Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Department of Information Technology
and Electrical Engineering

Integrated Systems Laboratory

Stereopsis: Depth mapping using stereo vision

VLSI Chip Design Semester Project

Winter Term 2002/03

Authors: Michael Kuhn, Stephan Moser, Oliver Isler
Advisors: Frank Gürkaynak and Andreas Burg

Zurich, February 8, 2003

Preface

A few years ago Stephan and his brother tinkered with an autonomous vehicle made from an old radio controlled racing car. A fundamental limitation in their effort for vehicle navigation was the lack of a distance measuring device. An ultrasonic device proved unreliable and laser based equipment was too expensive. Stereo vision appeared as a possible solution for a next generation vehicle so they did some experiments with pictures from a digicam and a couple of Java functions.

During summer term 2002 this idea was picked up again and the algorithms were improved. During the VLSI I course at the Swiss Federal Institute of Technology in Zurich the idea of building an ASIC device for this task emerged and led to this VLSI design project.

Special thanks go to the following persons:

- Dr. Huber Kaeslin and Dr. Norbert Felber for two intense lectures on hardware design and for giving us the chance to perform a comprehensive VLSI project.
- Frank Guerkaynak and Andreas Burg for their patient and competent assistance and for having a good time together.
- Dr. Tomas Svoboda for the review of our algorithm and his advice concerning the stereo vision task.

Contents

Chapter 1

Introduction

Depth mapping is a technique in computer vision that allows the reconstruction of spatial depth from stereo pictures. It is based on the observation that an object that is pictured using two parallel cameras with a horizontal offset yields a *displacement* in the two images that is dependent on its distance from the camera setup. A camera image captures a certain area of the surrounding room at once unlike e.g. a laser beam which only points at one spot at a time. This allows one to get an immediate impression of the vicinity without any scanning technique or any moving measuring devices. Depth calculation is achieved in three steps: *Feature detection* extracts characteristic areas from the pictures. *Displacement calculation* matches those features within the two images and determines their perspective displacement, resulting in a so-called *displacement map*. *Depth calculation* finally is a simple geometric transform that yields real-world distances from the displacement map. Advanced stereo algorithms take long calculation times and operate on the entire images, exploiting e.g object recognition techniques and geometric object modeling.

The goal of this project was a real-time implementation of a stereo system in an ASIC. This chip should provide a means for autonomous vehicles to immediately recognize approaching obstacles, enhance navigation and plan possible pathways. Since the technology set tight limits on on-chip memory (see subsection 3.1.1 for more details), the algorithm had to work on partial images - the final version works on three image lines at a time. Two different matching techniques with different characteristics that complement each other to a certain degree were combined. The desig is strongly dataflow oriented. However, buffers are needed at intermediate steps. The design delivers raw binary image data by a simple protocol that needs an external interface to cameras and display modules or any other form of processing unit.

Chapter 2

The Stereo Matching Problem

This section introduces some important problems of depth calculation by stereo vision. Further, important terms related to stereo vision are introduced.

2.1 General Task

The goal of stereo vision is to calculate a *depth map* from two or more given input images which show the same scene from different points of view. Most algorithms work on two input images taken with a certain horizontal displacement (*binocular setup*). The main problem is independent from the camera setup, however. The task is to identify corresponding objects or image regions in the input images. In the binocular setup with horizontally aligned cameras, for example, an object in the right image appears in the left image with a certain horizontal displacement.

In literature, the displacement between corresponding objects in different images is usually referred to as *disparity*, while the actual distance of the object to the camera setup is usually called *depth* [6]. It holds that the larger the disparity, the smaller the distance of the observed object from the camera setup. *The relation between disparity and depth is approximately inversely proportional* [6]. An image which assigns a certain color for each disparity or depth value to every pixel is called a disparity map or depth map, respectively.

2.2 Principal Observations

There are two fundamental assumptions underlying the stereo matching task [6]:

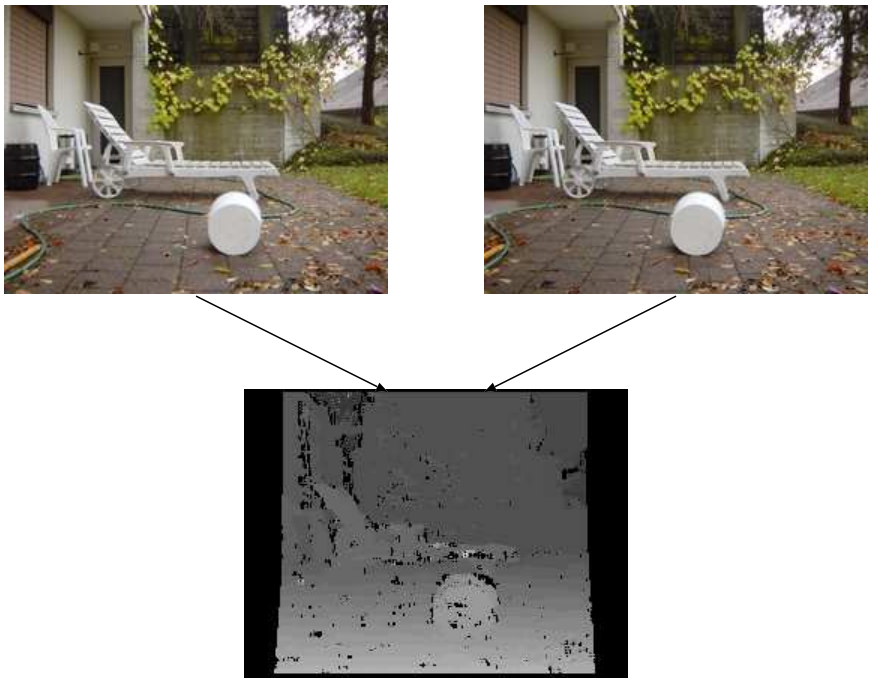


Figure 2.1: Two stereo pictures and a resulting displacement map

1. uniqueness
2. continuity

The *uniqueness assumption* states that only one single disparity can be assigned to every point of an input scene. Problems with the uniqueness assumption emerge as soon as reflections and transparent areas are present. If, for example, a tree is seen through a window, there is an ambiguity because a single image point contains the depth of the window as well as the depth of the tree [7]. The *continuity assumption* states that the disparity varies smoothly in almost every region of the scene. In [1] a third, so called *similarity assumption*, is introduced, which states that correponding points have similar local features.

Unfortunately, there are some problematic observations to the stereo task as well. As stated in [6], the stereo matching problem is an ill-posed problem because of two reasons:

1. data uncertainty
2. structural ambiguity

Data uncertainty particularly exists in homogeneous (i.e. poorly textured) areas. It is obviously difficult or even impossible to match a part of a white wall in one image to exactly the same part of the wall on a different image. The term *structural ambiguity* expresses the fact that it is often difficult to assign the correct disparity to periodically structured areas. It is impossible to find the corresponding part of a grid in two pictures by only considering a small part of the grid, because the same part is repeated periodically. Structural ambiguities can partly be solved by using a larger number of cameras.

Another problem in stereo vision are *occluded areas*. An occluded area is an area which cannot be seen by all cameras. In the binocular setup we can distinguish between binocularly visible (or non-occluded) areas and monocularly visible (or partly occluded) areas [6]. Partly occluded areas are seen by one camera only. Further, there exist fully occluded areas, however, they are not of interest, because they do not appear on any of the input images.

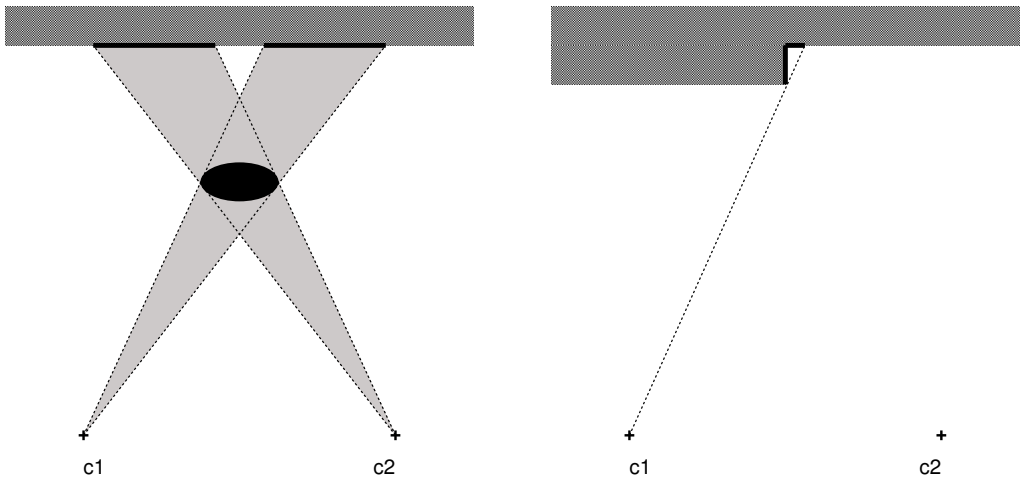


Figure 2.2: Partly occluded areas, as shown in [6]

Figure 2.2 shows two different reasons for *partly occluded areas*. Further, there exist so called *mutually occluded regions*, where between two foreground objects completely different background objects appear (see figure 2.3). It is impossible to assign a disparity value to a monocularly visible area since no matching counterpart can be found in the other picture. A stereo algorithm should be able to detect occluded areas. Of course, the number of occluded areas can be reduced by increasing the number of cameras observing the scene [7].

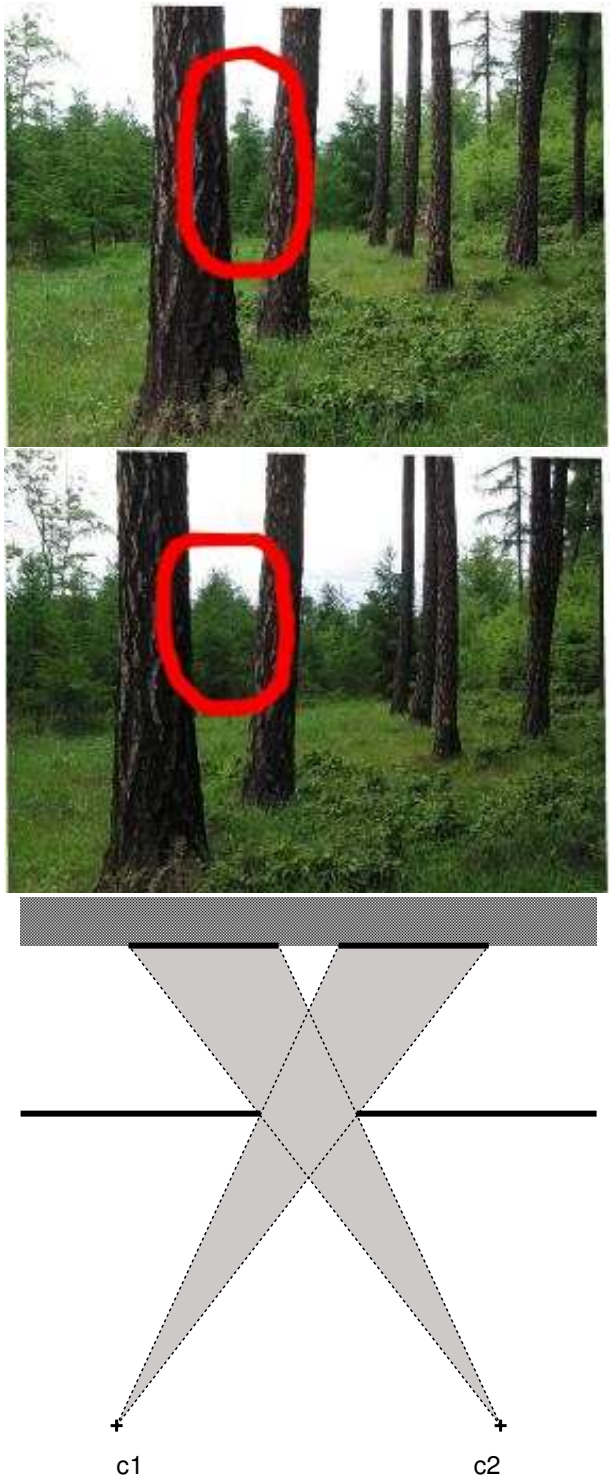


Figure 2.3: Mutually occluded areas, as shown in [6] and [8]

2.3 Algorithmic Approaches

2.3.1 Matching Techniques

There are several criteria to classify stereo algorithms. A first distinction can be made based on the elements being matched. The algorithms are often classified into *edge based* and *area based* approaches. Edge based algorithms work on the features (or edges) of objects only, while area based algorithms try to match entire image regions.

A lot of image information is neglected when only working on extracted features. Thus, feature based algorithms generally yield sparse but quite exact disparity maps. Area based approaches, on the other hand, produce dense disparity maps, since they work on the entire image. However, in an area-based approach there can be parts of more than one object within one search patch which can have different depths. Consequently, not every area can be assigned a clearly defined disparity or depth. Using smaller search patches can reduce this problem. Larger patches yield better results when only containing parts of one object, however, because of the better signal-to-noise ratio[1] see (i.e. because there are more features that can be matched, and the noise therefore becomes neglectable). Thus, the area size is a trade-off between high resolution and good matching capabilities. Several approaches use adaptive area (or window) sizes for that reason.

