Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2011

MSc THESIS

Modular RT-Motion USB

Serge Keyser

Abstract

During the course of this thesis, RTM-USB (Real Time Motion on Universal Serial Bus) has grown from a single board motor controller to a motion control platform. The original RTM-USB board, containing a CPU and two motor drivers, has been extended (*hardware* wise) with a network/bus interface which makes it easy to expand the hardware functionality of the platform. Research has been undertaken in order to see which hardware extensions would be interesting for implementation on this platform. In order to demonstrate the usability of the proposed modular concept a few extension modules are implemented in hardware and the software needed to connect these extension modules to the original RTM-USB board has been written. Additionally, FPGA applications for motion control have been studied. In parallel with this thesis there was a parallel activity, which made the original RTM-USB *software* more extendable. Both projects and the original RTM-USB hardware and software, compose the new RTM-USB platform.

CE-MS-2011-01

PHILIPS
sense and simplicity

Philips Applied Technologies

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science

Modular RT-Motion USB
Hardware part

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Serge Keyser
born in Leidschendam, Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Modular RT-Motion USB

by Serge Keyser

Abstract

During the course of this thesis, RTM-USB (Real Time Motion on Universal Serial Bus) has grown from a single board motor controller to a motion control platform. The original RTM-USB board, containing a CPU and two motor drivers, has been extended (*hardware* wise) with a network/bus interface which makes it easy to expand the hardware functionality of the platform. Research has been undertaken in order to see which hardware extensions would be interesting for implementation on this platform. In order to demonstrate the usability of the proposed modular concept a few extension modules are implemented in hardware and the software needed to connect these extension modules to the original RTM-USB board has been written. Additionally, FPGA applications for motion control have been studied. In parallel with this thesis there was a parallel activity, which made the original RTM-USB *software* more extendable. Both projects and the original RTM-USB hardware and software, compose the new RTM-USB platform.

Laboratory : Computer Engineering
Codenumber : CE-MS-2011-01

Committee Members :

Advisor:	Ben Juurlink, CE TU Ddelft
Advisor:	Georgi N. Gaydadjiev, CE TU Delft
Chairperson:	Koen Bertels, CE, TU Delft
Member:	Wouter A. Serdijn, Microelectronics, TU Delft
Member:	Arjan van Genderen, CE, TU Delft
Member:	Sait Izmit, Philips Applied Technologies, Eindhoven

I dedicate this thesis to those who made me who I am.

Contents

List of Figures

List of Tables

Acknowledgements

This document describes the work and research I did regarding the use and development of a costeffective motion control platform called: Real Time Motion - USB.

As I started my internship in Philips back in the fall of 2008 I had no idea how working inside a big company like Philips would be. After spending one year in Philips I can truely say that Philips offers a great amount of opportunities which are simply waiting to be picked up.

I had an amazing time working together with many colleagues of which a few I would like to thank in particular. First and most of all my gratitude goes to my thesis supervisor in Philips, Sait Izmit, who always carefully reviewed my work and who gave me great feedback which made it possible for me to succesfully finish of this project. Since this RTM-USB project also handles some software parts I want to thank my fellow student and colleague Widita Budhysutanto for his help and assistence where needed. Both my university supervisors, Ben Juurlink and Georgi Gaydadjiev, I would like to thank for comming over to Philips to see my intermediate presentations and for reading and correcting my thesis after I wrote it. Besides my coleagues I would like to thank the many friends I made in Eindhoven: Asif, Mafalda, Wouter, Sara, Navin, Akshay, Sima, Ulf and Armin thanks all for the time we spent together!

And last but certainly not least I would like to thank my mother for helping me out with thousand-and-one small and big things, without which I would not have been able to finish this thesis.

Serge Keyser
Delft, The Netherlands
January 28, 2011

1 Introduction

These days electronic hardware complexity rises by the day. Making custom electronical hardware to control a mechanical prototype setup is not beneficial anymore, due to long debug and design time. Hence the need arises for low cost, easy to use, highly modular prototyping electronics.
This thesis will provide a solution to this need by designing and manufacturing the hardware infrastructure needed for such a modular prototyping system.
In this chapter, Section 1.1 gives the motivation why and where exactely this thesis delivers added value, furthermore Section 1.3 shows which goals we need to meet in this thesis. The last section in this chapter, Section 1.4, explains how this thesis is organized

1.1 Motivation and Impact

Modular motion control electronics prototyping hardware (hereafter simply called "hardware") does exist, however costs are high (in the order of 10k Euros for a simple system like the BOSCH Rexroth NYCe 4000 [1]), it is complex to operate and over dimensioned in most cases. Such a high price/high complexity limits the number of people who are able to use the hardware. If it is possible to lower the complexity and cost, more people can do more research in less time at less cost. This is where the RTM-USB platform is positioned: A cost sensitive, easy to use motion control electronics prototyping platform which is within reach of almost any control engineer.

1.2 Introduction to RTM-USB

Two years ago the RTM-USB (Real Time Motion on Universal Serial Bus) project was started. Until the start of this thesis a board with a microcontroller and two DC motor drivers (see figure 1.1), has been designed and manufactured. The board can be controlled through a USB port using Matlab and/or Simulink. In order to turn the RTM-USB board from a single board into a prototyping platform it should be made possible to easily add extra hardware functionality. To do this in a structured way, both the hardware needs to be made modular and a unified way to connect all the modules together, needs to be defined.

1.3 Project Goals

The hardware where we start from is neither modular, nor does there exist a way to connect additional hardware. The main goal of this thesis is to extend the hardware of

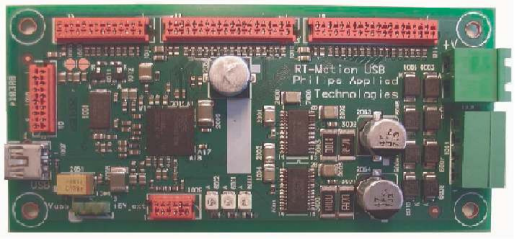


Figure 1.1: RTM-USB Board

the RTM-USB board with additional functionality which is usefull for motion control. In order to reach this goal we need to meet the follwing sub-goals

- Design the interconnect to connect the RTM-USB to the to be designed extension modules (see figure 1.2 for an overview).
- The to be designed interconnect should have one option to achieve high speed and one to have multiple extensions on the same bus (not necessarily at high speed).
- Hardware extensions for the RTM-USB board should be designed and made. Their functionality should be aimed at motion control.
- A test setup should be designed to show that the designed interconnect and hardware extensions are actually working as intentioned.

As a separate part of this thesis, the use of FPGAs (Field Programmable Gate Arrays see appendix C for an introduction) in motion control will be explored. The aim of this exploration will be at application which are simple but require a lot of calculations or an exceptional high data rate. We will look which *motion control* specific functionality can be implemented on such an FPGA and how a modular architecture (similar to the modular hardware platform) could be established.

1.4 Thesis Organization

This thesis is organized as follows: A literature survey is given in *Chapter 2*, in this literature survey the position of the RTMotion-USB hard- and software is discussed with respect to other real-time modular prototyping systems. This literature survey will also reveal from which options one can choose during the design process of the RTMotion Hardware Architecture. *Chapter 3* shows how the architecture for the hard- and software is designed, the designed architecture will later on be used to implement this hard- and software. Next to design, also the goals which should be met are introduced. Later on it is possible to measure and determine if these goals are met by the implemented

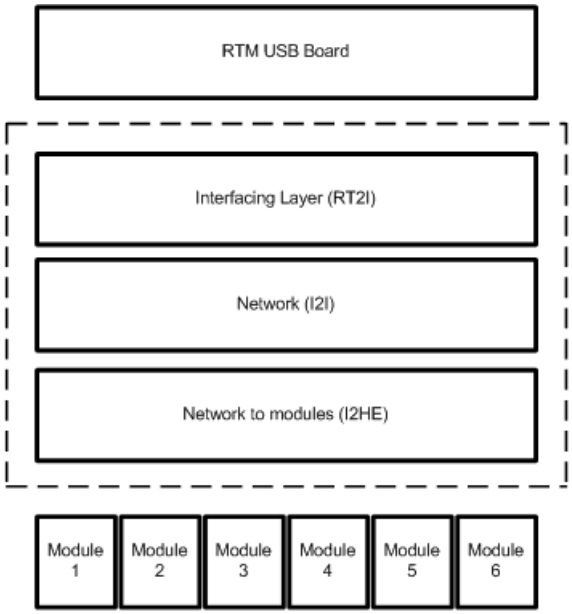


Figure 1.2: The blocks in the dotted area make up the Hardware framework for the RTM USB board

design. *Chapter 4* introduces the design of the actual hard- and software. Hardware schematics, PCB layout and code are discussed and explained. *Chapter 5* shows that the designed hardware meets the goals as set in *Chapter 3*. *Chapter 6* treats how the extensions as made in this thesis are used in a comercial product. As a conclusion to this thesis, *Chapter 7* gives a summary of the work done during the time span of this project. Furthermore suggestions for future improvements of the architecture, hardware or software are introduced.

In the appendices three important topics are discussed which are vital to understand everything in this thesis. Appendix A gives an introduction in the various motors which are being used in motion control and a very brief overview in which motor controller electronics are available. Appendix B gives an introduction in what encoders actually are (good in case the reader doesn't have a background in motion control) and how they are physically constructed. The last appendix, appendix C, gives a short introduction in the terminology of Programmable Logic Devices (especially for those who do not have a computer engineering or similar background).

Literature Survey 2

This chapter presents an overview of motion control platforms which are comparable to the RTM-USB motion control platform as it is after the finishing of this thesis. Paragraph 2.1 introduces and compares these systems. The results of this comparison are used in paragraph 2.2 to see which requirements should be met by the to be designed extension modules. Concluding the discussion of the extension modules, some use cases will be shown.

Following, the interconnect between the extension modules and the RTM-USB board is being investigated in paragraph 2.3. The definition of the interconnect as it is seen in this thesis will be defined. The paragraphs after this will discuss the various types of interconnect that are present in this project, several solutions for this interconnect are being discussed.

As the final subject of this chapter a review of the possibilities of FPGAs for motion control is given in paragraph 2.4. Also for FPGAs it holds that setting up a modular structure is important. Hence the different possibilities in FPGA software modules and software interconnect between these modules are explored. In order to select the right FPGA, there is always a need to give an estimation of how much resources a certain implementation would take. An estimation of the resource cost to implement the previously suggested FPGA software modules, is given.

2.1 Related Work

In this section related work is introduced. All the introduced systems are used for rapid motion control prototyping, either in a lab environment or in a learning (school) environment.

