD was described entirely in VHDL language. The VHDL routines perform the various functionalities as they were mentioned in the previous paragraphs of this chapter. This paragraph will give a brief overview of the architecture of the EPLD.

The EPLD operates in one of the following modes:
- idle mode
  The EPLD is waiting for a command from DSP or the TMS 34020
- image in mode (set by DSP)
  The EPLD generates all the required signals to transfer an image from the SBIP to the extension board.
- image out mode (set by DSP)
  The EPLD generates the required signals to transfer an image from the extension board to the SBIP
- test mode (set by TMS 34020)
  The EPLD generates the required signals to transfer an image from the SBIP's crossbar switch through the FIFO memories back to the crossbar switch.

In every mode, the TMS 34020 accesses have the highest priority. If it accesses the host DMA memory area, or writes to the hardware DMA port, the EPLD will change to idle mode, since these operations conflict with image transfer operations.
If the TMS 34020 performs an operation that does not conflict with the image transfer operations, like reading the test/error flags or writing the host address page, the EPLD will not change mode.
Also if the TMS 34020 wants to set the EPLD into test mode while an image is transferred to or from the memory on the extension board, this image transfer will be interrupted, and the extension board will be set into test mode.

Figure F.9 shows the architecture of the EPLD. The following units are found inside the EPLD.
- TMS interface
- DSP interface
- Video interface
- image transfer controller
- Host DMA interface
- Memory controller
- Test/reset

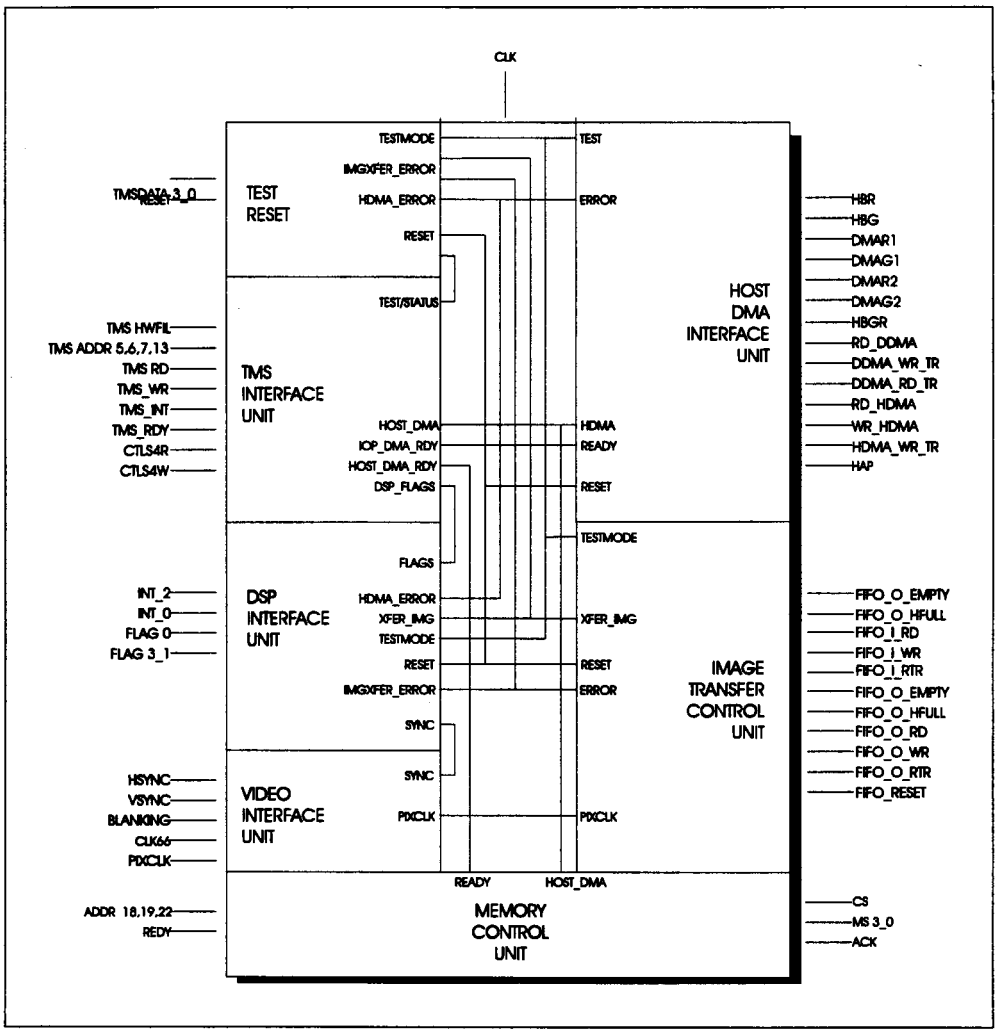


Figure F.9: Architecture of the EPLD.