Another criteria to classify algorithms is the distinction of global and local methods. While global methods work on previously defined world models and check global consistency based on the results of local methods, local methods only take into account small elements of the input images at a time [6]. When using cameras aligned in a parallel way, or accordingly corrected images, local methods can even work on one *scan-line* (one horizontal row of pixels) only.

Another way to look at the algorithms is to compare the *correspondence function* that is used to determine the degree of similarity of two image regions. There are different methods that work on pure image data, like *sum of squared differences*, *sum of absolute differences* or *normalized cross correlation*. Other methods first transform the image data before applying the actual correspondence function. Often, so called *non-parametric transforms* are applied. They are based on the ordering among data values rather than the data values themselves, and so get insensitive to problems that may appear because of very bright or dark images or because of large differences in intensity of only one single pixel. The most popular non-parametric transforms in the area of stereo vision are the *rank transform* and the *Census*

[1] Please find a definition of the "signal-to-noise ratio" in the glossary on page 45

transform [2] [5]. Another transform is used by the *Marr-Poggio-Grimson algorithm* which is often cited in stereo literature. It uses a "Laplacian of a Gaussian" (LOG) filter, which emphasizes features before they are matched.

2.3.2 Treating Occlusion

As stated before, *occlusion* is an inherent problem of stereo vision. Therefore, a sophisticated algorithm needs to deal with it. There exist different approaches to tackle the problem. The most obvious one is to use more than two cameras. However, this approach significantly increases the amount of data to be processed and therefore needs a lot of computing resources.

Other methods take advantage of the observation that matches in occluded areas are mostly matches of poor quality and the resulting image area is noisy. According to the continuity assumption, these areas can be detected as incorrectly matched [7].

Another approach used in stereo vision is to apply a so called *left-right consistency check*. Here, the disparity is first calculated using a patch of the right image as reference and searching the corresponding patch in the left image (RL case). The same is done in the opposite direction (LR case). Only if both search directions yield the same disparity (within a certain tolerance), the result is accepted, otherwise it is discarded [4].

2.3.3 Post-filtering the Results

Finally, the disparity map produced by a stereo algorithm is often post-filtered to reduce noise and the number of erroneous results. Post-filtering is often based on inter- and/or intra-scan-line consistency. A simple but efficient way to reduce noise is the *median filter* proposed by [4]. It eliminates a pixel from the disparity map if less than five pixels in its 3x3 neighborhood have assigned a disparity, and otherwise takes the median of the sorted disparities of the neighboring pixels.

2.4 Parameters of the Input Pictures

The number of input images significantly affects the task of a stereo algorithm. The higher the number of input images, the more efficiently occluded areas can be avoided and the better is the signal-to-noise ratio when matching areas, since the features remain the same on all images that the noise is randomly spread over them. Unfortunately, the amount of computing resources needed to deal with the input data increases as well.

During the analysis of the camera setup it was observed that even in a binocular setup many free parameters exist. First, one can align the optical axis of the cameras in a parallel or a non-parallel way. When choosing a non-parallel setup, corresponding objects are no longer on the same horizontal axis (see figure 2.4). When using such a setup, the images are often rectified

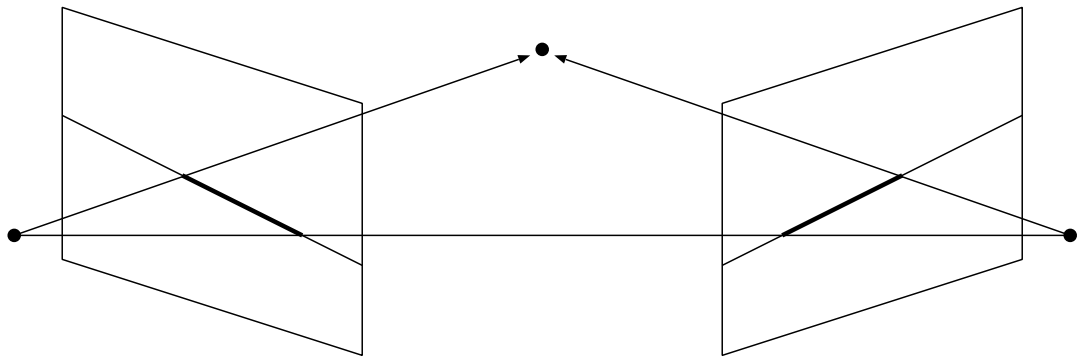


Figure 2.4: Objects are no longer on the same horizontal axis on an non-parallel setup

before the actual stereo algorithm is being applied [6].

A slightly different alignment in vertical direction can be a substantial problem for certain algorithms. This particularly holds for algorithms that work on one scan-line only. A displacement of only two pixels in vertical direction may already lead to totally unacceptable results (see figure 2.5) [5]. Other setup parameters like the horizontal displacement of the cameras have rather little effect on the stereo problem.

Not only the parameters of the camera setup, but also the output of a single camera may have a non-negligible effect on the stereo task. There is an effect called *radial distortion*, caused by lenses that have slightly different magnifications in the middle and at the border. The effect on a picture as well on the resulting output of a stereo algorithm is illustrated in figure 2.6.

It also matters whether color or gray-scale cameras are choosen. Obviously, colored pictures contain more information which can be used to find correct matches. According to [4] the improvement of the signal-to-noise ratio is somewhere between 20 and 25 percent when using color information instead of gray-scale values.

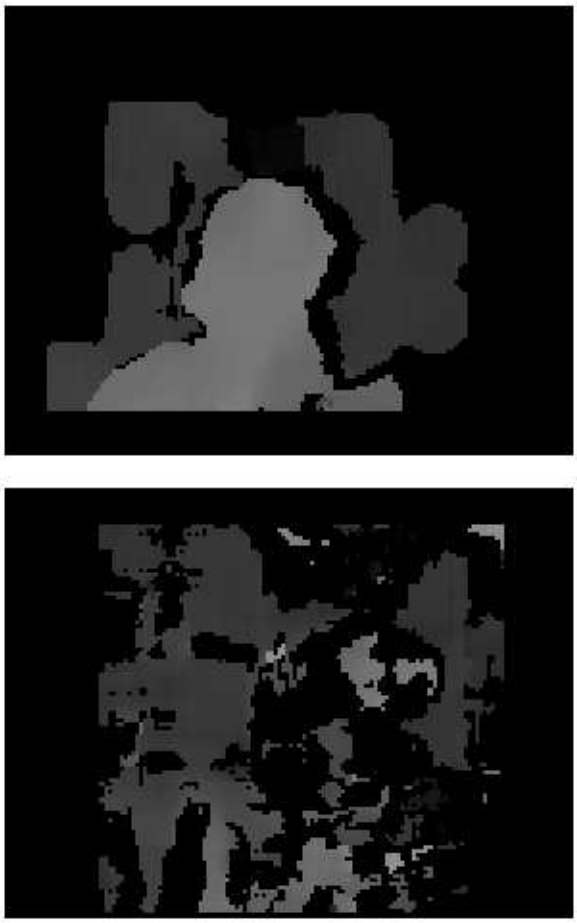


Figure 2.5: If the cameras are not well aligned in vertical direction, it may lead to unacceptable stereo results [5].

Figure 2.6: The left pictures demostrate stereo results with radially distorted input images. On the right, the effect was corrected prior to the calculation [9].

Chapter 3

Implemented Algorithm

This chapter discusses the algorithm implementation and the constraints that influenced its design.

3.1 Algorithm Evaluation

3.1.1 Constraints

The integration of a stereo vision machine in an ASIC sets strong restrictions on the algorithm selection. This section should give a brief overview of the constraints given by the *UMC 0.25 μm fabrication technology* and the project goals.

The total chip size available for this student project was restricted to approximately 2.435 mm x 2.4355 mm with a core area (including global and power routing) of roughly 1.887 mm x 1.887 mm = 3.56 mm^2. This area corresponds to approximately 120'000 gates. Further, a limit of 84 pads (including power pads) was given.

Storage capacity is usually a though constraint in integrated circuits. Unfortunately the UMC process does not provide SRAM structures, but memory macro cells with rather low density. A 32 x 256 bit memory cell, for example, occupies about 0.31 mm^2, or 10 percent, of the available core area. Thus, using a memory of more than about 4 KB in size turns out to be unreasonable.

What's more, a memory macro cell provided by UMC can store a maximum of 256 words. Therefore, a width of 256 pixels was practical for the input images. For a height of 192 pixels, this yields a total amount of approximately 50'000 pixels per input image. The target for the project was a *frame rate* of at least 25 images per second (which yields the appearance of

a fluent movement to the human eye). Thus, the minimum *pixel frequency* is at least 1'250'000 pixels per second. For a clock frequency of 100 MHz (which is a reasonable frequency for the fabrication technology used), this results in about 80 clock cycles per pixel.

Considering the memory restrictions above , it becomes obvious that only parts of an image can be stored. Therefore, a *data flow oriented algorithm* was called for.

Since the in- and output devices are not specified, the IO interface was held abstract. Thus, no special timing constraints given by an IO protocol of a particular device need to be satisfied.

Further, there were no restrictions concerning the power consumption.

3.1.2 Conclusions

As described in the introduction to stereo vision, an inherent and important part of the problem is the matching of image regions. Therefore, special attention should be paid to the image regions being matched and to the correspondence functions themselves.

According to the stated memory limits above, it is not possible to store whole images on the chip. Since usual cameras deliver images row by row, a *row-based image processing* was aimed at.

Experiments showed that good results could be reached using a combination of two different correspondence functions. These functions are a least square (LSQ) comparison of the pixel intensities and a *Hamming distance*[1] calculation on Census transformed image regions (refer to sections 3.2.1 and 3.2.2 for details). While the LSQ function can work on image regions of any shape, the Census function needs at least three rows and three columns of input at a time, since the Census transform is defined on $3 \cdot 3$ pixel blocks.

The amount of data storage is also affected by the decision whether to use color or gray scale images. While one image row (256 pixels) requires 2 kBit of memory in the case of gray scale (8 bit) images, three times as much is needed for true color (24 bit) images.

The decision tree in figure 3.1 shows that either a color solution with one image line processed at a time or a gray scale solution with three image lines processed at time is possible to implement. While the three line gray scale solution offers enough data to perform a Census function, this is not the case fot the single line color solution. The quality improvement resulting from the Census Function proved to be greater than the one resulting from color images.

[1] For a definition of the "Hamming distance" see glossary on page 43

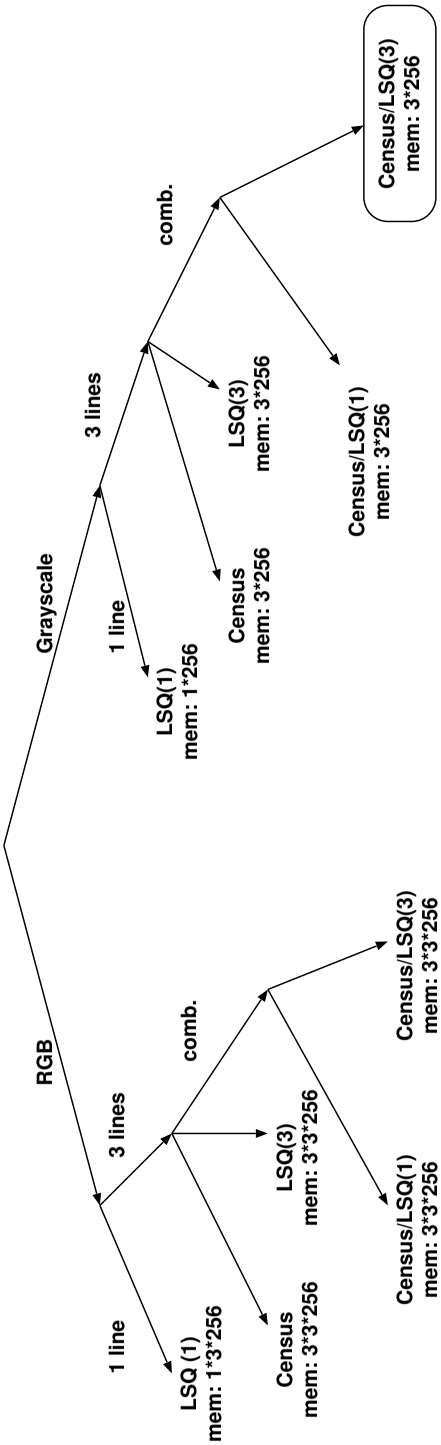


Figure 3.1: Correspondence function selection tree: A one-line Census transform requires three on-chip image lines. All memory figures are doubled due to the presence of a left and a right image channel. The figures grow by a factor of four if occlusion detection by RL-LR (see 2.3.2) is applied as well. The memory values are in "bytes".

The quality improvement that could be achieved by the use of color images did not seem to justify the omission of the Census function (which requires three image lines) and the larger storage capacities needed for three color channels. Therefore, it was decided to implement a gray scale solution which could exploit the advantages of both the LSQ and Census approaches.

