

TECHNICAL UNIVERSITY OF CRETE

MASTER OF SCIENCE THESIS

A Heterogenous System Approach To The RealTime Stereo Problem

Author:

Yiannis AGADAKOS

Supervisor:

Dr. Yiannis PAPAESTATHIOU

*A thesis submitted in fulfilment of the requirements
for the degree of Master of Science*

in the

Microelectronics Hardware Laboratory
Electronic and Computer Engineering



September 2013

Declaration of Authorship

I, Yiannis AGADAKOS, declare that this thesis titled, ' A Heterogenous System Approach To The RealTime Stereo Problem ' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand."

Albert Einstein.

UNIVERSITY NAME (TECHNICAL UNIVERSITY OF CRETE)

Abstract

Electronic and Computer Engineering

Master of Science

A Heterogenous System Approach To The RealTime Stereo Problem

by Yiannis AGADAKOS

The time of monolithic devices is soon to end as the demand for innovative applications increases and sophisticated technologies like augmented reality come to be. A hard constraint for such technologies is immense processing power, while also operating under real time constraints, conventional systems and software principles fail to win the challenge. However a new era of embedded devices is dawning as heterogeneous systems become readily available, the power hidden inside these tiny giants can bring applications previously unthought of within our grasp. By applying distributed embedded software engineering and following the principles of heterogeneous systems we have created a novel system that can solve the Stereo Vision challenge in Real Time with more than 40FPS in QVGA and with less than 2 Watts total power consumption while at the same time performing feature detection, furthermore the proposed architecture allows the same system to be used for any processing intensive work where parallel calculation is required, such as signal processing, by exploiting our novel client-server system developed. Lastly the design and implementation principles used and proposed in this thesis foresee and support the cooperation with a higher level of heterogeneous systems with more processing units such as the OpenCL enabled GPUs, soon to come. The proposed system can satisfy some of the most demanding processing requirements in real time, opening the door to applications previously available only to workstation machines.

Acknowledgements

This work would have been infinitely more difficult without the guidance and tutorship of my supervisor Dr. Yiannis PAPAESTATHIOU and my colleague Konsantinos Makantasis MSc, whose expertise in Computer Vision proved invaluable. . . .

Contents

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
Abbreviations	x
1 Introduction	1
1.1 Introduction	1
1.2 Goal	3
1.3 Background	3
1.3.1 Computer Vision	3
1.3.1.1 Stereo Vision	4
Epipolar Geometry	4
Epipolar Constraint	6
Monotonicity and Uniqueness Constraint	6
1.3.2 Embedded Systems	8
1.3.2.1 Real Time	9
FPGA	9
DSP	10
GPP	10
ASIC	11
1.3.3 Summary	11
2 Architecture	12
2.1 Relative Work	12
2.1.1 Embedded Universe	12
2.1.2 FPGA based	13
2.1.3 ASIC	13
2.1.4 Atom	14

2.1.5	DSP	14
2.1.6	Heterogeneous Systems	14
2.1.7	Summary	15
2.2	Our Approach	16
2.2.1	Initial Design	16
2.2.2	Assumptions and Theory boosts	17
	Epipolar Constraint	18
	Monotonicity Constraint	18
2.2.3	GPP Side	19
2.2.4	DSP Side	20
2.2.5	Assembling the Puzzle	20
3	Implementation	23
3.1	Introduction	23
3.1.1	GPP Implementation Overview	23
3.1.2	DSP Implementation Overview	24
3.2	GPP	25
3.2.1	Tools and Frameworks	27
3.2.2	Tests	27
3.2.3	Optimizations	28
3.3	DSP	28
3.3.1	Tools	28
3.3.2	Implementation	29
	3.3.2.1 MexFile	34
3.3.3	Amdahl's Law	39
	3.3.3.1 Window Impact	39
	3.3.3.2 Maximum Disparity	39
3.3.4	OMAP3530	41
3.4	Optimizations	41
3.4.1	Platform independent optimizations- general principles	41
	3.4.1.1 Disparity-Window Choice	41
	3.4.1.2 Compiler optimizations	42
	3.4.1.3 Optimized image data serialization and Loop Unrolling	42
	3.4.1.4 Memory Copy Impact	43
	3.4.1.5 SAD on one step	43
3.4.2	Architecture Dependent Optimizations	45
	3.4.2.1 Variable types	47
	3.4.2.2 Intrinsics	48
	3.4.2.3 Use of Restrict	51
	3.4.2.4 Configuration Files Optimizations	51
4	Results	53
4.1	Assumptions and Standards	53
4.1.1	Quantum Change and Incremental Speed Ups	54
4.2	Disparity and Window Size	59
4.2.1	Experimental Disparity Maps	59
4.2.2	The Million Disparity per Second Metric	60

4.3 Summary	60
5 Conclusions	64
5.1 Future Work	65
6 The build path, fallacies and pitfalls	67
6.1 Time Distribution	68
A The build path, fallacies and pitfalls	69
A.1 Time Distribution	70
B Device Schematics And Info	71
B.1 OMAP3530	71
B.1.1 Arm Cortex A-8	76
B.1.2 TMS320C64+	80
B.1.3 POWERVR	83
Bibliography	86

List of Figures

1.1	Software System Flowchart	1
1.2	Epipolar Geometry and pinhole model	5
1.3	Church Image Pair	6
1.4	Epipolar Line	7
1.5	Monotonicity and uniqueness constraints	8
2.1	Processing Pipeline	17
2.2	Epipolar Line	18
2.3	Monotonicity Constraint	19
2.4	Software System Flowchart	22
3.1	Code Profiler Lower Disparity	33
3.2	Extracted depth, 32 Disparity Levels, Window size of $W = 5 * 5$	33
3.3	Code Profiler	33
3.4	Code Profiler Second run	33
3.5	Code Profiler Output	40
3.6	Extracted depth, 128 Disparity Levels, Window size of $W = 5 * 5$	40
3.7	Datapath	47
3.8	Functional Units.	49
4.1	Performance chart, frames per second and joules per frame.	56
4.2	Performance impact of pixel block size and disparity levels	59
4.3	Disparity Maps for block size 5 and varying disparity levels	61
4.4	Disparity Maps for block size 7 and varying disparity levels	61
4.5	Disparity Maps for block size 5 and varying disparity levels	62
4.6	Disparity Maps for block size 13 and varying disparity levels	62
4.7	Million Disparities Metrics, summary chart	63

List of Tables

4.1	Stereo algorithm Run Times, unoptimized	54
4.2	Stereo algorithm Run Times, Quantum Step:Incremental Optimizations .	55
4.3	Stereo algorithm Run Times, Quantum Step:Incremental Optimizations .	58

Abbreviations

LAH	List Abbreviations Here
SoC	System on a Chip
GPP	General Purpose Processor
DSP	Digital Signal Processor
GPU	Graphics Processing Unit
API	Application Interface
IP	Intellectual Provider
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
O/S	Operating System
RTO/S	Real Time Operating System
BIOS	Real Time Operating System
OE	Open Embedded
TCP/IP	Transmission Control Protocol , Internet Protocol
UDP	Intellectual Provider
USB	Universal Serial Bus
WAR	Write After Read hazzard
RAW	Read After Write hazzard

To my beloved family. Especially to my parents Vasili and Katerina, thank you for patience and bottomless support, for raising me with ethics and for awakening me to the world of science. . . .

Chapter 1

Introduction

1.1 Introduction

As technology advances more powerful devices find their way in the hands of even the most casual user. Sophisticated software aids us in our ever more complicated everyday tasks, from the simple now everyday organizer and email enabled smartphones to application software that aids us in finding the constellations when we look at the stars, it is a new age and every aspect of our daily lives is under improvement, however this is bounded by the power and energy efficiency of our mobile devices, at least up to now. We have reached a point however, where in our humble opinion, the processing power and capabilities hidden in these handy portable devices, has exceeded the problems of such constraints such as processing power, memory and battery efficiency, and if properly handled, we believe, they can be used to provide solutions previously out of their reach.

A good candidate for such devices, is the area of computer vision, a powerful field of expertise that finds use in many distinct applications from medicine to astronomy



FIGURE 1.1: The Google Glasses project brings the dawn of augmented reality to the masses..

and civil environments. Despite its tremendous usefulness, the processing requirements and resources it required held its exposure to real time portable devices low. However, this is not the case anymore, as recent embedded devices have shown a host of computer vision applications, that found their way to our smart phones and smart devices in general, one of the first massively used app was the smile detection one, now present in nearly any compact camera, recently a popular trend is that of the QR¹ code where a user uses his smart phone to scan the special tag on a product and retrieve information about it finally as augmented reality comes of age its enchanting applications awe us with every appearance they make in our lives.

One of the last computer vision fields to appear on portable devices is that of stereo vision, its complex and demanding algorithms prohibit its use to portable devices where real time is a hard constraint, however it would provide even more flexibility and utility to our smart devices, imagine your phone or pda being able to estimate depth, even at close ranges, even from trivial applications like measuring objects or distances up to sophisticated f augmented reality applications where the information of depth will provide new possibilities.

Another recent trend in devices is the creation of hybrid systems a jump we saw happening in desktop personal computers and workstations with the establishment of CUDA and OpenCL technologies where a system now has more than one processing units and with vastly different architectures as well, allowing for distributed computing to provide with unforeseen results.

Unlike the aforementioned desktop world, in the embedded world a host of fledgling hybrid systems appeared recently combining in a single chip multiple processing units, bringing thus embedded programming and development new possibilities, a good representative of this category is the OMAP3530 system on a chip², this SoC in its Gumstix OveroFire host will also be used in this Thesis.

The work done in this Thesis will not only show that a device such as OMAP3530 is capable of running stereo vision in real time, it will also demonstrate the full design and development process for the newly arrived embedded hybrid devices, the power they bring and the pitfalls one must avoid in order to harness their power. It is the authors hope that the design principles for heterogeneous systems, presented in the following sections will be applied in a variety of processing intensive applications especially in the field of computer vision and telecommunications as the benefits of using only software to achieve results comparable with hardware based solutions are numerous and vast further more the incoming OpenCL technology is bound to arrive soon in the embedded world further increasing the power of such devices. On top of that, the fact that the entire

¹ QuickResponse Code

² hence referred to as SoC

embedded system is hosted on an O/S like Linux with the software universe this implies, is an attractive notion for every engineer at least!

1.2 Goal

In this Thesis we challenge the stereo image problem attempting to create a real time embedded system with at least 15 FPS on QVGA images while at the same time performing application code capable of performing feature detection or other heavy processing in real time while keeping the power consumption at a minimum. In order to achieve this we will use the gumstix system on a chip utilizing both the arm GPP and the TI DSP unit researching and implementing a data pipeline between them turning the system to a heterogeneous embedded system. It is the authors hope that through this long and tough journey the principles and design patterns that will steam from this work will find their way to numerous embedded systems in the fields of telecommunication, image processing and in general any field where real time performance on big data is required.

1.3 Background

Intro

The work done in this Thesis is based into two major scientific fields the world of Computer Vision and the field of Embedded Systems. Following is a the required background and the main notions that we will commonly refer to in the chapters to come. The hereby presented topics are by no means complete, as this demanding task would require several chapters on each own, the reader is redirected to the excellent existing bibliography on each field.

1.3.1 Computer Vision

with as few words as possible, computer vision is a field of expertise, where a system takes an image as an input and tries by using principles from the sciences of physics, anthropology and other sciences to explain and extract useful information. It is a science that attempts to extract, analyze and explain from 2-D imagery the 3-D real world, thus an inverse incomplete problem where complex mathematics and physics come into play. A well known adage is

”a picture is worth thousand words”

while the saying roots vanish in history, its meaning cannot be truer when it comes in computer vision where the information extracted from a simple can be used to detect objects and describe the image Lastly in the authors humble opinion two cornerstone books that interested readers should study are Gonzalez and Woods and the also excellent work of prof <http://szeliski.org/Book/>.

1.3.1.1 Stereo Vision

Intro The field of Stereo Vision is a sub field of Computer Vision, its main goal is to reconstruct the third dimension from two 2-D images, thus providing the distance information of a printed scene. This is attained by using the disparity theory where depth information can be found by comparing the horizontal location of an object from two different pictures taken with a specific technique to create a stereoscopic pair.

Stereo Vision is used in many applications, especially those that require the 3D dimension to properly operate such as unmanned vehicles where the system must avoid colliding into obstacles, though recently stereo vision is used from augmented reality devices such as smart tv's to detect hand gestures, lastly the extracted 3D dimension can be used to reconstruct the real world using stereoscopic pair of images thus it also finds use in medicine, archeology and any science that attempts to reconstruct a 3D object from 2D images.

Epipolar Geometry while not an actual geometry theory in the sense of Eukledian or other theories, is the subset of rules and constraints chosen to perform the task of stereo correspondence. There are several constraints that must hold true before one can use Epipolar Geometry to calculate the depth information of an image firstly the pictures of the scene must be taken by using cameras that satisfy the pinhole model (in practice all cameras do) then both camera sensors must be identical or rectified so that their corresponding image matrices are referring to the same basis system, lastly noise due to physical flaws or deficiencies must be corrected amongst which is the barrel vision effect where the camera stretches the image near the edges of the 2-D square image plane.

Once these constraints hold true then this theory can be applied and by using the following procedure on a stereoscopic image pair one can recreate with acceptable accuracy the distance of each point from the camera system, thus recreating the lost 3D information. Figures 1.2 and 1.3 show the epipolar geometry at work where the same object is shown in different corresponding location according to the point of view, it is precisely this displacement that that provides us with the depth information. The displacement between corresponding point, namely the difference between the location of appearance

of a point in the left and right image, is called a disparity and the image that contains all the disparities of a stereo image pair is called a *disparity map*, once we have the disparity map we can get the actual depth map by multiplying each disparity with the depth ratio which depends on the lens of the camera and the distance between the two camera center points which is also referred to as the *baseline*, the equation is $Z_i = \frac{B * F}{D_i}$ where $D_i = X_{i_L} - X_{i_R}$ where i is a pixel belonging to the left image, the generated image is called a depth map. While there are many different approaches to calculate disparities we will refer to the correspondence matching where as described above we try to locate an object in the image pair. At this point we must select an image as our basis image from the image pair this can be either the left image or the right image and in bibliography both are used, we will use the left image to select a point and then attempt to find its correspondent on the right image. This is an exhaustive problem and if attempted as such the processing it requires renders it unusable as each pixel must be tested for correspondence with each pixel from the left image luckily some constraints can be used that drastically limit the range of search in a point where we can achieve even real time performance on very large images these constraints are the Epipolar and Monotonicity constraints.

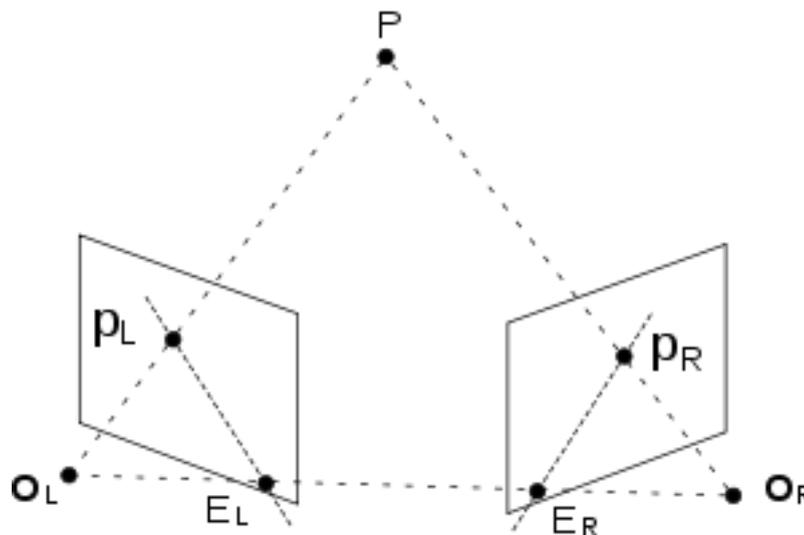


FIGURE 1.2: Epipolar Geometry and pinhole
Epipolar Geometry Model

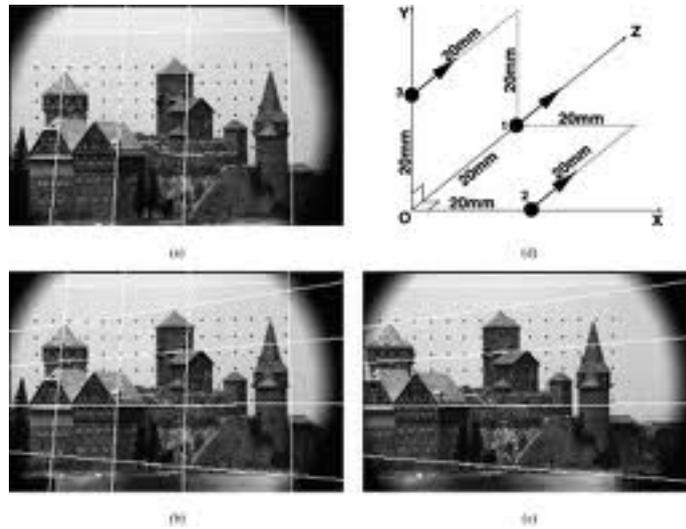


FIGURE 1.3: Church Image Pair
Stereo pair of an actual image, horizontal displacement is evident.

Epipolar Constraint As we mentioned before searching exhaustively for the match of each pixel of an image is prohibitively expensive as even the smallest images contain hundreds of thousands of pixels one imagine the vast number of calculations this easily propagates. However the epipolar constraint can drastically limit the search range for a corresponding pixel by denoting that a corresponding pixel can only occur on the same horizontal line as our target pixel (target is the pixel in the left image and corresponding is the match we try to find on the right image) this changes the problem to a 1-D search problem since we only have to search one line of pixels in the target image. In order to use this constraint however our images must refer to the same basis system and must be rectified vertically so as the only displacement present is the one in the horizontal dimension, this means that we also have to correct for the Barrel vision effect. With this constraint in effect the calculation cost of the stereo correspondence problem is low enough to attain real time calculation on stereo image pairs.

Monotonicity and Uniqueness Constraint Another important constraint that can be used to further speed up the search for the corresponding match is the so called *Monotonicity Constraint*, this constraint states with simple words, that if we find a point X_{i-1_R} on the right image that corresponds to a target X_{i-1_L} point on the left image

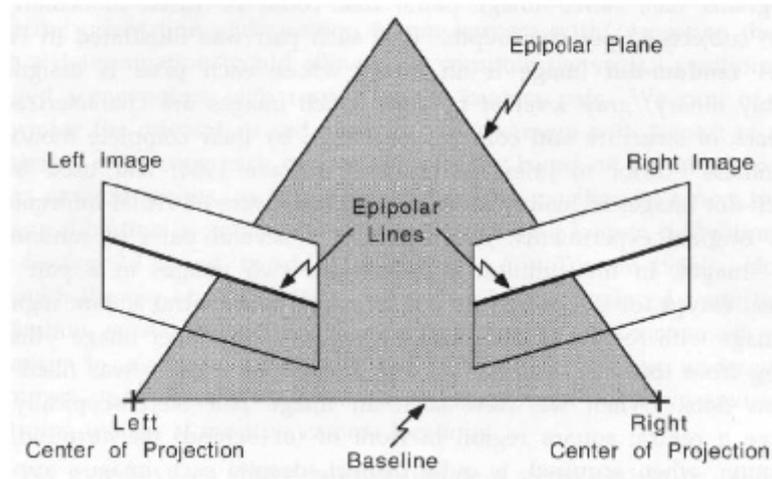


FIGURE 1.4: Epipolar Line

When a point appears on the left image then its corresponding match can only appear on the epipolar line on the right image.

then the subsequent point X_{iL} if it is after the point X_{i-1L} (on the horizontal axis) then it can *only* have a match on the right image in a position *after* the previously found point X_{i-1R} , lastly the uniqueness constraint states that each point can only correspond to one match. In order to get a feeling of how drastically this constraint helps, think of a line of pixels of an average sized image has 640 pixels (if the image is $480 * 640$) on the horizontal axis, this is the image also known as VGA, in order to fully match all points existing on the target frame with the corresponding frame we would thus need $640 * 640 = 409600$ operations by using the monotonicity and uniqueness constraint we would get 640 operations for the first pixel, 639 operations for the second so on and so forth down to 1 operation for the 639 pixel of the left image this is an arithmetic progress and each sum can be calculated by Gauss formula $\sum i = 1640 \frac{(P_1 + P_{640}) * 640}{2} = 102560$ operations since each VGA image has 480 lines that would mean a total reduction of $102560 * 480 = 49228800$ operations, this means that by using the monotonicity and uniqueness constraints we calculated the entire disparity map in approximately $\frac{1}{4}$ of the time previously required.

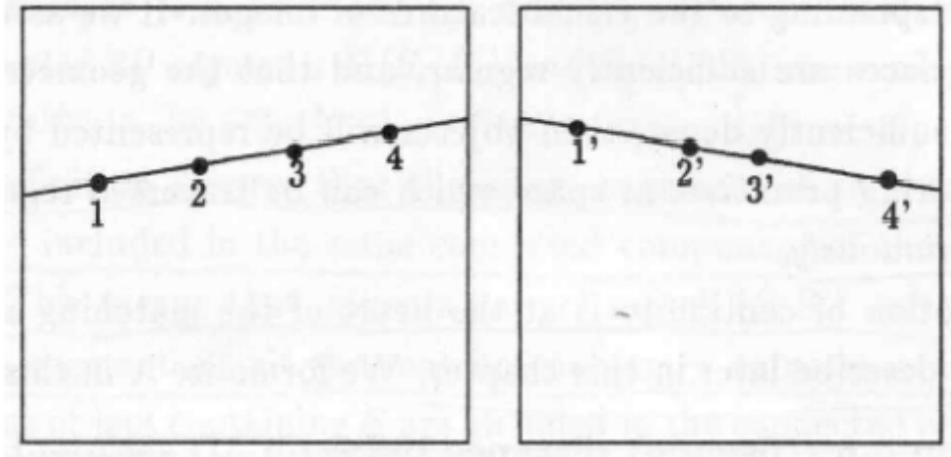


FIGURE 1.5: Monotonicity and uniqueness constraints

Subsequent matches can appear only sequentially on the corresponding frame and can only match one target.

1.3.2 Embedded Systems

A specialized computer system that is part of a larger system or machine. Typically, an embedded system is housed on a single microprocessor board with the programs stored in ROM. Virtually all appliances that have a digital interface – watches, microwaves, VCRs, cars – utilize embedded systems. Some embedded systems include an operating system, but many are so specialized that the entire logic can be implemented as a single program.