The TMS interface decodes the memory map as shown at the end of paragraph F.4.1. Therefor it requires 3 address lines, the decoded selection line HWFILL of the SBIP, and read and write signals.
The HWFILL signal is the output of an address decoder on the SBIP. It becomes active when the SBIP accesses memory in the range of 0x90000000 to 0x9FFFFFFF. Only four address lines were used for the local address decoding, since there was a shortage of IO pins on the EPLD. This means that the 16 address locations from 0x90002000 - 0x900020F0 are mirrored 15 times in the range 0x90002100 to 0x90002F00.
Also the 512 words ranging from 0x90000000 to 0x90002FE0 are mirrored again in the memory range from 0x90003000 to 0x9FFFF000.

In case the TMS accesses memory locations in the memory map, the TMS interface will activate the appropriate units in the EPLD:

- for a DMA accesses (host- or IOP- DMA), it activates the host/dma interface to perform the desired operation. In case of a host DMA, it also activates the memory controller, that in turn activates the memory select signals for the DSP's CS (internal DSP memory accesses) or one of the 4 MS lines (external memory accesses).

- to read status, reset errors, or set test mode, it activates the test/reset unit.

The DSP interface decodes the commands from the DSP via three input flags, and returns a status to the DSP via the output flag.
The command from the DSP, for an image transfer to or from the SBIP, the DSP interface activates the image transfer controller, which in turn will control the FIFO memories and generate DMA requests via the host/dma interface unit.
For each DSP command, the DSP-interface will set the corresponding status signal (vsync, frame id, hsync, error flags etc. see paragraph F.2.1) on the output flag and the DSP's external interrupt line. It receives these signals from the various unit's within the EPLD.

The Video interface is always active and performs the following tasks:
- regeneration of the blanking signal
- odd/even frame detection
- Rising pixel clock edge detection

This information is required for image transfers and is used by the FIFO controller unit.

The Image transfer controller unit contains the required functionalities to transfer the image to or from the extension board, or to transfer images in test mode. It is activated by the DSP interface or the test/reset unit, and uses the output of the Video interface. The image transfer controller unit controls the FIFO memory directly, and performs DMA requests via the HOST/DMA interface unit.

The host DMA interface unit performs tasks like IOP DMA transfers and Host DMA transfers. This involves controlling the various registers and buffers as explained in the previous paragraphs and performing DMA requests to the DSP.
To perform these tasks, this unit is activated by the TMS interface.
In case the TMS 34020 wants to perform a host DMA transfer, this unit will activate the memory controller unit, that will generate the required selection lines for the memory devices.
This unit also passes DMA requests from the Image transfer controller unit to the DSP.

The test/reset unit resets the entire EPLD at start up and it offers the possibility to set the extension board in a test mode. Further it provides the possibility for the TMS 34020 to read status information (about image and DMA transfers) from the EPLD.

The memory control unit is activated by the host DMA interface. In case the DSP grants it's busses, the memory control unit takes over the task of asserting the memory selection lines for the external SRAM. From 3 address lines of the DSP, it derives which of the four external memory banks or internal DSP memory is selected.

The figure does not show a number of unused connections between the EPLD and the SBIP and DSP. These were reserved for future use and are defined to be high impedant.

G Software design

The operation of the grey value correlator contains besides the hardware a considerable amount of software. This software consists of the application and system software on the DSP and of software for the TMS 34020 and the SBIP's host computer. The aim of this appendix is to explain how the software for the extension board was set up.

First the environment in which the software runs will be described in section G.1. Then the DSP software is described in section G.2. Section G.3 gives an overview of the functions for the SBIP and the host.

G.1 SOFTWARE ENVIRONMENT

Writing software for the extension board requires besides knowledge about the application, also knowledge about the system on which the software is running. The software has to interface to the software that is running on the SBIP and to the extension board hardware.

Paragraph G.1.1 will describe how the software of the extension board interfaces to the SBIP software. Paragraph G.1.2 explains some definitions that are used on the SBIP software and that are adapted on the extension board. Appendix F explained how the software has to control the extension board hardware. However, the memory was only explained briefly. Paragraph G.1.3 will give a more detailed description on the memory configuration of the extension board.

G.1.1 Software interface to the SBIP

The software interface between the DSP extension board and the SBIP was adapted from that between the SBIP and the host computer. This paragraph will give a short description on the implementation of the software interface between the SBIP and it's host computer.

In a configuration of an SBIP with a host computer (PC or VME based CPU) the SBIP operates as a slave of the host computer. After it is initialized it waits for commands from the host, it executes these commands, and returns the results to the host. The software on the SBIP contains in many cases only of a large library with vision functions with a software interface to the host. Via this interface the SBIP receives commands from the host to execute a function from the vision library, it then executes this function and returns the results to the host. In this way the SBIP performs the image processing and reports results to the host software. The host software performs the actual interpretation of these results. Thus the vision application is in fact partially running on the SBIP and partially on the host.In the SBIP environment, this is the most simple solution to solve vision problems. In cases where the SBIP operates stand alone, it will be necessary to implement the entire application on the SBIP itself, thus without using a host.