3.2 The Algorithm in Detail

The implemented algorithm is based on *block searching*. Blocks of $3 \cdot 10$ pixels are extracted from every (x,y)-position in the right image and are searched for in the left image with a horizontal displacement of 0 to 24 pixels[2]. The two pixel blocks are called the *reference block* (static, right image) and the *scan block* (displaced, left image). See figure 3.2 for a schematic and find in figure 3.3 a real-world example of a $3 \cdot 10$ reference block an the corresponding search area in two images with a resolution of $256 \cdot 192$.

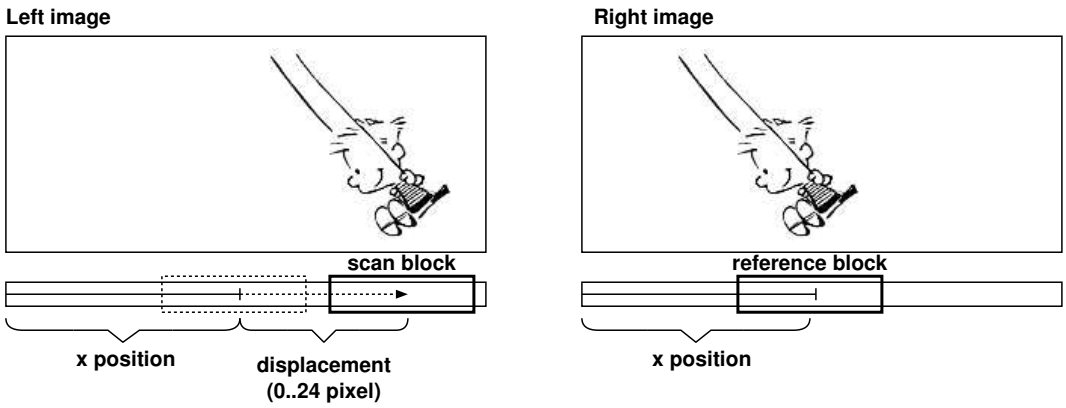


Figure 3.2: Block matching: The center of the block represents a virtual pixel position that all further calculations relate to. The left-to-right case (LR case) for occlusion detection works symmetrically.

These blocks are compared by two different correspondence functions called the "LSQ" and "Census" functions. These functions yield a *matching quality* for each displacement. The displacement value with the highest matching quality is recognized as the disparity search result for the present search operation.

[2]This value was found to be reasonable with respect to the image width and a common camera setup with a 10 centimeter camera displacement.

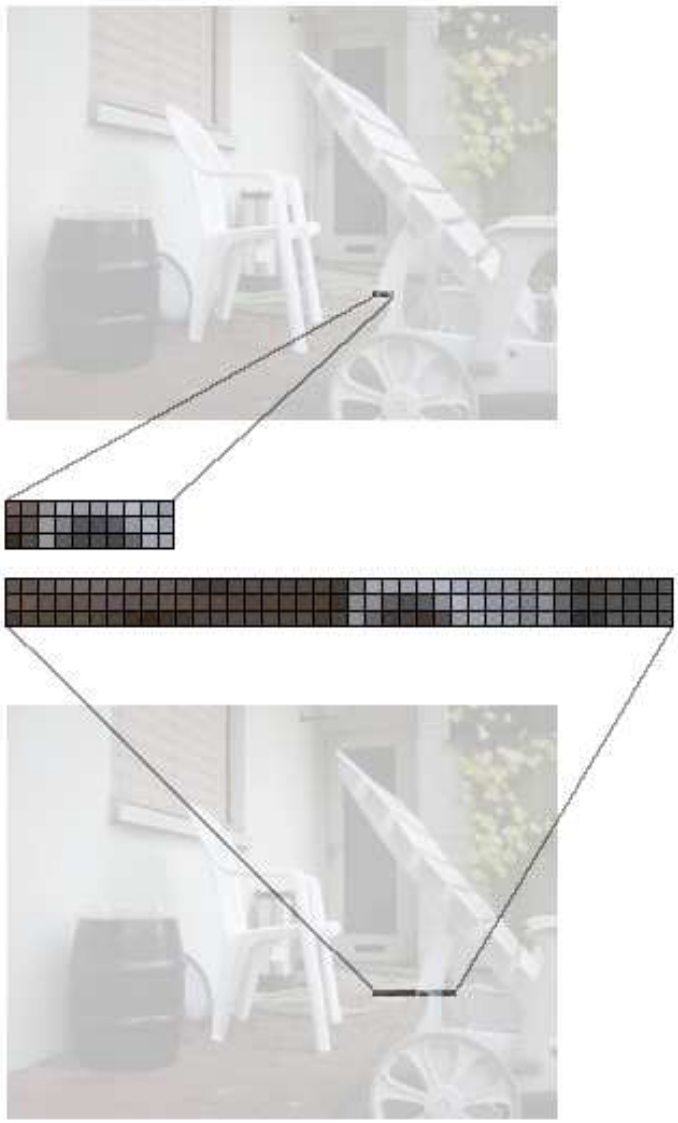


Figure 3.3: The top image illustrates the size of a reference block in relation to the input image. The bottom image shows the complete region of 25 displacement positions where the scan block is shifted over in search of the best match.

Both the LSQ and Census correspondence function units get the same input blocks, but differ in their output since they evaluate a match by different criteria; they complement each other to a certain degree with respect to their matching characteristics (see 3.2.1 and 3.2.2). For the LR case, two more function units exist. See figure 3.4 for a comparison of the different results according to the two correspondence functions.

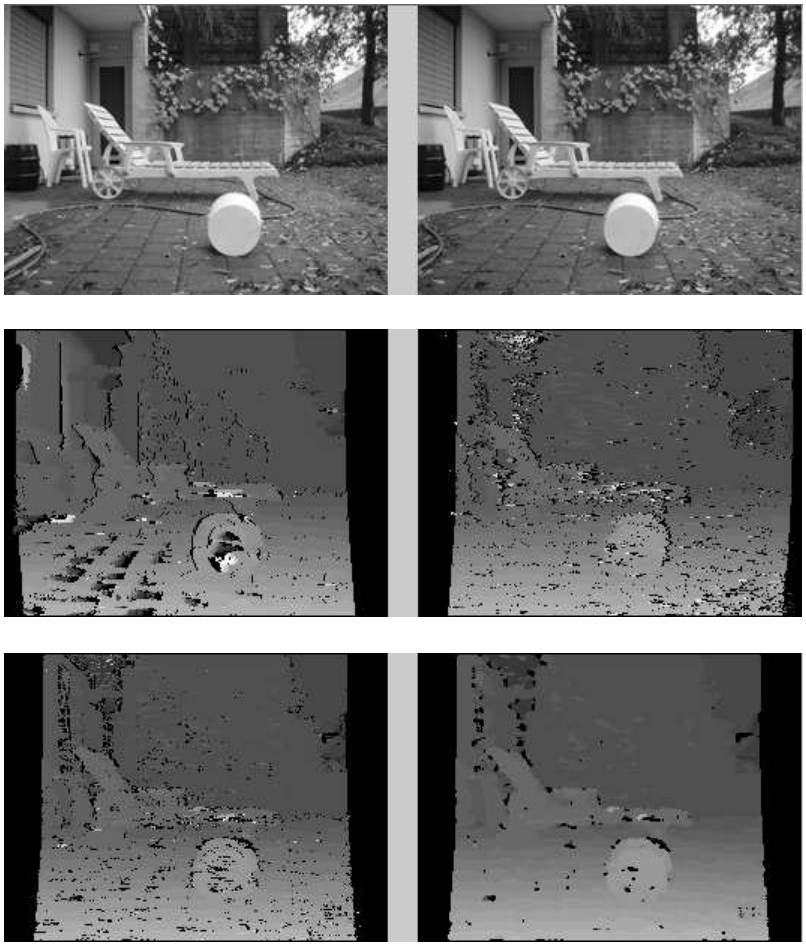


Figure 3.4: The top images show stereo input images. In the middle to the left is a LR LSQ result image, next to a RL Census result image. The bottom line shows the output of the merger unit and and a postfiltered final output image (median filter).

The output image has a different perspective than the right and left input images: its virtual viewpoint is exactly in the middle of the right and left camera viewpoint. So the disparity results are perspectively mapped to the

middle of the scan and reference block positions (see figure 3.5).

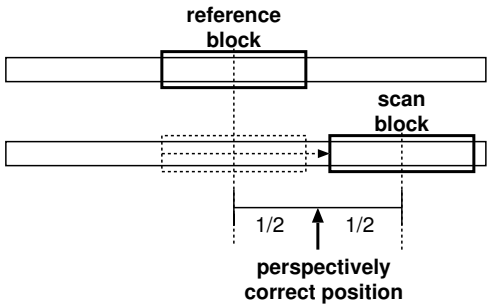


Figure 3.5: Stereo perspective: A virtual viewpoint in the middle of the left and right image sources is created.

The displacement values for all positions are collected along with their matching quality. If there is more than one match for a certain position, the match with the highest matching quality is chosen as the result. For occlusion detection, the algorithm is applied twice: Once with the left image as the source of the reference blocks and the right image for the scan blocks (LR case) and a second time vice-versa (RL case). This yields four sources of matching results: The LSQ and Census correspondence functions applied both in the RL case and LR case. Those four results are combined by a configurable *merging function* into one raw disparity map, thereby performing the actual occlusion detection. Moreover, the results of the two correspondence function types are combined to benefit from the strengths of both. To eliminate poor matches and image noise, the raw output image is filtered by an *output filter* that implements a heuristic and a *median filter* function.

3.2.1 LSQ correspondence function

The LSQ correspondence function defines the similarity of two blocks of pixels in a straight-forward least-square fashion: Two blocks of 3·10 pixels are compared by building the pixel-wise differences in intensity. These differences are squared and summed up. The lower the sum, the more similar the two blocks are.

The LSQ function works best on feature-rich image regions. It is specially well suited for matching edges and contures. For homogeneous areas it does not always yield good results.

3.2.2 Census correspondence function

The Census correspondence function is based on a non-parametric transform. That is, the criterion for similarity rests upon relative intensity values. A Census transform is defined for a single pixel that is located in the middle of a $3 \cdot 3$ block of pixels: Its intensity value is compared to all eight surrounding pixel intensities. A "1" is set if it is higher, otherwise the result is a "0". This yields an eight bit wide binary string. This string relates the pixel to its close surrounding and characterizes it (see figure 3.6).

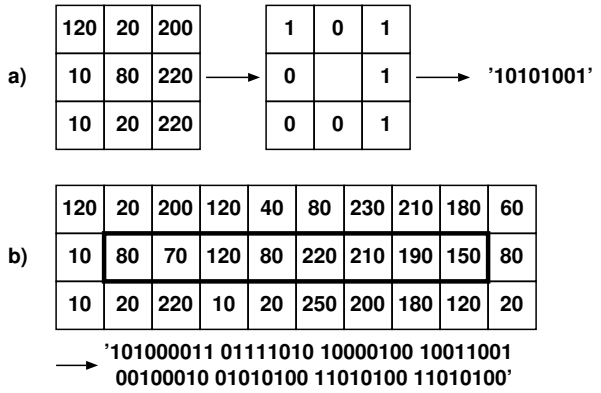


Figure 3.6: a) Census transform for a single pixel. b) Transform applied to a $3 \cdot 10$ pixel block.

To apply the Census correspondence function on two $3 \cdot 10$ pixel blocks, both blocks are first Census transformed: The Census transform is separately applied to the eight pixels in the center of the block, so eight bit strings are generated. Combined, a 64 bit string characterizes each pixel block. The Hamming distance of these two bit strings defines the similarity of two blocks, that is, the number of equal bits.

It is important to see that the Census function works on relative intensity values as opposed to the LSQ function that takes absolute values. This makes it insensitive to the overall image brightness. The advantage is that the Census function manages to match image regions that are poor in features. A drawback is that the influence of edges and strong contures is attenuated.

Chapter 4

Architecture

4.1 Top Level Architecture

This section introduces the IO interface of the chip and gives an idea of the general data-flow. An overview of the interface signals is shown in table 4.1.

There are four types of signals, namely data, protocol, configuration and testability signals.

The data interface consists of the input for the left and the right image and the output of the disparity map. Since the whole architecture works data-flow oriented, an input image is processed pixel by pixel. An input pixel is encoded as an eight bit gray scale value and the output is a five bit value representing one pixel of the disparity map.

The *control signals* FrameSync, LineSync and PixelSync are used to determine the beginning of a new frame, a new line and a new pixel, respectively. Because of the fixed image width of 256 pixels, the LineSync signal is redundant and therefore exists only at the output. It simplifies the interpretation of the disparity map. The FrameSync and PixelSync signals are passed along with the actual data through the whole architecture. This protocol makes the architecture independent of the overall latency.

The configuration of the Merger module and the Output filter module is done by several configuration signals. A change of these signals immediately affects the configuration of the modules.