³ This rather simple definition describes but the simplest embedded devices, Raj Kamal in his excellent book for embedded systems more completely describes such devices as

An embedded system is one that has computer-hardware with software embedded in as its most important component.

this definition in our opinion more correctly describes an embedded system and at the same time is allowed to scale with time as new devices come to be with unknown hardware at the time of this writing as this was true for older devices before the time that RS232 or Ethernet, blue tooth or wifi or camera sensors found their way to the embedded devices, software was and will be the key hardware component in an embedded systems heart. This paradox means that in an embedded system software is written to be so closely intimate with the underlying hardware that almost behaves as a hardware controller and while the traditional language to write such software was no other than the assembly language, advances to compiler technology and specialized tools allow the programmer to write almost as efficient code using higher level languages such as C or

³Definition taken from http://www.webopedia.com/TERM/E/embedded_system.html

VHDL. A word of note to keep from this introduction to embedded systems is that high performance software that runs on an embedded device is closely tied to every aspect of the underlying hardware and almost always requires heavy modifications to be ported to another device as application logic is closely related to the very microcode that controls the devices electronics.

1.3.2.1 Real Time

is a constraint that most embedded devices are required to hold true to, it is the constraint that denotes system behavior and its meaning is more apparent by the alternate name of this constraint which is *Reactive Systems* as it is evident it implies a systems response to events and interrupts in such a way that the initiator does not wait more than a specified amount of time, the reflexes of a human being are a good metaphor to the real time systems an action causes an immediate reaction, the feel of pain or discomfort to ones arm causes immediate retraction from the source of such discomfort with no delay. Of course there are various levels of real time responses the aforementioned example is more closely described from the so known *Hard* real time constraints where the corresponding actions cause an immediate reaction with no delay, with a miss constituting a complete system failure, softer constraints are the *Firm* and the *Soft* constraints. By their nature most if not all embedded devices fall under the Soft constraint which means that all embedded systems are required to respond within a range of time that is deemed acceptable in each application while exceeding these boundaries usually renders them useless.

Devices and Processors

In the universe of embedded devices exist many platforms and in contrast with the world of personal computers the choices an embedded engineer has to make in order to select a platform as the underlying hardware base for his application are quite a few each with its own pros and cons, we will attempt a brief description for each of the main families in the following paragraphs.

FPGA Field programmable gate arrays is a technology that with a few words aims to achieve hardware level performance when executing algorithms while avoiding the cost inherited with designing and implementing a microelectronic circuit. The exact way it achieves such a feat are beyond the scope of this Thesis, in short it formats an interconnected array of logical gates in such a way where the output of a logical function given an input matches the ground truth table of the implemented algorithm.

Commonly implementation of these systems is done in the VHDL language, while their major asset is their hardware level performance(albeit quite slower than their ASIC counterparts and typically less power efficient) with just software effort their down side is that reprogramming must take place on a special programmer device (typically) and can only perform the task they were programmed to do.

DSP Digital Signal Processors are specialized processors that contain typically more hardware resourses for addition and multiplication thus achieving higher throughput to signal analysis where the same operation must be done to a vast number of data. These devices while offering much higher throughput than a general purpose processor when SIMD calculations are required, they typically fall behind in performance to FPGA solutions their major strength is that they can be reprogrammed away from the lab and require no special hardware as all the application program in written in software and requires no internal structural change as is the case of FPGA's, this allows a good trade off for most applications between performance and robustness furthermore their power consumption is quite low typically lower than corresponding FPGA but higher than ASIC solutions.

GPP General Purpose Processors do not exist per Se in the embedded world while Intel has attempted to enter the embedded world by using the Atom processor that is a general purpose processor its power consumption is at least one order of magnitude higher than other embedded systems. Most general purpose processors are based on the ARM IP, while architectures from other Intellectual Providers also exist, in short these processors are able to run programs in a similar if not identical way to personal computers and their hardware follows a similar path typically one instruction is applied to a set of one or two operants per cycle commonly referred to as SISD model. Strengths of this platform are the easy access to software already compiled to run on those processors(since no special programming is needed other than to use an appropriate compiler) thus implementation to these platforms is easy and straight forward one simply uses a compiler that generates an output executable that runs on the target architecture, also commonly these processors are quite cheap since they target a higher market share, lastly power consumption is higher than all the aforementioned technologies and commonly one must have access to additional hardware in order to use them such as I/O controllers and memory. We must note here that while the micro controller family is a very important part of the embedded world we skip it here since they cannot cope with the goal of the current thesis stereo map calculation.

ASIC application-specific integrated circuit, is as its name implied customized to be used in one specific application and cannot be reprogrammed or reassigned there are a host of different technologies to create ASIC's as the idea begun approximately in 1980's, a common denominator is the absence of reprogramming and the narrowness of the scope of execution which typically limits the ASIC's to perform exactly the task it was created for. Strengths of the ASIC is its almost undisputed performance as nothing sort of a pure electronic solution such as a VLSI circuit can match its speed, while recent advancements in FPGA technology have certainly shortened the gap between them, ASIC retains the first place in performance and power consumption, on the down side it cannot be reprogrammed as an FPGA can so bugs and updates cannot take place save for those ASICs that have a microcode controller or BIOS and even then the updates are extremely limited when compared with FPGA or the more versatile DSP and GPP systems. Lastly the cost of creating such a system is higher by far than any of the aforementioned systems since it commonly requires custom hardware or lengthy design and implementation which is by far more expensive than all previous systems.

1.3.3 Summary

In this section we attempted to present the goal of this Thesis and provide the reader with at least the basic background and notions that appear in this work. While the fields that this work dabbles with are extremely complex a detailed description of either computer vision or the embedded world was unrealistic, however the principles and notions presented can be used as a seed for the reader to further research and study on the excellent and numerous bibliography on either field. In the chapters to come we will present:

Chapter 2 In chapter 2 we will show the relevant work in our field, discuss pros and cons and based on this discussion present our novel approach to the problem.

Chapter 3 has a detailed description of our implementation, with detailed steps of development and the reason behind each engineering choice.

Chapter 4 contains the results and their analysis.

Chapter 5 has conclusions and future work.

Appendices lastly contain informations about the tools used as well as fragments from the codes and configurations for the DSP.

Chapter 2

Architecture

2.1 Relative Work

While the problem of stereo vision is studied from the early 70's, and publications can be found with algorithms and performances from around that time(put reference here), it was not until the mid 90's that the first embedded platforms appeared[1, 2] .

Due to the technological limits at the time embedded systems that performed stereo vision, developed slowly but steadily from FPGA solutions that could perform stereo correspondence on small resolution systems to ASIC and later with intel ATOM and M processors; while the solutions that used DSP's where sparce and few within, a very important survey for relevant work until 2001 can be found in the excellent work of [3], another survey that sums up some results is [4].

Since the focus of this thesis is on real time stereo vision performed on embedded systems, we will focus on relevant work that presents results on those systems.

2.1.1 Embedded Universe

As we narrow our view to the embedded world we find three major contenders, FPGA[2, 5-7] based solutions are by far the most common, followed by Intel's initial portable cpu Atom, followed by DSP[8-10] standalone systems and a handful of ASIC[11, 12] platforms, lastly a new wave of heterogeneous devices that appeared recently have begun to be studied and show promising and extremely interesting results[13]..

2.1.2 FPGA based

The vast majority of the systems developed utilize FPGA's and provide excellent results, amongst them even some of the earliest work in this problem was developed as a solution to FPGA systems i.e the system named PARTS [14] using 16 FPGA could generate the outstanding for the time being 42 FPS at QVGA quality even earlier the pioneering work of Olivier Fauger[2] as et all on 1993 was able to produce stereo images of 256*256 at 8FPS using matrices of 4 by 4 networked FPGA. One of the most cited, recent papers, concerning FPGA based implementation is the work of S. Jin et al. [15] where in their work state that they have developed an FPGA solution that may perform as good as 30 or 60FPS on VGA images while operating on two different timings, while their results are impressive the window size of 15 is considered generally high and many other authors provide results for window sizes of 7 or 5, in their own work however they have developed their algorithm on a general processor as well to compare results and provide that on their 3.2Ghz Pentium 4? with SSE optimizations a FPS of 1.1 is achieved. Other systems namely the work of Masrani and MacLean [5] report 30 FPS but at double disparity levels (128) , in the work of Darabiha[6] et all an array of 4 FPGA is used to calculate sub QVGA stereo image at 20 FPS and with 20 disparity levels lastly in the work of Jia et all we see for VGA images approximately 30FPS using 64 disparity levels, it should be noted here that the usual window used in these works is 15 and in some cases it is not reported, however since the impact of window size in performance is significant in cases where it is not reported it is difficult to estimate the power of the implementation.

To sum up, performance is directly correlated with the size of the FPGA unit, furthermore FPS alone should not be the only metric since the window size, disparity levels and correlation functions must be considered. More recent systems such as the work of [16] report even better performance with newer Xilinx Virtex 5 where they can process more than 87fps in HD content, where for VGA the authors report more than 590 Fps. . A low cost implementation of[16]is [17] where a complete system is implemented.

2.1.3 ASIC

Since the cost of constructing an ASIC custom board is significant and prototyping is orders of magnitude slower than for the FPGA systems, the prior systems using ASIC's are significantly fewer; two of the most cited works in this field are the SAZAN system propose in the work of Kimura [11] that is able to produce QVGA stereo maps with 20FPS on 20 Disparity levels, also Woodfill et al developed the "The Tyzx DeepSea G2 Vision System" [12] based on the homonymous Deapsea processor that is able to produce

as much as 200FPS on 64 disparity levels. Lastly an inovative work performed in the aerospace field uses a novel architecture aimed for low power operation is introduced in the work of Diotalevi[18] et al that produces up to 25FPS at 348*288 image size with 16 disparity levels consuming approximately 75mW using the IntellaSys S40C18 unit.

2.1.4 Atom

As this unit is more like an underpowered x86 cpu, and operating systems using this unit are general purpuse results from works that used this unit as an embedded real time solution provided unacceptable results, at least for the real time constraint, as they failed to provide good quality depth map at an acceptable frame rate 0.1 to 2 FPS as reported from [4], most systems using this solution simply perform stereo calculation offboard in a base station system [19] .

2.1.5 DSP

DSP systems are an attractive candidate for the stereo problem since they provide parallel computation and can do so without the need of hardware level programming as FPGA's requiring just software level programming. One of the most used units in this field is Texas Instruments TMS320DSP family, roughly arround since 1983 though the TMS32010 model it has grown to a powerful and stable platform that according to the model can perform either fixed point or floating point arithmetic.

It is of no wonder as such, that almost all available work in this field is done using some model of this family, in the work of Nelson Chang et all [8] a DSP based solution is proposed using the TMS320C6414T-1000 model that can perform up to 50FPS on 16 disparity levels at above QVGA quality (384*288) using their proposed jigsaw template and the SAD correlation function. More recent work in the field are the two robot catcher systems of Lin and Chiew[8-10] that can perform real time stereo calculation and are able to catch items thrown at them from the distance of 4meters with 65% accuracy, again these systems utilize THMS320C64 dsp.

2.1.6 Heterogeneous Systems

A new highly promising option are Heterogeneous systems, systems that provide 2 or more processing units of different architecture that can use the best of multiple worlds, such as providing a general purpuse operating system such as Linux and yet having highly specialized capabilities such as a OpenCL enabled GPU or an in chip specialized hardware such as a DSP unit or a FPGA.

Due to the short history of these systems, prior work in these is limited, the only system that implements stereo vision on Heterogeneous system the author was able to find was in the excellent work of Goldman and Matthies [13] , where the authors used a GumstixFire unit with OMAP3530 SoC. Texas Instrument's OMAP3530 unit contains multiple operating units namely a GPU, CPU and a DSP system of the TMS320 family. Using a distributed processing model the authors report a FPS of 46 on QVGA images or 8 on VGA while at the same time using the general purpose processor to run independent feature tracking algorithms.

2.1.7 Summary

The aforementioned systems are by no means exhaustive as the work in this field is immense, it would require nothing sort of a survey for each category (FPGA, ASIC, DSP) to provide an exhaustive and complete state of the art, however the systems mentioned are amongst the most cited the author managed to find. Alas the only survey available, the work of [3] while excellent is a bit outdated and is not focused on embedded systems. Lastly another important notion the we must keep in mind are the metrics used to compare methods, the most common metric found in all publications, at least in the real time field, is the FPS without a sufficient frame rate any solution is not acceptable for real time applications, another commonly used metric is disparities per second commonly abbreviated as MDS (million disparities per second) also a useful metric especially in publications where power consumption is important such as [13] and [18] metrics that provide a view on consumption appear such as frame per joule, watt per frame or even disparities per joule. Finally a common metric for all stereo vision based works is the use of true disparities to error disparities ratio and the handling of non correspondences and occlusions, although for embedded applications where usually less accuracy is required (at least on pixel level), this metric is usually overlooked in favor of the aforementioned fps and energy metrics.

2.2 Our Approach

Considering the minimum requirements presented, the real time constraint and the demand that a variety of computer vision algorithms should be able to run in parallel we chose a heterogeneous system as the development platform that can adhere to these demands. More specifically we chose the OMAP3530 S.o.C in the gumstix implementation.

Let us justify the reason behind this choice, an A.S.I.C system is only available if made by a large company and even then the interoperability with another system is troublesome at best, an FPGA has a varying price from affordable to expensive yet again it suffers from interoperability issues and even though it can be programmed it is not feasible to do this outside of a lab, a DSP system alone would be sufficient to process the data yet it would suffer greatly to perform more than one operation, and lastly a micro controller alone would not be able to perform the required Stereo Vision task in real time.

A combination of the above systems however as presented before would combine the strengths of two (or more) worlds to solve the problem. From the presented heterogeneous systems 1 we chose the GPP+DSP for three main reasons, firstly the cost is lower, secondly the DSP can run in software any algorithm while the FPGA must be reprogrammed and lastly the community supporting the GPP+DSP platform was open source and with better support an important notion. Of the potential candidates we chose the gumstix implementation a powerful complete system on a chip that houses a GPP a DSP and a separate GPU amongst other components. This setup allows heavy processing tasks to run on the DSP system while allowing the GPU for graphic support and the GPP to run the operating system and any additional algorithms. The DSP system communicates with the GPP through a special driver called DSP-Link which is how all interprocessor communication takes place.

2.2.1 Initial Design

To fully exploit all benefits of this powerful system a modular architecture was designed where each component was assigned the task that can handle best, the GPP will be charged with running the O.S. control the cameras and perform any other task such as networking and general algorithms according to each application such as custom feature detection algorithms in the context of computer vision; The DSP unit will be tasked with running highly parallelized and heavy processing parts of the algorithm where in this context is the correspondence matching but in other applications a similarly heavy operation should be placed in the DSP, lastly the GPU will handle the graphics display

if it exists. Thus as shown in 2.1 a pipeline of data is formed starting from the CMOS sensors passing through the GPP to the DSP and back.

This design is chosen in belief that this processing chain will vastly accelerate the requested operation achieving the real time performance we require this hope is due to the technical specifications of the DSP system B.

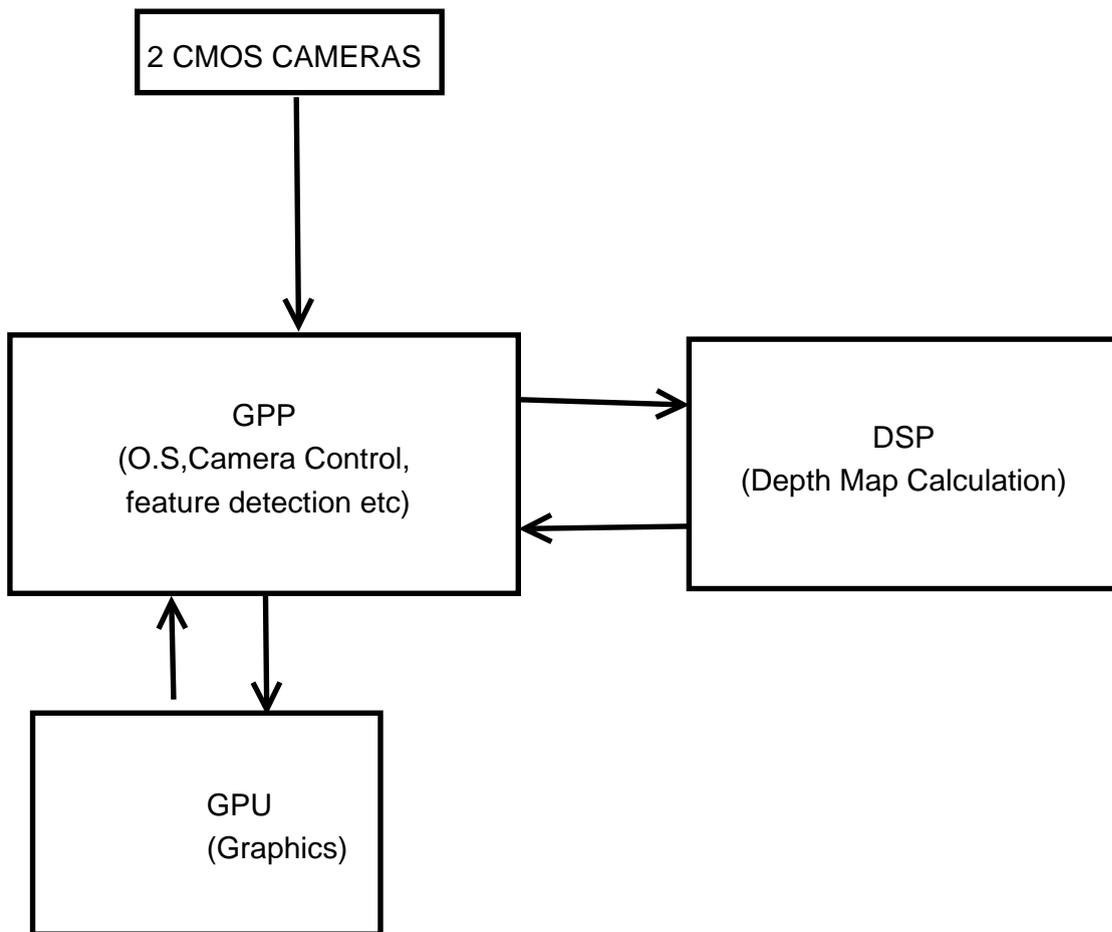


FIGURE 2.1: Data Flow .

2.2.2 Assumptions and Theory boosts

A very important step when trying to optimize a problem especially in resource limited systems with real time constraints is which assumptions can be made using mathematics or theory, where without sacrificing a lot of accuracy we can get significant speedups. In our problem the fastest stereo correspondence algorithm is the SAD while the profile shape[add reference here] algorithm proposed by Trippets et al, is faster its loss of accuracy in some applications may not be acceptable but the main reason we did not chose this algorithm over SAD is not accuracy it is rather the lack of implementations of

this algorithm in the field depriving us from the second major goal of this thesis which is the comparison of the heterogeneous system performance versus other approaches when using process heavy applications. Two major theoretical assumptions that can be employed for stereo vision are also the monotonicity constraint and the epipolar constraint.

Epipolar Constraint This constraint can be applied to all rectified stereo pair images and it means that a point x_{ij_l} where i = rows and j =columns cannot have a corresponding point x_{ij_r} in any other line than i , in other words it means that in a rectified stereo pair if we put the images side by side and try to find for a given point in the left image its match in the right image then if we draw two parallel lines one on the bottom edge and one on the top edge of the image pairs then the match has to be on a line passing through the interesting point and is parallel to the other two see figure 2.2. This important notion limits the search for the corresponding point to only one line instead of scanning the entire image without this constraint real time stereo vision might not be feasible. As a reminder at this point the line where two matching points exist (on the left and right image) is called an *Epipolar Line*.

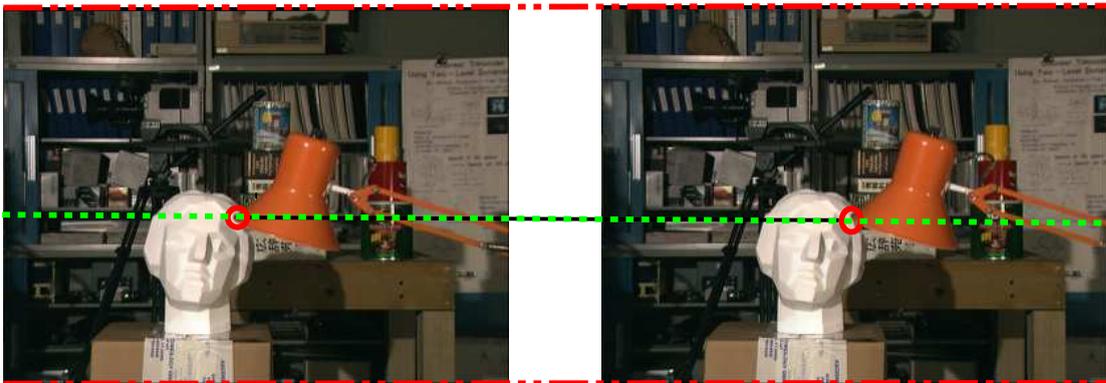


FIGURE 2.2: the dotted green line is the convex where corresponding matches are allowed to be.

Monotonicity Constraint This constraint first used by Davi Geige[ref], ChaMei[91 ref] further limits the required processing vastly limiting the convex where the match might be found by suggesting that not only such a match must be on the Epipolar Line but it must be in such a position on the Epipolar Line where previous points matched points cannot occur again; In stricter terms for i, j where i =rows and j =columns of an image array where I_l the left image of a stereo pair and I_r the right image if a point $x_i(j-1)_l$ has a corresponding match in the position $x_i(j-1+d)_r$ where d is the displacement, then the point x_{ij_l} cannot have a match to any point in n where $n = [0...j-1]$. See figure 2.3

This simple constraint is an assumption that is true in most cases for more information on when it does not hold please seek additional info in the aforementioned bibliography. Without this constraint in order to match a point in the epipolar line for an image of $W \times H$ pixels where W is the width of the image one would need W operations for each pixel (we use the notion operation to denote it the abstract process of correspondence matching be it SAD, SAS or any other of the aforementioned methods), that means $W \times O$ for each pixel thus for the entire line we would need $M = W * W * O$ operations and for the entire image a total of $M * H$ where using the monotonicity constraint the number of operations per epipolar line is $N(o) = \sum_{i=0}^W i^2 = \frac{n*(W-ds_i+1)}{2}$ in the worst case where $ds(i-1)$ with ds_i is 1 and ds_i is the distance of the matching element in the epipolar line from the start of the row. This constraint lowers the time required to find a correspondence to the sum of an arithmetic progress the difference between this and the W^2 which is the cost of not using this constraint greatly increases as W increases so for large images the use of this constraint is almost mandatory.