In the software interface, a function ID is assigned to each function of the vision library. When the host has to call a certain function on the SBIP, the software interface on the host side first writes the function arguments into a communication buffer. When it then writes the function ID at the top of the buffer. The TMS 34020, which is continuously polling the top of the buffer, detects that the host wrote the function ID. It reads in turn the function ID and passes the arguments from the communication buffer to the appropriate function. After the function is executed, it writes the return parameters into the communication buffer and resets the command ID to notify the host that the command is finished.

The vision library with the software interface between the host and the SBIP provide thus a transparent interface between the host and the TMS 34020. The functions that are available in the vision library on the SBIP are also present in a library on the host. The function implementation on the host just copies the function arguments that are passed to it, to the communication buffer. A command handler on the SBIP passes these parameters that it receives via the communication buffer to the appropriate function. The function implementation on the SBIP performs the actual algorithm.
This interface offers thus the possibility for the host to call any of the functions from the vision library. Since a certain function exists in both host- and SBIP environment, it is possible to develop and debug application software for the SBIP on the host computer, and once it is found to be working satisfactorily, the source code can be compiled with a TMS 34020 cross compiler, and ran on the TMS 34020 itself.

Of course there are restrictions on having the capability of compiling one and the same source for both the host and the TMS 34020 environments. The programmer should use predefined functions and macros for operations on images. Let's take the example that he likes to get the grey value of a pixel: This requires to load a pointer variable with the corresponding memory location where the grey value of that pixel is stored. Since the image is mapped in the SBIP's memory space, and not in the host's memory space, this pointer implementation will only work in the TMS 34020 environment, not on the host environment.
Therefor, the programmer should use the getpixel() function/macro, rather then manipulating a pointer though the image memory. In the host environment, the getpixel() function delegates the action to the SBIP via the communication buffer, the getpixel()-macro in the TMS 34020 environment is in fact the pointer operation. If the programmer keeps these kind of restrictions in mind, it is fairly easy to implement an application on the SBIP.

One of the requirements of the extension board is that the software interface should be very easy, such that other algorithms can be implemented on the extension board without much difficulties. It is therefor obvious to adapt as many as possible from the SBIP's software environment to the extension board. This has the following advantages:
- Programmers with SBIP experience are familiar with the software environment of the SBIP, and can therefor implement algorithms easier on the extension board.
- If the same structures are used, it will be easier to port existing SBIP software to the DSP.

Adapting the software structure from the SBIP resulted in a DSP library for the DSP, one for the SBIP and one for the host computer. The DSP-libraries on the host and the TMS 34020 are an extension to the current vision library.
The DSP operates as a slave of the TMS 34020, it receives commands from it, executes them and returns results to the TMS 34020.

Figure G.1 shows an overview of hierarchy of the host, the SBIP and the DSP extension board.