RTMotion-USB is a single board controller for motion control. The RTM-USB platform aims at motion control applications which are on a tight budget but still need high performance motion control. This controller has been successfully applied in several Philips internal projects (robot arms, medical devices,etc).

dSpace is a modular prototyping environment aimed at: motion control, automotive industrie, aerospace industry, etc. dSpace provides software and hardware which integrates seamlessly with simulation programs like Matlab. dSpace has been deployed during the prototyping phase of major projects (designing airplanes, helicopters, cars, etc). In theory it would be possible to use dSpace hard- and software in a final product, however in most cases this is too expensive.

Bosh Rexroth NYCe 4000 is a motion control system which is mainly aimed at high

precision motion control. This system is useful for both prototyping and production. The relative small size makes it easy to integrate in typical projects like semiconductor and medical machines. Also here a complete product is delivered consisting of both hardware and software.

National Instruments Labview is a widely used prototyping system, used from classrooms till factory testing of produced electronics. There is also hardware availabele for motion control, this hardware can be controlled using the accompanying software or simulation programs like matlab/simulink.

Elector E-blocks is a simple prototyping system which aims at the educational market, there are no items in the available hardware and software which are specifically targeted at motion control. Despite this disadvantage some parts can be used to do general signal processing which is useful for motion control. The hugh advantage is that the learning curve is very modest and the delivered software is user friendly.

After this short introduction, the above systems are compared on the functional level. From table 2.1 we can immediately see that the number of possible units in one single system is a lot higher for RTM-USB as compared to the competition. One might wonder what the usefulness is of having 128 modules (at some point the information from all these modules has to converge to a single point which which will overload the used USB bus). However as stated in table 2.1 the RTM-USB modules can run stand alone, which makes each unit eligible to do its own data processing. Using this fact it is possible to run very fast (local) control loops and a much slower global control loop (which means that the data load is more eavenly spread in time). This reasoning shows that the limit of 128 RTM-USB modules, as set in the USB standard, still yields a usable application. Furthermore the table shows that the RTM-USB board is performing in the mid range of the five compared motion control prototyping systems.

Table 2.1: High level properties of the compared systems

Name	RTM-USB	dSpace	NYCe 4000	NI Labview	E-Blocks
Target market (budget wise)	Low/Middel end	High end	High end	Middle end	Low end
Prototyping	Yes	Yes	Yes	Yes	Education
Production	Yes	No	Yes	Yes	No
Number of extension modules (Aimed at motion control)	0	5	20	4	
Units per system	128	16	62	4	1
Cost Price (EUR) (basic version)	classified	10k	8k	1k5	300
Capable of running stand alone	Yes	Yes	Yes	No	Yes
Max control loop frequency(kHz) using a typical setup with 1 encoder and 1 motor	10	10-20	20-30	1-2	0.5
More information	[28]	[10]	[1]	[29]	[11]

The above introduced motion control prototyping systems can now be compared (see table 2.2) on a number of aspects which are the most important aspects of motion control according to the future customers (employees of Philips Applied Technologies) which we interviewd viz.:

- *Interconnect* is both how the systems connect with the outside world and how they connect the extension modules internally.
- *Motor control* is a vital part of motion control, there are many types of motors, and in this category the motors are listed that are being supported by the various motion control systems.
- *Encoder counters* are the sensors which deliver feedback to see what the behaviour of the motor is, a motor control loop is usually based on such a sensor.
- *Digital and analog in- and outputs* are used to control the motors and additional sensors.

Please note that LabView and E-Blocks are not specifically targeted at motion control, therefore not all points of comparison are available.

Table 2.2: Comparison between prototyping systems for motion control

Name	RTM-USB	dSpace	NYCe 4000	NI Labview	E-Blocks
Control frequency	10kHz	32kHz	32kHz	–	–
Available Interconnect	USB	PCI Optical Ethernet HSS [1] PHS++[2]	Firewire	PCI Firewire USB GPIB Can (open)	USB Serial
Motor control					
Supported motors	DC	DC/BLDC/ Stepper/synchronous	DC/BLDC/Stepper	DC	DC
Encoders					
Supported encoders	Quadrature S0/S90	Incremental Quadrature Sin/COS	5V S0/S90 Quadrature Analog 10V absolute Endat 2.2 Panasonic incremental Panasonic absolute Endat 2.1 Hiperface Sine/Cosine	Quadrature	–
Pulse count per revolution(MHz)	0.225	1.25	0.4		
Number of channels	2	6	10		

Continued on next page

[1] High Speed Serial bus
[2] Proprietary local bus to connect all modules

Table 2.2 – continued from previous page

Name	RTM-USB	dSpace	NYCe 4000	NI Labview	E-Blocks
Analog IO					
ADC					
Number of bits (per channel)	16/10	16	12/16	16 [3]	10
Number of channels	8	5-32	2(scalable)	2	7
Differential/ single ended	single	single	single	single	single
Voltage input range (V)	0/3.3 0/3.0	-5/+5 -10/+10 +/-12 save	0/10 -5/+5 -10/+10	-15/+15 -5/+5 -10/+10	0/5
DAC					
Number of bits (per channel)	16	12/16	16	12/13/16	
Number of channels	2	5-32	2(scalable)	4	
Differential/ single ended	single	single	single	single	
Voltage out put range (V)	0/2.6	-5/+5 -10/+10 +/-12 safe	0/10 -5/+5 -10/+10	-10/+10	
Digital IO					
Number of IO	32	32(scalable)	100	100	42
Voltage output range (V)	0/3.3	0/5	0/5	0/1.8/2.5	0/5
Buffered	No	Yes	Yes	Yes	No

[3] These ADC cards are not made by National Instruments but by 3^{rd} party suppliers

Using table 2.2 we can score each section of the table to get a better understanding of the differences between the systems. There are 5 systems available, the system which performs best in one section gets a score of one, the second best a score of two, etc. This scoring method yields figure 2.1 in which a better system gets a lower score. The overal score can be observed in the last column.

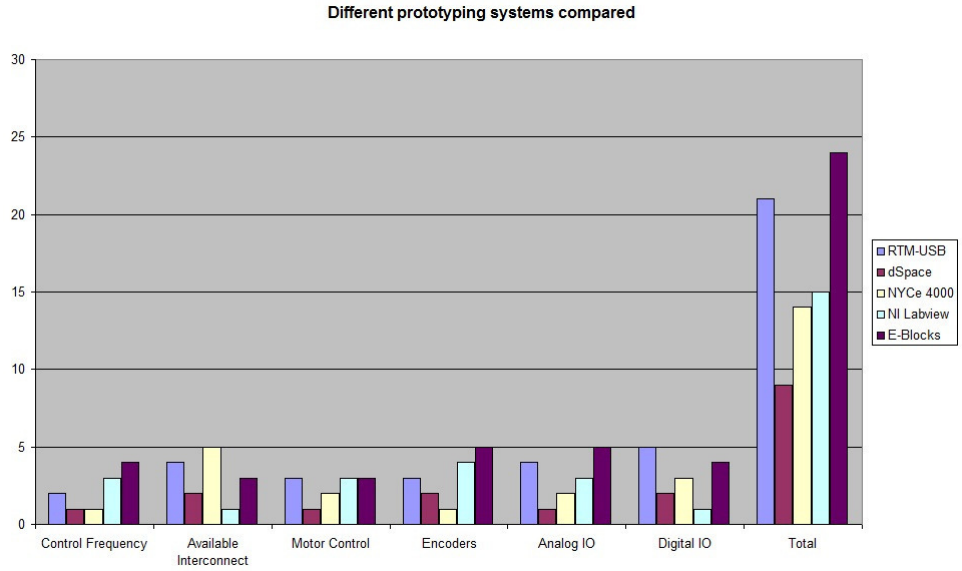


Figure 2.1: Comparison graph between available motion control prototyping systems (lower is better)

From this figure we can draw a few conclusions:

- dSpace is overall the most diverse system available
- NYCe 4000 and NI Labview are quite comparable with regards to supported functionality
- RTM-USB as it was before the start of this thesis is comparable with E-Blocks, which means that a lot of work to increase the functionality of the RTM-USB board needs to be done during the course of this thesis.
- If RTM-USB would become a serious player in the market of low cost modular prototyping motion control, the supported functionality should be equivalent to, or exceeding the functionality as delivered by NYCe 4000 (see also next point).
- dSpace is significantly better compared to the competition, inorder to not create "another dSpace" (with all the drawbacks that we want to avoid, in terms of cost and size) we should not try to support as much functionality as dSpace does.

Ofcourse there are many items which were not compared between the systems (financial cost, physical dimensions, accuracy (only partially discussed), etc) however it gives

a quick overview of where RTM-USB is currently positioned and where we want it to be at the end of this thesis.

In the previous comparison we took only commercially available systems into account since we also want to make a commercial system. However there are many more lowcost (prototyping) motion control initiatives who were being carried out as part of university projects / research and have never been commercialized. Some examples are:

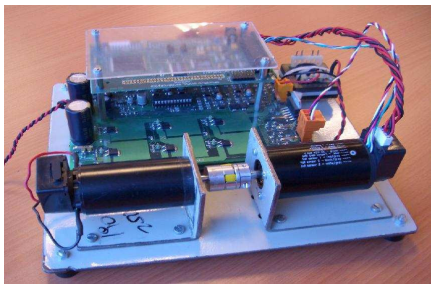


Figure 2.2: Low Cost Motion Control by the University of Katalan

"A New Low-Cost Motion Control Educational Equipment" by the university of Katalan [4] (see figure 2.2). This is a very low cost motion control system used in a class room setup. The whole unit is build around a Texas Instruments(TI) DSP which can be programed from the computer using the TI software.

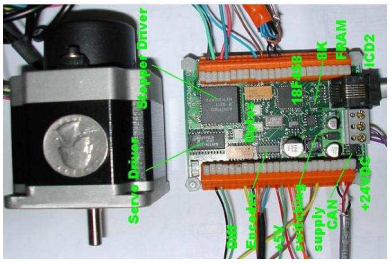


Figure 2.3: Low Cost Motion Control by Circuit Cellar

"A low cost USB-CAN Distributed Motion Control System" [27] Is the outcome of a design contest as organised by Circuit Cellar (http://www.circuitcellar.com). This system is based on a Microchip PIC microcontroller and a CAN bus (see figure 2.3). It can be used to run simple PID loops at a loop frequency of approximately 2Khz.