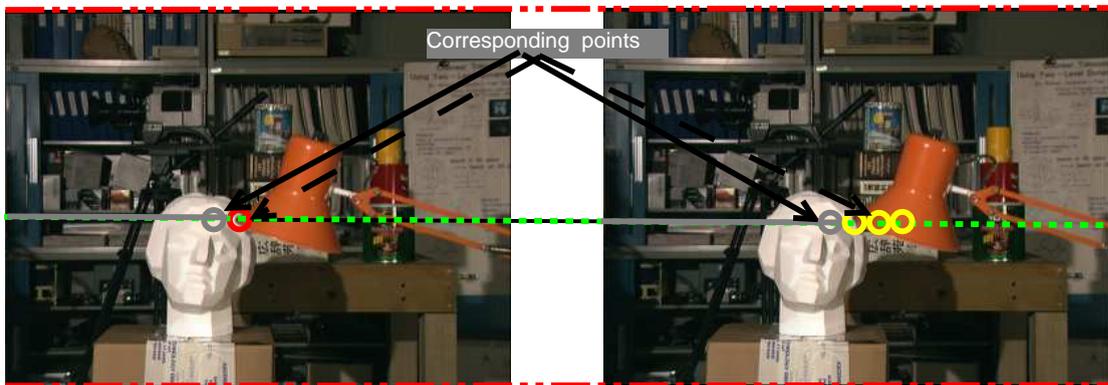


FIGURE 2.3: Grayed out part is the skipped part.

2.2.3 GPP Side

As our system is heterogeneous we are able to use each part to attain the best of both worlds, in the case of GPP¹ side our assets are immediate access to the Linux O/S ease of programming and access to the Open CV library along with a huge variety of ready to use frameworks such as the invaluable gstreamer API, this allows us to interconnect our application with all this applications to provide rapid development prototypes and applications that integrate easily with the work of others.

For our current application we used the Open CV framework in order to grab the image frames from the device nodes² perform image rectification and turn them to

¹General Purpose Processor

²CMOS Camera's

gray-scale and once the depth map is generated and brought back from the DSP side the visualization on the device is done using the same framework. The gstreamer API allows us to wrap up the entire application as a gstreamer plug in that can be used as part of a gstreamer pipeline ready to be used as a subcomponent even by simple users. Lastly any interoperability using the network is done using the gstreamer pipeline itself either broadcasting using UDP or in case of a more secure and reliable communication using the TCP plug in.

2.2.4 DSP Side

This is the part where all the processing intensive computations take place, the slave application that runs in the DSP³ is tasked with receiving a bundle of frames (left and right) and computing the depth map, upon completion it signals the GPP side application and transmits the depth map back, as memory copy operations are not negligible⁴ the DSP side transmits only the depth map without copying back the received left and right frames, for debug reasons you may override this behavior and transmit the source frames as well (see [A](#)).

The entire work of the stereo vision algorithm will be here finding the stereo correspondences, plane fitting and depth map generation all take place in this processor in contrast with other relevant work.

2.2.5 Assembling the Puzzle

Having studied the relative work done, provide us with insight on many pitfalls however the uniqueness of the current system required extensive analysis on top of that. Having identified the processing intensive parts of the task at hand, at least from a distant point of view , allows as to depict the profile of our target system in a few words it must be distributed and modular, it must be able to provide fast development environment to quickly incorporate computer vision algorithms and the software already available from libraries such as OpenCV, yet it must be able to process a consuming algorithm in the DSP in a seamless manner this leads us to create dedicated module that runs on the dsp with the sole purpose of performing the stereo vision task and nothing else while the, now , idle GPP is left with all its resources available save the few allocated for the interprocessor communication in its disposal to perform feature detection and any other task required while the DSP power is hidden behind a wrapper function that appears to the host program as an external C function. The loose coupling of

³Digital Signal Processor

⁴See Results Chapter

the components that operate in the GPP and DSP systems provide with the unique flexibility of updating each code independently as long as we adhere to the interfacing code that migrates data between the processors, and this is the third software module developed, the glue software that connects this modular system together.

In the following figure 2.4 we see a detailed flowchart of how the software system runs. The host processor ARM cortex A-8 initializes the client DSP application and the camera systems, it should be noted that the current camera systems are one usb camera and the e-con camera module where it resides on the special camera interface provided by the SoC, this limits the power consumption to a minimum. Once the camera nodes are configured they are synchronized through software by driving the picture capturing from two synchronized threads, the constraint in this case is relaxed real time which means that we have a somewhat narrow but not absolute time frame where the two snapshots must occur. In the second phase and once the DSP Bios system has booted our client application, the GPP side server app begins feeding the DSP client application with stereo pairs through the interprocessor channel, data serialization provides a robust and loose way to transfer data that will make the entire system easily portable and modular. Lastly when the DSP client application receives the stereo pair input it begins processing and once the disparity map is calculated it serializes the result in the buffer and transmits it back to the GPP side, while the processing is done on DSP the GPP host application carries on with feature detection or any other task required it should be noted that in case feature detection is enabled the user always sees the outcome of the previous frame to allow for the DSP to process the current frame. In the end when the user sends the TERM signal the process gracefully exits and before doing so it gracefully shuts down the DSP sub system as well.

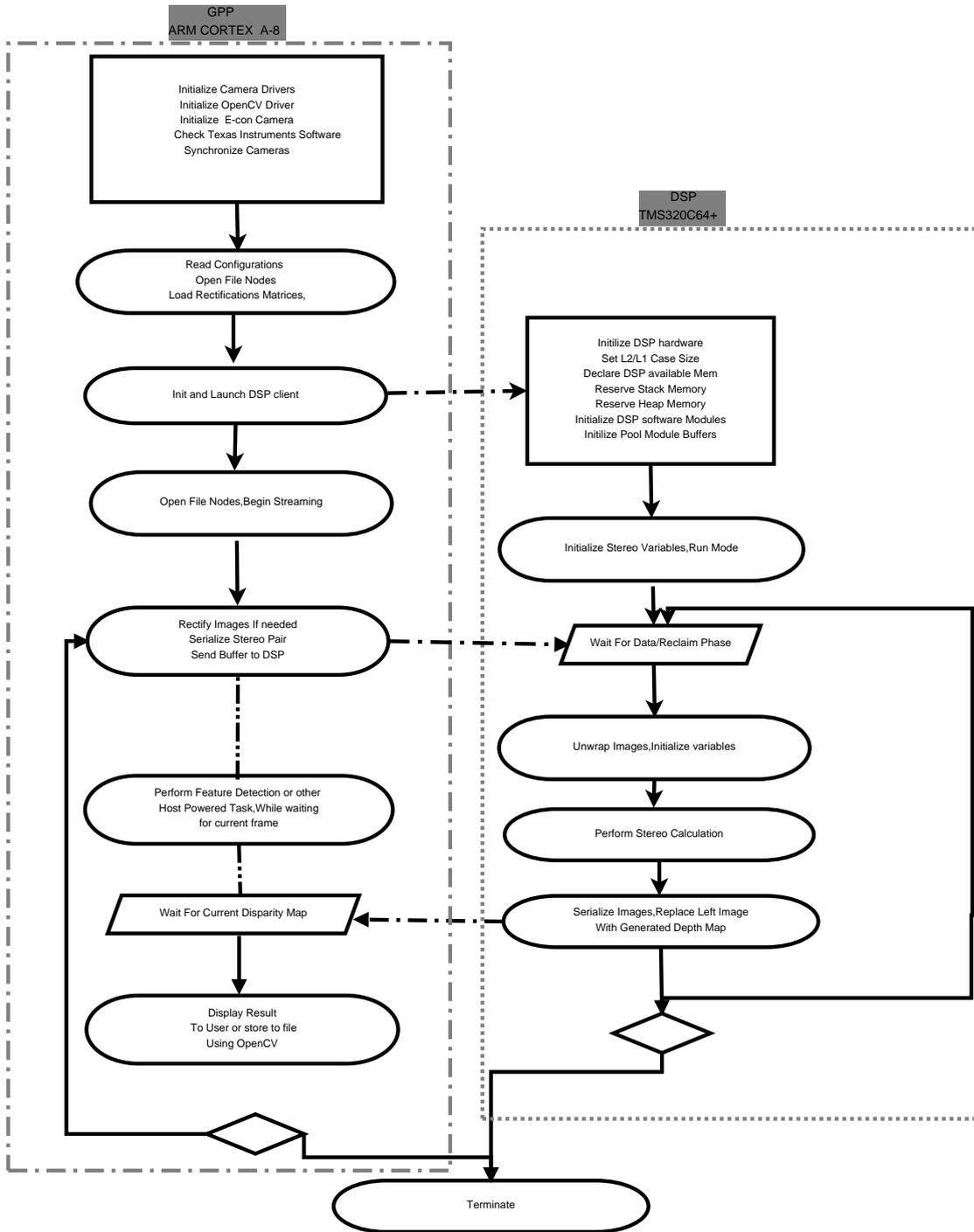


FIGURE 2.4: Processing Flowchart.

Chapter 3

Implementation

3.1 Introduction

Having done the draft overview of what processing unit will handle which task two independent but closely related main components have been identified the GPP side and the DSP side components. Since it is a rather complicated system with various components the implementation was top down , in the sense that we started from the least complicated but easiest to implement technologies and by confirming proper behavior and results drilled down to more complicated, more difficult to debug levels, but of course of much higher performance. Following is an overview of the development performed for both parts more detailed information about the tools and more can be found to [B](#).

3.1.1 GPP Implementation Overview

The development of the GPP side component since it is much easier to develop and test using the tools supplied by the Open Embedded community (which are no other than the GCC and GDB) can take place both natively on the Gumstix platform or we can start by writing the same program for a x86 architecture, verify its correctness and then cross compile to the armv7a architecture.

Both approaches are valid and the pros of one are the cons of the other, in a nutshell those are :

Native Style Developing code on the target machine directly.

- Writes code directly to the target platform, encounter any problems on the go and solve them without building on erroneous assumptions.

- Code depends only on available frameworks and API's that are already cross compiled.
- Easier to test application in intermediate states and visualize real results.

Build Host Style Developing code on a different machine than the target machine, usually with different architectures(creating a program on a x86 architecture machine to run on a RISC machine).

- Access to better IDE's or to IDE's programmers are more accustomed to.
- More powerful machines allow for faster compilation and better responsiveness.
- Programming for general architectures such as x86 provides access to the entire universe of software tools that might help debugging and prototyping leading to rapid development .
- In contrast to Native development the programmer is not connected via SSH or some console (eg USB) that might occasionally degrade or crash or become unresponsive due to a code bug.

The author chose Build Host style for the GPP side of the system, tried to keep third party libraries and frameworks at a minimum and when it was mandatory we took care to first verify availability of the library in question to the end system before building code on top of it, as was the case with Open CV and the gstreamer framework. Once the GPP side application was up and running and we where satisfied with robustness and performance the cross compilation procedure begun and ended without problems since each and every part was known to already be available at the time we chose it.

3.1.2 DSP Implementation Overview

The development of the application for the DSP side has a major difference than that of the GPP side, the Build Host approach is not available, at least not without using specialized software such as an emulator. The reason is simple, DSP's are SIMD processors where x86 with a few exceptions (specific operations) are SISD further more the compilers used to generate an executable that runs on the DSP are not available for x86 since each company that provides DSP systems has its own compiler; Moreover in the case of heterogeneous systems the DSP does not have access to the Linux operating system and is not managed by it directly, especially in the case of the OMAP3530 that we used, a special BIOS O/S is used to control and operate on the DSP in conjunction with specialized software that runs on the GPP side.

The above reasons are the most significant ones while not exhausting all of them, it is

clear we cannot use the approach of the GPP side application. Thus a hybrid model was used the DSP side application was divided into two main parts based on their functionality in the big picture. One part must perform the actual computation and generate the depth map and one part can be tasked with handling the entire communication between the GPP side and launching our algorithm, ensuring proper run and communicating the result back in the GPP side and the Linux O/S. In principle these two parts can be loosely connected and one can change the entire stereo vision algorithm with a completely different algorithm with only minimal changes to the other part, probably only the input and output buffers and maybe some control signals to reflect the requirements of the new algorithm.

It is evident this way that one of the parts is heavily based on the internals on the DSP BIOS system using the RTOS functions and conforming to custom standards, while the execution unit (the stereo vision part) only receives an input buffer and performs on it taking into account the architectural resources of the DSP but from a programmers perspective is just an algorithm that has access to some extra resources.

By noting the loose interoperability of the above design we prototyped the execution unit of the stereo vision algorithm using the MatLab system, this gave us the power of testing rapidly many different theoretical subcomponents for the algorithm as is for example the correspondence kernel we tested all SAD, SAS, DAS, SSD benchmarked its performance in the grand scale before choosing the best kernel for our requirements, which is SAD.

On the other hand the communication part was developed natively on the gumstix system, for all the reasons described in the first paragraph of this section, it might help the reader at this point to read the appropriate section in Appendix B, where we describe the operation and development of this communication unit with great detail see [AppendixB](#).

3.2 GPP

The GPP side is tasked with operating the CMOS sensors grabbing the stereo pair images performing any rectification necessary and communicating the image pair on the DSP side; That is by no means however the only tasks the GPP is running amongst them are the actual O/S and a host of processes required for networking and more applications such as additional feature detection and classification software that runs in parallel or in series with the depth map result.

Our implementation however is limited to the piece of software that performs two discrete tasks :

- Control and communicate data with DSP side.
- Operate camera sensors and display results to user whether on an on screen module or through network.

For the first part we used the LOOP GPP side application provided by Texas Instruments; Using this program as a basis we developed a GPP side server that launches the DSP side application and is responsible for communicating interprocessor data.

The second module of the GPP side application is the part that handles the camera sensors grabs the stereo pair, rectifies, smooths and clears the images and bundles them together in a byte buffer to be passed to the communicator so that it can be copied to the DSP side. After the DSP finishes the depth map generation it sends back the depth map frame using the same buffer that the initial source frames were transmitted, it should be noted at this point that the memory copy practice used is the Issue-Reclaim model (see App. A).

Finally the depth map is copied from the issue buffer to the memory space of the GPP application for further use and the issue buffer is available to receive the next frame and repeat the entire procedure. Once the depth map frame is received the user space application performs any additional tasks that required the depth map as for example depth estimation to avoid an object or feature detection while using the stereo information; the additional tasks depend on the actual user application, however it should be timed so that the user space application does not take more time than the time required by the DSP to process and copy one frame otherwise performance will suffer and we will have a lower frame rate due to the inability of the GPP to cope with the frame rate of the DSP.

Visualization is the final part of the GPP application where the result is presented to the user, in our SoC there is a GPU available to display using Qt or any other framework available to the OE distribution to graphically display the results on an on-board screen or transmit the result to a work station using either TCP/IP or UDP or even Bluetooth. Lastly the program passes some configuration parameters to control the DSP algorithm such as the correspondence window dimension $N * N$ and the maximum disparity levels D ; also there is an option to ask the DSP to try to maintain real time performance(15FPS) if possible for a given resolution, to calculate the required disparity levels and window we take into account the available disparities per second this SoC can calculate and approximate the best depth level and window, in case the required real time FPS are not attainable the maximum possible is used.

3.2.1 Tools and Frameworks

The tools used for the implementation of the GPP side are

- GCC version 4.2.2
- Open CV version 2.2
- GDB client and server version 4.2 .
- Gstreamer version 1.0

The development was done in the build host machine which was a x86 Corei5 machine running Ubuntu Linux 12.04. After verifying proper execution we used the available corresponding libraries, already cross compiled in our build host machine; transferring the cross compiled applications to the SOC was seamless and execution was verified using the armv7a version of the GDB software on target. Following this top down procedure we performed minimal development on the target board taking advantage of easier debugging and development to the buildhost machine leaving only platform specific bugs instead of application also, to be solved on the target system.

3.2.2 Tests

In order to verify proper procedure for the GPP application part, a test application to stand in place of the DSP client was implemented. We also tested the Open CV image read functions to handle the camera sensors with proper timing.

The DSP test application was used to test the buffer copy communication scheme using the Issue Reclaim model using the following scenario : using Open CV we controlled the camera sensors grabbed the stereo image pair concatenating both images on 1-D memory buffer array of character type (8 bit per element) notifying the DSP test application to copy it on the its memory space; the test application then performed a simple operation to each frame, wrote it back to the same 1-D memory character buffer array and notified the main application to copy it back to the its memory space; this served more than a test as the code that handles the stitching of the two frames to a linear character buffer and vica versa was later used also in the DSP side to unmangle the frames, due to the difficulty of debugging on the DSP side any bugs on even this simple operation would be troublesome to find. Lastly the output received from the test app was unmangled and displayed to the user. Note that this DSP test application does not run on the DSP it emulates the DSP-GPP issue reclaim model but it otherwise is a normal armv7a architecture executable that runs on the GPP.

3.2.3 Optimizations

Since this part was not computational intensive no optimizations besides those that came from the compiler directive `-O3` where performed, also since a lot of third party libraries are used the custom code we could optimize was rather insignificant.

3.3 DSP

The DSP unit that lies within the OMAP3530 SOC is the system that performs the most process intensive work, the stereo correspondence function; dense map generation; plane fitting and error correction, in reality the entire depth map operation lies in the DSP. Having no O/S at least not in the usual context means that this unit is able to work on the loaded application almost exclusively, context switches and interrupts from other processes that run on the DSP (typically those of the DSP BIOS system) are at a minimum, this important attribute allows for almost 1 to 1 ratio in CPU time.

From the above, it is clear, that practically only the user level application runs on the DSP system and as it was described in the overview section, this is in turn divided into two parts the communicator client and the stereo vision system. The communicator is based on the DSP side of the official Texas Instruments sample of the LOOP application with some necessary modifications to account for the drastically increased buffer requirements to house the stereo image pair interprocessor communication and a number of arguments required to parametrize the stereo vision algorithm such as the correspondence window size; disparity map and the real time flag, again for an exhaustive, analytical list see Appendix B. Since no debugger was in our disposal for the OMAP3530 platform the development process was twofold, firstly we prototyped the entire stereo map generation algorithm using the Matlab environment; verifying that from the available correspondence matching functions, that SAD were indeed the fastest and that its accuracy was acceptable; then working our way towards the DSP we migrated towards the final result in the process described in the following sections.

3.3.1 Tools

The tool chain used to develop DSP applications is proprietary and managed by Texas Instruments, the most important tool used was the compiler tool `cl6x` which is actually a script that calls a host of other tools amongst which is a gcc style compiler and a similar linker, while there are some significant deviations from the standard, someone experienced will find it easy to adopt it. From this compiler tool we receive the DSP executable which ofcourse can only be used in the DSP unit.

Another tool used to configure the DSP unit's many parameters, RAM memory section, cache size, code size, memory unit register mappings among many other things, is the TCONF tool; writing in a javascript like language the user may configure all DSP subsystems, for more information about this tool and configuration see AppendixB and the relevant excellent documentation manuals from Texas Instruments.

While there is also an emulator system that can be used to profile the executable and the map file generated it was not used since the process intensive parts of the algorithm were identified and weighted from our custom development procedure.

Instead we developed a debug mechanism based on the Notification module that allows for callback functions in the GPP side to be called from the DSP code albeit a single 32bit value at a time, even so its significance in the development process was invaluable. Lastly it must be noted that there is a debug development platform available from Texas Instruments to construct prototypes on the OMAP3530 system using the Code Composer Studio¹ suite which is also proprietary and requires the use of the debug platform to work; this was not used as it was not available to start with and the price was especially high, at the time of this writing it is approximately 2000\$, however the author believes that the use of this platform might greatly speed up the development process, especially for novice users.

3.3.2 Implementation

Since the DSP communicator is mostly based on the TI's DSP side LOOP code, we will focus on the implementation of the stereo vision part, note here that in place of the stereo vision kernel using the our modular architecture one can replace it with any process intensive program with minimal changes; as for the DSP communicator code the reader should check AppendixB in the DSP section.

From this point onwards when we refer to the application or program we will mean the stereo vision part of the DSP system. After the initial steps of the design where we created a high level view of the system as a flowchart, the next step was to implement the program to a high level easy to use language with equally powerful and easy visualization power, so that we can visualize the depth map and any intermediate arrays, for this we chose the Matlab environment. Thus the first version of the program was developed and it helped us see using the profiler where the most computational critical parts are, although a first estimation was available from the first step(designing the code), this helped both visualize the program and verify our assumptions. A first version of the code lies in 3.1

This version uses exclusively matlab functions most of which are optimized for x86

¹OMAP3530 EVM <http://www.ti.com/tool/tmdsevm3530>

architecture, the profiling results for this code appear in figure 3.3, the window is 5 by 5 and the maximum disparity level is 128, which is extreme and unnecessary for this image as for the current image pair in literature a maximum disparity of 16 is commonly used, however the increased disparity level shows us two things :

- The impact on performance of disparity levels is evident as a simple change from 128 to 32 drops the process time almost linearly 3.5.
- The importance of a fast correspondence function metric such as SAD since this function is greatly used, an increase on the computational time for example for a factor of 2 will almost directly decrease performance for an equal factor.

By analyzing the performance results we see that the vast amount of time is spent in two functions *findBestMatchInEpipolarLine* and the *SAD* functions, here we must be careful since the SAD function is called exclusively from the *findBestMatchInEpipolarLine* function we must not sum their independent time, the time that the *findBestMatchInEpipolarLine* takes to run includes that of the SAD function; nevertheless these two functions alone account for approximately 94% of the total execution time, note at this point that execution times might vary depending on system load yet the relative impact of these two functions should approximately be the same a simple example for the two profile maps(3.3 ,3.4) Using the formula $\frac{TotalTime}{FunctionTime} = \frac{100}{x}$ we get for two different runs with varying system load

$\frac{5.022}{4.750} = \frac{100}{x} = 94.5838\%$, $\frac{5.0132}{4.468} = \frac{100}{x} = 92.9\%$, for a lower maximum disparity we get different results $\frac{1.870}{1.670} = \frac{100}{x} = 78.7\%$

we see that while the important functions are again the *findBestMatchInEpipolarLine* and *SAD*, the fraction is ($\approx 80\%$) differs from the $\approx 94\%$ previously seen, this can be explained since while the program runs faster for lower disparities the impact to read and process the images are still the same as well as the overhead to call the functions thus the 0.3Seconds have a higher impact on the total lower time. In any case, *findBestMatchInEpipolarLine* and *SAD* remain the most process intensive and costly parts of the program and any optimization must, in order to have any effect at all, improve their operation times. Lastly in figure 3.6 we see the results of the algorithm.