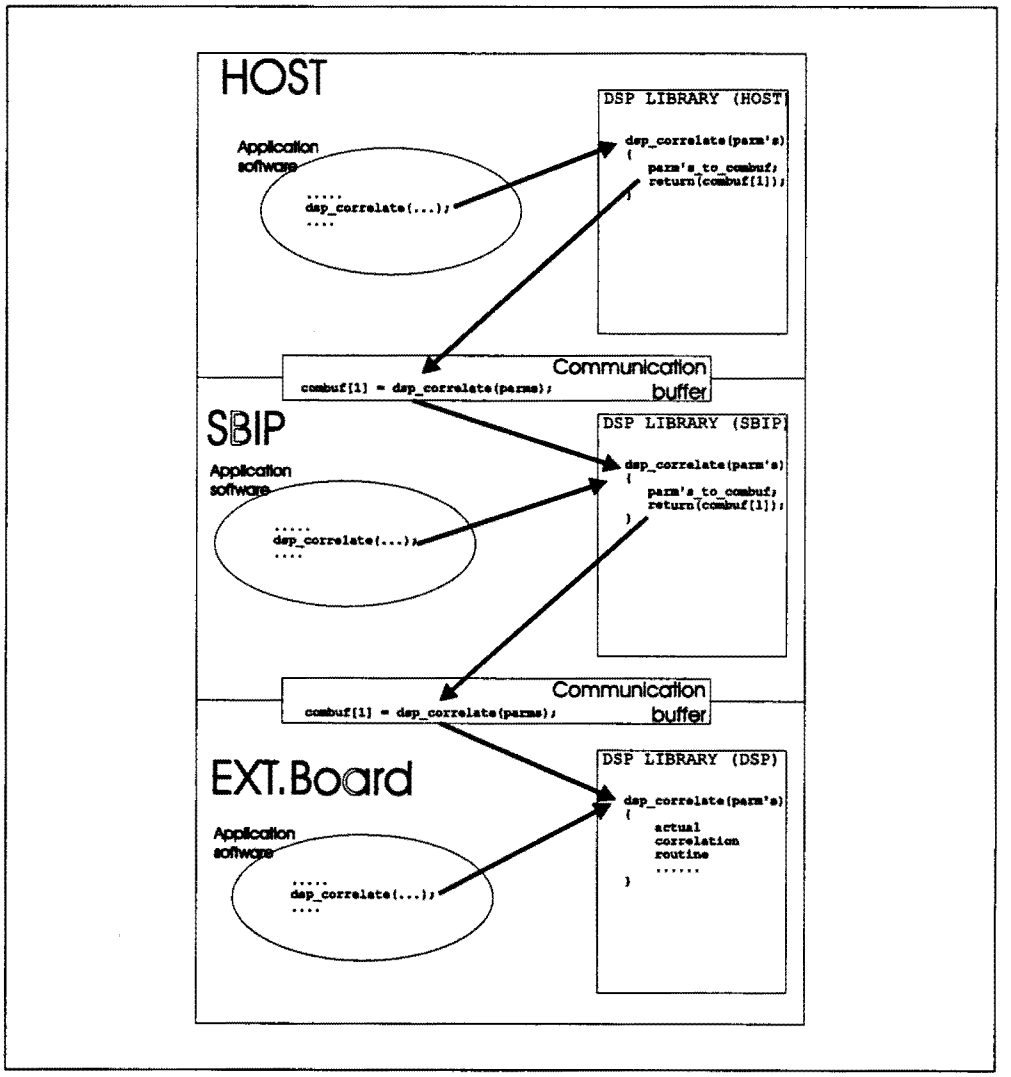


Figure G.1: Software interfaces host - SBIP - extension board.

There are basically two software interfaces that play a role: the one between the host and the TMS 34020 and the other between the TMS 34020 and the DSP.
The software interface between the host computer and the TMS 34020 provides the possibility to call the DSP routines also from the host computer. This is necessary to be able to debug the DSP functions also on host level.
The TMS 34020 passes parameters that it receives via this interface to the functions in the DSP library that is on the SBIP. These functions can either be called by the application software on the SBIP or by the host software interface. The DSP passes parameters that it receives from the SBIP to the appropriate function, after that function is executed, the results are passed back to the SBIP.

The software interfaces provide thus a transparent communication between the various processors. They also serve some other purposes, like booting the DSP and a properous hardware control of the extension board.

As shown in figure 4.2, DSP-function calls by the host are passed to the DSP via the TMS 34020. In fact, every command could be issued straight from the host computer to the DSP. For test purposes, it came out very handy to issue commands directly from the host to the extension board without intervention of the TMS 34020 CPU. For normal operation, however, it is a better practice to involve the TMS 34020 in passing commands to the extension board. In that case the DSP receives the tasks always from one and the same CPU, thus no conflicts like the host and TMS 34020 calling a function simultaneously, can occur.

G.1.2 Image definitions on the SBIP

This paragraph explains some definitions on images that are used in the SBIP software. Only the definitions that play a role for the description of the extension board software are explained. For a complete description of the definitions is referred to the SBIP's library manual.

Manipulations of images oftenly concerns only a small rectangular window in the image. Therefor, a type "Window" is defined in the SBIP software, this structure contains the following members:

- fromrow, fromcol
  Two integers, defining the row- and column- positions of the window with respect to the origin (upper left corner) of the image.
- width, heighth
  Two integers, defining the width and the height of the window.

This type was also adapted to the software on the extension board.

The SBIP stores images in a large video memory. Since images can have different sizes, each image that resides in the video memory has a structure associated with it. A type "Image" is defined in the SBIP software. This is a structure that contains the following image characteristics:

- im_start
  a pointer to the upper left corner of an image
- im_pitch
  an integer value that specifies the offset between two adjacent column pixels.
- im_window
  a Window that holds the size of the image
- im_mask
  a character that specifies which of the 8 bit planes are used by the image.
- im_name
  a string that can hold a name for an image
- im_number
  a character that can hold a rank number of the image

On the SBIP the first four members are to be specified, the two last can be used optionally. The Software on the DSP extension board uses the same structure for images, however, there are some differences.

The im_mask member remains unused on the extension board, since the DSP can not access separate bit planes like the TMS 34020 can. The parameter remained included in the Image structure for software compatibility purposes.

G.1.3 Memory configuration

As paragraph F.3.1 explained the memory map of the extension board, this paragraph describes how that memory is used by the software. The memory on the extension board consists of external SRAM (1M x 8 bits) and internal memory of the DSP (256 kb). The external SRAM is reserved to store images, the internal memory to store programs, variables (in block 0) and the template (in block 1). For the description of the configuration of the memory, the various declarations from the architecture file are shown in the table below. This table shows the various memory segments:

MEMORY BLOCK 1
seg_rth 0.5 k x 48
code segment for system initialisation (runtime environment)
seg_init 0.5 k x 32
data segment for system and variables initialisation data
seg_stak 1 k x 32
segment for runtime stack, used for passing parameters, and storing return addresses.
seg_pmco 12 k x 48
segment for storage of program code
seg_pmda -
segment for storage of user data stored in program memory space
seg_dmda 12 k x 32
segment for storage of user data stored in data memory space
seg_heap k x 32
segment for heap
seg_com 0.25 k x 32
segment for communication buffer.