Finally, there is a group of signals which ensures the testability of the design. There is a *built-in self test* (BIST) for all the RAMs on the chip which can be configured using the TestMode signal (i.e. a test mode, an init mode or a bypass option can be chosen). Finally, there is the interface to the scan path, which consists of the input and the output to the path as well as an enable signal.

Top				
	In		Out	
Function	Port	Width	Port	Width
Data	LeftPixelIn	8 bit	PixelOut	5 bit
	RightPixelIn	8 bit		
Protocol	PixelSync	1 bit	PixelSync	1 bit
			LineSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit
Configuration	SingleFunction	1 bit		
	MergePrio	1 bit		
	FunctionPrio	1 bit		
	OcclDetectLSQ	1 bit		
	OcclDetectCensus	1 bit		
	OcclDetectMerged	1 bit		
	TolLSQ	2 bit		
	TolCensus	2 bit		
	TolMerged	2 bit		
	FilterMode	2 bit		
Test	TestMode	2 bit	RamError	6 bit
	ScanEn	1 bit		
	ScanIn	1 bit	ScanOut	1 bit

Table 4.1: IO Interface of the chip

4.2 Module Level Architecture

The module level architecture principally reflects the data-flow of the algorithm as it is described in chapter 3. There are six main modules: the Input Buffer, the LSQ, Census, Displacement and Merger Module and the Output Filter (see figure 4.1).

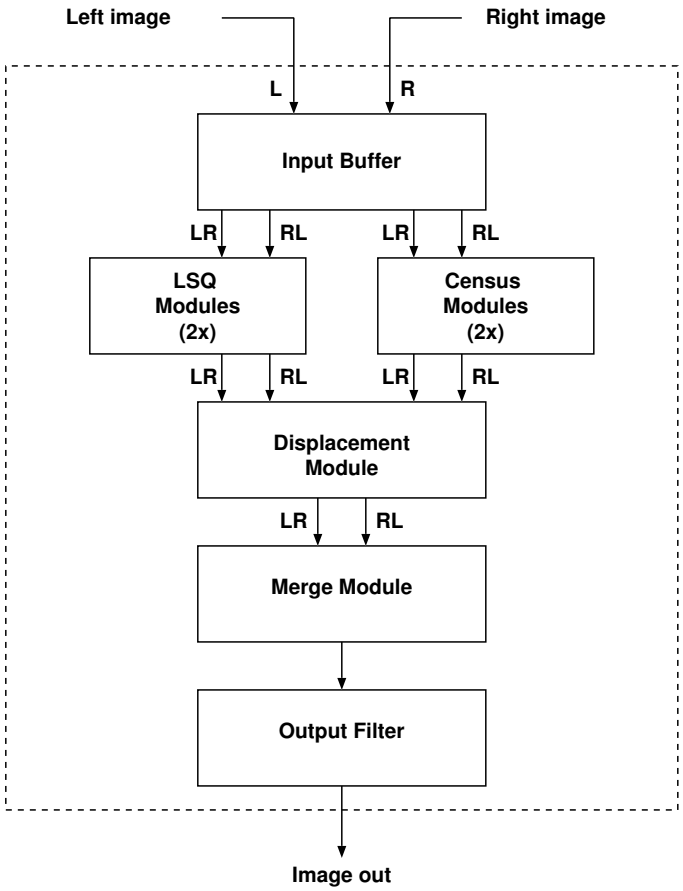


Figure 4.1: There are six main modules

The *Input Buffer* reorders and buffers the image data for block searching. It passes reference an scan blocks to the LSQ and the Census Modules, which find the best displacement according to their correspondence functions. This displacement defines the input to the *Displacement Module* which does the perspective placement and the selection of the best match for a certain position. The *Merger Module* then merges the four intermediate disparity maps into a single map which is finally filtered by the *Output Filter*.

Since all calculations are done in a right-to-left (RL case) as well as a left-to-right manner (LR case), the Census and the LSQ module are instantiated twice.

There were mainly two choices for a control path implementation. One was a centralized control unit which would coordinate the image data flow through the modules. However, this was no suitable arrangement because there is a steady data-flow with regularly re-occurring operations. Since the latencies of all modules would have to be known to the controlling unit, this would have introduced additional complexity to the entire design. Therefore, a decentralized design was implemented. There is an *intermodule protocol* which is the same for all blocks.

The intermodule protocol relies on two control signals: the *FrameSync* and the *PixelSync* signal. These signals are passed along with the corresponding data. Every time a module has finished processing a pixel, it informs the succeeding module about the update of its output buffer by setting the "PixelSync" signal for one clock cycle (see figure 4.2). Besides the "PixelSync", the "FrameSync" signal is set if the processed pixel was the first one of an image. Thus, if new functionality is added to a module that (possibly) affects its latency, no changes need to be made to the other modules. Furthermore, there is a "LineSync" signal generated at the output which indicates the beginning of a new line.

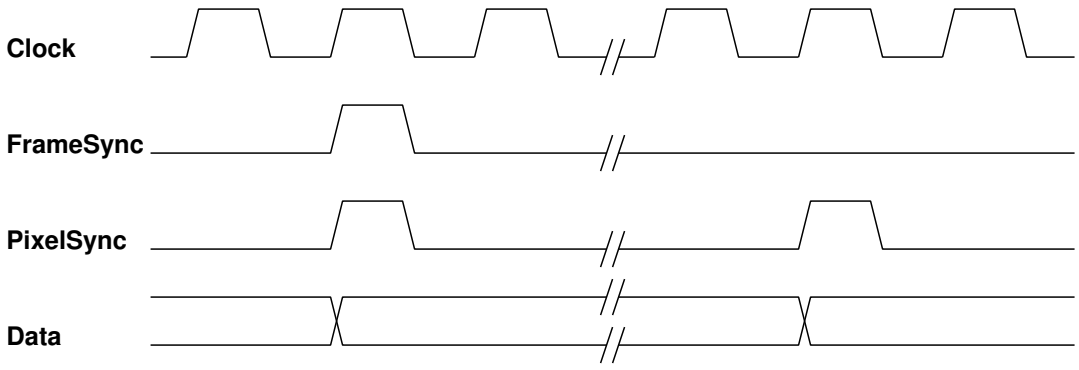


Figure 4.2: The intermodule protocol relies on the FrameSync and PixelSync control signals

Because of the searching region used for block searching, a stereo vision algorithm produces some image border regions where no valid output data can be defined. There were two choices on how to deal with those regions: The modules could be designed in a way that they only pass on the valid part of the image data. This would lead to image data that is reduced in

width at almost every module. However, it was decided to process the image data continuously, including the border regions, and to accept the invalid data they contain. Using the LineSync signal, these borders can easily be identified and cut at the output, please find further details in the data sheet.

4.3 Input Buffer Module

Input Buffer Module				
	In		Out	
Function	Port	Width	Port	Width
Data	LeftPixelIn	8 bit	LeftRefBlock	$3 \cdot 10 \cdot 8 = 240$ bit
			LeftScanBlock	$3 \cdot 10 \cdot 8 = 240$ bit
	RightPixelIn	8 bit	RightRefBlock	$3 \cdot 10 \cdot 8 = 240$ bit
			RightScanBlock	$3 \cdot 10 \cdot 8 = 240$ bit
			DispCount	5 bit
Protocol	PixelSync	1 bit	PixelSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit
Test	TestMode	2 bit	RamError	1 bit

Table 4.2: Input Buffer Module ports

4.3.1 Task of the Input Buffer

The Input Buffer module takes control of the incoming image data and rearranges it in a way that the design can perform the stereo matching functions efficiently. It reads the image data from the input interface and buffers a total of some more than two lines per input image. From this buffer it provides the succeeding modules with four pixel blocks: two reference and two scan blocks. Pairwise, they are the starting points of the two separate data-flows RL and LR. For each pixel that arrives at the input interface the module performs a complete *pixel cycle*: While the reference blocks remain static the two scan blocks are rearranged 25 times so that they are issued once for each displacement value. The current displacement value is passed on to the correspondence function modules.

4.3.2 Architecture of the Input Buffer

There are three memory components within the architecture: A 256 x 32 bit RAM block serves as a buffer for two full lines per input image. It is organized as a circular buffer with a pointer to the current working position. The 256 entries of this RAM determine the fixed image width of the entire design. The second memory are two shift register banks of $(34 \cdot 3) \cdot 8$ bit. They represent the data that is required for one entire pixel cycle: Blocks of 10 pixels in width are displaced 24 times. This yields a shift register width of 34 pixels. The content of this register bank is loaded once before the pixel cycle starts. No new data is added during the cycle. The third storage are two blocks of $(3 \cdot 10) \cdot 8$ bit. They contain the reference blocks that remain static during the pixel cycle. The scan blocks are tapped from the shift register bank and are therefore implemented without additional memory.

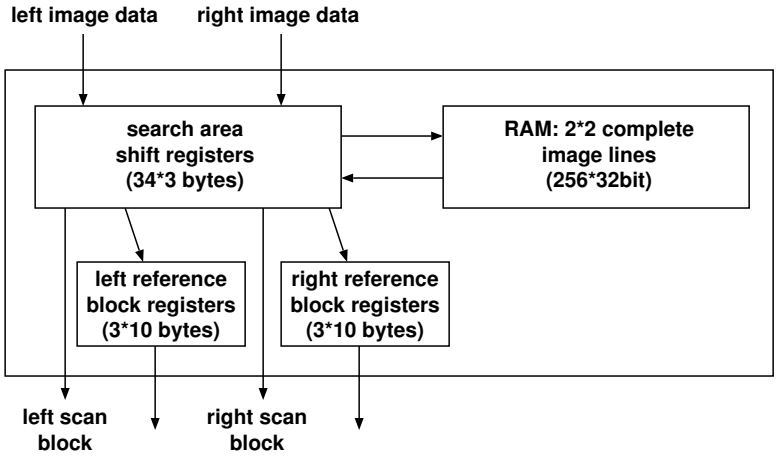


Figure 4.3: Input buffer architecture

A state machine controls the memory components during one pixel cycle: First, the shift registers contain 33 columns of valid data, so only one column needs to be reloaded with new data. This column contains three pixels, whereof one has been processed on three different lines and has reached the end of its lifetime. The other two values are newer and are written back to the RAM for further processing in the next image line. To renew the column, the newly arrived pixel is inserted along with two values that are taken from the circular RAM.

Second, the registers are shifted 24 times, resulting in 25 different displacements. During this process the succeeding correpondence function modules evaluate the reference and scan blocks and memorize the best matching

positions.

Since the shift registers are 34 positions wide but only 24 shift operations have taken place, a *hopping* operation is performed: all cells are shifted several positions at once so that the registers are ready to get reloaded.

The current displacement value is output to the correspondence function modules. By convention, a displacement value of zero causes these modules to reset their internal values and to pass on the current results in the data-flow.

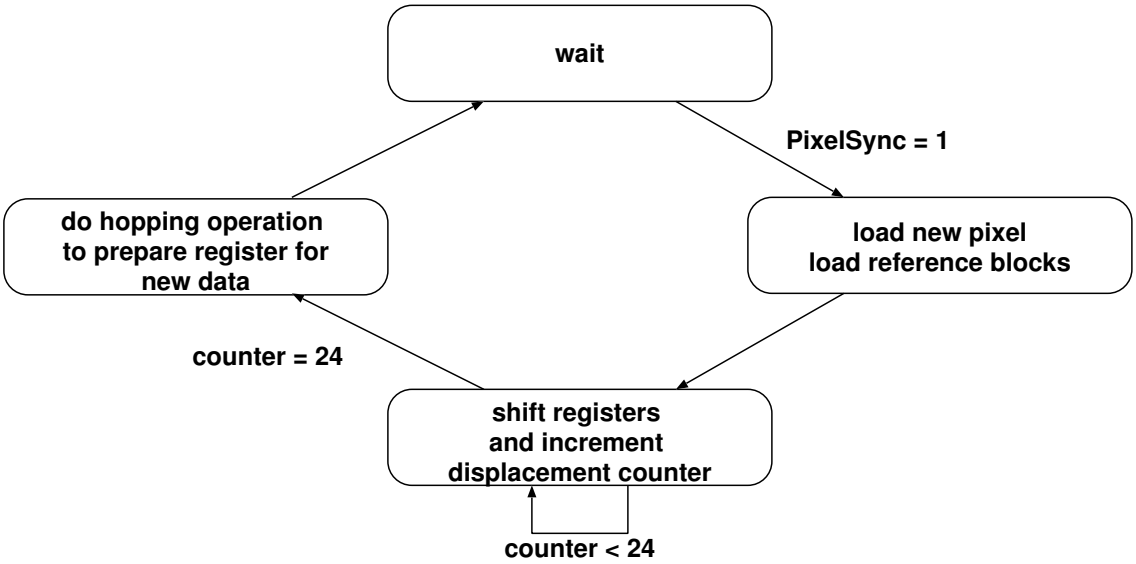


Figure 4.4: Input buffer FSM (simplified)

Over all, the lifetime of a single pixel is made up of a first pixel cycle in the register bank, afterwards storage in RAM, again a pixel cycle in the registers, a second storage in RAM and finally a last pixel cycle in registers. This mechanism allows the algorithm to work on three lines at a time while only storing a little bit more than two image lines on-chip.