Both these projects are, unlike the systems we compared earlier, not on the open market and these projects need a lot more development before they can be commercialized. This is why they were not take into account in the comparison on tables 2.1 and 2.2

2.2 Extension Modules

This section presents a list of modules which are interesting to implement for the RTM-USB project. A selection (see paragraph 3.1) of this list is implemented in hardware. Besides this list a calculation is given on how much load each module puts on the final system. This discussion is given in terms of bandwidth (bits/s). If a final system is composed of these extension modules, it will just suffice to add up all bandwidth requirements and to compare this to the capabilities of the interconnect, if the bandwidth needed is less then the bandwidth available on the interconnect then it is possible to implement the system. See chapter 3.1 for the capabilities of the interconnect. If the network is not capable to suite a composed system, possible ideas to improve the interconnect are to be found in section 2.3. Finally some use cases for the extension modules are discussed. One of these use cases will be used as a vehicle to demonstrate the capabilities of the implemented hardware.

2.2.1 List of extension modules for RTM-USB

In this paragraph we introduce a number of extension modules which could be implemented to extend the RTM-USB platform. The modules are grouped in 4 different categories: Functional, Connectivity, User Interface and Miscellaneous Extensions.

Functional Extensions

1. **Digital IO:** Simple in and output pins, groups of 8 would be preferable (since this is one byte of data). Input and output buffers should be constructed such that the input output voltage levels can be chosen by the end user. Usefull levels for IO voltages are 0/5V -5/5V -10/10V -12/12V since most sensors for motion control use these levels (see also table 2.2)

2. **FPGA board:** Contains an FPGA to do signal processing. The FPGA can be used for all kinds of processing, ranging from counting encoders to speech recognition. A list of these applications is given in Chapter 2.4.2

3. **DSP board:** Contains a simple DSP which can easily be programmed in C, this is easier than an HDL needed to program an FPGA. The DSP could be used for all sorts of motor control applications.

4. **Encoder board:** Contains the logic to count the pulses coming from a motor encoder (at a maximum of 11MHz). This board will work as a buffer between RTM-USB and the encoder since RTM-USB will never be fast enough to count all pulses at the speed by which they are coming from the encoder.
   Supported encoders could be of the type:
   - Sin/Cos
   - Incremental
   - Absolute
   - Quadrature

A study on how these encoders work and how they should be used is given in appendix B.
If a motor is moving at a slow pace (less then 10 rpm), the number of counted pulses stays constant, hence it appears as if the motor is standing still. In order to avoid this problem we can count the time between encoder pulses (this is equal to taking the derivative of the number of pulses over time) by which we can avoid coming up with a motor speed of zero.

5. **DA/AD converters:** Use these to control motion control power amplifiers and to read the sensor values from the a sensor board. Typical values of the ADC are:
   - Number of bits: 16
   - Number of channels: 8/16
   - Differential/single ended: Selectable
   - Input voltage range: 0/10, -10/+10, 0/5, -5/5 V

   Typical values of the DAC are
   - Number of bits: 16
   - Number of channels: 8/16
   - Differential/single ended: Selectable
   - Output voltage range: 0/10, -10/+10, 0/5, -5/5 V

   On this board also some simple filtering (low pas anti-aliasing) and biasing should be done. Care should be taken that the resistors in these filters can be easily replaced depending on the application where the filters are used for.

6. **Memory:** One board with ram and flash memory. Adding an SD slot would provide us with a removable data storage which would be usefull for data logging. The problem is that for SD cards a license is needed to control them in SD mode, however most sd cards (not the small ones) have an SPI mode in which they can be controlled. For SPI, only a license is needed if the SPI controller is implemented in an FPGA. Otherwise no license is required.

7. **Motor amplifier:** This board contains the power chips which directly drive the motors. Various types of motors should be supported: DC, AC, BLDC, stepper, piezo actuators (for more information on these terms see Appendix A). The board it self is controlled by either the network or the AD/DA converter board for more accuracy. There should be different boards for different power levels (0-50W and 50-150W).

8. **Filters:** A simple board with a few low pass filters for common frequencies (two of each filter):
   - 500 Hz
   - 1000 Hz
   - 2000 Hz

9. **Direct digital synthesis:** Use the DA converter board and the FPGA or DSP and the memory board to generate wave shapes which are not repeating and have a specific wave shape (the memory should at least be 64k samples deep to do useful work for motor control). Types of signals:
   - White noise
   - Setpoints (for actuator control)
   - Sinusoid

Connectivity Extensions

10. **USB-host:** USB on the go (OTG) can be used to both connect to a host computer or to an external peripheral. This is particularly useful if the RTM-USB system will be a stand alone system, since this gives an easy way to connect user input devices like a keyboard or mouse.

11. **EtherCat:** This would be a bridge between the local bus and the EtherCat bus. This is particular useful if the RTM-USB would be used in an industrial application/environment. A suggestion is the following module: Beckhoff, FB1111-0140 which has the size of a post stamp (15 X 10 mm).

12. **Ethernet:** This is not a realtime bus however it is a really generalized way of communication (every computer has an ethernet port these days) and ethernet functionality is cheap and easy to implement. This extension would be used to input user data or to output status messages to the computer. An ENC28J60 from Microchip could be used, this ehternet chip is controlable over SPI. The problem in this case is that most chips go to the MAC layer but the Ethernet stack has to be run on the RTM-USB (which might compromise the desired realtimeness of the RTM-USB)

13. **CAN:** CAN bus is highly used in automotive/robotics applications, there fore it would be a good choice to extend the RTM-USB board with this bus. Any simple microcontroller (Microchip PIC,Atmel AVR, etc) with the support for a ninth data bit on its serial port could be used here to act as a bridge between the CAN bus and the RTM-USB board.

14. **RS485:** Rs485 is a physical layer of many different protocols (DMX, Profibus, etc) and, hardware wise, very easy and cheap to implement. This can again be done using a serial port on any microcontroller, the only hardware which should be added is an differential line tranceifer such as Texas Instrument's sn65176B.

15. **Radio Remote Control:** Use a simple 2.4Ghz remote control module to setup a robust wireless link which does not depend on a line of sight (as with IR remotes) and can control lots of modern day home equipment (i.e. Philips living colors lamp).

16. **PLC interface:** This module has been considered however there is not much standard for a PLCs. Most PLCs just connect to the outside world using one of the above busses/protocols. We just add it here to show there is no need to consider implementing such an interface in the future.

17. **Optical communication:** Optical interfaces are being used in order to galvanically separate the various parts of an interconnected system (which eliminates noise introducing ground loops). A simple optical module could be made with a TOSLINK module like the ones that are used for SPDIF data communication (with a data rate of 6Mb/s with Non Return To Zero, NRZ, endcoding).

User Interface Extensions

18. **Debug board:** This board contains a number of LEDs, switches and a simple two line display. In case the RTM-USB is becoming a standalone system, this board will make it easy to get some feedback from the hardware.

19. **Display:** A color display can display data in a user friendly way. If combined with a touch screen it is possible to let the end user adjust parameters in the device. It is not supposed to be a video screen since the data supply (the network/bus used in this system) will not be fast enough to supply the video data to the screen. The FPGA board could be used as a graphics controller. There are numerous displays available on the market, brand names are Sharp, Hitachi, Kyocera, etc.

20. **Audio:** This can be an extension which can replay messages to the user, if a microphone is included speech processing can be done which creates an additional user interface.

21. **IR remote control:** Use a simple (and inexpensive) TV/VCR/DVD/etc. remote control to let the user input data into the system. As a bonus an IR led could be added to the board in order to generate IR signals with which remote IR controllable devices (i.e. tv, cd player, dvd player) could be controlled.

22. **VGA camera:** Use a camera to recognize the user, or any other objects, raw camera data can not be transported over the network/bus (this would take too much bandwidth) hence the data should be processed locally.

Miscellaneous Extensions

23. **Experimenters board:** Simple board with a field of holes and a connection to the interconnect. The board contains a simple micro-controller which can be used to interface one's experiment to the RTM-USB board.

24. **Sensor simulation board:** Sensors are always depending on how the outside world behaves. It is hard to control the outside world, for debugging purposes it would be good to be able to control the sensor values in hardware.

This should be a resistive type sensor simulator since most sensors have a resistor like behaviour.

25. **Location determination:** It is important to know where the RTM-USB board is connected, especially in the case when multiple boards are being used in the same system. This functionality could be implemented as a separate module or each of the extension modules could get a memory to store its ID from which the main computer could determine what it is controlling (in the case of a robot, a leg, a wrist, etc.) This is something that should not be done in software or hardware (by the user) since if a mistake is made during initialisation, a board is initialised as being a knee joint of a robot but in reality is a wrist, we might end up with a damaged robot).

2.2.2 Use cases for the extension modules

In motion control several areas can be defined. Each area has its own demands in terms of accuracy, cost, area, safety, etc. Motion control applications can be divided into three categories, these and their specific characteristics can be found in table 2.3

Table 2.3: Main areas of motion control

	Low End	Middle End	High End
Loop frequency	0-2KHz	0-10KHz	0-30KHz
Application area	Toys	Office	Medical
	House hold	Pick and place Robotics	Semiconductor Robotics
Accuracy	Milli meter	Micro meter	Nano meter
Most popular motor types	Stepper	DC	DC
	DC	BLDC	BLDC
		Stepper	Stepper
			AC
System cost	O(10)	O(100)	O(1000)

The aim of the RTM-USB platform is in the middle to high end production and prototyping range. Please keep in mind that low cost does not mean low precision. Typical applications in the low to middle end area are:

- Domestic robotics
- Motion control prototyping
- Medical equipment
- Electrical car control

In order to demonstrate the capabilities of the hardware as designed in this thesis we will use a humanoid robot application. We do this for three reasons: the application is already available, which creates an easy testbed. The application is also a good example

of a low cost high precision system (exactly those systems that we want to target with the RTM-USB board). Last but not least, this demonstration could prove the hardware useful, leading to a real mass production of the work that has been done in this thesis.

2.3 Module Interconnect

The extension modules which will be designed in this thesis work, need to be connected to each other and to the original RTM-USB board, this interconnect should be both reliable and simple (cost effective) to implement. This paragraph is only there to show what options are available for use in the interconnect, the real design choises with corresponding reasoning can be found in chapter 3.
In this design there will be a parallel and serial interconnect. First the interconnect architecture is defined, with the emphasis on a serial interconnect, see paragraph 2.3.1. The subsequent sections each introduce a distinctive piece of the interconnect and different implementations are being explored. Following the serial interconnect, section 2.3.5 dicusses the parallel interconnect. Now that the interconnect is fully defined we need a protocol to send data over this interconnect, how this could be implemented is described in paragraph 2.3.6.