MEMORY BLOCK 1
seg_tpl 64k x 16
segment for storing the template

EXTERNAL SRAM
seg_img1 1M x 8
segment for storage of image

Data and program memory spaces

The DSP has a data and a program memory space, in both spaces variables can be stored. The C compiler for the DSP has special extensions to specify whether variables are to be stored in data or program memory. On the extension board, however, all variables are stored in the data memory space, the program memory space is only used for storage of the program and the template. The template has to be stored in program memory to be able to perform single cycle multiply accumulates. The external SRAM is configured as data memory.

G.2 EXTENSION BOARD SOFTWARE

As explained in the previous section, the extension board contains a library with functions that can be called by the TMS 34020. The three following paragraphs explain the routines that run on the DSP: the DSP initialization, the image transfer and the correlator software.

G.2.1 DSP Initialisation

After booting the DSP, some initialization has to take place: the run-time environment has to be set up and some variables have to be set to default values.

The C run-time environment controls dynamic memory allocation, system initialisation, stack usage etc. The initialisation of the C run-time environment is derived from the system-architecture file, a header file with interrupt tables. The required routines for initialisation and memory allocation are linked with the user objects to an executable during the link process.
After a reset, the processors jumps to the initialization procedure of the run-time environment. After the initialization of the run-time environment is completed, function `main()` is called.

After initialisation of the run-time environment, some DSP registers and variables have to be initialized. This is done by the `dsp_init()` routine, it is called from function `main()`. This routine sets the default image transfer parameters and some of the DSP's control registers. This routine can also be called from the TMS 34020 and the host via the software interfaces.

G.2.2 Image transfer routines.

As became clear in the section F.5, some software is required to transfer images from the sbip to the extension board. This software consists of a routine `dsp_getimage()` that is called initially when the image is to be transferred, and an interrupt service routine `vdma_in()` that becomes active once per transferred image line. The `dsp_getimage()` function can be called by the application software on the extension board, the SBIP and the host. This function accepts one argument: a pointer to an Image type that specifies the image properties.

To specify how the image is transferred, the routine `dsp_settransferparam()` can be called. This routine specifies whether the image is transferred in interlaced- or non-interlaced - mode, and the size of the image. These parameters are set to Interlaced 512 rows, 512 columns by default during initialisation of the DSP.

The operation of the image transfer is basically as follows:

a. The `dsp_getimage()` routine initializes the various registers that are used for the image transfer, and sets up a DMA transfer for the first image line from the external port to the internal line buffer. Further it prepares the first transfer from the buffer in the internal DSP memory to the image memory already.

b. After a line is transferred from the external port to the buffer in internal DSP memory, the interrupt handler `vdma_in()` becomes active and initiates a transfer from the internal buffer to the image memory.
If not all image lines are transferred, the interrupt server starts the next transfer from the external port to the internal buffer. (note that this transfer is performed simultaneously with the transfer from the buffer to the image memory.)
Further the server prepares the next DMA transfer from the internal buffer to the image memory.

The design of the interrupt service routine is such that the next DMA transfer from the external port to the internal line buffer is set up as quickly as possible. From paragraph F.5.2 it became clear that there is approx. 2.5 us. available for setting up this next DMA transfer. This requires, that no other interrupts may have priority over the DMA interrupt during the horizontal sync pulse. In that case the DSP has some 75 instructions to set up the next transfer.

By designing the interrupt handler such that the DMA transfer for the following image line is prepared after the transfer for the current line is started, the response time for a starting the next DMA transfer is very fast. After the server becomes active, it just loads the registers of the controller, and restarts the DMA controller. Further the alternate register set allows a single cycle context switch to the work registers for the image transfer. The image transfer routines use alternative registers r8 to r15 and the alternative index register sets 0 to 3. These registers may not be overwritten by other software during the image transfer.

The image transfer procedure that was described here shows just a simple overview, in practice, also frame - synchronisation and de-interlacing have to be handled by the image transfer software.

FRAME SYNCHRONISATION

Frame synchronisation is performed by the `dsp_getimage()` function.
When this routine is invoked, it awaits the first rising edge of the vertical sync pulse. This is done by setting the appropriate command on the output flags for the EPLD. The Vsync will then appear on the DSP's input flag.
When the Vsync pulse is detected, the routine sets all parameters for the image transfer and sets up the first DMA transfer. The parameters for the image transfer are: the image size, and transfer method (interlaced/non-interlaced), pointers to the internal memory buffer, and destination image, the window that is to be extracted from the incoming image and the sub-sample factors in row and column direction. Then it sets the command ID for the get image on the output flags and exits.

DE-INTERLACING

Before the de-interlace procedure is explained, the interlace protocol of the TMS 34020 is explained here first.