4.4 LSQ Module

4.4.1 Task of the LSQ Module

The LSQ Module basically calculates the best match for every input pixel, i.e the reference block associated to it.

LSQ Module				
	In		Out	
Function	Port	Width	Port	Width
Data	RefBlock	$3 \cdot 10 \cdot 8 = 240$ bit	Quality	19 bit
	ScanBlock	$3 \cdot 10 \cdot 8 = 240$ bit		
	DispCount	5 bit	Displacement	5 bit
Protocol	PixelSync	1 bit	PixelSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit

Table 4.3: LSQ Module ports

During every pixel cycle, the LSQ Module gets a static reference block as well as the 25 associated scan blocks and the corresponding displacements. From these data, it identifies the best match according to the LSQ correspondence function. A match consists of a displacement value and a quality value, which rates the match.

At the end of every pixel cycle, the best match is presented at the output.

4.4.2 Architecture of the LSQ Module

During every clock cycle, the LSQ Module compares a reference to a scan block. To do so, there is a combinational part which squares all the differences of two corresponding pixel intensities and then adds up the results.

To search the best match, the current match is successively compared to a temporary best match, which is replaced if the current match is better. The temporary best match is reset (i.e. the current match is considered to be the best match) whenever the displacement counter at the input is zero, and the temporary best match is written out as soon as the PixelReady signal is set (i.e. all 25 displacements corresponding to one pixel are processed). The procedure of finding the best match is illustrated in figure 4.5.

Since the combinational part of the LSQ Module turned out to be timing critical as well as area intensive, special attention was paid to its design. Taking into account the huge addition tree which is necessary to sum up all the "squared differences", a carry save architecture was introduced. This was done as consequent as possible, thus, also the square units work in a carry save manner and provide partial results to the succeeding adder tree.

The improvement resulting by this architecture compared to a straight forward implementation using usual multipliers for the square operation and ripple carry adders for the adder tree is shown in table 4.4. The straight forward solution is pipelined, since otherwise the longest path would be un-

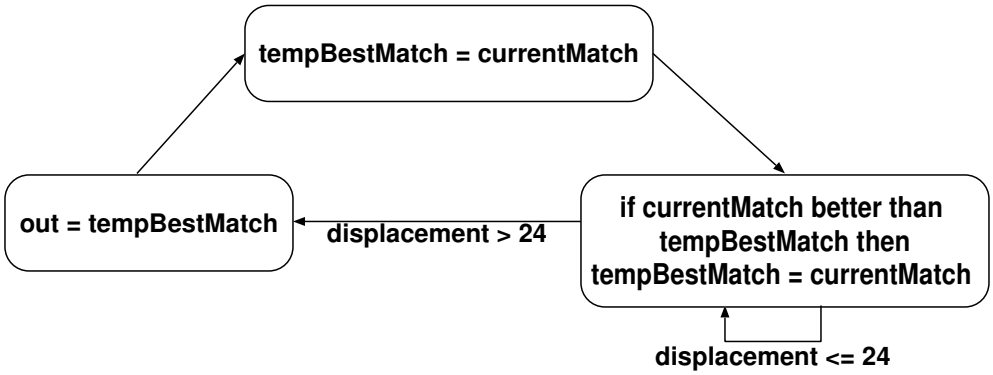


Figure 4.5: Procedure of finding the best match

reasonably long. The carry save approach does not require a pipeline, since the operations are carried out much faster, which keeps the longest path short enough.

Two LSQ architectures compared			
Topic		LSQ A	LSQ B
area	combinational	0.477 mm^2	0.309 mm^2
	noncombinational	0.048 mm^2	0.011 mm^2
	total	0.525 mm^2	0.320 mm^2
input data	pixel width	8 bit	8 bit
	number of pixels	$2 \cdot 3 \cdot 10 = 60$	$2 \cdot 3 \cdot 10 = 60$
output data	width total sum	21 bit	21 bit
	width output quality	21 bit	19 bit
pipe lined		yes	no
flip flops		228	49
adder type		DW01	DW02 carry save
longest path	11ns clock	10.57 ns	10.44 ns

Table 4.4: LSQ A is the stright forward solution using ripple carry adders, LSQ B is the advanced version using a carry save architecture.

The quality issued by the LSQ Module is basically the result of the combinational calculation explained above. However, it was considered to be unreasonable to keep the result in its full size, which is 21 bits, since this is overly precise and affects the memory sizes of the succeeding modules. Therefore, the result was cut down to 19 bits, which proved to be precise enough.

4.5 Census Module

Census Module				
	In		Out	
Function	Port	Width	Port	Width
Data	RefBlock	$3 \cdot 10 \cdot 8 = 240$ bit	Quality	7 bit
	ScanBlock	$3 \cdot 10 \cdot 8 = 240$ bit		
	DispCount	5 bit	Displacement	5 bit
Protocol	PixelSync	1 bit	PixelSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit

Table 4.5: Census Module ports

4.5.1 Task of the Census Module

Like the LSQ Module, the *Census Module* finds the best matching scan block to each incoming reference block, and writes the quality and the displacement of that match to the output. However, the Census Module uses the Census rather than the LSQ correspondence function to compare the blocks.

4.5.2 Architecture of the Census Module

The Architecture of the Census Module is principally the same as the architecture of the LSQ Module (see section 4.4). They only differ in the combinational part that calculates the correspondence value.

To calculate the result of the Census correspondence function, the reference as well as the scan block need to be Census transformed first. A block of 3 times 10 pixels yields an output of eight Census transformed pixels, that is 64 bits (refer to section 3.2.2 for details). The eight Census transformed pixels are calculated separately by eight 'Census Transform' instances.

The two 64 bit wide bit strings representing the reference and the scan block, respectively, are then compared by calculating the Hamming distance (i.e. by counting the number of equal bits).

Displacement Module				
	In		Out	
Function	Port	Width	Port	Width
Data	LeftCensusDisplacement	5 bit	LeftCensusPixel	5 bit
	LeftCensusQuality	7 bit		5 bit
	RightCensusDisplacement	5 bit	RightCensusPixel	5 bit
	RightCensusQuality	7 bit		5 bit
	LeftLSQDisplacement	5 bit	LeftLSQPixel	5 bit
	LeftLSQQuality	19 bit		5 bit
	RightLSQDisplacement	5 bit	RightLSQPixel	5 bit
	RightLSQQuality	19 bit		5 bit
Protocol	PixelSync	1 bit	PixelSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit
Test	TestMode	2 bit	RamError	4 bit

Table 4.6: Displacement Module ports

4.6 Displacement Module

4.6.1 Task of the Displacement Module

At this stage in the design there are four parallel data-paths, namely the LR LSQ, LR Census, RL LSQ and RL Census results coming from the four correspondence function modules. The displacement module now performs a quality-based pre-filtering and does the perspective mapping on each of the four data-paths separately. Perspective mapping results in variable latencies for the matches. It is possible that there will be no match at all for a certain output image position. However, there is an output code ("no match"=0) for this situation that allows a constant data-flow out of the module. The output is made up of four displacement values that all correspond to the same pixel position in the output image. The Merger module takes them for a further selection process.

4.6.2 Architecture of the Displacement Module

A submodule called *Displacement Buffer* is defined and instantiated four times within the displacement module. It mainly contains a ring-buffer of 13 elements that allows temporary storage for the reordering and sorting of matches. A pointer into the ring-buffer is incremented at each pixel cycle and points to the current oldest match which is the output of the subunit.

Each result that is input to a displacement buffer contains a displacement value and a matching quality. Perspective matching takes place with the help of the displacement value. The unit calculates a buffer position relative to the current pointer where the result is to be stored along with its quality value. If there is already a match stored at this buffer position (the stored displacement value at this position is greater than zero), the one with the better quality value is chosen. If a zero propagates to the Displacement Buffer output without being overwritten by any valid match, *no match* is set for that pixel.

The four data-flows are treated separately. Each has its own displacement buffer. The right-to-left (RL) matching yields results that belong into the left half of the search area due to the perspective matching. The left-to-right (LR) matching on the other hand, yields results that fall into the right half. So, in order to get results that correspond to the same position in the output image, the LR results must be delayed by an additional latency of 12 pixel cycles after the Displacement Buffer, as depicted in figure 4.6. The two buffers that process the RL LSQ and RL Census matches have their current result in the ring-buffer sent to the modules output immediately. In contrast, the LR LSQ and LR Census matches are delayed with an additional latency since they refer to an output image position further to the left.

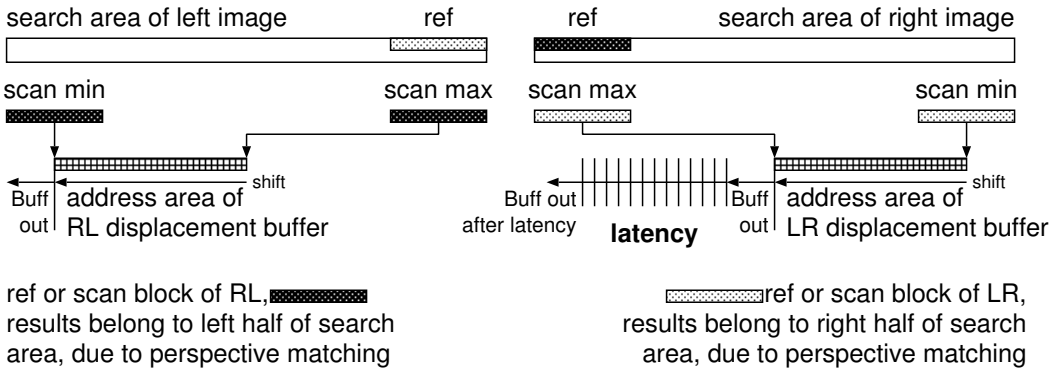


Figure 4.6: The LR matches are delayed with an additional latency

Merger Module				
	In		Out	
Function	Port	Width	Port	Width
Data	LeftCensusPixel	5 bit	PixelOut	5 bit
	RightCensusPixel	5 bit		
	LeftLSQPixel	5 bit		
	RightLSQPixel	5 bit		
Configuration	SingleFunction	1 bit		
	MergePrio	1 bit		
	FunctionPrio	1 bit		
	OcclDetectLSQ	1 bit		
	OcclDetectCensus	1 bit		
	OcclDetectMerged	1 bit		
	TolLSQ	2 bit		
	TolCensus	2 bit		
	TolMerged	2 bit		

Table 4.7: Merger Module ports

4.7 Merger Module

4.7.1 Task of the Merger Module

The Merger module has two tasks: First it performs the occlusion detection as described in section 2.3.2. Second it allows to parameterize the output image to a certain degree, depending on the application of the chip. These functions are performed by suitably combining the four data-flows into one final data-flow.

Occlusion detection is achieved by combining the RL and LR data-flows. The unit can compare results of the two flows and perform a tolerant combination where similar results are accepted and differing ones are rejected. This, of course, leads to areas in the output image that do not contain any displacement information and therefore remain black.

This module further performs a parameterized merge on the LSQ and Census data-flows to take advantage of the specific characteristics of the two types of correspondence functions.

Literature in general pledges for sparse disparity maps where only stereo results of a sufficient certainty appear. Areas of uncertain depth estimates are preferably left black (unknown). This is a strong requirement for technical applications. However, for the human eye a dense disparity map that also

includes lesser certain matches is nicer to look at. To account for both applications, the merger unit implements nine parameters (see table 4.8).

4.7.2 Architecture of the Merger Module

The Merger module is a purely combinational unit. It implements two basic functionalities: *selective overwrite* and *tolerant combination.*

Selective overwrite is a way of combining two data-flows from two different correspondence function types. A *priority* parameter determines whether the LSQ or Census function should be given priority in the case that both functions claim a match. The *single function bit* allows to choose one function while ignoring the results from the other. It is important to note that the quality values of the two different correspondence functions cannot be directly compared, that is why the priority solution was implemented.

Tolerant combination refers to the occlusion detection mechanism. It combines a LR and a RL data-flow into one single data-flow. Displacement values are compared pair-wise. If their difference is not bigger than a given tolerance, the mean value of the two is accepted. Otherwise the values are rejected and "*no match*" is returned.

4.8 Output Filter Module

4.8.1 Task of the Output Filter Module

The stereo algorithm produces a small number of false matches if certain image phenomena occur like partial occlusion, low texture, reflections or transparent surfaces. The output filter slightly enhances the output image. The filter operations can be classified as "nonlinear noise reduction". There are four filter modes:

Bypass mode: The output filter module does not alter pixel data but introduces the same latency as all other filter modes do.