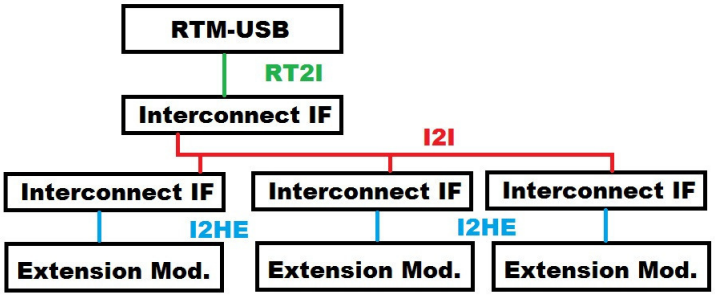


Figure 2.4: Architecture overview with the names of the different interconnect.

2.3.1 Interconnect Architecture

The interconnect which is used to connect the extension modules and the RTM-USB board has to be both flexible and cost effective to implement. From the design specifications we have to implement both a serial and parallel bus. The reasoning behind this is that an n bits wide parallel bus has a raw data throughput of n times the serial bus at the same speed. In case a hardware extension uses more bandwidth than the serial interconnect can deliver, a switch could be made from serial to parallel. Even though this compromises the modularity of the setup, it would still be possible to setup a system involving the RTMotion-USB board.

In order to discuss the interconnect we define three classes of interconnect: (see also figure 2.4

- between RTM-USB and the interconnect interface (RT2I)

- between the interconnect interfaces themselves (I2I)
- between the interconnect interface and the hardware extension (I2HE)

A setup with a separate interconnect interface adds extra hardware cost and complexity, in comparison to interconnect where hardware extensions are directly connected to the "RTMotion-USB's" on board processor, however this layer of abstraction makes the whole system more modular and less dependable on the processor which is on the RTM-USB board (which makes it easier to switch to a different processor when needed).

2.3.2 Connection from RTM-USB to the interconnect interface (RT2I)

In order to keep the hardware implementation as simple (and as cost effective) as possible preferably a peripheral from the LPC2888 processor on the RTM-USB board should be used. An overview of the LPC2888 peripherals, available for data communication, and their properties can be found in table 2.4

Table 2.4: LPC2888 Peripherals properties

Peripheral	I^2C	I^2S	Memory bus	LCD bus	GPIO
Max frequency (MHz)	1.4	2.5	30	5	3
Max speed MBits/s	1.4	2.5	480	40	48
Bus type	Tri-state	Audio	16 bit parallel	8 bit parallel	16 bit parallel

2.3.3 Connection from interconnect interface to interconnect interface (I2I)

This interconnect will be implemented using one of the busses listed in table 2.4 however the interconnect interface should implement an abstraction layer between the used processor (in this case ARM7) and the network. Such that in the future the current processor might be replaced without affecting the interconnect.

Table 2.5 lists which available busses could be used in the I2I interconnect. There are a lot more serial busses available, however this will be a low cost system, therefore the aim is at the low cost hardware, easy (in terms of programming effort) to implement, serial busses.

Which bus will be implemented, depends on the load on the bus. The bus load will be calculated in chapter 3.2.

2.3.4 Connection from interconnect interface to hardware extension module (I2HE)

In this part of the interconnect there are several possibilities.

Table 2.5: Comercially available serial busses

Bus	I^2C	I^2S	RS485	LVDS	SPI	1-Wire
Max frequency (MHz)	1.4	2.5	20	500	10Mhz	0.111
Max speed MBits/s	1.4	2.5	20	800	10	0.111
Bus type	Tri-state	Audio	Differential serial bus	Differential serial bus	Serial With chip Enable	Serial Single wire

1. Convert the I2I bus type into another format which is usefull for the peripherals on the hardware extension module, for example to one of those mentioned in table 2.5.

2. Convert the incoming serial data stream into a parallel one.

3. Incoming serial data is used in such a way that it controls the peripherals on the extension module directly without conversion.

Item 1 is easy out of a user's, hardware design and modularity, point of view. However this requires extra space in the interconnect interface and the user needs to tinker on the network interface, which is in turn not user friendly.
Item 2 enables the user to simply connect anything without having any knowledge of the interconnect or the interconnect interface, since the protocol is handled by the I2I interconnect the parallel port is transparent as seen from the side of the processor hence this is easy to use. If the user wants to control anything, that does not have a parallel interface, additional intelligence (micro controller, CPLD, etc) is required, which will add to the module's cost.
Item 3 is the most easy solution from the I2HE interconnect designer's point of view since nothing has to be designed. However this is the most demanding solution for the user, every time an other hardware module is designed, the interconnect interface needs extra software and the extendability(which comes with the provided modularity) will be lost.

2.3.5 Parallel bus

This is the parallel bus that will be used next to the serial bus. The parallel bus is easy to implement, there are not much possible schemes to choose from. The only possible way would be to implement a simple 8/16 bit parallel bus. On the RTM-USB board there are 2 parallel ports which each expose 16 bits. In this case it is just simple GPIO, which can only run at 3MHz. The maximum data throughput using GPIO is still 48Mbit/s (3M * 16) which is faster than most serial solutions mentioned in table 2.5. The additional benefit of using the parallel port can be seen during the debugging of the modules. Since the GPIO is so elemetary, we do not need to worry about errors in the communication, hence it is easy to rull out errors, which allows for easy debugging. In future revisions

of the RTM-USB board the memory bus of the on board processor could be used to transfer data. The memory bus is a lot faster than GPIO (see table 2.4).

2.3.6 Interconnect Protocol

In order to communicate on the I2I interconnect a protocol needs to be designed. Existing protocols (like Modbus, Linbus, Sercos, etc.) put way to much overhead on the simple system which is being proposed here.

The following requirements could be provided by the protocol for the RTM-USB platform (depending on the type of I2I and demands of the final application):

1. **Error detection**: Wrong data could be detected, most of the time the data is thrown away and the new data of the next iteration of the control loop will be used. Care should be taken that the system goes in hold mode if there are too many errors in the transmission.
   It is also possible to refrain from doing any error detection, since there will an update of the value every loop iteration. Even though this saves bandwidth and processing power, good sanity checking is inevitable since bad data can lead to a runaway of the system.

2. **Single master/multimaster**: Determines who can control the interconnect. In order to keep everything realtime it is best to start with a protocol which only allows one master. Later on, multi master extensions could be made if required, and time permits.

3. **Initialization**: In order to see which modules are currently on the bus, the protocol could provide a way to issue an inventory command which returns the address of the module, the functionality and possible other parameters.

4. **Lightweight**: Since the interconnect will not be able to handle high data rates, as little as possible overhead should be spend on the protocol.

The available options to deal with the above constraints are:

- **Error detection:**
  - CRC checking : Works on multiple parts of data, needs some hardware, perfect to implement in a CPLD or FPGA.
  - Parity checking : Works mostly on one byte of data, is weaker than CRC checking but easier to implement.
  - Sanity check : Works on a whole setpoint sent. Compares the previous setpoint with the current one, if the current one differs too much, the current received setpoint will be disgarded.
  - Hash : Works on the whole setpoint, however multiple values lead to the same hash number, hence if data is corrupted ""good enough" it might lead to the same hash as the original data. Hash functions are separated in such a way that this problem will not occur too often [13]

- **Single master/ multi master:**
  - Single master: deterministic, relatively easy to implement, low software foot print, data could only be acquired by polling.
  - Multi master: non deterministic, needs a way to schedule communication, supports interrupts, easier to extent the interconnect.
- **Initialisation:**
  - Tristate bus : give each module which is manufactured, a universal unique ID, and using an anti-collision scheme, as is popular in RFID protocols, read the IDs.
  - Try all possible addresses and look for response: if the number of possible addresses is not too high (max 256) it might be possible to try all addresses and see if a response comes back.
  - User set :The user enters which devices are on the bus, and the electronics just assumes those modules are present.

More information on the protocol used and its design can be found in section: 3.2

2.4 How to use FPGAs in motion control applications

These days FPGAs are used in many applications. This section focuses primarily on using FPGAs in motion control applications, where high speed and the ease of hardware design are needed. An introduction into FPGAs is given in appendix C. There is no time during this thesis project to implement an FPGA module hence the FPGA part for this thesis is limited to this literature survey. Paragraph 2.4.1 presents how FPGAs can be used in control applications. Continuing on this topic it is being shown which functionality (Intellectual Property, IP) would be particular useful for the RTM-USB platform. In order to be able to estimate which FPGA should be used, the resources used by these functional blocks is estimated, using this estimation an FPGA will be selected.

2.4.1 Applications on FPGA

The combination of motion control and FPGA is relatively little used since most of the times motor control applications have a low financial budget (i.e. electrical motors in office machines and household appliances). On the other hand FPGAs are very well suited for machines where performance demands outweighs the cost (factory robots, air craft, etc)[33] A few examples where FPGAs are used to do motion/motor control can be found in [16] [22] Most of the solutions are ad-hoc solutions (not modular) if anything should be implemented on an FPGA to be used with the RTM-USB board, a way to make the FPGA software modular, should be found. A way to make a modular design possible would be to use a Network/Bus On Chip (NOC/BOC) such as: NXP Æthereal NOC[24], Wishbone BOC[26], Altera Avalon BOC [2], Core Connect BOC [5], Open

source NOC [19],Xpipes NOC [6]. Please note, the only network of the above list which provides realtime communication out of the box, is Æthereal [24] [19], the busses can be made realtime with some effort on the protocol side.

One of the developments which are currently a hot subject of research in motion control on FPGA are fuzzy logic controllers [32] [15] [17] mainly to do low level control loops (motor torque/speed control). The FPGA part is being used here to implement the controller, the controller it self is being fed with setpoints from somewhere else (micro controller, computer, etc). This is done since the software in micro controllers (programmed in C) is easier to adjust for future users as compared to coding an FPGA (in some HDL or using an HDL generator).

2.4.2 Useful FPGA modules for motion control

In order to determine which functions are useful to implement on an FPGA, a list of functions has been compiled. These modules can be found in literature and have been proven to be useful for motion control. For all these modules it holds that they contain a lot of shifts and multiplications which run at high speed. Both tasks can be achieved more easy in hardware (FPGA or CPLD) than in software (micro processor).

1. **Stepper motor control:** As proposed and implemented by [14] a stepper motor controller is one of the possible modules to implement. The algorithms as proposed by the paper shows that implementation of a stepper motor driver in an FPGA allows for easy experimenting with different algorithms. Since a stepper motor controller uses a lot of bit shifting, it is a good candidate to move from software on a processor (where shifting is expensive) to an FPGA (where bit shifting is easy).