In interlaced mode, one image is transferred in two frames. Since the video standards have an odd number of image lines (eg. 625 lines for CCIR), one of the frames ends with a half line, and the other begins with a half line.
Even though the image size on the SBIP has an even number of image lines (eg. 512), the TMS 34020 transfers line 0 and line 511 only partially. So in interlaced

mode, the TMS 34020 transfers actually only 511 whole image lines instead of 512. Since the TMS 34020's documentation is not really clear about the number of pixels that are being transferred in line 0 and line 511, and setting up the DMA transfers require an exact pixel count for each line, the image lines 0 and 511 are ignored on the extension board. The memory locations of these lines are defined, but they will not be written during image transfers.
Thus if an image is transferred from the memory on the SBIP to the extension board, there are the following differences:
- Line 0 and Line 512 on the extension board contain no information.
- The entire image on the extension board is shifted one pixel to the left (see paragraph F.5.3).

De-interlacing is mainly done by the `vdma_in()` routine, the `dsp_getimage()` routine only plays a role in the initial phase of the image transfer. When the `dsp_getimage()` routine is invoked, it sets an "interlaced" - flag, and checks if the first frame to be transferred contains the odd- or even-image lines.
In case the first frame contains the odd image lines, the image memory pointer is advanced to line 1.
In case the first frame contains the even image lines, the image memory pointer is advanced to line 2 (line 0 is not use in interlaced mode),
The blanking signal in line 0 is suppressed in hardware, thus line 0 will not be transferred.
The `vdma_in()` routine then becomes active every time one image line is transferred to the internal buffer. In interlaced mode, this routine advances the pointer for the DMA transfer to the external memory one image lines, to leave an empty space in memory to insert a line during the next frame.
After the first frame is transferred, it sets the pointer back to line 1 or line 2 (depending on the fact whether the first frame contained the even or odd lines respectively).

G.2.3 Correlator software

Even though there was a C source available for the grey value correlator application, some time was spent on enhancing this algorithm. A part of this C source has been rewritten in assembly language, to achieve the desired execution time.
The search algorithm from the existing source was left unchanged.

The correlator application consists of three C-routines that contain the search algorithms, and three assembly language routines that perform calculations of the correlation coefficient.

The main routine `dsp_correlatecoarsefine()` accepts as arguments:
- pointer to an Image type specifying the image
- pointer to an Image type specifying the template
- an integer specifying the number of maximums that the global search should remember.
- an integer specifying the maximum sub-sample factor that the algorithm should use during the coarse search.
- a pointer to an array of structure where the function should store the results of the search

This routine performs the actual search algorithm as described in section 1.2. This function performs multiple exhaustive searches. Therefor it calls the `dsp_exhaustsearch()` function.

The second C-function, `dsp_exhaustsearch()`, accepts the same arguments as the `dsp_correlatecoarsefine()` function. It performs an exhaustive search with the template in the image, and stores the data for the best matches in the results structure.
The `dsp_exhaustsearch()` first calls the third C-routine `dsp_setcorrelparm()` to set up a structure with the required data for the assembler routines, then it calls the assembler routines, for calculation of the correlation coefficients of the template at the various positions within the image.

The `dsp_setcorrelparm()` function sets up the required data for the assembler functions. The function arguments are the same as the ones for the two previous functions except for the last structure: the `dsp_setcorrelparm()` requires a pointer to a structure were it can store the various parameters for the assembler routines. The function calculates the required data for the assembler routines and writes this data into the structure. The function calls an assembler routine `calcvart()` to calculate the sum of the template pixels and the approximated variance of the template pixels. This according to the sub-sample parameters, passed to the function.

There are two more assembler functions involved in the exhaustive search: the `dsp_correlate1()` and the `dsp_correlatenext()` functions. In an exhaustive search, the template is shifted column wise from left to right and row wise from top to bottom of the window. A correlation coefficient is calculated every time the template is shifted one position by one of these correlation routines. Every first correlation coefficient in a row (this is when the template is at the left most position) is calculated using the `dsp_correlate1()` function. The rest of the correlation coefficients is calculated with the `dsp_correlatenext()` function since this saves a considerable amount of time. The `dsp_correlatenext` function makes use of calculations, that were done for the previous correlation coefficient.

Both correlation functions take a pointer to a structure that was set up by `dsp_setcorrelparm()`.
- The `dsp_correlate1()` function requires besides the structure also a pointer to pixel in the image where it has to calculate the correlation coefficient.
- The `dsp_correlatenext()` function requires besides the structure a specification of the position with respect to the previous calculation (one step left, right, up or down), where it has to perform the calculation of the correlation coefficient.

The various routines are callable from the SBIP and host environment.

NUMERICAL ACCURACY OF THE CALCULATIONS.

In the correlator algorithm, many calculations are being performed. This paragraph deals with the numerical properties of all these calculations.

Most calculations are performed using integer arithmetic, for these calculations no inaccuracies will occur as long there is no overflow on variables. Since the summation loops perform maximum 2^{16} MAC's on 8-bit numbers, the result can be represented by a 32 bit (unsigned) integer. Thus in the MAC loops, there will be no inaccuracies.