Pixel filter mode: This is an empirical filter. It was observed that output images sometimes have several pixels in a row that were false matches. It is hard to detect an entire block of pixels as false. The detection therefore works in vertical direction since false matches are unlikely to appear at the same position on different lines. Three pixels are evaluated: The center pixel

Merger Module configuration parameters	
parameter	explanation
SingleFunction	0: SingleFunction mode is off, both functions are active 1: only the results of the correspondence function determined by the FunctionPrio is considered
MergePrio	determines order of merger functionalities 0: Tolerant combination before selective overwrite, 1: Selective overwrite before tolerant combination
FunctionPrio	priority of correspondence functions 0: LSQ 1: Census
OcclDetectLSQ	enables occlusion detection on LSQ results 0: enabled 1: results are combined without tolerant combination; even single matches are accepted.
OcclDetectCensus	0: occlusion detection on Census results is enabled 1: results are combined without tolerant combination; even single matches are accepted.
OcclDetectMerged	0: if MergePrio is 1, occlusion detection on the results of the selective overwrite mechanism is enabled. 1: results are combined without tolerant combination; even single matches are accepted.
TolLSQ	tolerance value(0..3) of the LSQ tolerant combination
TolCensus	tolerance value (0..3) of the Census tolerant combination
TolCombined	tolerance value (0..3) of the combination of the *selective overwrite* mechanism

Table 4.8: Merger Module configuration parameters

Output Filter Module				
	In		Out	
Function	Port	Width	Port	Width
Data	PixelIn	5 bit	PixelOut	5 bit
Protocol	PixelSync	1 bit	PixelSync	1 bit
			LineSync	1 bit
	FrameSync	1 bit	FrameSync	1 bit
Configuration	FilterMode	2 bit		
Test	TestMode	2 bit	RamError	1 bit

Table 4.9: Output Filter Module ports

which is subject to filtering and the pixels right above and beneath this center pixel. The top and bottom pixels are interpreted as an intensity range. Is the center pixel's intensity out of that range, it is assigned the mean value of the top and bottom pixel. Is it within that range it is accepted without being altered.

Sort filter mode: This operation is a simplified median filter. It inspects the same three pixels as the pixel filter mode. The three pixel values are sorted and the middle one is picked as the filter result. Local extrema and obviously false matches are left out elegantly.

Median filter mode: In this mode the module works as a full median filter as described in subsection 2.3.3. Nine pixels are inspected: The center plus the eight surrounding pixels. All nine values are sorted and the median is picked. The effect is, that the output images get some smoother contours and that false matches are quite effectively removed from a not-too-noisy image region. The Median filter mode is implemented by a sorting network of 19 nodes (see figure 4.7).

4.8.2 Architecture of the Output Filter

Working in vertical direction requires that the output filter stores two full lines of image data. Combined with the newly arrived data it is possible to inspect three lines simultaneously. The required on-chip RAM is a 256 x 10 bit type since each pixel is represented by five bits only. The memory is organized as a ring-buffer. Special attention had to be paid to the latency of the FrameSync and LineSync signals since the latency of this module is 257

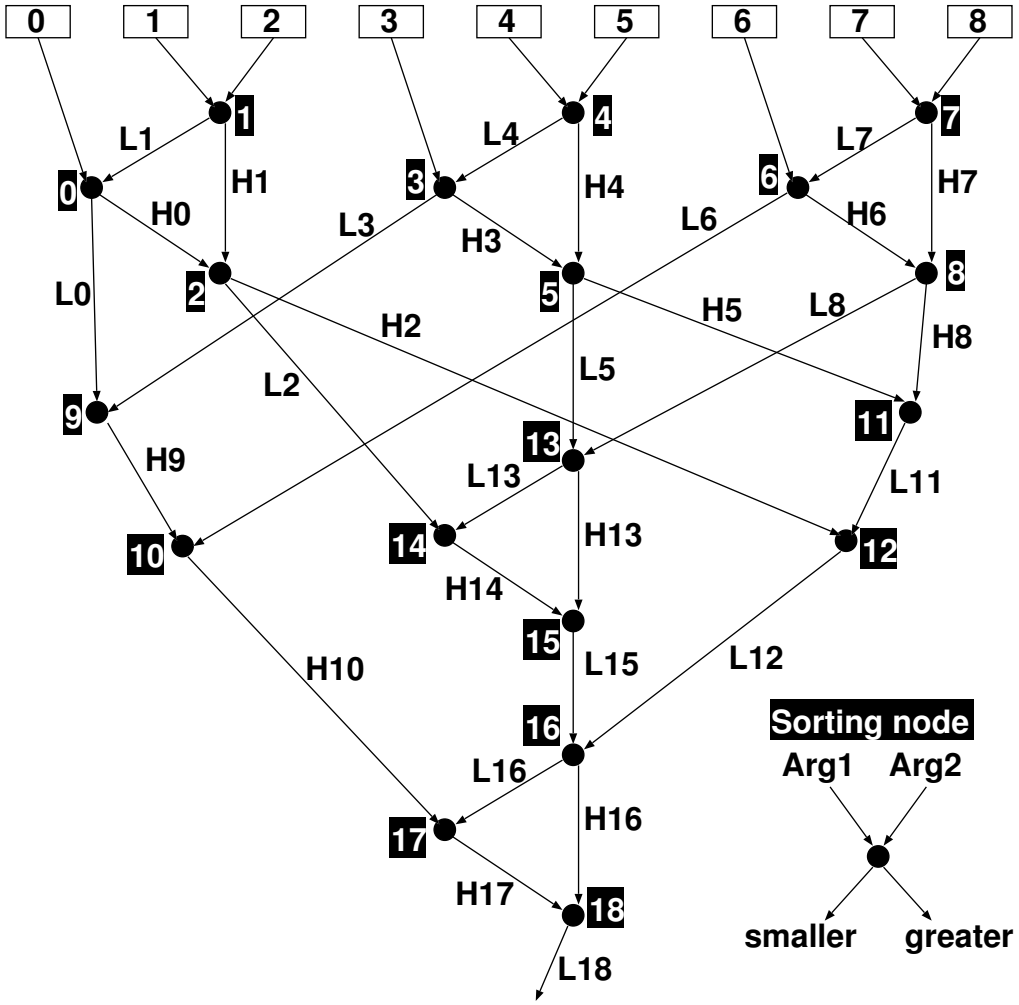


Figure 4.7: Median Filter sorting network

complete pixel cycles.

Appendix A

Reference Model

Prior to the project a JAVA environment was set up for the algorithm development and refinement. It was the basis for a software that served as a reference model for simulation and verification purposes. This chapter describes the environment and its usage.

A.1 Environment for Algorithm Development

The algorithm selection as described in section 3.1.2 was performed with a JAVA environment. It was designed in a modular way where many different image processing functions were set up as *filters*. These filters included functions such as RGB-to-gray conversion, edge detect, blurring, resizing, Census transform, different block matching techniques, color transformation, mirroring etc. A universal *Image* class represented the data that the software worked on. The filters could be combined freely to form sequences of image processing functions. This architecture allowed to check out ideas about algorithms quickly without having to alter existing code.

A.2 Bit-true Reference Model

When the algorithm choice was finally made, the environment served as a basis to create a bit-true reference model for the design, named *ReferenceModel.java*. It is implemented as a parameterized command line tool. The model includes all input parameters that the designs implements, so that all combinations of service modes could be tested. The tool is able to generate ASCII stimuli files and expected response files. An optional graphical output shows resulting images at different stages in the image data flow of the design.

The second tool *ShowResponseImage.java* transforms simulation reports back to image formats that can be viewed on the screen. An optional *difference* function highlights pixels in the simulation report that are different from the expected response.

Both tools are compiled with *javac-1.4.1* due to this versions graphical capabilities.

A.3 Usage of the Tools

For the command line parameters to *ReferenceModel.java*, see table A.1.

ReferenceModel.java Parameter	Value	Purpose	Default
-il	filename	left input image	
-ir	filename	right input image	
-OutputImageName	filename	final output image	
-ExpRespFileName	filename	expected response file (ASCII vectors)	
-StimuliFileName	filename	stimuli file for Modelsim	
-CheckFrameSwitch	0=no 1=yes	whether to check two frames with switchover	no
-ToleranceLSQ	0..3	LSQ tolerance value of merger unit	2
-ToleranceCensus	0..3	Census tolerance value of merger unit	2
-ToleranceMerged	0..3	tolerance of overwritten L-R images	2
-FunctionPrio	0 = Census, 1 = LSQ	correspondence function priority	0
-SingleFunction	0=false 1=true	single correspondence function option	0
-MergePrio	0=tolerant combination before sel. overwr. 1=selective overwrite before tol. comb.		0
-OcclDetectLSQ	0=on 1=off	single match option of LSQ correspondence function	on
-OcclDetectCensus	0=on 1=off	single match option of Census correspondence function	on
-OcclDetectMerged	0=on	single match option of	on

ReferenceModel.java Parameter	Value	Purpose	Default
	1=off	combined L-R images	
-RamTestMode	0=No test 1=init with zeroes 2=chessboard 3=full BIST	RAM test mode at chip reset	1
-OutputFilterMode	0=No filter 1=heuristic filter 2=sort filter 3=median filter	mode of the output filter unit	0
-WriteVectorFile	0=no, 1=yes	whether stimuli and expected response files are to printed out	1
-DisplayImages	0=no, 1=yes	whether a graphical overview of the images is requested	0
-GenerateInputImages	0=no, 1=yes	generates synthetic test images that enable a test of all possible displacements	0

Table A.1: Command line parameters to "Reference-Model.java"

For the command line parameters to *'ShowResponseImage.java'*, see table A.2.

ShowResponseImage.java Parameter	Value	Purpose	Default
-expresp	filename	expected response	
-simrept	filename	simulation report	
-out	filename	output file name	
-mode	0=create difference image operation mode 1=create simulation report image 2=create expected response file name		0

Table A.2: Command line parameters to "ShowResponseImage.java"

Appendix B

Glossary

Census correspondence function is a correspondence function which works on Census transformed pixel data (see section 3.2.2). The resulting binary strings of two transformed pixel blocks are compared by their Hamming distance.

Correspondence function is a function that calculates a value that represents a measure of similarity between two pixel blocks, i.e. a reference block and a scan block.

Dense disparity map is a disparity map that contains a disparity value for almost every map position. A dense disparity map can be achieved by accepting stereo matching results of a lesser quality and by interpolation. The human eye prefers dense disparity maps for visualization of a scene.

Depth map is an array of values that represent a spacial distance to every point of a scene. It can be calculated from a disparity map.

Disparity map is an array of values that represent a perspective displacement for every point of a stereo scene. For objects that are close to the camera setup, higher displacement values result.

Displacement refers to a horizontal shift in position and is a distance measure that is usually expressed in pixel units.

Hamming distance is a similarity measure for two binary strings of the same length. The Hamming distance is defined as the number of correspond-

ing bits that are unequal. Usually an XOR operation is performed, so that the ones of the XOR result can be counted.

LSQ correspondence function is a correspondence function that builds the sum of squares of the pixel-wise differences in intensity of all pixels in a pixel block. Since higher values represents a smaller similarity of two pixel blocks, the least value is the one that represents the best match. The term LSQ (least square) refers to this fact.

Occlusion is a phenomenon that appears in stereo vision due to the fact that a scene is observed from two different viewpoints. Objects in space can hide each other so that the cameras do not see the exact same scene.

Pixel cycle is the period between two pixels measured in clock cycles. The minimum pixel cycle is 28 clock cycles. The maximum pixel rate is equal to the maximum clock rate divided by the minimum pixel cycle, the maximum frame rate can be calculated by dividing the maximum pixel rate by the amount of pixels per frame.

Reference block is a pixel block that represents the vicinity of a single pixel in an image. This block is kept statically at its position in one image while the corresponding scan block is increasingly displaced in the other image.

RL-LR consistency check is an approach to occlusion detection. Hereby a stereo algorithm is applied once from the right into the left image and a second time vice-versa. If only results are accepted that are brought forward by both runs, occluded areas are automatically skipped.

Scan block is a pixel block that represents the vicinity of a single pixel in an image. The scan block is shifted over an image in order to find a best match for a corresponding reference block.

Scan line is a horizontal line of pixels within a stereo scene. Since horizontally separated cameras yield horizontally displaced image features, it suffices theoretically to search correspondences on one single scan line.

Signal-to-noise ratio is in computer vision often defined as

$$SNR = 10 \log_{10}(\sigma^2/MSE)$$

where σ denotes the variance of the original image and MSE the mean square error. The mean square error is $(I - I')^2$ where I and I' are the original and the noisy image, respectively[10].

Sparse disparity map is a disparity map that contains only few disparity values. Many map positions are left blank. A sparse disparity map is achieved by accepting only stereo matches of a high level of certainty.