2. **FPGA Encoder counter:** This is a basic element in motion control and the FPGA hardware is perfectly suited [23] to implement this kind of counters / controllers. Because an FPGA (real hardware) can be driven to high counting speeds ($> 30Mhz$) relatively easy as compared to a microcontroller. Furthermore counters are easy to make at a custom length (32/16/12, etc. bits) in an FPGA.

3. **Software commutation:** In brushless direct current (BLDC) motor applications commutation (energizing the coils inside the motor at the right time) is important. Normally sensors are used to detect the rotor position. However it is also possible to work without sensors and just calculate the position of the rotor, as proposed and implemented by [20].

4. **PID controller:** PID controllers can be implemented relatively easy as demonstrated by [30] however care should be taken that the controller is implemented in such a way that it is optimized for the hardware structure of an FPGA e.g. it should be easy to map the implementation on to a set of Look Up Tables (LUTs)

5. **PWM controller:** This block should be implemented as a separate block. In order to be usefull for motion control a minimum PWM frequency of 100kHz should be used.

6. **Arithmetic units:** The basic blocks of any signal processing system are **M**ultiply **AC**cumulate (MAC) functions. These functions perform a multiplication and addition in one cycle and are particularly useful for filtering (next item). However in motion control MACs can be used in PID loops as can be seen in the following example: A simple PID loop would look like:

```
error = setpoint - current_position
integral = integral + error*dt
deriv = 1/dt * (error - prev_error)
out = Kp * error + Ki * integral + Kd * derivative
prev_error = error
```

   Concluding from the above pseudo code we can calculate that one PID controller loop consists of:

   - 4 subtractions/additions
   - 4 multiplications
   - 1 division

   The division is questionable since it can be considered as multiplication of the inverse of a constant. Hence per loop iteration we can identify 4 MACs. If this loop is run at 10Khz we need to be able to proces 40E3 MACs/s per PID loop. A single FPGA running on 40 MHz could do 40 MMACs/s, or 1000 PID loops. This is more than enough for the case where this FPGA will be used (approx 10-20 loops maximum). For more complex controllers (Multiple Input Multiple Output,MIMO) more calculations need to be done depending on the controller and the loop speed.

7. **Filtering:** Since motion control sensors (like for instance encoders) are always used in a noisy environment, filtering needs to be done. Filtering can be done in the analog domain, however these will always be fixed filters. To implement fixed analog filters really well, a lot of effort has to be put into designing analog hardware. Besides this, good analog components are not really economical to use (large component spread). If however a digital filter is used, a fairly cheap and weak analog filter can be used. As an example we can take the following numbers:

   - Pulse frequency 3 MHz (as given in the specifications for the encoder extension, see paragraph 2.2.1)
   - Pulses per revolution (PPR) : 2000 as suggested in the interview with future users
   - Revolutions per minute(RPM): 5000 as suggested in the interview with future users

Now it is possible to do 18 times over sampling (5000RPM = 83.33 RPS, 2000 PPR, 3Mhz/(83.33*2000) = 18). Due to this over sampling factor we can do filtering very well in hardware. Options to do filtering are:

- Peak detection, count 14 times a 1 out of 18 counts, than it must have been a 1, otherwise 0. This method has the disadvantage that the output will always lag by 1 count.
- FIR /IIR (Finite/Infinite Impulse Response) Filter these filters are really well known theory and can easily be implemented in an FPGA. The performance will be much better than the peak detection circuit, however much more resources are needed for the FIR/IIR circuit.

8. **Miscellaneous:** There are various blocks which are nice to have but are not an absolute necessity.
   - *Sercos III interface:* SERCOS (SErial Realtime COmmunications System) is a serial bus which connects all modules inside a realtime motion control network. This bus is an widely used bus, and connects via ethernet or optical links. Easy IO is an already free available implementation of Sercos for low cost low end FPGAs[25].
   - *Ethercat:* Beckhoff uses an FPGA to control an Ethercat PHY (PHYsical layer chip). In order to send data over Ethercat, a MAC (Media Access Controller) should be implemented in HDL code.[31]
   - *Complete controller:* As proposed by [15] a complete controller could be implemented on an FPGA. Such a controller would have everything to work stand alone (control a motor and provide a user interface to the outside world) All parts are connected using a standard bus. This bus could be a comercial bus (avalon bus, as used in [15]) or an opensource bus like Silicore's Whishbone bus [26] or even a network on chip (NOC) like Æthereal [24] [21] [22].

2.4.3 FPGA resource estimation

In this paragraph, the resources needed by each module, as proposed in section 2.4.2, are estimated. This is just a rough estimation in order to be able to select an FPGA device. The resources needed vary depending on the implementational complexity. The table below lists each function and the approximate needed number of logic elements (LE, a measure of the useable size of the FPGA) which is needed to implement the function. Furthermore for each module an FPGA is suggested. For the really small modules a cheaper CPLD (Complex Programmable Logic Device) will be suggested, since these devices are easier to use (the configuration remains in place even when the supply voltage is switched of, this in contrast to FPGAs where the configuration is lost after power down). There are a number of FPGA vendors in the market, in order to decide which vendor can be taken as a reference to map the functionality on, table 2.6 is created [4]. This table presents an overview of the different vendors and their main

[4]For the scores given in the table 2.6 it holds that the more "+" signs are given the better the manufacturer scores (i.e. the more pluses in "Device Cost" the cheaper the devices are etc.).

benefits and drawbacks. The columns "Software userfriendlyness" and "Availability of devices" are based on data from other users as found on the internet.

Table 2.6: FPGA Vendors compared.

Vendor name	Device Cost	Software Cost	Software Userfriendly	Availability of devices
Actel	−−	Free	Unknown	−−
Altera	++	Free	++	++
Atmel	−−	Unknown	Unknown	++
Xilinx	++	Free	–	++
Lattice	+	Unknown	Unknown	−−

As a reference, Altera FPGAs are taken since Altera scores best in table 2.6. Additionally it has the most userfriendly software (which is not necessarily the best, however if a large user base of non computer engineering people is to be attracted, as is the case for the RTM-USB platform, user friendliness may cost some performance).

Table 2.7: Resources needed for implementing motion control modules

Module number	Number of Logic Elements	Sugested FPGA	Cost(Eur)
Stepper motor	320	ep2c8	28.00
Encoder counter	240	epm240	5.10
Softwar comutation	523	ep2c8	28.00
PID controller	400	ep2c8	28.00
PWM controller	14	epm3064	2.10
Arithmetic units	> 1000	ep2c8	28.00
Filtering	> 1000	ep2c8	28.00
Miscelaneous	± 8000	ep3c16	29.10

The figures in table 2.7 for estimated number of LE are taken from the papers which are cited in the previous paragraph. From this table we can see that the FPGA ep2c8 would be a good choice to use if we want to implement an FPGA module on which we can run all the above mentioned functionality. It is however very well possible to use an ep2c16 which is only 1.5 euro more expensive but holds double the amound of Logic elements.

2.5 Chapter Summary

In this chapter it is shown which direct competitors are available for the RT-USB board, four different competitors have been found. Al the competing systems have been compared in order to give the RTM-USB board a background. A list of hardware extension modules has been created in which several modules which are useful for motion control are listed. Since all these modules should be connected to the RTM-USB board, the interconnect is defined and split up in three parts. For each part there are several ways

listed on how this interconnect could be implemented. In the final part of the chapter it was looked into the use of FPGA's in motion control. A list of FPGA applications has been made, as well as an overview of FPGA resources needed for each of these applications. The next chapter will show which of the hardware modules will be designed and which kind of interconnect will be implemented.

3 System Design

In the previous chapter the possibilities of the RTM-USB platform, regarding the extension modules and interconnect, have been indicated. In this chapter these possibilities will be narrowed down to a subset which will be implemented in real hardware during the time span of this thesis. Paragraph 3.1 gives a list of modules which will be designed and implemented, which interconnect topology and protocol are chosen is discussed in paragraph 3.2. As the final part of this chapter, a selection will be made out of the FPGA implementable functional blocks in paragraph 3.4.

3.1 Extension Modules

3.1.1 Extension Modules Selection Method

Since time is the limiting factor which prevents all modules listed in paragraph 2.2.1 to be implemented, a selection needs to be made. The selected demo application where we will use the RTM-USB board and the created extension modules is a robot arm. This arm is a typical example of a motion control problem. The accuracy that is needed for this application falls into the middle-end precision (typical control loops of $< 1kHz$, with a typical data transfer rate of $< 5MBit/s$, accuracy: millimeters).

In order to see what electronics we need for this robotics arm we first need to know how the arm works internally. As we take a closer look into the mechanics of the robot arm we see that it works with series elastic actuation (SEA). The mechanical realisation of this principle is displayed in figure 3.1.

This SEA principle is used to measure the external forces on the arm. By measuring the stretching of the spring and using hooke's law (F = -k$\mathbf{x}$, with k the spring constant and $\mathbf{x}$ the displacement of the spring) it is easy to measure how much force is applied externally to the arm. It is important to know these external forces, since we have to counteract these in case we have to maintain a steady position. Furthermore in humanoid robotics it is especially important to know these fources since we need to be able to limit the force that is being exerted by our application to the outside world (otherwise the robot might cause someone serious injury). To continue with figure 3.1 it can be seen that we need a motor, an encoder and an analog port (to measure the spring displacement from the optical sensor) to control a joint in this arm. Since this principle can be seen in most of the joints of the arm we consider controlling this as the main purpose of our electronics. Hence we need to select modules from the list in paragraph 2.2.1, which will help us to control this mechanical structure.

3.1.2 Selected Extension Modules

To be able to control the previous discussed mechanic model we need:

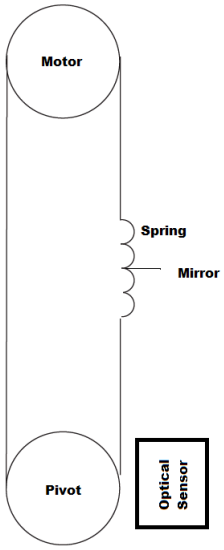


Figure 3.1: The principle of Series Elastic Actuation is all about measuring the stretching of the spring.

1. Encoder counter
2. Motor amplifier
3. Analog IO

The encoder counter is the most basic building block from any closed loop motion control system. The encoder counter can be implemented using an off the shelf standard chip, however these are hard to get. The chip which was used previously (HCTL 2032 from Agilent) worked sufficiently good, however it is only obtainable in a relative big Dual Inline Package (6cm^2) and its price is quite high (aprox 15 Euro in small quantities). Therefore we looked for other ways to implement the encoder counter and we looked back to the literature study on FPGAs in motion control (paragraph 2.4.2) and we can easily use a small CPLD which can fulfill the same functionality as the HCTL2032 at only one third of the price and one fourth of the size.