Following our plan we work towards the target platform, having estimated our bottlenecks, the next step is studying their behavior in native c code and verifying proper results by comparing with our Matlab code, the jump to C code will not only provide an intermediate step to the target platform but also reveal new optimization possibilities that were not available in the Matlab level. To still keep the power that the Matlab environment provides we will create the identified process intensive parts using Mex files as described in the following subsection..

```

1 function[ret] = stereo()
2 tic;
3 import yagad.*;
4 Left=rgb2gray(imresize(imread('tsuL.png'),1));
5 %Tsukuba Left Image
6 Right=rgb2gray(imresize(imread('tsuR.png'),1));
7 %Tsucuba Right Image
8 %Resize was used to benchmark image width impact using the monotonicity constraint.
9
10 [h,w]=size(Left);
11
12 window=+5;
13 range=floor(window/2);
14 A=zeros(h+2*range,w+2*range);
15 B=A;
16
17 A(1+range:end-range,1+range:end-range,:)=Left(:,:,1);
18 B(1+range:end-range,1+range:end-range,:)=Right(:,:,1);
19
20
21
22 depthA=calculateDepth(A,B, size(A),window);
23 ret=depthA;
24 figure(1);
25
26 imshow(mat2gray(ret));
27 %colormap(jet);
28 toc;
29 end
30
31 function[depthMap]=calculateDepth(A,B,dim,window)
32 depthMap=zeros(dim);
33 h=dim(1);
34 w=dim(2);
35 range=floor(window/2);
36 lastVal=0;
37
38 for y=1+range:window:h-range
39     for x=1+range:window:w-range
40         cp_x=findBestMatchInEpipolarLine(A(y-range:y+range,x-range:x+range,1),
41             B(y-range:y+range,x-range:end,1),128,window);
42         if(cp_x==0)
43             depthMap(y-range:y+range,x-range:x+range)=lastVal;
44         else
45             depthMap(y-range:y+range,x-range:x+range)=cp_x-1;
46             lastVal=cp_x-1;
47         end
48     end
49 end
50 end
51
52 function [bestMatch] =findBestMatchInEpipolarLine(referenceWindow, epipolarLine
53     ,maxDisparity,window)
54     [h,w]=size(epipolarLine);

```

```
55     bestResult=900;
56     bestMatch=0;
57     result=0;
58     range=floor(window/2);
59     for x=1+range:(w-range)
60         result=SAD(referenceWindow, epipolarLine(:,x-range:x+range,1));
61         if result<bestResult
62             bestMatch=x;
63             bestResult=result;
64             if result==0%if perfect match is found
65                 return
66             end
67         end
68         if x>maxDisparity
69             return
70         end
71     end
72 end
73
74
75 function[sq] = SAD(referenceWindow, correspondingFrame )
76     diff=(referenceWindow-correspondingFrame);
77     sq=sum(sum(abs(diff)));
78 end
```

LISTING 3.1: Matlab pilot code

Profile Summary

Generated 12-Jul-2013 17:15:04 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
stereo	1	1.820 s	0.052 s	
stereo>calculateDepth	1	1.568 s	0.098 s	
stereo>findBestMatchInEpipolarLine	4466	1.470 s	0.743 s	
stereo>SAD	132634	0.726 s	0.726 s	
imshow	1	0.137 s	0.003 s	
newplot	2	0.060 s	0.003 s	
newplot>ObserveAxesNextPlot	2	0.057 s	0.002 s	
graphics/private/cla	2	0.055 s	0.005 s	
cla	2	0.055 s	0.000 s	
imutils/private/limitSize	1	0.055 s	0.003 s	
imutils/private/constrainToWorkArea	1	0.049 s	0.049 s	
setdiff	4	0.047 s	0.004 s	
setdiff>setdifflegacy	4	0.043 s	0.017 s	
imread	2	0.040 s	0.002 s	
imagesc/private/readpng	2	0.036 s	0.003 s	

FIGURE 3.1: A smaller disparity level drops execution time almost linearly, however the impact of the identified process intensive functions is still prominent.

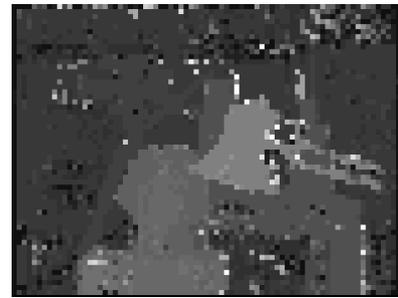


FIGURE 3.2: Extracted depth map, 32 Disparity Levels, Window size of $W = 5 * 5$. While the presence of outliers is evident, the quality trade off for performance is acceptable even with elementary plane fitting and error correction.

Profile Summary

Generated 12-Jul-2013 17:17:57 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
stereo	1	5.022 s	0.002 s	
stereo>calculateDepth	1	4.847 s	0.097 s	
stereo>findBestMatchInEpipolarLine	4466	4.750 s	2.453 s	
stereo>SAD	471970	2.296 s	2.296 s	
imshow	1	0.110 s	0.007 s	
newplot	2	0.075 s	0.003 s	
graphics/private/cla	3	0.067 s	0.011 s	
setdiff	5	0.054 s	0.005 s	
setdiff>setdifflegacy	5	0.049 s	0.020 s	
cla	2	0.049 s	0.002 s	
newplot>ObserveAxesNextPlot	2	0.049 s	0.000 s	
imread	2	0.040 s	0.001 s	
imagesc/private/readpng	2	0.036 s	0.002 s	
imutils/private/basicImageDisplay	1	0.028 s	0.003 s	
ismember	5	0.027 s	0.004 s	

FIGURE 3.3: Code profiler first run. It is clear that the most intensive parts of this code are finding the best match in the Epipolar Line and the computational cost of the correspondence matching function(in this case SAD)

Profile Summary

Generated 15-Jul-2013 15:42:31 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
stereo	1	5.132 s	0.003 s	
stereo>calculateDepth	1	4.865 s	0.097 s	
stereo>findBestMatchInEpipolarLine	4466	4.768 s	2.490 s	
stereo>SAD	471970	2.277 s	2.277 s	
imshow	1	0.203 s	0.006 s	
newplot	2	0.168 s	0.002 s	
newplot>ObserveFigureNextPlot	2	0.120 s	0.005 s	
clf	1	0.115 s	0.055 s	
graphics/private/cla	3	0.073 s	0.011 s	
setdiff	5	0.059 s	0.006 s	
setdiff>setdifflegacy	5	0.053 s	0.028 s	
newplot>ObserveAxesNextPlot	2	0.046 s	0.001 s	
cla	2	0.045 s	0.001 s	
imread	2	0.039 s	0.003 s	
imagesc/private/readpng	2	0.033 s	0.001 s	

FIGURE 3.4: Code profiler second run. Another run with different system load provides similar results

3.3.2.1 MexFile

Using the option of Mex files we are able to write our code in C, while at the same time verifying intermediate and final results with an already working program, rapidly. As expected the functions `findBestMatchInEpipolarLine` and `SAD` were migrated to Mex files and the calling entry mex function took the place of the `calculateDepthMap` (see 3.4 line:31), since the requirement was to be called from the already developed Matlab code, no other development was necessary other than simply writing the code in a Mex file, compiling it and calling it in place of the previous Matlab functions the result of the stereo.mex file was the depth map. By studying the C code in 3.4 one will identify the entry function `mexFunction` that glues native code and the Matlab environment and the stereo function which is in place of `calculateDepthMap`, `findBestMatchInEpipolarLine` with its corresponding function and lastly in place of `SAD` there are two native functions one to perform the Absolute Differences element by element and one to sum the resulting matrix to a number. Lastly close observation to the following code segment will reveal a peculiar operation, we put all the data to a structure byte by byte, the structure thus contains the stereo image pair, this is done because as planned from the beginning of this analysis in the target platform we will need to pass the data to the DSP using a buffer, thus by creating the code wrap now we can test and verify proper serialization and deserialization of data (DSP will also return data in a buffer).

```

1   for(x=0;x<n;x++){
2       unsigned char * offsetR=rightImage+x*m;
3       unsigned char * offsetL=leftImage+x*m;
4       for(y=0;y<m;y++){
5           testImage->right.data[y][x][0]=*(offsetR+y);
6           testImage->left.data[y][x][0]=*(offsetL+y);
7       }
8   }
9   stereo(testImage, window);
10
11  for(x=0;x<n;x++){
12      unsigned char * dstOffset=res +x*m;
13      for(y=0;y<m;y++){
14          *(dstOffset+y)=(unsigned char)testImage->depth.data[y][x][0];
15      }
16  }

```

LISTING 3.2: serialize data

As expected, writing the process intensive functions in native code and calling them through the mex file mechanism accelerated the algorithm greatly for the 128 Disparities the run time as verified by profiler output (3.3 is just above 1 second in contrast with the 4.7 it took in native matlab code. Note that no optimizations are done yet other than turning the process intensive code to native code. The goal of this step however was

not to optimize the code as explained above we rather wanted to close in our distance towards the target platform and have a bug free code to work with on the target platform having debugged as much as possible in this higher level of development and indeed we now have a C code that performs stereo vision with verified results and does so by correctly serializing image data in a buffer and back².

Results for this intermediate code in 3.5 and generated depth map 3.6.

```

1 function [c] =stereonative()
2     tic;
3     window=5;
4     rawA=rgb2gray(imread('tsuL.png'));
5     rawB=rgb2gray(imread('tsuR.png'));
6
7     c=stereo(rawA,rawB,window);
8
9     imshow(mat2gray(c));
10    toc;
11 end

```

LISTING 3.3: Mex matlab side pilot code, as evident all processing is done in the stereo Mex function, using Matlab enviroment only for visualization and input.

The native C code below performs the stereo map calculation, no optimization is implemented at this point and some helper functions are in the stereo.h header file.

```

1 #include "stereo.h"
2 #include "mex.h"
3 #include "matrix.h"
4
5
6 char * printMatrx(const int * M,int row, int col);
7 void absoluteDiffs(int* res ,const int * A, const int* B,int rows,int cols);
8 int sumOfMatrix (const int* A,int rows,int cols);
9 int findBestMatchInEpipolarLine( LargeImagePair pair,int currentX, int currentY,
10                                int maxDisparity, int window);
11 void assignRowValues( int * target ,int startX,int endX, int value);
12 void loadElementsWindow(int * dst, int * start,int window, int cols, int rows);
13 void stereo(LargeImagePair * testImage,int window);
14 #define MAX_DISPARIITY 128
15
16 void mexFunction(int nlhs,mxArray *plhs [],int nrhs, mxArray *prhs []){
17     int x=0;
18     int y=0;
19     int n,m=0;
20     unsigned char * leftImage;
21     unsigned char * rightImage;
22     unsigned char * res;
23     int window=5;
24
25     void * buffer;
26     mxArray *A,*B,*R;

```

²The image structure is in the stereo.h header file, check Appendix C for the source code.

```

27     mwSize dims[2];
28     LargeImagePair * testImage;
29     buffer= (void*) malloc(sizeof(LargeImagePair));
30     testImage= (LargeImagePair*)buffer;
31     A=prhs[0] ; B=prhs[1] ; R=plhs[0];
32     m=mxGetM(A);    n=mxGetN(A);
33     dims[0]=m; dims[1]=n;
34     if(nrhs==3){
35         window=*mxGetPr(prhs[2]);
36     }
37
38     plhs[0]=mxCreateNumericArray(2, dims, mxUINT8_CLASS, mxREAL);
39
40
41
42     leftImage=(unsigned char *)mxGetData(A);
43     rightImage=(unsigned char *)mxGetData(B);
44
45     res=(unsigned char *)mxGetPr(plhs[0]);
46     for(x=0;x<n;x++){
47         unsigned char * offsetR=rightImage+x*m;
48         unsigned char * offsetL=leftImage+x*m;
49         for(y=0;y<m;y++){
50             testImage->right.data[y][x][0]=*(offsetR+y);
51             testImage->left.data[y][x][0]=*(offsetL+y);
52         }
53     }
54     stereo(testImage, window);
55
56     for(x=0;x<n;x++){
57         unsigned char * dstOffset=res +x*m;
58         for(y=0;y<m;y++){
59             *(dstOffset+y)=(unsigned char)testImage->depth.data[y][x][0];
60         }
61     }
62     free(buffer);
63 }
64
65
66 void stereo(LargeImagePair * pair,int window){
67     int h =IMAGE_HEIGHT_LARGE;
68     int w= IMAGE_WIDTH_LARGE;
69     int range=window/2;//this will keep floor(window/2)
70     int x=0;
71     int y=0;
72     int k=0;
73     int lastVal=0;
74     int correspondingX=-1;
75     int val=lastVal;
76
77     for (y=range;y<(h-range);y=y+window){
78         for (x=range;x<(w-range);x=x+range){
79             correspondingX=findBestMatchInEpipolarLine(*pair, x,y,
80                 MAX_DISPARITY,window);
81             val=lastVal;

```

```

82         if (correspondingX != -1) {
83             val = correspondingX - x;
84             lastVal = val;
85         }
86         for (k = (y - range); k < (y + range + 1); k++) {
87             assignRowValues (&pair->depth.data[k][0][0], x, window, val);
88         }
89     }
90 }
91
92 }
93
94
95
96
97
98
99 /**
100  * This function finds the best match of a window
101  * inside an epipolar line
102  *
103  */
104 int findBestMatchInEpipolarLine ( LargeImagePair pair, int currentX,
105     int currentY, int maxDisparity, int window) {
106     int bestResult = 900;
107     int h = IMAGE_HEIGHT_LARGE;
108     int w = IMAGE_WIDTH_LARGE;
109     int range = window / 2; //this will keep floor(window/2)
110     int x = 0;
111     int y = 0;
112     int bestMatch = -1;
113     int result = bestResult;
114     int * diff = (int*) malloc (sizeof (int) * window * window);
115     int * referenceWindow = (int*) malloc (sizeof (int) * window * window);
116     int * correspondingWindow = (int*) malloc (sizeof (int) * window * window);
117     int * leftIndex = &pair.left.data [0] [0] [0];
118     int * rightIndex = &pair.right.data [0] [0] [0];
119     int giveUpThreshold = currentX + maxDisparity;
120     rightIndex += (currentY - 1) * IMAGE_WIDTH_LARGE;
121     leftIndex += (currentY - 1) * IMAGE_WIDTH_LARGE ;
122
123     loadElementsWindow (referenceWindow, leftIndex, window, currentX - 1, w);
124
125     for (x = currentX; x < (w - range); x++) {
126         loadElementsWindow (correspondingWindow, rightIndex, window, x - 1, w);
127         absoluteDiffs (diff, referenceWindow, correspondingWindow, window, window);
128         result = sumOfMatrix (diff, window, window);
129         if (result < bestResult) {
130             bestMatch = x;
131             bestResult = result;
132         }
133         if (result == 0) {
134             break;
135         }
136         if (x > giveUpThreshold) {

```

```
137         break;
138     }
139 }
140
141 free(correspondingWindow);
142 free(referenceWindow);
143 free(diff);
144
145     return bestMatch;
146 }
147
148
149
150 void loadElementsWindow(int * dst, int * start, int window, int col, int NoCols){
151     int y=0;
152     int x=0;
153     int *dstOffset;
154     for(y=0;y<window;y++){
155         int *offset=(start+y*NoCols+col);
156         int *dstOffset=(dst+y*window);
157         for(x=0;x<window;x++){
158             *(dstOffset+x)=*(offset+x);
159         }
160     }
161
162
163 }
```

LISTING 3.4: Mex pilot code

3.3.3 Amdahl's Law

Having profiled our code, at least from a high level, we identified the most time consuming functions from Amdahl's Law³ we know that if we want to speed up our code then we should focus on the most time consuming parts, keeping in the back of our mind that the DSP available to us is a SIMD component with 2 independent datapaths as well. Thus we will turn our attention on the stereo correspondence and SAD functions exclusively. Using the formula $maximumspeedup \leq \frac{p}{1+f \cdot (p-1)}$ thus since f is from our analysis up to here approximately $1 - f \approx 94\%$, $\rightarrow f = 6\% \rightarrow f = 0.06 \Rightarrow Speed_{Max} = \frac{p}{1+0.06 \cdot (p-1)}$ where p is the number of times we increased the speed of the aforementioned functions, we can expect an almost linear 1 to 1 relationship between overall program speedup and function speedups.

3.3.3.1 Window Impact

As we increase the window size we have two effects :

- Blurring, higher window means that more pixels are taken into account to find a correspondence reducing noise effects but also reducing resolution, a good trade off seems to be a window of 7. But as usual in most computer vision problems, the best window size depends on the application.
- Speed higher window size means that once a correspondence is found it applies to all the elements of the window, greatly increasing speed. For real time applications the author believes a window size of 7 provides the optimum balance between time and quality.

3.3.3.2 Maximum Disparity

Similar to the size of the correspondence window, maximum disparity has a major impact on performance; a lower maximum disparity will end a correspondence search sooner than not allowing for the entire process to complete faster albeit the optimum match may not be found, but this is only true for items closer to the camera and in applications where mid field detection is required that may not cause interference. Again according to the application maximum disparity levels might range from as low as 8 to 128. For real time application 16 to 32 disparity levels provide the fastest results while holding acceptable levels of accuracy. Generally the higher the maximum disparity levels the lower will be the frame rate.

³Amdahl's law is mainly applied to speedup from parallel processors but it can also be used to estimate speedup from code optimizations

Profile Summary

Generated 15-Jul-2013 16:38:08 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
stereonative	1	1.399 s	0.001 s	
stereo (MEX-file)	1	1.241 s	1.241 s	
imshow	1	0.109 s	0.008 s	
newplot	2	0.073 s	0.003 s	
graphics\private\clo	3	0.065 s	0.011 s	
setdiff	5	0.052 s	0.006 s	
setdiff>setdifflegacy	5	0.046 s	0.021 s	
cla	2	0.045 s	0.002 s	
newplot>ObserveAxesNextPlot	2	0.045 s	0.000 s	
imread	2	0.037 s	0.001 s	
imagesci\private\readpng	2	0.032 s	0.002 s	
imuitools\private\basicImageDisplay	1	0.026 s	0.004 s	
clf	1	0.025 s	0.001 s	
newplot>ObserveFigureNextPlot	2	0.025 s	0.000 s	
imagesci\private\pngreadc (MEX-file)	2	0.023 s	0.023 s	

FIGURE 3.5: All the processing is done natively

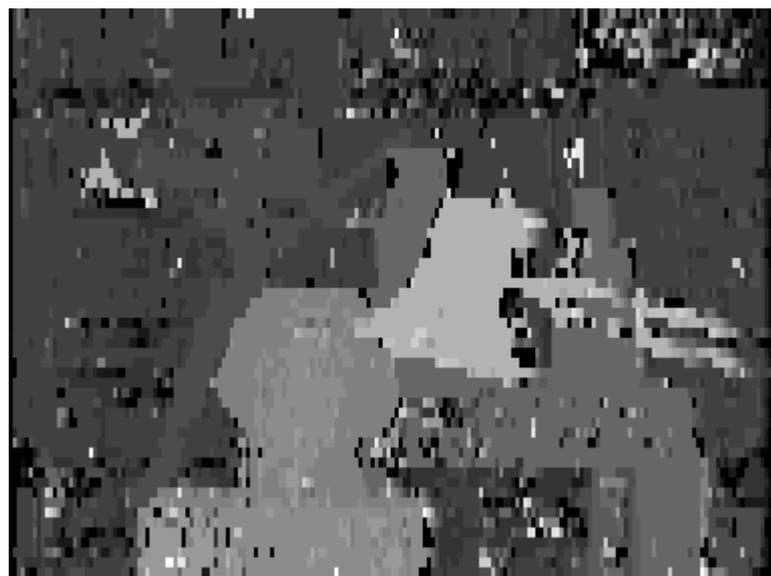


FIGURE 3.6: Disparity map as calculated from the above native code and visualized in Matlab.

3.3.4 OMAP3530

Having identified our goals and with a tested working code(albeit at a a higher level) we migrate our code towards the DSP unit where the optimization step will take place, from now on development will be on target platform. A small intermediate step was to cross compile the C code from the Mex file into armv7a architecture which is the GPP unit of the OMAP3530 system, the jump was seamless as the native C code required minimal adjustments (if any) at the same step we incorporated the Open CV framework to control the camera sensors and until then to be the I/O interface for image reading and handling see Appendix C for the code.

3.4 Optimizations

We can divide all the optimizations performed into two major categories, firstly is the simple code logic optimizations that would be applied anywhere regardless of the underlying architecture such as (programmer defined)loop unrolling or even simply smarter code; secondly are those optimizations that *are* architecture depending and as such *cannot* be applied to all systems. Before we continue however lets identify the hardware resources exported to us via the DSP architecture and see what can be exploited(and what must be avoided).

3.4.1 Platform independent optimizations- general principles

3.4.1.1 Disparity-Window Choice

Window size affects the generated map quality, generally window sizes of higher dimensions offer robustness against noise and provide with better correspondences; performance suffers however in a similar way as for each corresponding block more pixels are taken into account, thus increasing the number of required operations. The maximum level of Disparities as stated multiple times affects the number of blocks that the algorithm will search before selecting a match as best match, this affects performance significantly since higher disparities will force the algorithm to process more blocks for each pixel; implicitly this creates a window where objects that would generate higher disparities, and are thus closer to the camera sensor, are filtered away selectively creating a map that contains only objects in a certain distance, this can be desirable or not. In any case the above parameters drastically affect the depth map generation both performance and quality wise and must be tweaked according to demands to provide with optimum results.

3.4.1.2 Compiler optimizations

Since size is not an issue in this application and the entire program fits easily on the code memory region of the DSP, the compiler optimizations given where -O3 which performs a host of optimizations all aimed towards performance. The impact of even this simple action gave a significant performance boost, when able always optimize using this flag.