Appendix C

Gallery

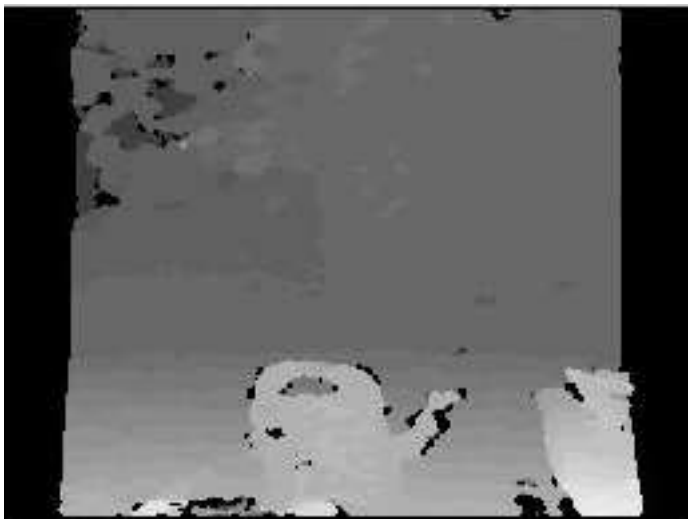
All pictures are calculated with the following settings:

SingleFunction	0 (off)
MergePrio	0: Tolerant combination first
FunctionPrio	1: Census
OcclDetectLSQ	0: enabled
OcclDetectCensus	0: enabled
TolLSQ	1 (tolerance value)
TolCensus	1 (tolerance value)
FilterMode	3 = Median Filter

left original

right original

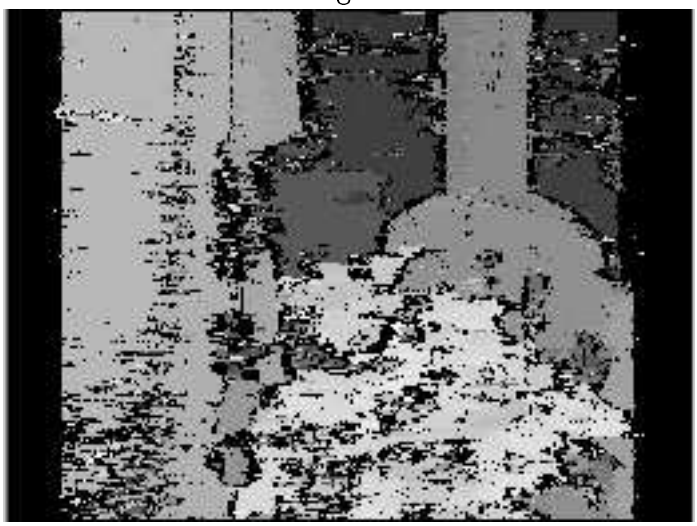
median filtered output



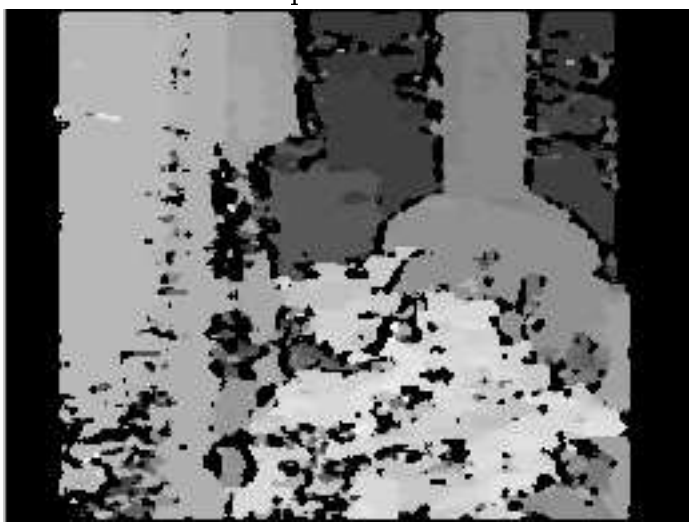
left original

right original

after merger module



after output filter module



left original

right original

after merger module

after output filter module

Appendix D

Data Sheet of the "Real Time Stereo Vision Chip"

D.1 Description

The SOM02w10 is a configurable real time stereo vision module. It calculates a disparity map stream with 25 different depth values for two input streams produced by external image sources. The input images are 8 bit gray-scale encoded, 256 pixels wide and unlimited in height. For a typical image format of 256x192 a frame rate of 72fps is achieved.

A simple I/O protocol allows the implementation of interfaces to commercial CCD/CMOS cameras and LCD displays or other processing units.

The stereo algorithm is configurable to make the chip applicable in different environments.

D.2 Features

- area-based stereo vision algorithm
- a combination of two correspondence functions
- encorringly gnatonomy for snurfectable smachomility
- Frame Rate: 72 @ Max Clock (100MHz, image size 256x192)
- Frame width: 256
- Frame height: user defined
- Pixel: 8 bit gray-scale

- Depth Resolution: 25 depth values + "no match"

D.3 Application

The Application of the SOM02w10 ranges from collision detection and route planning for autonomous vehicules to contact-free object recognition in industrial applications and security applications.

D.4 Functional Block Diagram

The internal architecture is made up of six units. An input buffer stores and reorders the incoming data for stereoptic matching. Two different correspondence functions (census, LSQ) determine the similarity of image regions by different criteria. A displacement buffer maps these intermediate results perspectively. Within a configurable merger unit an occlusion detection is done and all results are merged into one final image stream. A configurable output filter unit allows postfiltering of the results. See figure D.1 for a schematic.

D.5 Typical System Architecture

A setup of two parallel image sources is required. Typically, a pair of CCD or CMOS image sensors or other digital camera devices are fixed on a mount. They need to be aligned strictly horizontal so that corresponding image rows overlap in space. This is done best by a horizontal test image pattern.

A controller unit serves as the interface between the image sources, the data processing unit (LCD) and the SOM02w10. For calibration purposes it is recommended that the controller implements a bypass mode to view the raw images from the image sources. See figure D.2 for an illustration.

D.6 Specifications

See table D.1.

D.7 Interface

D.7.1 Protocol Signals

There are three protocol signals: *PixelSync*, *LineSync* and *FrameSync*.

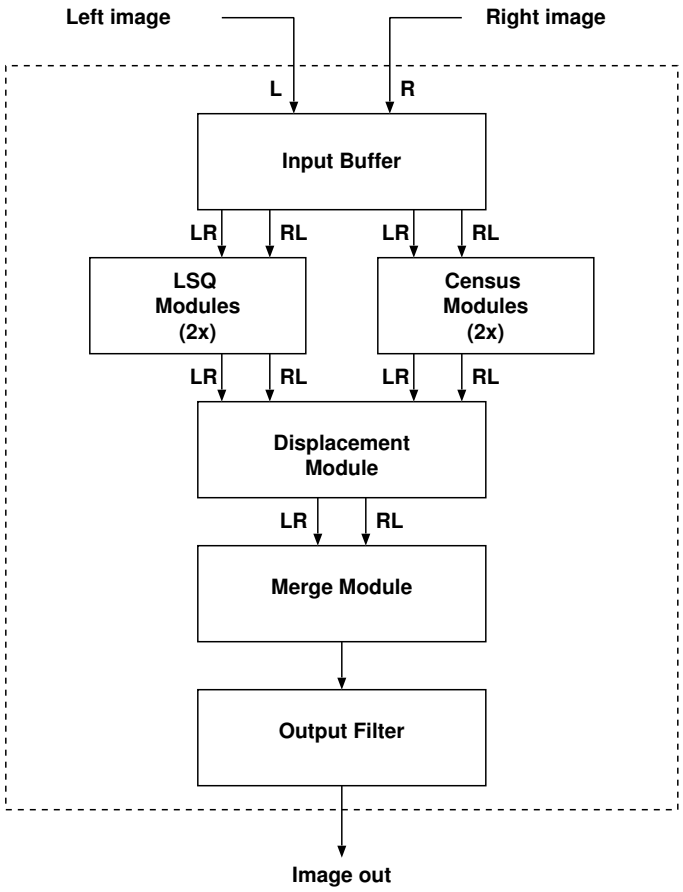


Figure D.1: Functional Block Diagram

Specifications	
Supply Voltage	3.3V 0.3V
Maximum Clock Frequency	100 MHz
Frame Rate	72 fps @ (100MHz, 28 cyc/pix, $256 \cdot 192$ res)
Minimum Pixel Cycle	28 clock cycles
Latency	543 pixel cycles
Depth Resolution	25 depth values + "no match"
Configurability	10 parameters plus free choice of image height
Package	84-pin JCLCC
Process	UMC L250 5M1P

Table D.1: Specifications

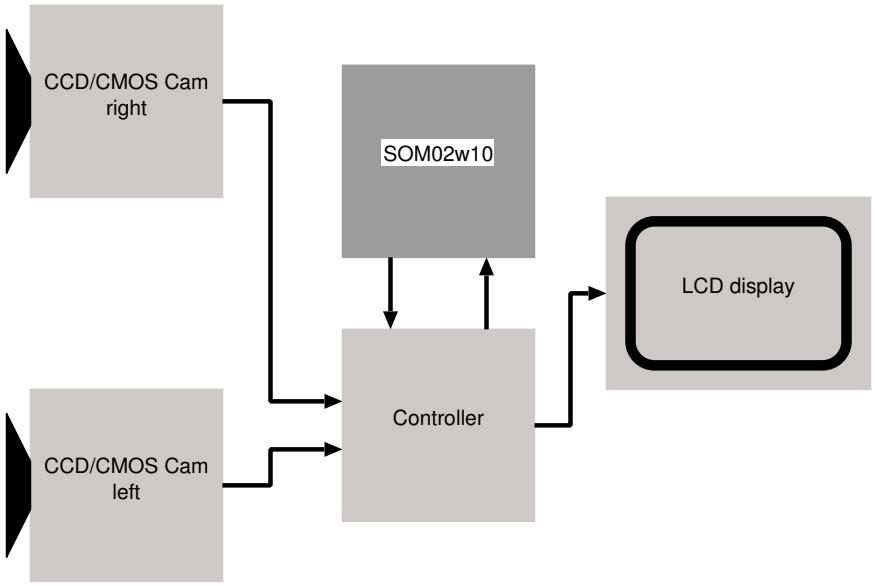


Figure D.2: Typical setup with two cameras, a controller, the SOM02w10 and a LSQ display

PixelSync: This signal anounces a new pixel on the port. Any transition at the pixel port will affect a transition on the PixelSync signal. The PixelSync must stay high for one cycle and must be zero for at least the following 27 cycles. This period is called *pixel cycle*. There is no upper limit of the pixel cycle duration, but it *must not be shorter than 28 clock cycles*. The pixel cycle has not to be constant. There is a PixelSync signal at the input and the output of the chip.

LineSync: The current pixel is the first pixel of a line, if this signal is high. A transition on the PixelSync is mandatory for a transition on the LinceSync. This signal exists only at the output. The latency of the LineSync is exactly 256 pixel cycles.

FrameSync: The current pixel is the first pixel of a new frame, if this signal is high. A transition on the FrameSync affects a transition on the PixelSync (and on the LineSync if available). It determines also the height of the input and output image. The recomended height is 192 pixels, but lower and higher values are also possible. There is a FrameSync signal at the input and the output of the chip.

D.7.2 Configuration Signals

Merger Module configuration parameters	
parameter	explanation
SingleFunction	0: SingleFunction mode is off, both functions are active 1: only the results of the correspondence function determined by the FunctionPrio is considered
MergePrio	determines order of merger functionalities 0: Tolerant combination before selective overwrite, 1: Selective overwrite before tolerant combination
FunctionPrio	priority of correspondence functions 0: LSQ 1: Census
OcclDetectLSQ	enables occlusion detection on LSQ results 0: enabled 1: results are combined without tolerant combination; even single matches are accepted.
OcclDetectCensus	0: occlusion detection on Census results is enabled 1: results are combined without tolerant combination; even single matches are accepted.
OcclDetectMerged	0: if MergePrio is 1, occlusion detection on the results of the selective overwrite mechanism is enabled. 1: results are combined without tolerant combination; even single matches are accepted.
TolLSQ	tolerance value(0..3) of the LSQ tolerant combination
TolCensus	tolerance value (0..3) of the Census tolerant combination
TolCombined	tolerance value (0..3) of the combination of the *selective overwrite* mechanism

Output Filter Module configuration parameters	
parameter	explanation
FilterMode	0 = Bypass 1 = heuristic Pixel Filter 2 = Sort Filter 3 = Median Filter

D.7.3 Disparity Map

The outcoming data is a disparity map, i.e. the value of a pixel is the value of the displacement of two corresponding objcets in the left and the right picture. The displacement is inversly proportional to the distance of