Right now the RTM-USB board is able to control Direct Current motors (two times 150 Watt), this would be enough to control the robotics arm application. However this motor controller is not modular (it is too much integrated with the processor, so it would be good to make a motor controller which is more modular. Furthermore BLDC motors are favourable over DC motors (see appendix A) but they are not being used in applications where the RTM-USB board is being used, simply because the RTM-USB board does not support this kind of motors. Implementing a Brushless DC controller would both enhance the RTM-USB board and broaden its usage.

For the BLDC motor amplifier we found several amplifiers, see table A.2 in appendix A. From these chips we selected the A3936, partially because Alegro DC motor drivers with a similar electrical interface were already used in the current version of the RTM-USB (hence simplifying the porting of the existing software drivers). But most of al because these were the only chips available that claim to go up to a maximum of 150 Watt without the need for external cooling. The solution we chose for this motor controller is a single chip solution, hence we need very little additional external components. The drawback of this solution is that in case we need a higher power amplifier, the chip and the PCB design needs to completely be redone. In order to overcome this problem a solution could be made with a separate power stage. The controller always stays the same, however an additional power stage (which is easy to design) can be changed uppon the needs of the user. This is further discussed in appendix A.

Just as a motor controller is already available, the analog IO is also available on the RTM-USB board. If we look back to table 2.2 we see that digital IO is the other kind of IO that is usefull to have. Preferably this IO should be able to cope with several different voltages (+-20V/+-12 Volt/+-10V/+-5V, etc.). These voltage levels are both for the input and output voltage levels and should be defined by the user.
In order to do this we have to implement a level converter. For up conversion there are three ways to do this, see figure 3.2, 3.3 and 3.4. In figure 3.2 a level converter using a mosfet can be seen, this is a simple design and can handle frequencies up to 10MHz and voltage levels from -20 till 20 volt. However this output will never be able to supply current, since any supplied current will cause a voltage drop over R1. Figure 3.3 presents a converter which makes use of two mosfets, this is a common structure which is mostly used in integrated semi conductors, the design can easily handle frequencies beyond 10 Mhz. The problem is that q2 can not be fully closed unless its gate voltage is exactly the same as the user defined output high voltage level. Hence this mosfet will always stay in the resistive area if the output high voltage is more than 3.3V or less then 0V (the voltages which we use to drive this output amplifier). To overcome this problem we can use a mosfet driver (not shown in the figure), however this will double the area taken by each output converter. The final converter design we have, can be seen in figure 3.4 here we use an opamp in a single package which is able to cope with the user supplied voltage levels. The output can supply a maximum of 50mA the only disadvantage is that the output frequency is lower (max 5MHz, sine wave) compared to the other solutions, however the fact that we can supply enough output current at the right voltage levels and the fact that our output frequency will be less than 5MHz (depending on the final interconnect we choose) makes that the circuit in figure 3.4 will be used in the digital io design as an output driver.

For the inputs we need an additional circuit (see figure 3.5). The circuit on the right in figure 3.5 is useless since it assumes that the voltage levels are known, which is of course not true. The left circuit in figure 3.5 gives the final answer, this circuit clips the incoming voltage on -0.6 and 3.3 Volt (due to the voltage drop over the diode). This clipping happens independent of the incoming voltage level, however the current that goes through the diode will vary with the incoming voltage since the resistor's value is fixed. For voltage levels of -20/+20 V the current going through the diode is acceptable

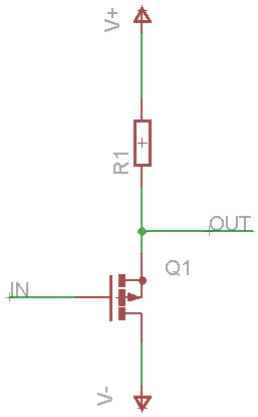


Figure 3.2: Level converter to upconvert the outgoing voltage, using a single MOS and a pull up resistor.

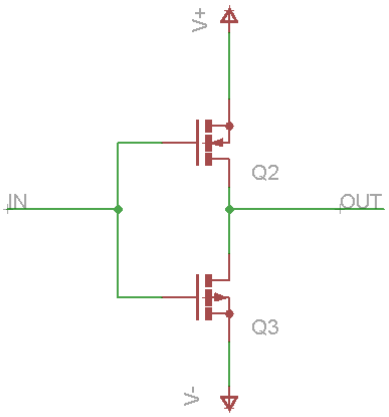


Figure 3.3: Level converter to upconvert the outgoing voltage, using a double MOS.

if R4 is chosen around 1kΩ.

The digital IO are split in 8 inputs and 8 outputs, this is done because implementing each pin as both input and output would render a much more complex PCB design and in general for motion control applications (reading sensors) there is no need to dynamically change the direction of the pins (from input to output or the other way around).

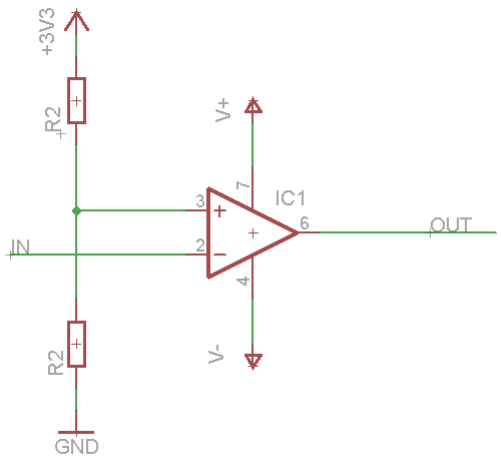


Figure 3.4: Level converter to upconvert the outgoing voltage, using an opamp.

Figure 3.5: Level converter to clip the incoming voltage (left) or to scale the incoming voltage (right).

3.2 Interconnect

3.2.1 Interconnect Selection Method

Before we can start the discussion about which interconnect to use we first need to know what the load on the interconnect will be. To calculate the load of each module on the interconnect, the data usage of each module needs to be determined. To gain insight in the possibilities of the different scenarios for the interconnect, an excell sheet is made which is used to calculate how much bandwidth is needed for different uses of the modules, see 3.6. This excel sheet can also be used in case a new configuration of the extension modules is build, in order to see if the configuration will fit into the bandwidth capabilities offered by the module interconnect. The following properties of each module can be varied.

- No of bits per channel
- No of channels
- Number of modules
- Loop frequency

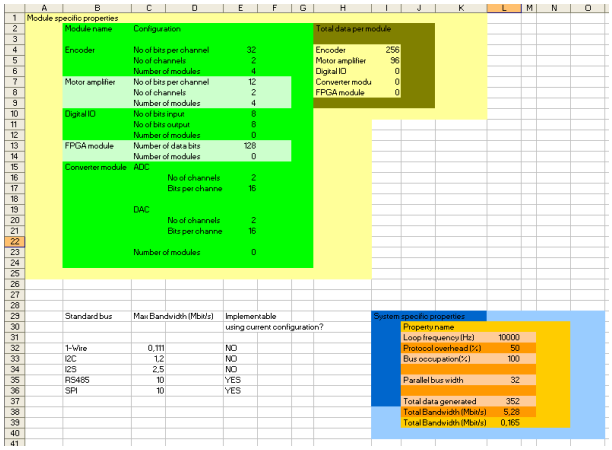


Figure 3.6: Excell sheet used to calculate the bandwidth usage of several different modules

- Bit width (for parallel bus)
- Bus occupation (which percentage of the time the bus is idle)
- Protocol overhead (how much of the bandwidth is spend on addressing, error correction, etc)

The minimum bandwidth needed for a system with 4 modules (two motor controllers, one encodercounter and one digital IO which is a typical motion control application) at a loop frequency in the order of 1-5kHz is approximately 5-10 Mbit/s.

3.2.2 Selected Interconnect

Based on the insight in the data usage and demands for the interconnect, gained in the previous paragraph, it is now possible to select the interconnect.

If we look back at Section 2.2 there are three types of interconnect:

- RTM-USB to the interconnect interface (RT2I)
- Between interconnect interfaces themselves (I2I)
- Between the interconnect interface and the hardware extension (I2HE)

For the connection between RTM-USB and the interconnect interface (RT2I), one of the ARM7 peripherals from table 2.4 needs to be selected. The above calculated bit rate of 5-10 Mbit/s, only leaves room for the following peripherals:

- **Memory bus** Can't be used since it is not available due to the RTM-USB's board design.
- **LCD bus** Could be used since it is available and it is hardware supported, hence easy to use from a software point of view.

- **GPIO** Could be used since 16 GPIO ports are available on the RTM-USB board's connectors, however if using GPIO we would end up emulating the LCD bus protocol on the GPIO bus. The advantage of having a 16 bits bus is partial nullified by the fact that additional handshaking signals need to be used or a complex bus protocol needs to be defined (which poses extra demands on the interconnect interface).

From the above list we can conclude that with the current design of the RTM-USB board the LCD bus is the cheapest and easiest to use solution. However if in the future the RTM-USB board would be redesigned it would be best to use the memory bus. The interconnect interface could just be put in the memory space of the controller, simplifying the protocol to communicate between processor and interface. Another advantage, in favour of the memory bus, appears due to the internal buildup of the ARM7 processor. The ARM7 has the memory controller on the same bus as the processor (AHB, Advanced Host Bus) whilst the LCD controller is on an other bus (APB, Advanced Peripheral Bus). To go from AHB to APB a bridge needs to be passed, which takes extra time and could cause extra delay to the internal functioning of the processor.

The next step in the interconnect is the connection between the interconnect interfaces (I2I) a bus type from table 2.5 needs to be chosen. With the calculated bandwidth, we can simply rule out a few busses, the ones that remain are:

- **SPI** This is the most cost sensitive solution, there are a lot of chips that implement SPI and convert it to parallel IO, the cost of implementation are low, however the noise immunity and speed properties are worse than those on RS485. Additionally we are not free to design our own protocol, we have to stick to the SPI protocol.

- **RS485** This bus is really well shielded for noise (due to the fact that it is a differential bus) the speed is twice as much as the SPI bus (inc case of SPI limited by the available SPI to parallel converters) however the cost to implement RS485 is a lot higher, approx 4 times)

- **LVDS** This bus standard has both a really high bandwidth ($> 200Mbit/s$) and an implementational cost which outweighs the bandwidth benefits by far for this application, hence there is no need to use it here.

Concluding from the above list, there are two winning scenarios. Both SPI and RS485 have equal good properties, but in different fields. Practice has to turn out which fields play a meaningful role. Therefore two setups will be made from which it will be determined which bus, either SPI or RS485, will be used for the connections between interconnect interfaces. Table 3.1 lists the properties of each bus.