3.4.1.3 Optimized image data serialization and Loop Unrolling

Another general optimization is writting both frames simultaneously on the buffer using the same loop, , in the folowing code snippet the procedure starts by reading the stereo image pair, and then serializes it to the aforementioned character buffer, exploiting the fact that all images have the same dimensions we serialize using the same loop avoiding extra overhead, furthermore high loop unrolling is adviced at least for this loop. Loop unrolling pragma directives while available to other compilers (like cl6x which we exploit in the DSP side later) are not available as a standard on GCC⁴ and in any case are architecture dependant , loop unrolling might not be optimum for all loops so fine grained control is required otherwise generally increasing the loop unrolling count for all loops might lead instead to performance decrease.

```

1 Mat colorI=imread(rightImage.c_str(),1);
2     Mat imageR ;cvtColor(colorI, imageR, CV_RGB2GRAY);
3     colorI=imread(leftImage.c_str(),1);
4     Mat imageL ;cvtColor(colorI, imageL, CV_RGB2GRAY);
5
6     for(y=0;y<IMAGE_HEIGHT_LARGE;y++){
7         int offset= y*imageR.cols;
8         char *trip,*tlip,*dlip;
9         rptr=imageR.data + offset;
10        lptr=imageL.data + offset;
11
12        trip=rImageBuffer+offset;
13        tlip=lImageBuffer+offset;
14        dlip=dImageBuffer+offset;
15
16        #pragma unroll (IMAGE_WIDTH_LARGE)
17        for(x=0;x<IMAGE_WIDTH_LARGE;x++){
18            testImage->right.data[y][x][0]=rptr[x];
19            trip[x]=rptr[x];
20            testImage->left.data[y][x][0]=lptr[x];
21            tlip[x]=lptr[x];
22            dlip[x]='0';
23        }
24    }

```

⁴The BOOST preprocessor library provides this functionality

LISTING 3.5: Optimized data serialization code, using Open CV imread on lines 1-3 we read the image frames from filepath

3.4.1.4 Memory Copy Impact

We evaluated the memory copy operation between the DSP and the GPP using the issue reclaim model, as it will be shown in the next chapter (results), memory copy is *not* trivial, it must be taken under account and minimize interprocessor memory copies to a minimum. The initial application copied a buffer that contained the left frame, the right frame and room for the resulting depth map, for the Tsukuba Image pair this was $Image_{left} = 384 * 288 * 8bit + 384 * 288 * 8bit + 384 * 288 * 8bit$ to store the resulting disparity map, to a total of $Buffer = 331776bytes$, for each frame this buffer had to be copied once to the DSP and then back to the GPP side thus each frame required $Total = GPP_{toDSP} + DSP_{toGPP} = 663552Bytes$ or 648Kbytes, with our tests about 50Mb/sec can be copied between processors that means that the maximum frame rate achievable with this scheme is $FPS = \frac{50 * 1024Kb}{648} \approx 79$ just by the limit of the memory copy operations alone. The optimization performed here cut the memcopy requirements in $\frac{2}{3}$, by sending the image pair alone from the GPP to the DSP and reclaiming the stereo depth map in the place of the left image.

One might consider sending just the depth map from the DSP side, such an action was not possible in the issue reclaim model, since the issued buffer must be reclaimed whole. In the current scheme our hard limit is thus. $FPS = \frac{50 * 1024Kb}{648 * \frac{2}{3}} \approx 118$, of course this is not achievable since we haven't even taken under account the actual depth map frame calculation time, which is the actual application, yet it will allow us to achieve higher frame rates since less time will be used in memory copy of the data between the processors hence more time will remain for the processing.

3.4.1.5 SAD on one step

Improving the stereo correspondance estimation is expected to provide significant speedup, again by reengineering code we exploited the SAD functionality as part of one function avoiding recurring costs of both loop overhead and stores by almost half. The following code is the SAD functionality unoptimized (reminder, this code runs on DSP) on figure 3.6 we see 2 different functions performing the parts of the SAD function one to calculate the matrix of absolute differences, and the next to sum all the elements. The next optimization which again can be practiced in any architecture is performing the SAD calculation in one step as shown in 3.7, in this snippet the entire calculation is done on

one step without keeping the intermediate matrix of differences, since this functionality is used a vast amount of times it has a significant impact on performance hence the benefits from this optimization are substantial.

```

1
2
3 findBestMatchInEpipolarLine(...){
4 ...
5
6     absoluteDiffs(diff,referenceWindow,correspondingWindow>window>window);
7     result=sumOfMatrix(diff>window>window);
8
9 ...
10 }
11
12
13
14 /**
15  * square matrices only
16  */
17 void absoluteDiffs(pixel * res ,const pixel * A, const pixel* B,small rows,small cols){
18     int i, j;
19     for( i=0; i<rows;i++){
20         const pixel * offsetA=A+i*rows;
21         const pixel * offsetB=B+i*rows;
22         pixel * offsetRet=res+i*rows;
23         for( j=0; j<cols;j++){
24             *(offsetRet+j)= abs ((*offsetA+j))-*(offsetB+j));
25         }
26     }
27 }
28
29 /**
30  * SUM ALL ROWS AND COLUMNS OF A MATRIX
31  *
32  */
33 int sumOfMatrix (const pixel* A,usmall rows,usmall cols){
34     int res=0;
35     int j,i;
36     for(i=0; i<rows;i++){
37         const pixel * offset=A+i*rows;
38         for( j=0; j<cols;j++){
39             res+=*(offset + j) );
40         }
41     }
42     return res;
43 }

```

LISTING 3.6: Naive SAD

```

1
2 findBestMatchInEpipolarLine(...){
3 ...
4     calcAndSumDiff(...);
5 ...

```

```

6 }
7
8
9 int calcAndSumDiffs(const pixel * A, const pixel * B){
10     int i, j;
11     int result=0;
12     for( i=0; i<WINDOW;i++){
13         const pixel * offsetA=A+i*WINDOW;
14         const pixel * offsetB=B+i*WINDOW;
15         for( j=0; j<WINDOW;j++){
16             result+= (pixel)abs((*offsetA+j))-*(offsetB+j));
17         }
18     }
19 }
20 return result;
21 }

```

LISTING 3.7: Fast compact SAD

3.4.2 Architecture Dependent Optimizations

The general optimizations can only take us so far, while important the results gained where modest in compare to what really the powerful DSP unit can give us. In order to use its power however we must exploit its hardware resources and change our code to confront to its custom capabilities.

By studying "*TMS320C64x/C64x+ DSP CPU and Instruction Set*" document we find out that the DSP unit is composed of two independent data paths that may work independantly on their register files , furthermore each datapath actually is a pipeline containing 4 functional units each with an an execution model that closely resembles Tomasulo;s reservation stations model. This means that we must exploit instruction issuing in such a way as to fully occupy all functional components if possible; our code will need reenging to support this. Furthermore the manual states that the number of operations each unit may perform depend on the size of operants, ranging from one 32bit operation to 4 simultaneous 8 bit operations for the .L units (there are two .L units one for each datapath namely .L1 and .L2), this means that where possible we must limit the variables to the minimum available container (8bit if possible), in the following figures we see figures 3.7,3.8⁵ from the "*TMS320C64x/C64x+ DSP CPU and Instruction Set*" document that show the datapath architecture and a brief description of the functional units, it is evident that all functional units have areas where they excell such as the .L units that may perform *simultaneous adds/subs* on 8bit packed variables, where the .S units can only perform 32Bit additions/subs. Also it is evident

⁵Reproduced from document id:spru732j, pages 21,29-30, publicly available from <http://www.ti.com/lit/ug/spru732j/spru732j.pdf>

that multiplications can only occur (in hardware level) on the .M units, while the .D units are used for memory operations. Also we find in the manual that all operations besides memory (some) operations conclude in one cycle (see pages 519-523 of manual). Lastly we verify our assumption, that performance is directly related to the abundance of the available and issued instructions (by keeping all functional units occupied if possible) from section Performance Considerations⁶ and we quote

”The C64x/C64x+ DSP pipeline is most effective when it is kept as full as the algorithms in the program allow it to be. It is useful to consider some situations that can affect pipeline performance.”

In the same section the instruction fetch model depicts that up to 8 instructions can be fetched and issued in one cycle (14 for the C64+ model via packed headers), this places another constraint and goal at the same time: issue as many instructions as possible up to the maximum allowed number, while of course applying caution so as to not overload one component in favour of others since then no speedup will be gained. In the same documentation to take into account the previous statement there is section 9.C-F, p.627 a detailed mapping between instructions and corresponding functional unit exists to optimize code.

To sum up, the TMS320C64+ unit⁷ is a powerful processing unit with strong parallel computation capabilities, whose power can only be leveraged if properly exploited by taking under consideration a number of things, most prominent of which are the capabilities of each functional unit, architecture depended commands and the differential impact each variable has depending on its size to program performance, lastly the Warp like fetch and issue model implicitly dictates that code is written in such a way that SIMD principles can be applied. With all these in mind we further develop the stereo correspondence and depth map generation code to reach the maximum attainable performance this is described in the following sections Variable Types, Intrinsic, Loop Unrolling and the Use of Restrict. for a more detailed description the user is directed to study the aforementioned excellent document.

⁶Ch4.3, p.527

⁷the one apparent to Gumstix OveroFire via OMAP3530

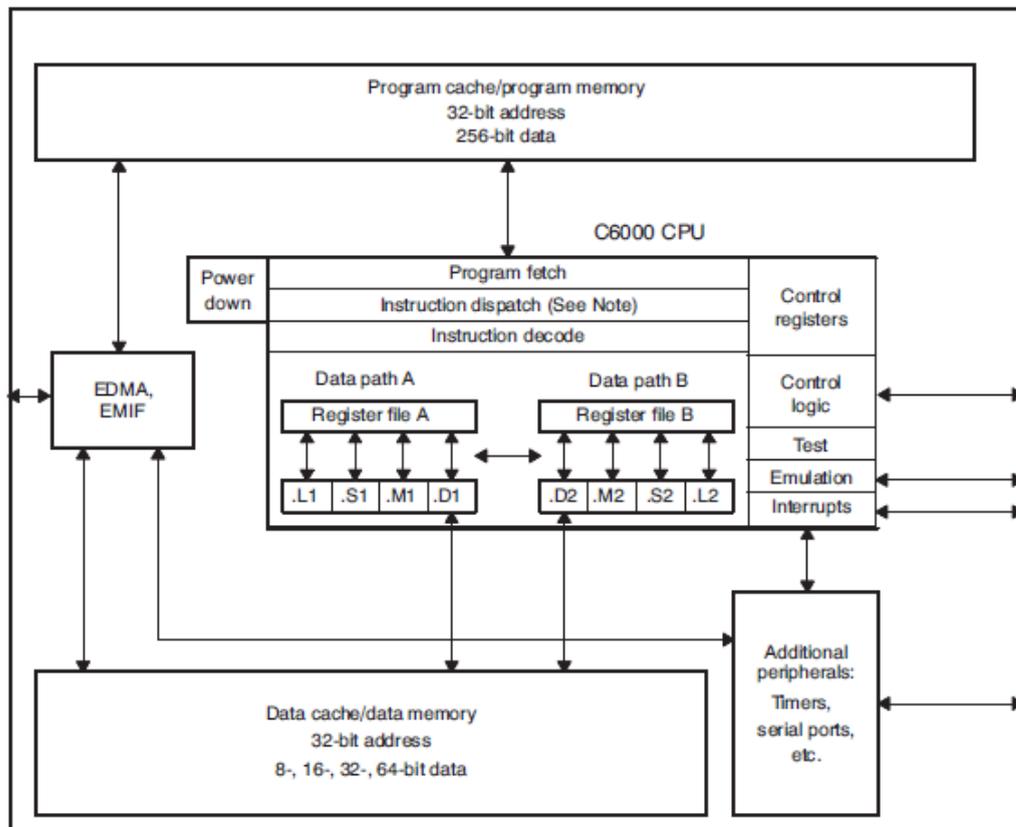


FIGURE 3.7: 2 Independent identical datapaths, A and B each with 4 functional units (.L,.S,.M,.D)

3.4.2.1 Variable types

Since image intensities vary from 0...255 we can use for pixel container the `uint88` type. Also most results such as the result from the matching function can fit into a 16 bit unsigned integer and since it is the sum of absolute values, unsignedness will not be an issue. Restricting sizes on all numerical values such as counters in loop code segments might not give the expected result since the .L unit that performs the 8 bit packed operations is already utilized leaving the .S,.D units that can only perform 32 bit ops alone thus counters are not `uint8` but rather simple integers lastly to perform 8bit packed operations on .L units we need to pack them first to a 32bit register, this is done by the .S unit.

To sum up the data flow to utilize all 6 components (remember .M units are not used currently) goes like this : The .S units pack 16 8bit values(8 intensity pixels) to 2 32

⁸unsigned 8 bit integers values range from 0 to 255

bit registers, the .L units subtract the previous packed values from .S while the .D units increment the counters and load from memory if necessary. Operation wise we need 4 pack operations, 2 8bit subs(.L units) , 2 adds/mem accesses which is 8 Operations plus the occasional sift that might take place to the .M units, for a total of 8 instructions. It is evident that a big portion of these optimizations is done by the compiler optimization -O3 (if we implicitly use .L with exclusive operations for this module then newly issued ADD or SUB operations burden .D module if idle, with a restriction in cross path allowances) from the compiler tools as will be shown in the Results Chapter.

3.4.2.2 Intrinsic

Before we continue on the details of why and which Intrinsic we used in this application a small introduction is required. We quote from the 2 major processor architecture currently on the market

Intel⁹ Intrinsic are assembly-coded functions that allow you to use C++ function calls and variables in place of assembly instructions. Intrinsic are expanded inline eliminating function call overhead. Providing the same benefit as using inline assembly, intrinsic improve code readability, assist instruction scheduling, and help reduce debugging. Intrinsic provide access to instructions that cannot be generated using the standard constructs of the C and C++ languages

Arm¹⁰ Intrinsic functions and data types, or intrinsic in the shortened form, provide access to low-level NEON functionality from C or C++ source code. They use syntax that is similar to function calls. Software can pass NEON vectors as function arguments or return values, and declare them as normal variables.

Intrinsic provide almost as much control as writing assembly language, but leave the allocation of registers to the compiler, so that you can focus on the algorithms. Also, the compiler can optimize the intrinsic like normal C or C++ code, replacing them with more efficient sequences if possible. It can also perform instruction scheduling to remove pipeline stalls for the specified target processor. This leads to more maintainable source code than using assembly language.

So in sort, intrinsic give access to exclusive capabilities of the underlying hardware and do so by providing a c/c++ style function that when interpreted by the compiler provides inline high performance assembly code. In our application for example we

Table 2-2. Functional Units and Operations Performed

Functional Unit	Fixed-Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit arithmetic operations Quad 8-bit arithmetic operations Dual 16-bit minimum/maximum operations Quad 8-bit minimum/maximum operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) Byte shifts Data packing/unpacking Dual 16-bit compare operations Quad 8-bit compare operations Dual 16-bit shift operations Dual 16-bit saturated arithmetic operations Quad 8-bit saturated arithmetic operations
.M unit (.M1, .M2)	32 × 32-bit multiply operations 16 × 16-bit multiply operations 16 × 32-bit multiply operations Quad 8 × 8-bit multiply operations Dual 16 × 16-bit multiply operations Dual 16 × 16-bit multiply with add/subtract operations Quad 8 × 8-bit multiply with add operation Bit expansion Bit interleaving/de-interleaving Variable shift operations Rotation Galois Field Multiply
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit constant offset Loads and stores with 15-bit constant offset (.D2 only) Load and store doublewords with 5-bit constant Load and store nonaligned words and doublewords 5-bit constant generation 32-bit logical operations

FIGURE 3.8: 4 independent functional units per Datapath, each with its own strengths and exclusive Intrinsics.

denoted the need to calculate the difference matrix between the left and right frame in a window each time, this as shown requires the subtraction of 2 8bit values each time that in order to gain the parallel speedup we downscaled to 8bit variables; this scenario works perfectly well when in the equation

$$R = A - B \quad (3.1)$$

where R is the result and A,B the values of an 8bit pixel on the left and right correspondance window respectively, if $A \geq B$ then the outcome stored in R is correct however if $A \ll B$ then we have an overflow since R,A, and B are unsigned and the result in R is wrong i.e $R = 1 - 2$ we get R=255 while we would obviously want a -1 and via the absolute operator a 1. This problem conventionally means to upscale it to 16bit or perform a simply test case code like the one in 3.8, however experimentaly upscaling to 16bit or doing this code has exactly the same cost.

```

1 //typical code to calculate absolute
2 uint8 R,A,B;
3 ...
4
5 R=A-B;
6 (A<B){
7 R=B-A;
8 }
9
10
11 //second style
12
13 uint8 R,A,B;
14 ...
15
16 R=(uint8)abs(A, B); //ABS function upscales A, B to 32 bit.
```

LISTING 3.8: 8bit Absolute calculation using conditionals

By using Texas Instruments Intrinsic however for the TMS320C64+ DSP we can use this function `_subabs4(A,B)` that subtracts and then returns the absolute 8bit result of the 8bit opperrants A, B, the new code is shown in 3.9, the discerning reader will note that we did not pack the values to 32bit this is because by experiments it is shown that performance does not suffer and that due to implicit issuing of multiple commands by the scheduler and loop unrolling multiple 8bit commands are issued to the .L units providing with the same speed as if using the pack and then issue a single command model. Note that the pramga unroll directive instructs the compiler to unroll the loop WINDOW times, furthermore the result is 16 bit since $W^{11} * W = \text{Max}(256)$ (windows more than 16 do not provide acceptable accuracy in literature), $256 = 2^8 \rightarrow 2^8 * 2^8 = 2^{16} \equiv 16\text{Bit}$

¹¹W is the WINDOW size of the correspondence search and assignment window

it will be shown in the result section that the impact of this optimization along with loop unrolling was immense.

```

1 Int16 calcAndSumDiffs(const pixel * A, const pixel * B){
2     small i, j;
3     Int16 result=0;
4     #pragma unroll (WINDOW);
5     for( i=0; i<WINDOW;i++){
6         const pixel * offsetA=A+i*WINDOW;
7         const pixel * offsetB=B+i*WINDOW;
8         #pragma unroll (WINDOW);
9         for( j=0; j<WINDOW;j++){
10            //call specialized function to return abs of 8bit uint
11            result+= _subabs4(*(offsetA+j),*(offsetB+j));
12        }
13    return result;
14 }

```

LISTING 3.9: 8bit Absolute calculation using intrinsics

3.4.2.3 Use of Restrict

No optimization in SIMD units might be complete unless we account for RAW and WAR hazzards. To ensure correct execution the scheduler makes sure that no instructions that have WAR dependencies execute simultaneously and waits for one to finish; however this is not known in advance in arrays where pointer operations might alter the location where a pointer references and that may lead in improper and erroeneous results. This is why the compiler cannot parallelize code that exists in a loop where the destination is an array such as in the following snippet 3.10, the compiler does not know in advance if the destination address in line 10 is accessed in following loops so it must wait for its execution to finish before issuing another, it is the programmers job to tell the compiler that no RAW hazzards exist and that he may perform loop unrolling issuing simultaneous assignments; we achieve this by using the *restrict* keyword as shown in 3.11 in our application this is used in numerous locations see AppendixB for the final source code and where restrict is used. By supplying the compiler with the required insurances that no memory accesses occur in the specified locations other than the ones apparent, it may issue parallel operations that greatly accelerate the procedures in question.

3.4.2.4 Configuration Files Optimizations

The last optimizations are the ones that took place in the .tcf and .tci files that describe to the TCONF tool our DSP application and its requirements in memory and subsystems. Also the OMAP3530_ALL_CONFIGURATION.c file was altered to account for

```
1 void loadElementsWindow(pixel * dst, pixel
2     * start, Int16 col, Int16 NoCols){
3     small y,x;
4     pixel * base=start+col;
5     #pragma unroll (WINDOW);
6     for(y=0;y<WINDOW;y++){
7         pixel *offset=(base+y*NoCols);
8         pixel * dstOffset =(dst+y*WINDOW);
9         #pragma unroll (WINDOW);
10        for(x=0;x<WINDOW;x++){
11            *(dstOffset+x)=*(offset+x);
12        }
13 }
```

LISTING 3.10: Code with potential WAR hazzards, is left unoptimized by the compiler.

```
1 void loadElementsWindow(pixel * restrict dst, pixel
2     * restrict start, Int16 col, Int16 NoCols){
3     small y,x;
4     pixel * base=start+col;
5     for(y=0;y<WINDOW;y++){
6         pixel *offset=(base+y*NoCols);
7         pixel * dstOffset =(dst+y*WINDOW);
8         for(x=0;x<WINDOW;x++){
9             *(dstOffset+x)=*(offset+x);
10        }
11 }
```

LISTING 3.11: Usage of restrict keyword by the programmer allows the compiler to issue multiple writes in parallel.

our increased requirements in memory pool size and for the custom location and layout of the available RAM.

Chapter 4

Results

Overview

Due to our top to bottom implementation methodology we were able to benchmark and study many intermediate programs and see the same algorithm evolve and speed up as we worked our way to the DSP level. The author feels, besides the contribution of providing a real time implementation for the stereo vision problem, that these intermediate results and the relative speedups presented are an interesting byproduct of our work, since this was done incrementally one can see how each discipline affects code performance independently and estimate impact on his own work.

In the following sections we will present our novel implementation results in the metrics commonly used in the field of stereo vision, where as described in the 2 are mDs, FPS and mD/Joule; also to serve as a reference we will provide the execution times of the algorithm in the following cpu's :

- Core i5 sandybridge 3.2Ghz
- Core(M) i5 sandybridge 2.1Ghz
- The ARM9 600Mhz unit of the omap 3530.

4.1 Assumptions and Standards

Table 4.1 contains results from the initial algorithm with no optimizations. Special note must be made on how we estimated energy, using the reported cpu time from tools such as gprof and time and the consumption of the unit as declared by the factory. Also the code running in the Corei5 machine is the one presented in the implementation section

TABLE 4.1: Stereo algorithm Run Times, unoptimized

Platform	Clock	Watt	TPF ²	FPS	EPF ^{3]}	md/S	md/J	md/Ghz ¹
Corei5	3.200	80.000	0.500	2.000	160.000	3.539	0.044	1.105
Corei5(M)	2.100	35.000	1.200	0.833	29.167	1.475	0.042	0.702
ARM9	0.620	0.643	6.000	0.166	0.107	0.294	0.459	0.476
DSP	0.430	0.248	0.083	11.905	2.953	21.065	84.942	48.990

¹ Estimated energy , time to generate frame * power consumption (Watt)

² Time per Frame in seconds.