The following formula shows a the calculation that is to be performed for a 2 dimensional normalized grey value correlation coefficient for a template of R rows x C columns:

$$r(I_{i,j},T) \triangleq \frac{R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c} \cdot T_{r,c} - \left(\sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c}\right) \cdot \left(\sum_{r=1}^{R}\sum_{c=1}^{C} T_{r,c}\right)}{\sqrt{\left[R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c}^{2} - \left(\sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c}\right)^{2}\right] \cdot \left[R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} T_{i+r,j+c}^{2} - \left(\sum_{r=1}^{R}\sum_{c=1}^{C} T_{r,c}\right)^{2}\right]}} \qquad \text{(G.1)}$$

$T_{r,c}$ represent the grey values of the template pixel at row i, column j.
$I_{i+r,j+c}$ represent the grey values of the image pixel from an image detail taken from row i, column j, at an offset r,c within the image detail.

The algorithm calculates in fact $|r(I_{i,j},T)| * r(I_{i,j},T)$ to avoid the calculation of the square root. The actual calculation that is performed by the algorithm has thus the following form:

$$|r(I_{i,j},T)| \cdot r(I_{i,j},T) \triangleq \frac{|C(I_{i,j},T)| \cdot C(I_{i,j},T)}{C(I_{i,j},I_{i,j}) \cdot (C(T,T))} \qquad \text{(G.2)}$$

With :

$$\begin{aligned} c(I_{i,j},T) &= R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c} \cdot T_{r,c} - \sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c} \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} T_{r,c} \\ C(I_{i,j},I_{i,j}) &= R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c}^{2} - \left(\sum_{r=1}^{R}\sum_{c=1}^{C} I_{i+r,j+c}\right)^{2} \\ C(T,T) &= R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} T_{i+r,j+c}^{2} - \left(\sum_{r=1}^{R}\sum_{c=1}^{C} T_{r,c}\right)^{2} \end{aligned} \qquad \text{(G.3)}$$

Actually, there are three identical terms to be calculated:

$$c(A,B) = R \cdot C \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} A_{r,c} \cdot B_{r,c} - \sum_{r=1}^{R}\sum_{c=1}^{C} A_{r,c} \cdot \sum_{r=1}^{R}\sum_{c=1}^{C} B_{r,c} \qquad \text{(G.4)}$$

Or in a one dimensional notation:
In which is:

N the number of pixels in the template (R * C) , N < 64 k pixels -> N is a 16 bits number

$$c(A,B)=N\cdot\sum_{i=1}^{N}A_i\cdot B_i-\sum_{i=1}^{N}A_i\cdot\sum_{j=1}^{N}B_j \qquad (G.5)$$

A_i,B_i 8 - bits pixel grey values.

The summation loop:

$$N\cdot\sum_{i=1}^{N}A_i\cdot B_i \qquad (G.6)$$

will yield a 48 bit result :

- The multiplication of the 8 bit values A B yields a 16 bit result
- the summation loop consists of maximum 2^{16} summations (maximum template size of 64k pixels). Thus the result of a summation loop < 2^{16} * 2^{16} = 2^{32}.
- The value N is a 16 bits value, the total result is thus represented by a 16+32 = 48 bits value.

The summation loops:

$$\sum_{i=1}^{N}A_i\cdot\sum_{j=1}^{N}B_j \qquad (G.7)$$

will also yield a 48 bit result :

- The summation terms, the template and image pixels are 8 bit's values
- One summation loop consists of 2^{16} (64k) summations, thus the result of a summation loop < 2^{16} * 2^{8} = 2^{24}.

Thus the calculation of c_{ab} will yield a 48 bit result.

Now care is to be taken, since these two 48 results are being subtracted. The subtraction can not be performed in 32 bits floating point format: If the two variables, being subtracted from each other are large almost equal values, the result of the subtraction might be a very small result. With a 32 bit floating point format, the relative error might become very large. Therefor the 48 bit subtractions are performed with integer arithmetic. The results of these subtractions are casted to floats, then the normalized correlation is calculated using floating point math.
2The normalized correlation coefficient is normalized to 2^{10}. Since the algorithm calculates the square value of the correlation coefficient the result of the calculation is normalized to 2^{20}. This value is casted to an integer.

G.3 HOST/SBIP SOFTWARE DESCRIPTION

The previous section explained the software that is running on the DSP. Most of the routines that were explained (eg. `dsp_getimage()`, `dsp_correlatecoarsefine()`, `dsp_exhaustsearch()`) can be called from the SBIP and the host via the software interface. There is also some specific software for the host computer and the SBIP. This section will explains these routines: paragraph G.3.1 describes how the DSP is booted, paragraph G.3.2 how the host can perform 32 memory accesses in the SBIP's memory and paragraph G.3.3 explains how templates are downloaded from the SBIP to the extension board.

G.3.1 Booting the DSP

After powering up the SBIP and the DSP extension board, the software for the SBIP is to be downloaded from the host first, and then the software for the DSP is to be downloaded. The programming tools of the DSP produce a so called "loader file", that contains the code for the DSP and the initialisation data for variables. This loader file contains besides the actual user program, a small loader program that takes care that the various code blocks are placed at the appropriate memory locations, and that variables are being initialized.