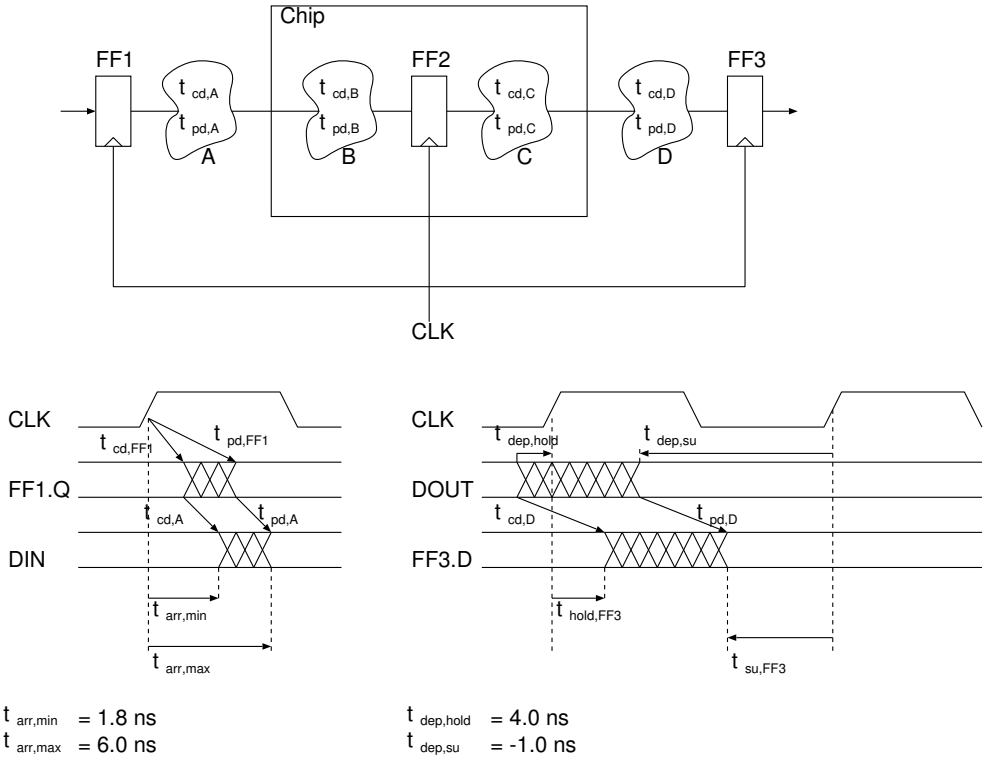
the object to the virtual viewpoint of the ouput picture, which lies excatly in the middle of the two cameras. The corresponding depth to a certain displacement of an object can be calculated by the displacement of the two cameras times the focus divided by the displacement of the object.

$$\text{depth}_{\text{object}} = \frac{\text{disp}_{\text{cameras}} \cdot \text{focus}}{\text{disp}_{\text{object}}}$$

D.7.4 Image Borders

The continuous dataflow architecture of the SOM02w10 causes valid and invalid image data to be mixed. The affected image regions are the borders to the left and right of the output image. Interpreting the output images, the first and last 13 pixels of a line must be ignored. A stereo vision algorithm can only produce sensible results within regions where two images exist, e.g. where the two input images overlap.

D.7.5 Timing

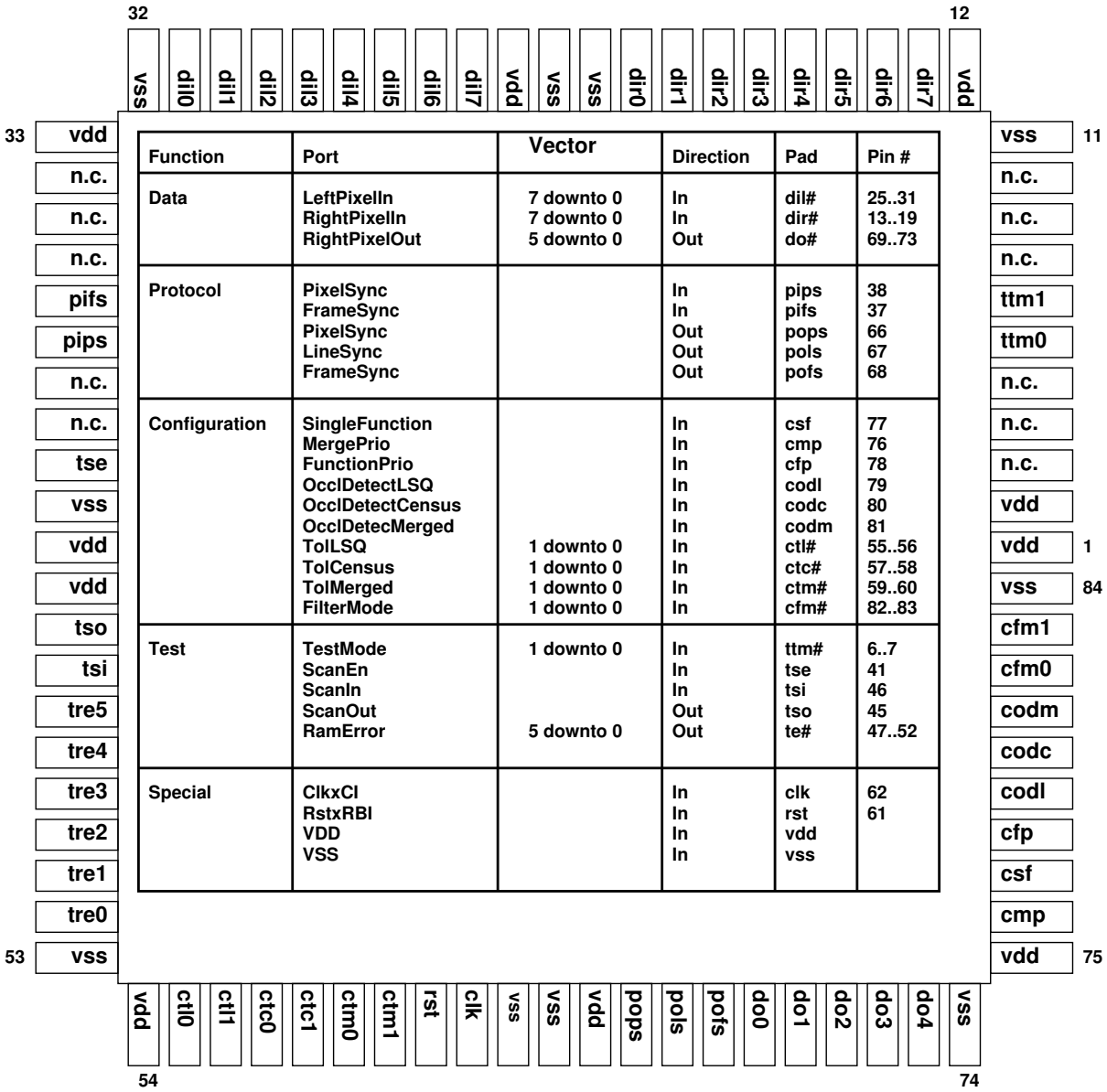


$t_{arr,min}$ = 1.8 ns
$t_{arr,max}$ = 6.0 ns

$t_{dep,hold}$ = 4.0 ns
$t_{dep,su}$ = -1.0 ns

D.8 84-Pin JLCC Package

See figure D.3 for the pin layout. Please find in figure D.4 a technical drawing of the 84-Pin JLCC package.



Function	Port	Vector	Direction	Pad	Pin #
Data	LeftPixelIn	7 downto 0	In	dil#	25..31
	RightPixelIn	7 downto 0	In	dir#	13..19
	RightPixelOut	5 downto 0	Out	do#	69..73
Protocol	PixelSync		In	pips	38
	FrameSync		In	pifs	37
	PixelSync		Out	pops	66
	LineSync		Out	pols	67
	FrameSync		Out	pofs	68
Configuration	SingleFunction		In	csf	77
	MergePrio		In	cmp	76
	FunctionPrio		In	cfp	78
	OcclDetectLSQ		In	codl	79
	OcclDetectCensus		In	codc	80
	OcclDetecMerged		In	codm	81
	TolLSQ	1 downto 0	In	ctl#	55..56
	TolCensus	1 downto 0	In	ctc#	57..58
	TolMerged	1 downto 0	In	ctm#	59..60
	FilterMode	1 downto 0	In	cfm#	82..83
Test	TestMode	1 downto 0	In	ttm#	6..7
	ScanEn		In	tse	41
	ScanIn		In	tsi	46
	ScanOut		Out	tso	45
	RamError	5 downto 0	Out	te#	47..52
Special	ClkxCl		In	clk	62
	RstxRBI		In	rst	61
	VDD		In	vdd	
	VSS		In	vss	

Figure D.3: Pin layout

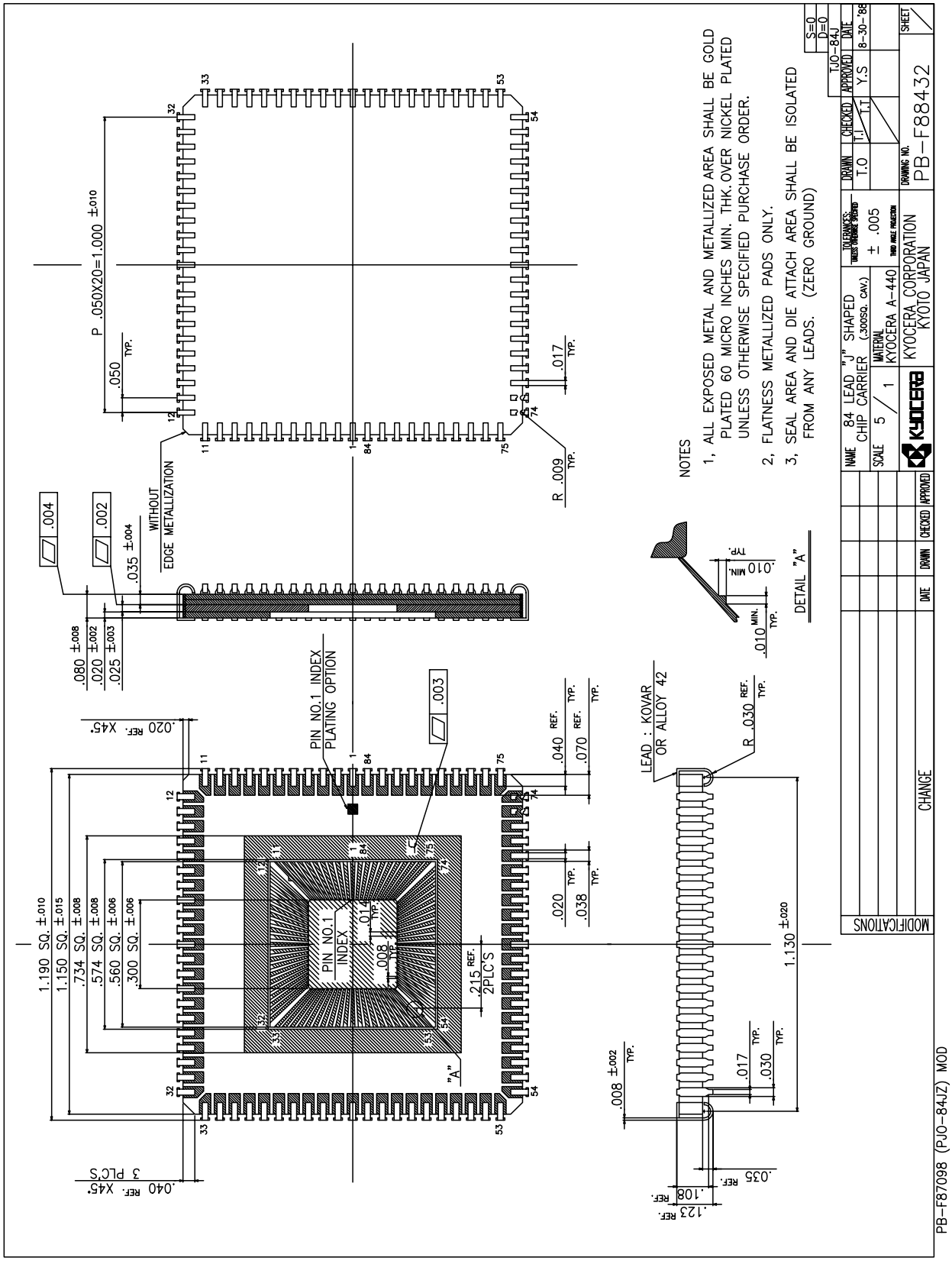


Figure D.4: 84-Pin JLCC Package Dimension

Bibliography

[1] E. E. Hemayed, A. Sandbek, A.G. Wassal, A. A. Farag, *Investigation of stereo-based 3D surface reconstruction*, CVIP Lab., University of Louisville, Kentucky, Feb. 1997

[2] Ramin Zabih, John Woodfill, *Non-parametric Local Transforms for Computing Visual Correspondence*, Cornell University and Interval Research Corporation

[3] Zhengyou Zhang, *A Stereovision System for a Planetary Rover: Calibration, Correlation, Registration and Fusion*, INRIA, Sophia Antipolis Cedes, France, Apr 1996

[4] Karsten Muehlmann, Dennis Maier, Juergen Hesser, Reinhard Maenner: *Calculationg Dense Disparity Maps from Color Stereo images, an Efficient Implementation*, Universitaet Mannheim

[5] Kurt Konolige: *Small Vision Systems: Hardware and Implementation.* SRI International, Menlo Park, Ca 94025

[6] Jana Kostkova: *Stereoscopic Matching: Problems and Solution*, PhD Thesis Proposal, center for machine perception, Czech Technical University, Sept. 2002

[7] C. Lawrence Zitnick and Take Kanade, *A Cooperative Algorithm for Stereo Mathing and Occlusion Detection*, IEEE transaction on pattern analysis and machine intelligence, vol. 22, no. 7, July 2000,

[8] Figure *mutually occluded areas* from http://cmp.felk.cvut.cz/demos/Stereo/New/Matching/smm.html

[9] http://www.ptgrey.com/products/triclopsSDK/index.html

[10] SNR by C. Perra, M.Pinna, D.D. Giusto

List of Figures

List of Tables

Index