³ Energy per Frame in Joule

under matlab and shows only the time required to run the .mex function the overhead calling the mex file is negligible as tested by calling an empty dummy stereo function the system reported zero running time hence time we deemed that calling overhead was negligible and used the time reported from the Matlab performance utility, at this point we must note that no cpu level optimizations are done to this code and our target is not to optimize it for x86 rather to provide us with a first impression and serve as a base for comparison . The time reported in the Mobile Core i5 it is the same code running natively and the O/S is Linux Ubuntu 12.04. For the calculation of the disparities we used a window of 7 and 16 disparity levels, furthermore to calculate md/S we multiplied FPS with the as stated calculated disparities per frame. Lastly in the seventh column we propose the use of million disparities per Ghz as we feel it provides a more intuitive metric when it comes in comparing processing units and can provide the reader with a sense of future improvement in case of overclocking or higher clocked similar systems. As for the frame per second metric, 100 frames were issued for each platform in 10 batches at random intervals with varying system load and average time of execution was estimated with each batch execution contributing equally. In the case of the DSP unit where no actual processing time could be estimated with sufficient accuracy we instead considered the worst case of 100% processing and maximum power consumption (as if all DSP modules are busy 100%).

Lastly we used the Tsukuba stereo pair as it has become the norm for benchmarking stereo algorithms, the images dimensions are width:384 and height:288 $\approx 110K$ pixels , we remind the user that the closest standard is the QVGA with width:340 and height:240 $\approx 76k$ pixels, this means that $\frac{N_{pixelsTsukuba}}{N_{QVGApixels}} \approx 1.44$, multiplying the provided Tsukuba FPS with this factor will provide a very rough estimation on how fast this system will provide using the QVGA standard images.

4.1.1 Quantum Change and Incremental Speed Ups

A common notion in computer science is when debugging or performing a change, do so but a single step at a time, this practice is sometimes referred to as quantum change and the figures following show how each quantum change affects our algorithm and the

TABLE 4.2: Stereo algorithm Run Times, Quantum Step:Incremental Optimizations

No	Platform	<i>Clock</i>	<i>Watt</i>	<i>TPF</i> ²	<i>FPS</i>	<i>EPF</i> ³	<i>md/S</i>	<i>md/J</i>	<i>md/Ghz</i> ¹
1	Corei5	3.200	80.000	0.500	2.000	40.000	3.539	0.044	1.105
2	Corei5(M)	2.100	35.000	1.200	0.833	42.000	1.475	0.042	0.702
3	ARM8	0.620	0.643	6.000	0.166	5.466	0.294	0.459	0.476
4	ARM86)	0.620	0.643	4.950	0.202	3.183	0.357	0.556	0.577
5	DSP	0.430	0.310	0.084	11.905	0.026	21.066	67.953	48.990
6	OC DSP	0.520	0.375	0.057	17.544	0.021	31.044	82.783	59.699
7	Fast Correlation	0.520	0.375	0.046	21.645	0.017	38.300	102.134	73.654
8	SUB_ABS4	0.520	0.375	0.039	25.523	0.015	45.162	120.433	86.850
9	RESTRICT	0.520	0.375	0.034	29.553	0.013	52.293	139.449	100.564
10	Opti.MemCopy	0.520	0.375	0.030	33.380	0.011	59.065	157.507	113.586
11	DSP Only ⁵)	0.520	0.375	0.023	43.860	0.009	77.609	206.957	149.248

¹ Estimated energy , time to generate frame * power consumption (Watt)

² Time per Frame in seconds.

³ Energy per Frame in Joule

⁴ With all optimizations

⁵ All in DSP ,without any memory transfer between processors With ALL implemented optimizations.

⁶ With optimizations in log level, *Not NEON*.

provided speedup. The measurement method, is the same batch method of test runs with batches of 100 frames per time as described in the previous section.

To study the impact that memory copy between DSP and GPP has on our application, we performed the calculation of a batch of 100 frames with two approaches, firstly the usual method of single frame transfer between DSP and GPP was performed with the GPP providing a new frame once the previous depth map was returned by the DSP, on the other hand we modified the DSP application to run a calculate 100 depth maps of the same frame so as to avoid interprocessor communication and started the measurement after memory copy finished.

It must be noted here that the outcome is not only the actual memory transfer delay due to memory copy operation itself, hidden inside is the overhead the Issue Reclaim communication model inherently contains since the issuer is not instantly notified when his issued buffer is ready to be reclaimed and vice versa . Regardless of the exact reason however the memory impact on our application is significant and this evidently shown in 4.2.

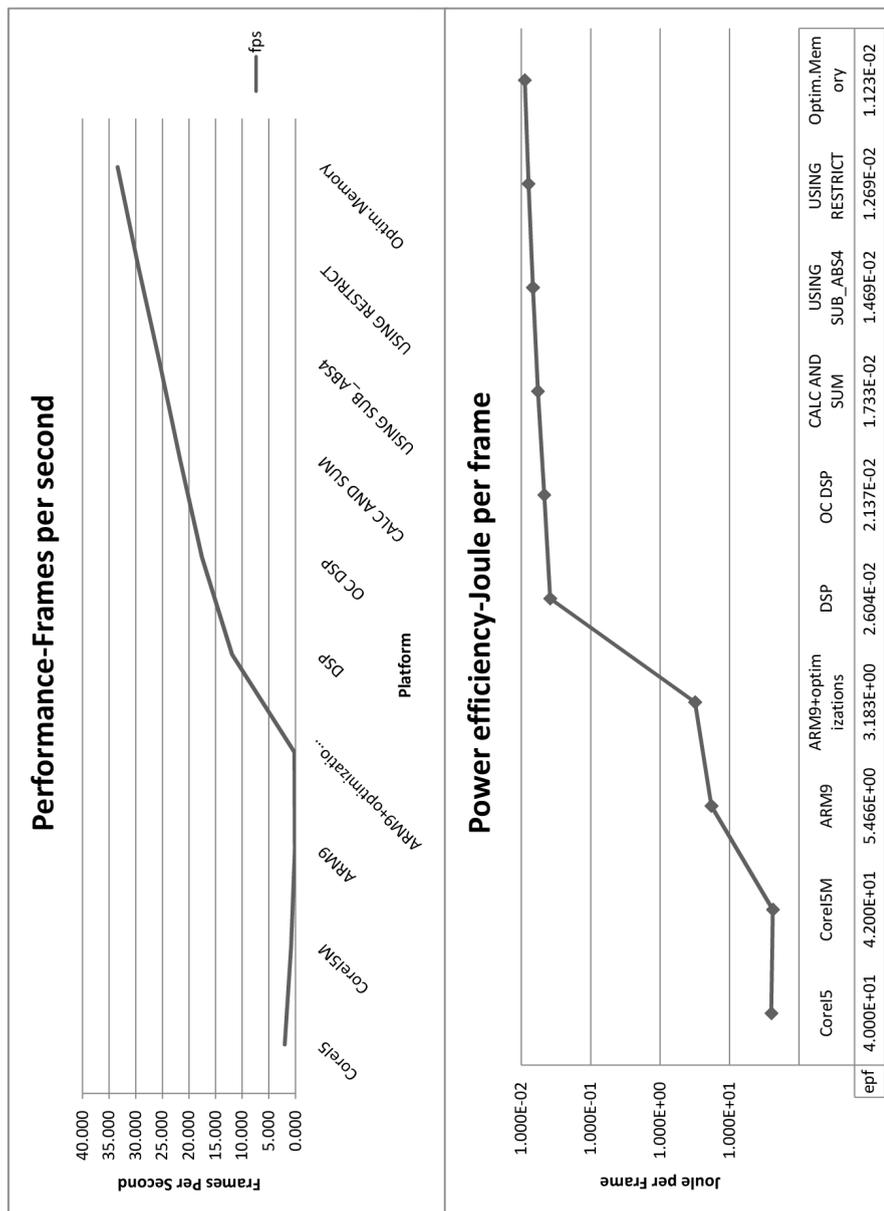


FIGURE 4.1: Performance chart.

Result Analysis

The power consumption of the OMAP3530 ArmCortex8 and of the DSP subsystem was taken from the tool that Texas Instruments provides to estimate power consumption according to usage is available for download for free from the company site¹, the author finds no reason to dispute the tools results as they system has been tested to operate under maximum power consumption with all subsystems operating at 100% utilization consumed less than 2Watts as expected.

From row 7 to 12, only the over clocked version of the DSP appears as it provides better results and while it consumes more 75mW more power it is yet more efficient md/J makes it attractive. Another interesting result on the overclocked version of the DSP unit is that the md/Ghz seem to increase which means that when the DSP operates at this frequency it does not simply raise its clock frequency it seems that additional improvements come into play otherwise the difference in the md/Ghz should be minimal between the two different clock rates on the same system (a higher clock rate should link to an equally amount of higher fps yet the ratio should be the same).

In row 11 the impact the interprocessor communication has on performance is evident. Observing the unoptimized memory copy, where a buffer of three images in size is transferred two times per frame, once towards the DSP and back, provides a FPS of 29.553 (row 10, the best result with all optimizations but memory optimization), yet without any memory copies by running the program exclusively on the DSP side performing 100 depth map calculations in a loop we receive the outstanding result of 43.860 FPS, in short from 39ms per frame we dropped to 23ms per frame or else 67.64% of the 34ms seems to be the cost of the actual calculation the rest appear to be memory copy and issue reclaim overhead, this means about 11ms for a duplex transfer of $Buffer = 3 * images * ImageSize$, $Buffer = 3 * 384 * 288 * 1Byte = 331776$, $Total = Buffer * 2 = 663552$ so $663552Bytes = 648Kb \approx 0.632Mb$ per Image yet we showed that 11ms per frame is the impact of the complete memory operation (memory copy +issue reclaim and rest delays) when we have 29.553 FPS in the case with all optimizations²all with the exception of memory buffer optimization), we can estimate the complex memory bandwidth for our application as $PracticalMemoryBandwidth \approx 1S/11ms * 0.632Mb = 57.45Mb/S$, we also know that without memory transfers, we need 23ms per frame just for the depth map operation and that in order to perform the calculation we need the two frames (Left and Right Image) from the GPP side and we need to return the calculated depth map, that places a hard constraint on the size of the data we *absolutely* have to copy and that is : $SizeDepthMapImage_{bytes} + Size_{LeftImage} + Size_{RightImage} = 110592 + 110592 + 110592 = Buffer/2 \approx 324Kb$, if we assume that by cutting the buffer

¹ http://processors.wiki.ti.com/index.php/OMAP3530_Power_Estimation_Spreadsheet

²(

TABLE 4.3: Stereo algorithm Run Times, Quantum Step:Incremental Optimizations

No	Platform	Speedup
1	ARM8	1.000
2	ARM8+Optimizations	1.717
3	DSP	58.930
4	OC DSP	1.474
5	CALC AND SUM	1.234
6	USING SUB_ABS4	1.179
7	USING RESTRICT	1.158
8	Optimizing MemCopy	1.129
9	Total Speedup	165.231

¹ We use the execution time it takes to run the initial algorithm takes to run in the ARM8 unit in OMAP3530, at 600Mhz.

size in half will lead to an equal amount of reduction in memory copy time then our estimation is 5.5ms; that would be if the actual memory copy operation was the most important factor, that would lead us to an expected result theoretically of approximately 32.5 PS by $\frac{1000ms}{Time_{Execution_per_Frame}+Time_{memory_transfers}} = \frac{1000ms}{23ms+5.5ms} \cong 35.08$, as the reader will note entry 11 provides the result for Optimizing with MemCopy and the result of 33.380 is distant from the one we theoretically (even though quite close) calculated, the reason is two fold, firstly memory copy operation is not the only factor so a linear reduction in buffer size will not lead to a linear reduction in total time since the Issue Reclaim overhead remains the same, secondly and perhaps most importantly is the issue reclaim model the issuer *must* reclaim the same buffer, that means that once we issued a buffer for the 2 input frames and the buffer was of size X we must reclaim a buffer of the same size, thus our theoretical calculation does not hold true, we must calculate for a buffer equal to 4 images even though we need , with these two additional constraints in mind we get the new result taking the assumption that the impact of a new frame will add amount of time equal to its size $\frac{1000ms}{Time_{Execution_per_Frame}+Time_{memory_transfers}} = \frac{1000ms}{23ms+6.67ms} \cong 33.7$, the theoretical result is a bit faster than the measured one but that is to be expected since there is also the overhead of the issue reclaim model and also our DSP application has also an overhead each time it receives a buffer to launch the kernel and perform some initializations each time where in the batch execution is hidden all of these and more small delays can be roughly calculated if we add a variable D and tweak it to touch our experimental results due to the ad hoc nature of the problem it is very difficult to analytically calculate exactly the delay but using the above method we have roughly 0.33ms hidden latencies due to the aforementioned reasons and potential interrupts even from the DSP Bios O/S.

Another interesting table is 4.3 , that shows the speedup gained from each quantum step

4.2 Disparity and Window Size

Disparity levels and the size of the correlation window (which in our application is also the size of the block assignment) have a direct effect on performance, as shown in figure 4.2 the higher the disparity levels the less the performance while increasing the correlation window provides usually better results the most important effect in performance is the size of the block where we assign the same value for all pixels, the bigger this window the faster the algorithm runs, since in effect it reduces the size of the image, yet produced images have lower resolution and small objects might get lost, the opposite is true for reducing the block. The results presented are for the initial version of the algorithm and we only show the results collected from the DSP, as always 10 runs of 100 single frame batches were used.

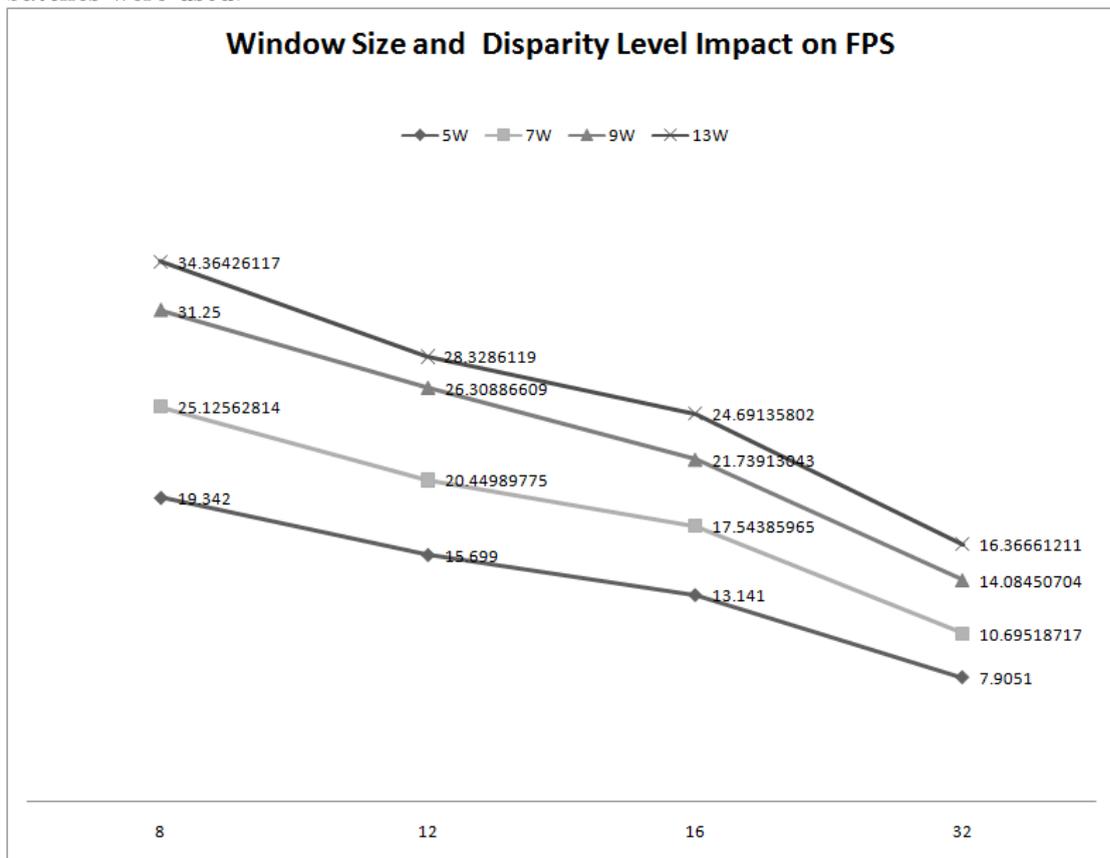


FIGURE 4.2: FPS degrades with higher disparity levels and with smaller pixel block sizes

4.2.1 Experimental Disparity Maps

We used our stereo algorithm and by tweaking pixel block sizes and disparity levels on the same image pair we generated the following disparity maps, we present it for the reader to get a feeling of the trade offs one makes when adjusting these very important

parameters, some applications where only rough information and topology is required might suffice on using 8 disparity levels and large block size but benefit from an extremely high framerate where others might have stricter requirements. In case small disparity levels are chosen then implicitly the user loses objects close to the camera (since the items close to the camera will have quite large disparities) while retaining those in the far field the exact distance can be tweaked by adjusting camera focus.

4.2.2 The Million Disparity per Second Metric

During our tests we noticed that the application did not respond linearly to the increase of the disparity levels as such the disparity per second metric without providing the corresponding disparity levels that provided for the measurement is incomplete, for instance we have a measured maximum disparity of 55md/s in the best case with 16 levels of maximum disparity however when testing for 32 disparity levels while our FPS suffers greatly it does not fall as much thus actually providing for an increased md/s of 1.22, thus had we used 32 Disparity levels instead of 16 for our comparisons our md/s metric would be 72 md/s instead of 55 md/s, it is clear that the md/s metric alone is ambiguous and should be followed by the pixel block and window correlation. In our application specifically this behavior is justified because as analyzed in the previous chapter, about 30% of the time is spend on memory transfers and the buffer transporting the data is unaffected by any change in disparity hence while our formula for calculating disparity is doubled (16 to 32) the corresponding FPS does not drop for the same amount thus leading us to a higher disparity per second.

4.3 Summary

In this section we presented the results of our implemented system. We can summarize the gathered data into two main categories the performance of the DSP system and what was the impact of each optimization step by step (table 4.3, the second interesting category is how our stereo vision approach of having a pixel block that matches the size of the correlation kernel affects performance and a host of disparity maps were presented in figures 4.4 ,4.6. Also the relative debunking of the metric million disparities per second, at least for hybrid systems should not go unnoticed as an increase to disparity levels did not lead to the expected drop in FPS, due to the memory bottleneck previously described leading us to a higher relative disparity from the reported 59.064 md/s to 89md/s.

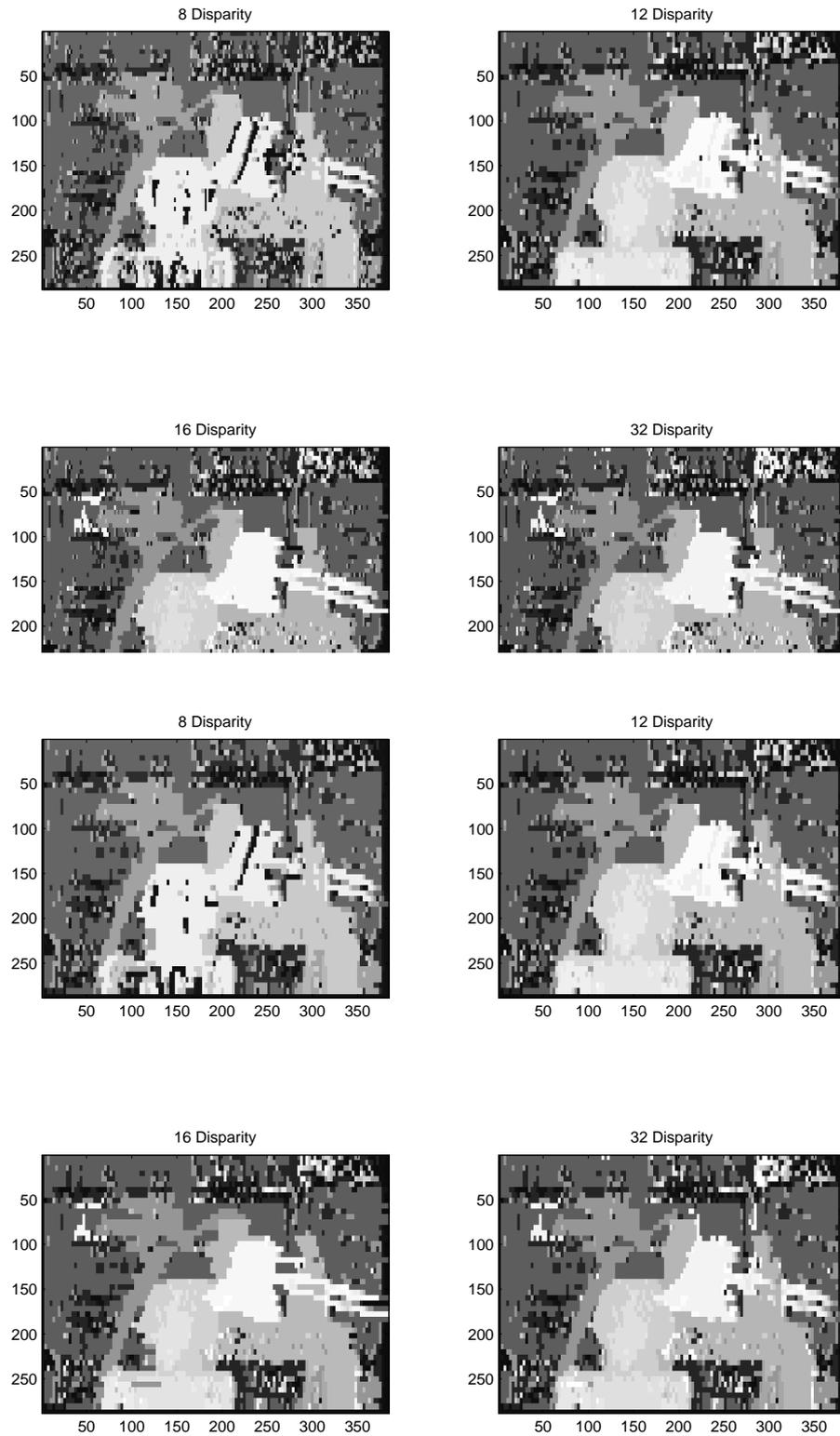


FIGURE 4.4: Disparity Maps for block sizes 5 (first four) and 7 (later)