Booting of the DSP is done by the host computer, because the loader file is written on a disk. Before the host starts the actual boot procedure, he first performs the following procedure:

- The module ID and hardware revision ID of the extension board are read and checked.
- A reset pulse is generated to bring the extension board into a defined state

After this is done, the loader file is read from disk and written to the DSP. The setup of the DMA transfer of the DSP is not necessary, since this setup is part of the reset procedure in the DSP. During reset, the DSP sets up a DMA transfer of 256 words internally. The first 256 words of the loader file contain a the loader program for the DSP. After these 256 words are downloaded, the DSP runs this loader program. The loader program takes further care that the entire user program is placed at the appropriate memory locations, and that variables are initialized. The data is written to the external port FIFO buffer (via host DMA) as 16 byte words. The DMA controller and the loader program will take care that data is being packed to either 48 bits for program code, or 32 bits for initialisation of variables.

As described in the paragraph F.3.2, it is also possible to configure the extension board for EPROM booting, by installing a jumper on the appropriate location. The downloading procedure is similar, the DSP detects the presence of the configuration jumper, and performs the DMA transfer from the boot memory device instead of from the external port buffer. The contents for the boot EPROM are also produced by the boot loader utility by passing the appropriate command line option to the utility. For the command syntax is referred to the "ADSP 21000 family - Assembler tools & Simulator manual".

G.3.2 Single 32 bit SBIP memory access

Certain operations require 32 bit memory accesses in the TMS 34020 memory space to control the extension board hardware properly. Via the SBIP's host interface, however, only 16 bits memory accesses can be performed. 32 bit's accesses by the SBIP's host are possible, but such an access is executed as two consecutive 16 bit accesses, yielding the 32 bit result. In case of a 32 bits device in the TMS 34020 memory space, first the lower 16 bits are accessed, then the 16 upper bit's are accessed. Thus the 32 bit's device is accessed twice.

There are two cases in which two 16 bit accesses to/from a 32 bits device are disastrous: during the setting of the host address page and during host DMA.

The host address page is latched in a 24 bit hardware register, it requires a 32 bit access (8 bits are ignored). If this address page register is accessed twice (each access 16 bits) only the last access will be stored in the register. During the first write, the 16 lower data lines contain valid info which is latched in the 16 lower bit's of the latch, the invalid data on the 16 upper data lines is also latched in the register. During the second write, the lower 16 address lines contain invalid data, and the 16 upper address lines contain valid data. During this write action, the upper 16 bits of the register are written with valid data. the lower 8 bits that were written in the first write cycle, are overwritten with invalid data.

During 32 bits DMA transfers to the extension board, the DMA controller considers every access to be a 32 bits access. So if two consecutive 16 bit accesses were done to obtain a 32 access, the DMA controller interprets this as being two 32 bit DMA transfers.

To omit this problem, two user functions `setsbip2memoryltms()` and `getsbip2memoryltms()` were written for the SBIP to perform a 32 bit write - and a 32 bit read - action in the SBIP memory respectively. The 32 bits parameters are passed to/from the SBIP as two 16 bit values via the software interface. The functions on the SBIP perform the actual (32 bit) memory access.

This function is only required for 32 bits memory accesses by the host. 32 bit accesses by an application program on the SBIP will be done in one cycle anyway and thus do not require a specific treatment.

G.3.3 Downloading templates

The grey value correlator application requires that a template can be downloaded from the TMS or host computer. A routine was written to download templates to the DSP extension board. It downloads a template that is stored in the SBIP memory, to the extension board using a host IOP DMA transfer. The routine was written for the SBIP because it uses 32 bits memory access for the DMA transfer. If the template is to be downloaded from disk, the `loadimage()` function from the SBIP can be used to transfer the template from disk to SBIP memory, and the `dsp_gettemplate()` function to transfer the template to the extension board. (Eventually a routine can be written to download the template straight from the host to the extension board.)

The procedure is as follows:

- The TMS 34020 issues a command to the DSP with the information of the template (size, start address), which in turn sets up a DMA transfer from external port to the internal DSP memory.
- Then the TMS 34020 resets the host DMA error bit in the EPLD (see paragraph F.4.2) and performs the data transfer. For this transfer the TMS 34020 packs two pixels in one 32 bits value and writes them to the DMA channel. The two pixels are packed in one 32 bits value because the DMA controller of the DSP does not support transfers to the 16 bits internal memory space in the DSP. The transfer is therefor to be performed to the 32 bits shadow memory space in the DSP.
- After the transfer of the template the error status from the DSP is read to check for errors.

The `dsp_gettemplate()` routine is available on the host and SBIP only. It is not callable by an application that is running on the extension board, because it involves the template to be downloaded by the TMS 34020. As the DSP is a slave of the TMS 34020, it can not order the TMS 34020 to download the template. The `dsp_gettemplate()` function being called on the host passes the function parameters via the communication buffer to the SBIP, where the actual routine is invoked.