Two setups were made to test the differences between RS485 and SPI as shown in figure 3.8. and figure 3.7 respectively. An overview of the used components in the test setup can be found in table 3.2. In the SPI setup the SPI bus is controlled by a Microchip PIC18F4550 because this is both easier to program, compared to the ARM7 chip on the RTM-USB board, and it has an on board SPI peripheral whereas the ARM7 chip needs an external chip to make an SPI port. The SPI bus on the PIC18F4550 has comparable

Table 3.1: SPI versus RS485 hardware

	SPI	RS485
MAX Bandwidth	10Mbit/s	20Mbit/s
Number of chips needed	3	4
Footprint (cm^2)	2.5	4
Effort to implement	small	large
Effort to extend	large	small
Noise sensitivity	high	low
Max number of devices	8	32

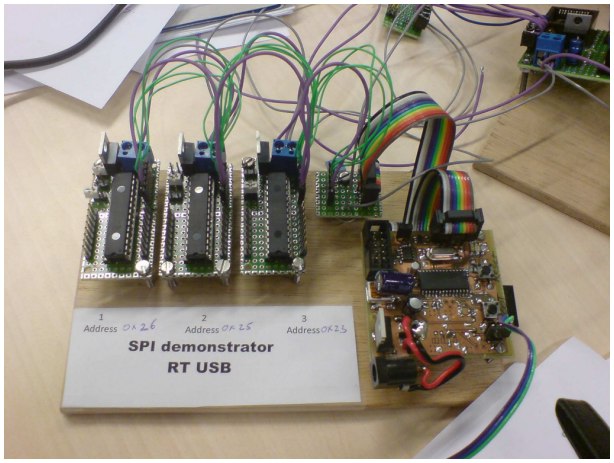


Figure 3.7: Test Setup for the SPI I2I communication

characteristics to an SPI port which will be attached to the ARM7 chip on the RTM-USB

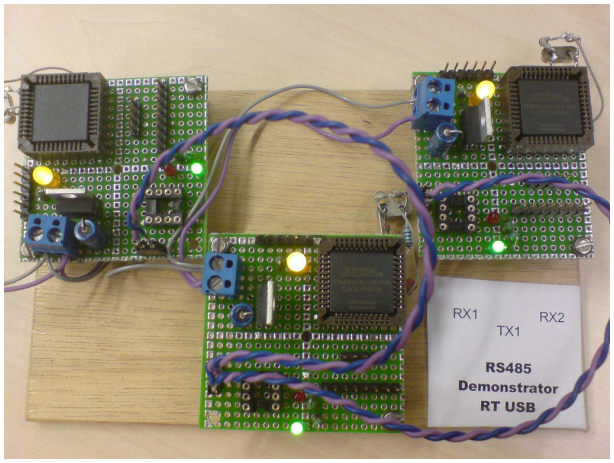


Figure 3.8: Test Setup for the RS485 I2I communication

Table 3.2: SPI versus RS485 setup

	SPI	RS485
Sending chip	PIC18F4525/s	Altera EPM3064
Receiving chip	MCP23S17	Altera EPM3064
Max data rate measured	10Mbit/s	20Mbit/s
Implementation finished	yes	no

board.

During the implementation of the test setup we found out that the physical size of the RS485 solution was bigger (1.5 to 2 times) than we originally had thought. This was caused because it was thought that the CPLD device could be obtained in a smaller SMD package, however this specific device was not available in a smaller package. If we look at a more expensive (in terms of financial cost) devices there are smaller devices available however these become too expensive to be of any use for us. Furthermore the SPI solution was much easier to implement, occupied a much smaller (about 3 times) area and the cost were less. These disadvantage were too big to continue working on the RS485, hence the SPI solution was chosen.

The last part of the interconnect is the interconnect interface and the extension module (I2HE), we need to choose from the options as depicted in paragraph 2.3.4. Since we choose to go for the SPI implementation using the SPI to parallel converter from Microchip (MCP23S17) we are only left with a parallel connection on the I2HE interconnect. This also limits us (severely) on the options we have regarding the protocol which is used on the interconnect, but this will not prevent us from creating a useful working system as we can see in the next chapter about the implementation of the system (chapter 4)

3.3 Interconnect Protocol

In the previous paragraph the SPI to parallel converter (MCP23S17) was chosen for the I2HE interconnect, from the options which are given in paragraph 2.3.6. We are not able to do error detection, since this would require extra intelligence like a micro controller or CPLD. As will be seen in the measurements chapter (see chapter 5) this will not pose a problem.

Multi master is possible and can be done on the driver level in the ARM7 micro controller. However for this thesis we start with single master since this will be easier to debug then a multi master system and once a single master system is acquired it can be extended to a multi master system.

For the initialisation we choose to use a system which is set by the user, the user enters which modules (which addresses) are on the bus and the software assumes that the modules are present. This is again done to have as little as possible intelligence controlling the modules, because all intelligence includes software which needs to be maintained which in turn makes the project more complex. In the end ideally the only

software we want to maintain is the software on the ARM7 processor.

3.4 Designing motion control applications on an FPGA

In this section we look into which interconnect and modules could be implemented on an FPGA. Unfortunately there will be no time during this project to implement anything regarding the FPGA however it is good to know for future refrences what is possible to implement.

3.4.1 Interconnect between functional blocks on an FPGA

As proposed in paragraph 2.4.1 some interconnect to connect all modules in the FPGA should be used. The requirements for this interconnect are: flexible, cost effective and realtime. From the solutions as proposed in paragraph 2.4.1 only Æthereal NOC is realtime, the problem however with this network type is that it is quite heavy and license fees are due. If we look into the Wishbone bus on chip [26] we find a simple lightweight structure, with no license fees (since it is open source), easy to implement and posing little overhead on the FPGA, unfortunately this bus is not realtime. In order to make it realtime, a scheduler should be made to route the data through the interconnect in real time. Another disadvantage of a bus is that the capacity of the bus is shared with all the masters and slaves on that bus (Time Division Multiplexing, TDM), in case a NOC was used multiple (local) data streams could have run over the same interconnect without limiting each others bandwidth. Even though these draw backs are severe, the benefits as mentioned above will outweigh them to a large extent.

3.4.2 FPGA motion control functionality

As already discussed previously in this thesis, there will be no implementation of FPGA code, other than that needed for the interconnect or the motion control extension modules. However in this paragraph we would like to make a proposal for modules which would be implemented on an FPGA in case we had more time left. A selection is made from the list given in paragraph 2.4.2. The following modules could be implemented:

- Stepper motor controller
- Encoder counter
- PWM Controller
- PID Controller
- Filtering

These blocks are all selected for the same reason: they are both basic building blocks of most motion control applications and they can show directly the added functionality of the FPGA. The encoder counter is a special case since we are going to use it for this thesis in a CPLD (unlike the other modules).

3.4.3 FPGA Selection

The FPGA vendor was already selected in paragraph 2.4.3. Using their catalog, a selection of devices can be made. From table 2.7 it is easy to see which FPGA is most beneficial to choose. The ep2c8 (8000 LE) FPGA is a good choise for most of the projects, however for only 1.10 Euro [1] more an FPGA which is twice the size and which is made in a newer technology can be obtained (ep3c16, 16000 LE) besides that the ep3c16 is the biggest FPGA which is still hand solderable (which is an advantage since all the prototypes will be soldered by hand). So for the FPGA board the ep3c16 FPGA from ALTERA is selected.

The smaller CPLDs can be used to implement simple modules (such as counters, PWM generator). In case an FPGA is used, the counter design could be simply transferred from the CPLD to the FPGA.

3.5 Chapter Summary

In this chapter a selection of the extension modules has been made. This selection (brushless DC motor controller, encoder counter and digital IO) will be implemented, see next chapter. In order to connect all these modules together an interconnect bus is selected. Several busses were considered, of which only RS485 and SPI remained as useful candidates. After making two test setups s choice is made for SPI since it is most cost effective, relatively easy to implement and easily extendable by using standard available devices.

A list of FPGA functional modules which could be implemented has been compiled. Unfortunately there is no time during this thesis to implement an FPGA extension module. Several interconnect implementations have been investigated. The main outcome is that the interconnect should be real-time and that a real-time Network On Chip (NOC) is hard to implement, and not strictly needed for the low requirements of the RTM-USB platform.

In the next chapter the selected extension modules will be implemented, we will show how to implement them and which components have been chosen for the hardware implementation.

[1] Prices from http://nl.digikey.com, November 25^{th}, 2008

4 Implementation

In the previous chapters, insight was gained into which modules and interconnect could be *implement and which* would be *implement. Modules and interconnect were designed from a high level view. This chapter will show what was done and needed to implement the modules and interconnect, which problems were encountered and which solutions were found to these problems.*

4.1 Used design tools

The software tools used for the design are freely available as student editions. Of course with some limitations (see table 4.1) but for this project all the limitations were not a problem at all. The only exception to this, is the compiler which was used for the software on the ARM7, for this we used a licensed tool (license was already available at Philips).

Table 4.1: Design tools

Design tool	Used for	Manufacturer	Limitations	website
Eagle	Schematic capture and pcb design	Cadsoft	Max PCB size 10*8 cm Only 1 schematic per design	www.cadsoft.de
Quartus	Verilog design and synthesis	Altera	High end devices not supported Some IP not supported No synthesize optimizations	www.altera.com
uVision 2	ARM7 code compilation	Keil	No limitations (license)	www.keil.com

4.2 Extension Modules

The extension modules of which the functional design has been done in the previous chapters are implemented, schematics of these extension modules can be found on appendix D at the back of this thesis.

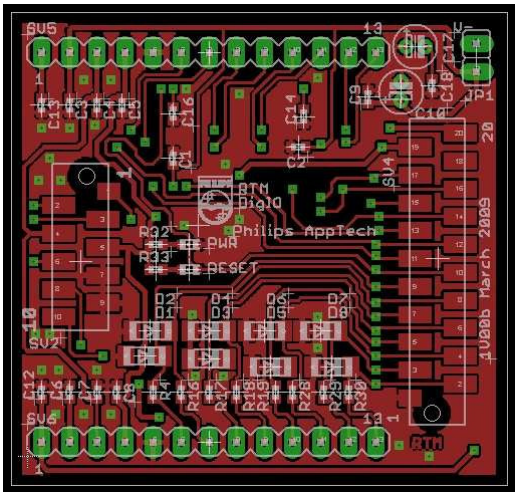


Figure 4.1: Digital IO PCB design

4.2.1 Digital IO

The implementation of this module is quite straightforward, the final result of the design can be seen in figure 4.1. The only problem encountered was the opamp, which is used to shift the output levels up to a higher voltage compared to the input voltage, had a gain bandwidth product (GBW) which was too low. Taking an opamp with a significantly higher GBW solved the problem. In future revisions it might be best to consider using a comparator, since they are faster then opamps and also at a lower cost. However this comes at a price: almost all comparators have an open drain output, hence they can not source current which yields the need for an extra amplifier. The results obtained with the current opamps can be reviewed in the Chapter 5.