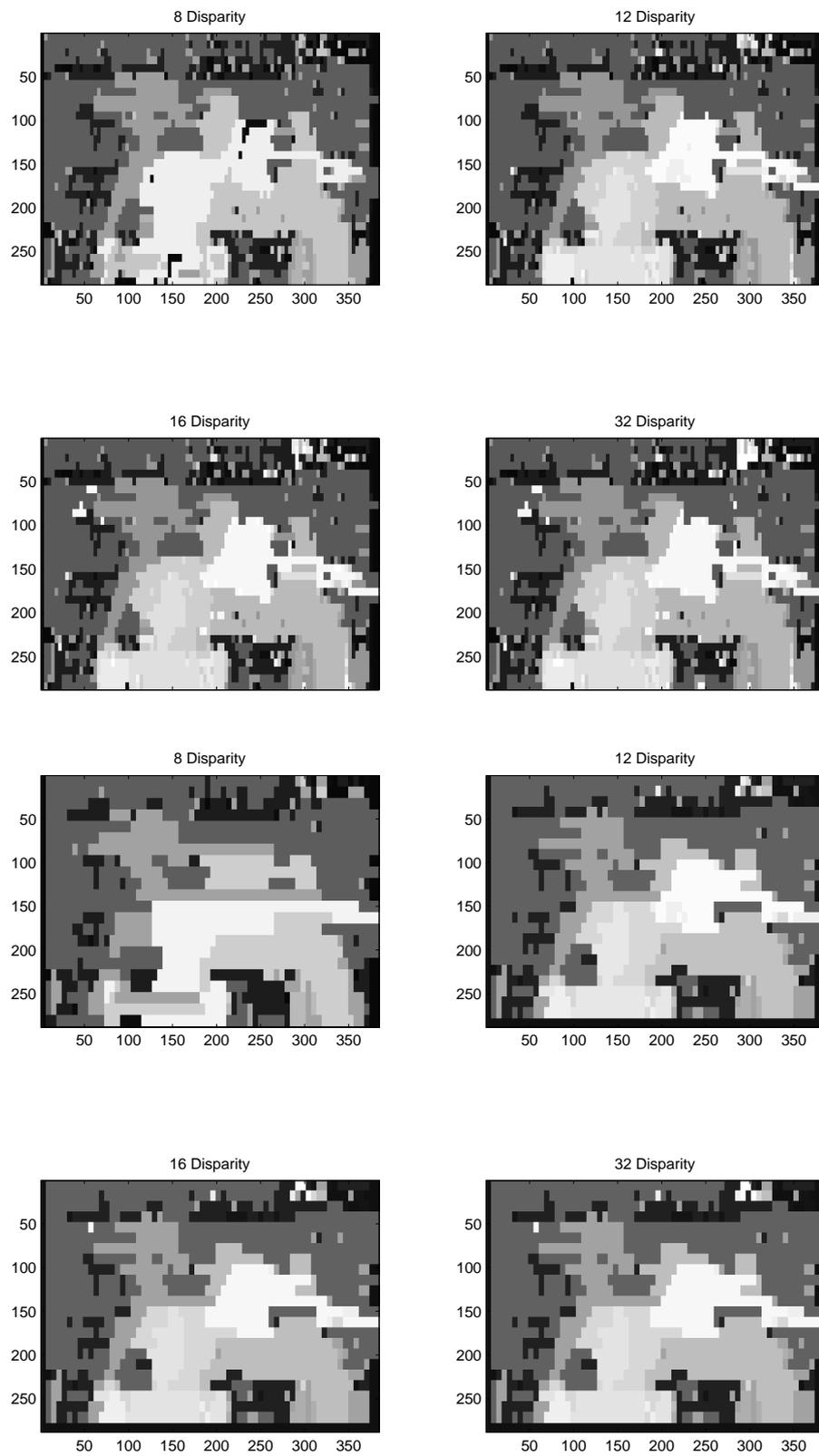


FIGURE 4.6: Disparity Maps for block sizes 9 (first four) and 13 (later)

Lastly in figure 4.7 we see all the commonly used metrics (in million disparities) along with the newly proposed million disparities per Ghz metric.

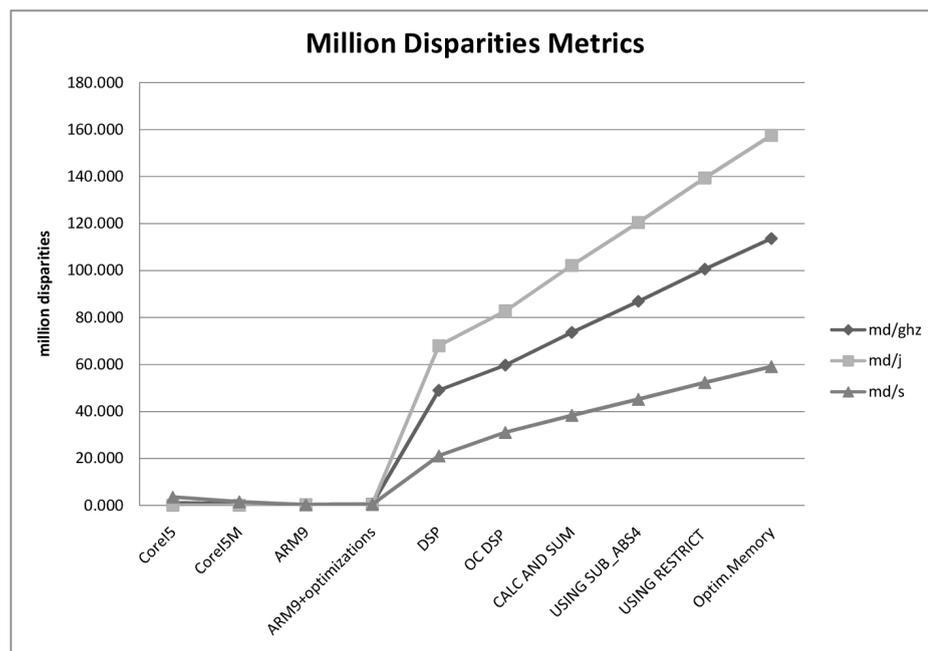


FIGURE 4.7: MillionDisparitiesMetrics summary chart

Chapter 5

Conclusions

Conclusions

The problem studied in this Thesis, was a rather complicated one, merging challenges from two distinct fields, the field of embedded computers and the field of computer vision. During this rather long journey a lot of the aforementioned challenges were solved, step by step, applying the iterative research and development process analyzed in chapter 3, until reaching the final results. In this process however we designed and implemented a system that can compete with the current state of the art in both energy efficiency, with less than $4 * 10^{-3}$ Joule per frame, marking it as one of the most power efficient systems in the field, with more than 33 frames per second in $384 * 288$ images. The results greatly exceeded our expectations but our contribution does not stop with the proposed novel system. The byproducts of this work show interesting results and arouse some questions about the commonly used metrics such as md/sec as we showed that it is not valid for all systems, as in our case memory bandwidth is an issue, hence the time cost of copying a memory buffer remains unaffected by increasing disparity thus leading to higher md/sec when increasing disparity levels while not necessarily providing better results. While this might be negligible for monolithic systems it should not be overlooked for modular or hybrid systems. Furthermore the proposed metric of md/ghz (million disparities per Ghz) shows a new, more consistent in our opinion way, of comparing similar systems; denotes at the same time architectural differences and pros one system might have to another (see DSP vs CPU on figure 4.7) in our case it allowed us to notice that over clocking our system has other hidden effects, that lead to higher performance and this is easily noticed from the increase in md/Ghz.

Our proposed novel system also bridges the world of linux and that of DSP, providing the benefits of both worlds, an important notion for development and research alike when

coupled with the design and implementation practices proposed; the low cost and high availability of such devices also provide an easily available platform; the possibility that hybrid devices such as the one used in this Thesis will be massively available by many, signals that research and implementation on such novel devices has only just begun and will certainly see more use. However as important all the aforementioned results might be, the author believes, that the biggest contribution of this work is bringing hybrid and modular embedded systems in the spotlight unearthing their power, showing that with proper design and data flow one might achieve tremendous results in otherwise complicated and previously hands off problems, for the embedded world , due to high processing demands. Using our proposed client server interprocessor design one can simply remove the stereo system code and insert his own while at the same time following the practices mentioned and evaluated in this work, with minimal change, since our interprocessor medium is simply a buffer one can achieve massive speedups with minimal development. Lastly while our initial goal was to generate 15FPS on QVGA images we achieved more than 40 FPS bringing our implementation on par with even some FPGA systems while at the same time having the arm processor free to be used for any other task creating a system that is on par with the state of the art having only as rival at the time of this writing the work of [13] who have mentioned 46 FPS. Lastly we have showed the weakness of md/S metric to describe performance unless accompanied by the disparity levels performed especially for heterogeneous systems. Lastly in the field of Stereo Vision and especially in the level of embedded devices, we have studied the commonly used metrics that are applied when comparing and evaluating work in this field. We believe and have so highlighted, that some issues arise in the million disparity metric, as there is ambiguity on how to calculate this rather commonly used metric for example in our system we receive higher disparity metric when using 64 disparity levels even though the test image requires only 16 and benefits slightly to no from this increase. The aforementioned issue directly affects all disparity metrics (md/joule, md/second, md/Ghz and so on) we thus propose that the disparity levels used to calculate the million disparity levels are always mentioned so there can be a more precise evaluation.

5.1 Future Work

From the work done in this Thesis there are two main paths for improvements and potential future work. Firstly in our currently used system not all power has been exploited, since we have an ARM Cortex A-8 unit that has NEON capabilities it would be possible to use the floating point unit using the NEON instruction set to calculate VGA or higher depth maps by splitting a stereo image frame between the processors, this is possible because using the epipolar constraint a match can only occur in a horizontal line allowing us

to split frames in as many pieces as required as long as each part contains entire lines it is possible to simply resynthesize from the depth map parts a complete and correct total image depth map. For applications requiring VGA or higher resolution images this can provide with results similar to those presented in this work for the Tsukuba Image Pair(384*288).

Secondly future devices show promise or advertise that OpenCL enabled GPU's will replace current solutions like the PowerSVG core present in our system, providing an OpenCL enabled GPU alongside DSP and a CPU unit provides immense capabilities for our stereo application a further division will provide access to real time performance on even higher resolution images with minimal code where for other applications extending our client server model with issue reclaim buffers will allow for a software pipelining to solve extremely demanding problems.

Another interesting and important improvement that can benefit the proposed system is the use of two camera sensors on the camera interface bus, instead of using only one in the camera interface and one USB, this will surely lead to lower power consumption but the use of a common bus will need clever multiplexing and camera synchronization possible requiring custom hardware. As for the software we currently use to calculate the disparity maps since the .M units of the DSP are not used it would be feasible for one to replace the corresponding function with a more complex one possible providing better results while sustaining the current FPS. Lastly new hybrid units with different hardware such as the FPGA system Zinc that provides a GPP unit with an FPGA could be implemented and compared with the proposed system. Exploration and implementation of similar processing intensive problems using the proposed software application .

Chapter 6

The build path, fallacies and pitfalls

A simple overview of the tasked perform will reveal that several distinct and vastly different platforms are implicated, as is true to even mundane tasks that involve multiple different components such a case is errorprone and even a simple error or miss configuration can cause mayhem and countless hours of delay. Due to prior experience with similar embedded work, we tried to avoid and prepare for errors and save time and energy, however a host of problems were encountered and dealt with while working in this thesis. Mainly the source of troubles was the interprocessor communication as debugging and monitoring was difficult , since excellent debug tools exist to monitor situations that occur in the ARM7 Cortex A-8 processor or the DSP processor (Texas instrument's Code composer studio line is the main one) there are no tools that allow you to adequately configure and monitor both of the processors operating in the current gumstix package where complexity arises even more by the different modules. Having the linux operation system provided a host of tools to ease our task but the poor documentation and relatively few similar projects (only one actually) provided with limited insight. Furthermore the absence of the omap3530 development kit that provided such functionality using the external jtag hardware further implicated our debugging process, thus evaluating all these problems we followed the top down procedure analytically described in the implementation section, the particular steps followed allowed us to debug our source c code using the matlab environment mainly for visualization purposes thus at the end of the procedure we had a c program that was verified to work properly and had no bugs, even more we managed to profile and identify the most processing intensive parts of our system and this aided us in designing our system.

6.1 Time Distribution

The work presented hereby took approximately 800 hours to complete. A rough estimation of the time allocation to each phase during the work in this Thesis follows :

- Background And Theory : 20 %
- Matlab Profiling and Debugging :15 %
- GCC profiling in native Code :15 %
- DSP transition problem resolution and cross compilation: 40 %
- OpenEmbedded Toolchain and Environment Generation :10 % As it is evident the lack of tools and the shortage of examples and relative work in the platform took its toll as an otherwise well defined task consumed substantial amounts of time, however the task might not be achievable at all without the proper profign and debugging performed in the matlab level as the clean code generated through that step allowed us to focus elsewhere without doubt of our programmes correctness. Time spend in Background and relevant Theory cannot be overestimate even more so a higher time allocation would be better fitting. Laslty the open embedded generation task was a tedious at the time the vast majority of this Thesis took place, however recent advancements to the Yocto project allows for a much cleaner and faster solution and is strongly recommended that the older distribution and build system is avoided all togher. As for the GCC cross compilation chain, the tools are state of the art and its correspondance with the normal linux procedures for compilation provided with a friendly and well defined development process where we could design,develop and test seamlessly while exploiting our programming knowledge, the same cannot be said for the TI dsp tools as incopatibilities with normal GCC options provided problems where they could be avoided an example to this case is the -o flag where as known in the gcc environment provides with the name of an output file where in the ctg6l compiler it is a completely different option that invokes the linker.

Appendix A

The build path, fallacies and pitfalls

A simple overview of the tasked perform will reveal that several distinct and vastly different platforms are implicated, as is true to even mundane tasks that involve multiple different components such a case is errorprone and even a simple error or miss configuration can cause mayhem and countless hours of delay. Due to prior experience with similar embedded work, we tried to avoid and prepare for errors and save time and energy, however a host of problems were encountered and dealt with while working in this thesis. Mainly the source of troubles was the interprocessor communication as debugging and monitoring was difficult , since excellent debug tools exist to monitor situations that occur in the ARM7 Cortex A-8 processor or the DSP processor (Texas instrument's Code composer studio line is the main one) there are no tools that allow you to adequately configure and monitor both of the processors operating in the current gumstix package where complexity arises even more by the different modules. Having the linux operation system provided a host of tools to ease our task but the poor documentation and relatively few similar projects (only one actually) provided with limited insight. Furthermore the absence of the omap3530 development kit that provided such functionality using the external jtag hardware further implicated our debugging process, thus evaluating all these problems we followed the top down procedure analytically described in the implementation section, the particular steps followed allowed us to debug our source c code using the matlab environment mainly for visualization purposes thus at the end of the procedure we had a c program that was verified to work properly and had no bugs, even more we managed to profile and identify the most processing intensive parts of our system and this aided us in designing our system.

A.1 Time Distribution

The work presented hereby took approximately 800 hours to complete. A rough estimation of the time allocation to each phase during the work in this Thesis follows :

- Background And Theory : 20 %
- Matlab Profiling and Debugging :15 %
- GCC profiling in native Code :15 %
- DSP transition problem resolution and cross compilation: 40 %
- OpenEmbedded Toolchain and Environment Generation :10 % As it is evident the lack of tools and the shortage of examples and relative work in the platform took its toll as an otherwise well defined task consumed substantial amounts of time, however the task might not be achievable at all without the proper profign and debugging performed in the matlab level as the clean code generated through that step allowed us to focus elsewhere without doubt of our programmes correctness. Time spend in Background and relevant Theory cannot be overestimate even more so a higher time allocation would be better fitting. Laslty the open embedded generation task was a tedious at the time the vast majority of this Thesis took place, however recent advancements to the Yocto project allows for a much cleaner and faster solution and is strongly recommended that the older distribution and build system is avoided all togher. As for the GCC cross compilation chain, the tools are state of the art and its correspondance with the normal linux procedures for compilation provided with a friendly and well defined development process where we could design,develop and test seamlessly while exploiting our programming knowledge, the same cannot be said for the TI dsp tools as incopatibilities with normal GCC options provided problems where they could be avoided an example to this case is the -o flag where as known in the gcc environment provides with the name of an output file where in the ctg6l compiler it is a completely different option that invokes the linker.

Appendix B

Device Schematics And Info

B.1 OMAP3530

As stated numerous times, the OMAP3530 S.o.C, is a powerful heterogenous system developed by Texas Instruments; this system contains numerous processing units and subsystems; the material presented hereby is property of Texas Instruments¹ corporation and serves only as a reference in the context of the work done in this Thesis and for academical purposes only.

- Omap3530 S.o.C schematic
- Omap3530 S.o.C datasheet

¹<http://www.ti.com/>

1 OMAP3530/25 Applications Processor

1.1 Features

- **OMAP3530/25 Applications Processor:**
 - **OMAP™ 3 Architecture**
 - **MPU Subsystem**
 - Up to 720-MHz ARM Cortex™-A8 Core
 - NEON™ SIMD Coprocessor
 - **High Performance Image, Video, Audio (IVA2.2™) Accelerator Subsystem**
 - Up to 520-MHz TMS320C64x+™ DSP Core
 - Enhanced Direct Memory Access (EDMA) Controller (128 Independent Channels)
 - Video Hardware Accelerators
 - **POWERVR SGX™ Graphics Accelerator (OMAP3530 Device Only)**
 - Tile Based Architecture Delivering up to 10 MPoly/sec
 - Universal Scalable Shader Engine: Multi-threaded Engine Incorporating Pixel and Vertex Shader Functionality
 - Industry Standard API Support: OpenGLES 1.1 and 2.0, OpenVG1.0
 - Fine Grained Task Switching, Load Balancing, and Power Management
 - Programmable High Quality Image Anti-Aliasing
 - **Fully Software-Compatible With C64x and ARM9™**
 - **Commercial and Extended Temperature Grades**
- **Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x+™ DSP Core**
 - **Eight Highly Independent Functional Units**
 - +Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle
 - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
 - **Load-Store Architecture With Non-Aligned Support**
 - **64 32-Bit General-Purpose Registers**
 - **Instruction Packing Reduces Code Size**
 - **All Instructions Conditional**
- **Additional C64x+™ Enhancements**
 - Protected Mode Operation
 - Exceptions Support for Error Detection and Program Redirection
 - Hardware Support for Modulo Loop Operation
- **C64x+ L1/L2 Memory Architecture**
 - 32K-Byte L1P Program RAM/Cache (Direct Mapped)
 - 80K-Byte L1D Data RAM/Cache (2-Way Set-Associative)
 - 64K-Byte L2 Unified Mapped RAM/Cache (4-Way Set-Associative)
 - 32K-Byte L2 Shared SRAM and 16K-Byte L2 ROM
- **C64x+ Instruction Set Features**
 - Byte-Addressable (8-/16-/32-/64-Bit Data)
 - 8-Bit Overflow Protection
 - Bit-Field Extract, Set, Clear
 - Normalization, Saturation, Bit-Counting
 - Compact 16-Bit Instructions
 - Additional Instructions to Support Complex Multiplies
- **ARM Cortex™-A8 Core**
 - **ARMv7 Architecture**
 - Trust Zone®
 - Thumb®-2
 - MMU Enhancements
 - **In-Order, Dual-Issue, Superscalar Microprocessor Core**
 - **NEON™ Multimedia Architecture**
 - **Over 2x Performance of ARMv6 SIMD**
 - **Supports Both Integer and Floating Point SIMD**
 - **Jazelle® RCT Execution Environment Architecture**
 - **Dynamic Branch Prediction with Branch Target Address Cache, Global History Buffer, and 8-Entry Return Stack**
 - **Embedded Trace Macrocell (ETM) Support for Non-Invasive Debug**
- **ARM Cortex™-A8 Memory Architecture:**
 - **16K-Byte Instruction Cache (4-Way Set-Associative)**



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

POWERVR SGX is a trademark of Imagination Technologies Ltd.

OMAP is a trademark of Texas Instruments.

All other trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2008–2009, Texas Instruments Incorporated

- 16K-Byte Data Cache (4-Way Set-Associative)
- 256K-Byte L2 Cache
- 112K-Byte ROM
- 64K-Byte Shared SRAM
- Endianness:
 - ARM Instructions - Little Endian
 - ARM Data – Configurable
 - DSP Instruction/Data - Little Endian
- External Memory Interfaces:
 - SDRAM Controller (SDRC)
 - 16, 32-bit Memory Controller With 1G-Byte Total Address Space
 - Interfaces to Low-Power Double Data Rate (LPDDR) SDRAM
 - SDRAM Memory Scheduler (SMS) and Rotation Engine
 - General Purpose Memory Controller (GPMC)
 - 16-bit Wide Multiplexed Address/Data Bus
 - Up to 8 Chip Select Pins With 128M-Byte Address Space per Chip Select Pin
 - Glueless Interface to NOR Flash, NAND Flash (With ECC Hamming Code Calculation), SRAM and Pseudo-SRAM
 - Flexible Asynchronous Protocol Control for Interface to Custom Logic (FPGA, CPLD, ASICs, etc.)
 - Nonmultiplexed Address/Data Mode (Limited 2K-Byte Address Space)
- System Direct Memory Access (sDMA) Controller (32 Logical Channels With Configurable Priority)
- Camera Image Signal Processing (ISP)
 - CCD and CMOS Imager Interface
 - Memory Data Input
 - RAW Data Interface
 - BT.601/BT.656 Digital YCbCr 4:2:2 (8-/10-Bit) Interface
 - A-Law Compression and Decompression
 - Preview Engine for Real-Time Image Processing
 - Glueless Interface to Common Video Decoders
 - Histogram Module/Auto-Exposure, Auto-White Balance, and Auto-Focus Engine
 - Resize Engine
 - Resize Images From 1/4x to 4x
 - Separate Horizontal/Vertical Control
- Display Subsystem
 - Parallel Digital Output
 - Up to 24-Bit RGB
 - HD Maximum Resolution
 - Supports Up to 2 LCD Panels
 - Support for Remote Frame Buffer Interface (RFBI) LCD Panels
 - 2 10-Bit Digital-to-Analog Converters (DACs) Supporting:
 - Composite NTSC/PAL Video
 - Luma/Chroma Separate Video (S-Video)
 - Rotation 90-, 180-, and 270-degrees
 - Resize Images From 1/4x to 8x
 - Color Space Converter
 - 8-bit Alpha Blending
- Serial Communication
 - 5 Multichannel Buffered Serial Ports (McBSPs)
 - 512 Byte Transmit/Receive Buffer (McBSP1/3/4/5)
 - 5K-Byte Transmit/Receive Buffer (McBSP2)
 - SIDETONE Core Support (McBSP2 and 3 Only) For Filter, Gain, and Mix Operations
 - Direct Interface to I2S and PCM Device and TDM Buses
 - 128 Channel Transmit/Receive Mode
 - Four Master/Slave Multichannel Serial Port Interface (McSPI) Ports
 - High-Speed/Full-Speed/Low-Speed USB OTG Subsystem (12-/8-Pin ULPI Interface)
 - High-Speed/Full-Speed/Low-Speed Multiport USB Host Subsystem
 - 12-/8-Pin ULPI Interface or 6-/4-/3-Pin Serial Interface
 - Supports Transceiverless Link Logic (TLL)
 - One HDQ/1-Wire Interface
 - Three UARTs (One with Infrared Data Association [IrDA] and Consumer Infrared [CIR] Modes)
 - Three Master/Slave High-Speed Inter-Integrated Circuit (I2C) Controllers
- Removable Media Interfaces:
 - Three Multimedia Card (MMC)/ Secure Digital (SD) With Secure Data I/O (SDIO)
- Comprehensive Power, Reset, and Clock Management
 - SmartReflex™ Technology
 - Dynamic Voltage and Frequency Scaling (DVFS)
- Test Interfaces
 - IEEE-1149.1 (JTAG) Boundary-Scan Compatible

- Embedded Trace Macro Interface (ETM)
- Serial Data Transport Interface (SDTI)
- 12 32-bit General Purpose Timers
- 2 32-bit Watchdog Timers
- 1 32-bit 32-kHz Sync Timer
- Up to 188 General-Purpose I/O (GPIO) Pins (Multiplexed With Other Device Functions)
- 65-nm CMOS Technology
- Package-On-Package (POP) Implementation for Memory Stacking (Not Available in CUS Package)
- Discrete Memory Interface (Not Available in CBC Package)
- Packages:
 - 515-pin s-PBGA package (CBB Suffix), .5mm Ball Pitch (Top), .4mm Ball Pitch (Bottom)
 - 515-pin s-PBGA package (CBC Suffix), .65mm Ball Pitch (Top), .5mm Ball Pitch (Bottom)
 - 423-pin s-PBGA package (CUS Suffix), .65mm Ball Pitch
- 1.8-V I/O and 3.0-V (MMC1 only), 0.985-V to 1.35-V Adaptive Processor Core Voltage
0.985-V to 1.35-V Adaptive Core Logic Voltage
Note: These are default Operating Performance Point (OPP) voltages and could be optimized to lower values using SmartReflex™ AVS.
- Applications:
 - Portable Navigation Devices
 - Portable Media Player
 - Advanced Portable Consumer Electronics
 - Digital TV
 - Digital Video Camera
 - Portable Data Collection
 - Point-of-Sale Devices
 - Gaming
 - Web Tablet
 - Smart White Goods
 - Smart Home Controllers
 - Ultra Mobile Devices

1.3 Functional Block Diagram

Figure 1-1 shows the functional block diagram of the OMAP3530/25 Applications Processor.

OMAP Applications Processor

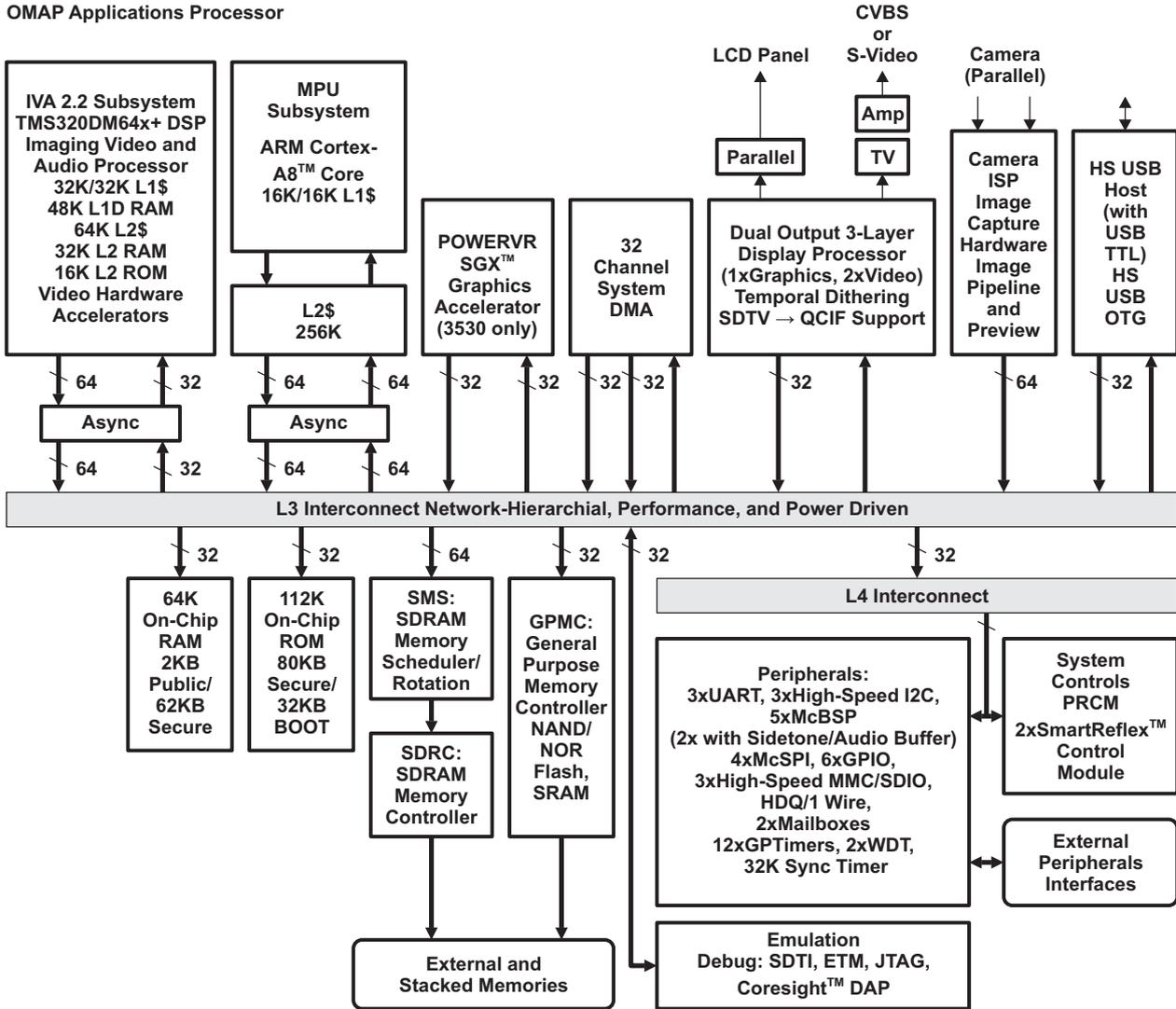


Figure 1-1. OMAP3530/25 Functional Block Diagram

B.1.1 Arm Cortex A-8

- Arm Cortex A-8 S.o.C schematic

Cortex[™]-A8

Revision: r2p1

Technical Reference Manual

ARM[®]

Copyright © 2006-2007 ARM Limited. All rights reserved.
ARM DDI 0344D

Cortex-A8

Technical Reference Manual

Copyright © 2006-2007 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
18 July 2006	A	Confidential	First release for r1p0
13 December 2006	B	Non-Confidential	First release for r1p1
13 July 2007	C	Non-Confidential	First release for r2p0
16 November 2007	D	Non-Confidential	First release for r2p1

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Some material in this document is based on *ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic* and on *IEEE Std. 1500-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

<http://www.arm.com>

B.1.2 TMS320C64+

- TMS320C64+, Taken from page 13 of the of TI "TMS320C6413, TMS320C6410 Fixed-Point Digital Signal Processors" Data Manual with Literature Number SPRS247F.
- Datasheet The following schematic is from the TMS320C64x+ DSP Megamodule manual with Literature Number: SPRU871K.

1 Features

- **High-Performance Fixed-Point Digital Signal Processor (TMS320C6413/C6410)**
 - TMS320C6413
 - 2-ns Instruction Cycle Time
 - 500-MHz Clock Rate
 - 4000 MIPS
 - TMS320C6410
 - 2.5-ns Instruction Cycle Time
 - 400-MHz Clock Rate
 - 3200 MIPS
 - Eight 32-Bit Instructions/Cycle
 - Fully Software-Compatible With C64x™
 - Extended Temperature Devices Available
- **VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core**
 - Eight Highly Independent Functional Units With VelociTI.2™ Extensions:
 - Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle
 - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
 - Load-Store Architecture With Non-Aligned Support
 - 64 32-Bit General-Purpose Registers
 - Instruction Packing Reduces Code Size
 - All Instructions Conditional
- **Instruction Set Features**
 - Byte-Addressable (8-/16-/32-/64-Bit Data)
 - 8-Bit Overflow Protection
 - Bit-Field Extract, Set, Clear
 - Normalization, Saturation, Bit-Counting
 - VelociTI.2™ Increased Orthogonality
- **VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x™ DSP Core**
 - **L1/L2 Memory Architecture**
 - 128K-Bit (16K-Byte) L1P Program Cache (Direct Mapped)
 - 128K-Bit (16K-Byte) L1D Data Cache (2-Way Set-Associative)
 - 2M-Bit (256K-Byte) L2 Unified Mapped RAM/Cache [C6413] (Flexible RAM/Cache Allocation)
 - 1M-Bit (128K-Byte) L2 Unified Mapped RAM/Cache [C6410] (Flexible RAM/Cache Allocation)
 - **Endianess: Little Endian, Big Endian**
 - **32-Bit External Memory Interface (EMIF)**
 - Glueless Interface to Asynchronous Memories (SRAM and EPROM) and Synchronous Memories (SDRAM, SBSRAM, ZBT SRAM, and FIFO)
 - 512M-Byte Total Addressable External Memory Space
 - **Enhanced Direct-Memory-Access (EDMA) Controller (64 Independent Channels)**
 - **Host-Port Interface (HPI) [32-/16-Bit]**
 - **Two Multichannel Audio Serial Ports (McASPs) - with Six Serial Data Pins each**
 - **Two Inter-Integrated Circuit (I²C) Buses**
 - Additional GPIO Capability
 - **Two Multichannel Buffered Serial Ports**
 - **Three 32-Bit General-Purpose Timers**
 - **Sixteen General-Purpose I/O (GPIO) Pins**
 - **Flexible PLL Clock Generator**
 - **On-Chip Fundamental Oscillator**
 - **IEEE-1149.1 (JTAG†) Boundary-Scan-Compatible**
 - **288-Pin Ball Grid Array (BGA) Packages (GTS and ZTS Suffixes), 1.0-mm Ball Pitch**
 - **0.13-μm/6-Level Cu Metal Process (CMOS)**
 - **3.3-V I/Os, 1.2-V Internal**

VelociTI.2, VelociTI, and TMS320C64x are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

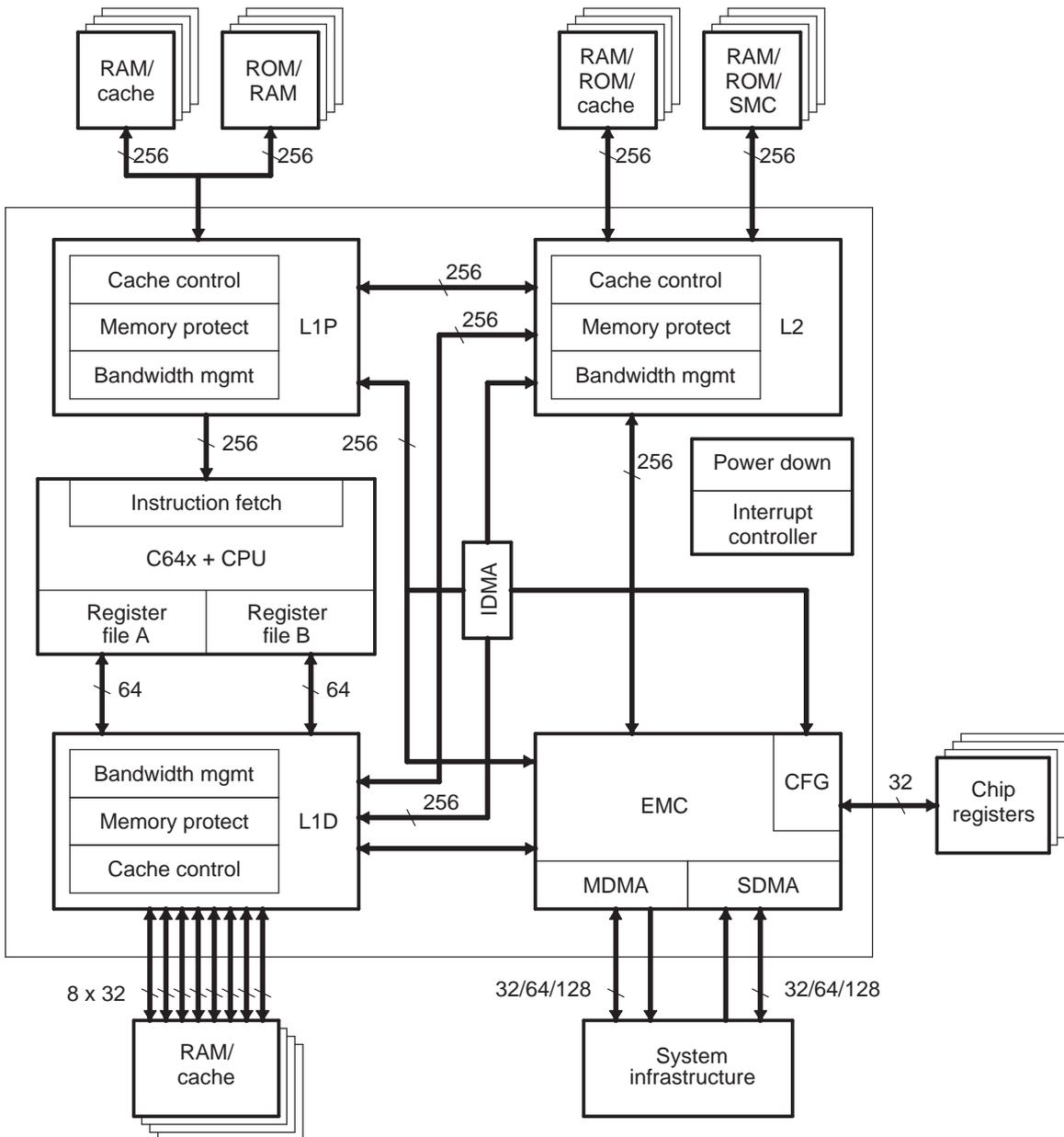
† IEEE Standard 1149.1-1990 Standard-Test-Access Port and Boundary Scan Architecture.

1.1 Introduction

The C64x+™ megamodule includes the following components: C64x+ CPU, Level 1 program (L1P) memory controller, Level 1 data (L1D) memory controller, Level 2 (L2) memory controller, Internal DMA (IDMA), bandwidth management (BWM), interrupt controller (INTC), power-down controller (PDC), and an extended memory controller (EMC).

A block diagram of the megamodule is shown in [Figure 1-1](#).

Figure 1-1. TMS320C64x+ Megamodule Block Diagram



B.1.3 POWERVR

- Powervrsgx530

Graphics Accelerator (SGX)

This chapter describes the graphics accelerator for the device.

Topic	Page
5.1 Introduction	184
5.2 Integration	187
5.3 Functional Description	189

5.1 Introduction

This chapter describes the 2D/3D graphics accelerator (SGX) for the device.

NOTE: The SGX subsystem is a Texas Instruments instantiation of the POWERVR® SGX530 core from Imagination Technologies Ltd.

This document contains materials that are ©2003-2007 Imagination Technologies Ltd.

POWERVR® and USSE™ are trademarks or registered trademarks of Imagination Technologies Ltd.

The 2D/3D graphics accelerator (SGX) subsystem accelerates 2-dimensional (2D) and 3-dimensional (3D) graphics applications. The SGX subsystem is based on the POWERVR® SGX core from Imagination Technologies. SGX is a new generation of programmable POWERVR graphic cores. The POWERVR SGX530 v1.2.5 architecture is scalable and can target all market segments from mainstream mobile devices to high-end desktop graphics. Targeted applications include feature phone, PDA, and hand-held games.

5.1.1 POWERVR SGX Main Features

- 2D graphics, 3D graphics, vector graphics, and programming support for GP-GPU functions
- Tile-based architecture
- Universal scalable shader engine (USSE™) – multithreaded engine incorporating pixel and vertex shader functionality
- Advanced shader feature set – in excess of Microsoft VS3.0, PS3.0, and OpenGL2.0
- Industry-standard API support – Direct3D Mobile, OpenGL ES 1.1 and 2.0, OpenVG v1.0.1
- Fine-grained task switching, load balancing, and power management
- Advanced geometry direct memory access (DMA) driven operation for minimum CPU interaction
- Programmable high-quality image anti-aliasing
- POWERVR SGX core MMU for address translation from the core virtual address to the external physical address (up to 4GB address range)
- Fully virtualized memory addressing for OS operation in a unified memory architecture
- Advanced and standard 2D operations [e.g., vector graphics, BLTs (block level transfers), ROPs (raster operations)]
- 32K stride support

5.1.2 SGX 3D Features

- Deferred pixel shading
- On-chip tile floating point depth buffer
- 8-bit stencil with on-chip tile stencil buffer
- 8 parallel depth/stencil tests per clock
- Scissor test
- Texture support:
 - Cube map
 - Projected textures
 - 2D textures
 - Nonsquare textures
- Texture formats:
 - RGBA 8888, 565, 1555
 - Monochromatic 8, 16, 16f, 32f, 32int
 - Dual channel, 8:8, 16:16, 16f:16f

Bibliography

- [1] R.B. Porter and N.W. Bergmann. A generic implementation framework for FPGA based stereo matching. In , *Proceedings of IEEE TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications*, volume 2, pages 461–464 vol.2, 1997. doi: 10.1109/TENCON.1997.648244.
- [2] Bernard Hotz Olivier Faugeras. Real time correlation-based stereo: algorithm, implementations and applications.
- [3] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *IEEE Workshop on Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings*, pages 131–140, 2001. doi: 10.1109/SMBV.2001.988771.
- [4] Nalpantidis Lazaros, Georgios Christou Sirakoulis, and Antonios Gasteratos. Review of stereo vision algorithms: From software to hardware. *International Journal of Optomechatronics*, 2(4):435–462, November 2008. ISSN 1559-9612, 1559-9620. doi: 10.1080/15599610802438680. URL <http://kth.diva-portal.org/smash/record.jsf?pid=diva2:463163>.
- [5] D.K. Masrani and W.J. MacLean. A real-time large disparity range stereo-system using FPGAs. In *IEEE International Conference on Computer Vision Systems, 2006 ICVS '06*, pages 13–13, 2006. doi: 10.1109/ICVS.2006.6.
- [6] A. Darabiha, J. Rose, and J.W. Maclean. Video-rate stereo depth measurement on programmable hardware. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings*, volume 1, pages I–203–I–210 vol.1, 2003. doi: 10.1109/CVPR.2003.1211355.
- [7] M.A.I. Manzano, D.L.A. Ojeda, M. Devy, J.L. Boizard, and J-Y Fourniols. Stereo vision algorithm implementation in FPGA using census transform for effective resource optimization. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009. DSD '09*, pages 799–805, 2009. doi: 10.1109/DSD.2009.159.

- [8] N. Chang, Ting-Min Lin, Tsung-Hsien Tsai, Yu-Cheng Tseng, and Tian-Sheuan Chang. Real-time DSP implementation on local stereo matching. In *2007 IEEE International Conference on Multimedia and Expo*, pages 2090–2093, 2007. doi: 10.1109/ICME.2007.4285094.
- [9] Chyi-Yeu Lin and Yi-Pin Chiu. The DSP based catcher robot system with stereo vision. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2008. AIM 2008*, pages 897–903, 2008. doi: 10.1109/AIM.2008.4601780.
- [10] Chyi-Yeu Lin Chyi-Yeu Lin, Yi-Pin Chiu Yi-Pin Chiu, and Chi-Ying Lin Chi-Ying Lin. Robot catching system with stereo vision and DSP platform. *DeepDyve*, January 2011. URL <http://www.deepdyve.com/lp/institute-of-electrical-and-electronics-engineers/robot-catching-system-with-stereo-vision-and-dsp-platform-r50gyFb3QQ>.
- [11] S. Kimura, T. Shinbo, H. Yamaguchi, E. Kawamura, and K. Nakano. A convolver-based real-time stereo machine (SAZAN). In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages –463 Vol. 1, 1999. doi: 10.1109/CVPR.1999.786978.
- [12] J.I. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck. The tyzx DeepSea g2 vision system, ATaskable, embedded stereo camera. In *Conference on Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06*, pages 126–126, 2006. doi: 10.1109/CVPRW.2006.202.
- [13] S.B. Goldberg and L. Matthies. Stereo and IMU assisted visual odometry on an OMAP3530 for small robots. In *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 169–176, 2011. doi: 10.1109/CVPRW.2011.5981842.
- [14] J. Woodfill and B. Von Herzen. Real-time stereo vision on the PARTS reconfigurable computer. In *, The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1997. Proceedings*, pages 201–210, 1997. doi: 10.1109/FPGA.1997.624620.
- [15] Seunghun Jin, Junguk Cho, Xuan Dai Pham, Kyoung-Mu Lee, Sung-Kee Park, Munsang Kim, and J.W. Jeon. FPGA design and implementation of a real-time stereo vision system. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(1):15–26, 2010. ISSN 1051-8215. doi: 10.1109/TCSVT.2009.2026831.
- [16] Apostolos Dollas Sotiris Thomas, Kyprianos Papadimitriou. Fpga-based architecture and design for real-time 3d stereo vision. Proc VLSI SoC,Instabul, 2013.

-
- [17] Apostolos Dollas Georgia Rematska, Kyprianos Papadimitriou. A low-cost embedded real-time 3d stereo matching system for surveillance applications. 2013.
- [18] F. Diotalevi, A. Fijany, M. Montvelishsky, and J. Fontaine. Very low power parallel implementation of stereo vision algorithm on a solar cell powered MIMD many core architecture. In *2011 IEEE Aerospace Conference*, pages 1–13, 2011. doi: 10.1109/AERO.2011.5747451.
- [19] Markus Achtelik, Abraham Bachrach, Ruijie He, Samuel Prentice, and Nicholas Roy. {Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments}. page 7332. SPIE, 2010. URL <http://dspace.mit.edu/handle/1721.1/52660>.