4.2.2 Encoder Counter

For this module we chose a CPLD from Altera. This module was not as straightforward as the previous one. Since it had to be usable with both differential and single ended encoders. A simple trick (setting one of the differential inputs to half the voltage supply) was used to enable the encoder counter to work with both type of encoders. The hardware implementation (see figure 4.2) was right in one go, however the software (for in the FPGA) showed a few minor latching problems (see figure 5.4) this was easily solved in the Verilog code. A schematic overview of the Verilog code can be seen in figure 4.3. The final encoder counter is able to count up to two channels in either pulse mode (counting the pulses coming from an encoder) or time mode which counts the number of clock pulses between the edges of encoder pulses. This last implementation is particularly useful to have when the motor is running at low speed. All counters are 32 bit long, which is comparable to most of the commercial available chips.

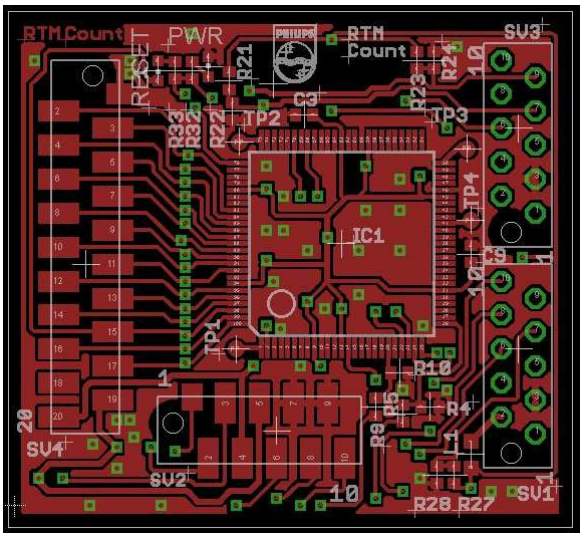


Figure 4.2: Encoder Counter PCB design

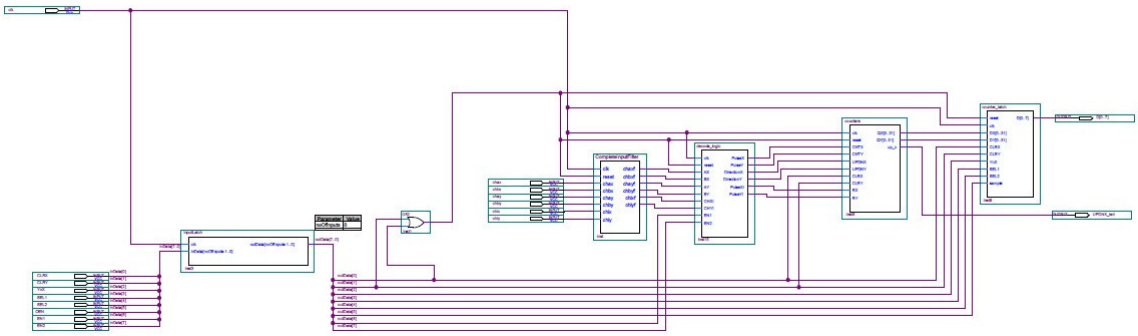


Figure 4.3: Encoder Counter Verilog design

4.2.3 Motor Amplifier

The motor amplifier is based on an amplifier from Alegro (see paragraph 3.1.1) which can be controlled using an analog voltage from the RTM-USB board or an analog voltage from a DAC on the motor controller which can be controlled over SPI. Using the datasheet and scaling the components to the right values an implementation that worked in the first time was made (see figure 4.4. It was found out that the datasheet of the test motor which was used to test the amplifier showed the wrong connections. Unfortunately this costed one motor (one of the hall sensors (see appendix B) was blown up). Once the right datasheet was used this motor controller functioned properly.

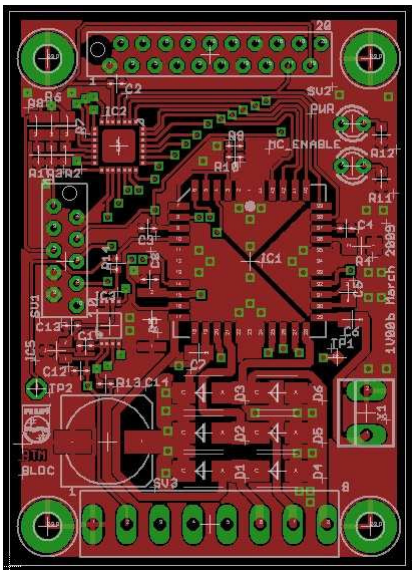


Figure 4.4: Motor Amplifier PCB Design

4.3 Interconnect

4.3.1 Hardware

Two different bus architectures were considered: SPI bus and RS485. As was seen in the previous chapter, SPI was more effective to implement in terms of cost, (software) maintenance effort and physical dimensions. The chip that was finally taken as the receiver was an MCP23S17 SPI to parallel port expander from Microchip, which will be placed on all hardware expansion modules.

From the processor side there should be data send onto the SPI bus. The RTM-USB board is not equipped with an SPI bus, hence we need to come up with an interface to glue the RTM-USB board to the SPI bus (called RT2I in paragraph 3.2). For this interface a choice of devices can be used (see table 4.2). The CPLD solution is both the fastest and most cost effective solution. An advantage we get for free here is that in case we take a bigger FPGA it would even be possible to put the SPI bridge and the encoder counter (both written in Verilog) together such that they can be put on one board. Using this CPLD the SPI bridge board (see figure 4.5) was created. This is a board with four SPI channel outputs and a parallel input. Four channels were constructed since this number of channels fitted in the available space and the four channels together give a serial data speed which is equal to the parallel data speed (40Mbit/s).

On to each channel at most eight modules can be connected, which is a limitation of the MCP23S17 SPI chip's addressing capabilities. The parallel port of the SPI bridge board connects to the LCD control port of the RTM-USB board, in this way both GPIO (easy to debug) and the LCD bus (faster connection) can be used for sending data over to the SPI bridge. Some Verilog code was written to split the incoming data into four channels and to shift out the data on the SPI channels, the code can be configured using parameters, hence if more SPI channels are needed it is simply a matter of changing the

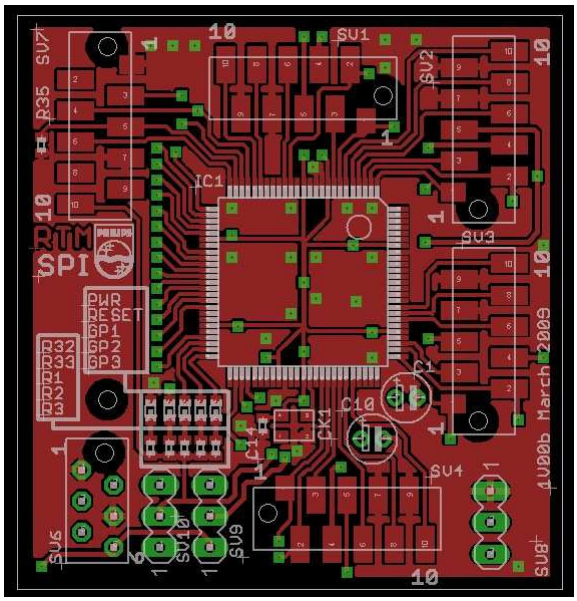


Figure 4.5: SPI PCB Design

parameters in the code and no rewriting of the code itself has to be done. Again here the design was done tidy and both hardware and software worked almost in one time (there were some minor bugs, but nothing structurally wrong).

Table 4.2: Bus master controllers

Controller	SPI speed	Package	Internal/external Xtall	Cost (Eur)
Tiny48	6Mbit/s	QFN28/32 TQFP32	external	1.10
Atmega 128	8Mbit/s	QFN64/32 PDIP TQFP BGA	external	7.69
CPLD (EPM240)	40Mbit/s	TQFP100	external	2.00
8051 (C8051F52X)	6Mbit/s	10DFN	external	2.75
ARM7 (LPC2100)	10Mbit/s	LQFP48	internal	3.30
MSP430 (2013)	10Mbit/s	TSSOP14	external	2.47
PIC16F726	4Mbit/s	28SSOP	external	2.20
PIC18F4550	11Mbit/s	TQFP44	external	6.87

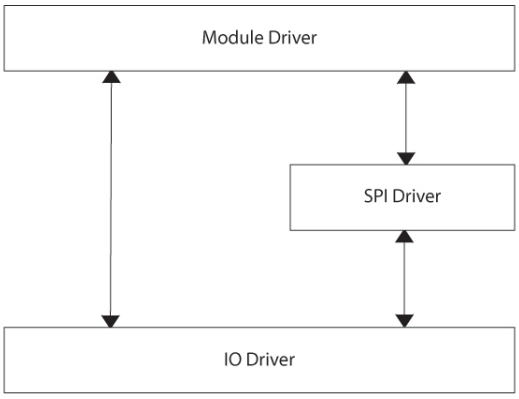


Figure 4.6: Driver Software design

4.3.2 Software

Besides the software in the CPLD (written in Verilog) device drivers had to be written to control all the modules from the RTM-USB board. It was chosen to make a hardware independent layer for each hardware module. This layer talks to the lower IO port driver directly (in case the modules are directly connected to the parallel port of the RTM-USB board) or to the SPI driver in case the modules are connected over the SPI bus (see picture 4.6). All software is written in the C language.

4.4 Cost Calculation

Since this whole project is about designing a real system which can be manufactured in a moderate quantity, the cost needs to be taken into account. This chapter takes all the solutions which will be implemented as can be found in chapter 3 and makes a rough cost calculation. This cost will be the financial cost only. Other notions of cost are being left out.

4.4.1 Extension Modules

Per module the cost price of the components can roughly be calculated, by picking the main components and add a 10 % for the additional components, table 4.3 gives such an overview.

Prices are taken from nl.digikey.com as per may 1, 2010 Each module has additional financial cost which is caused by the network interface, see paragraph 4.4.2 for the cost per module. It might be good to notice that the price of the encoder counter board is half the price of comparable commercial solutions (for which price only the chip is obtained, a board should still be designed).

Table 4.3: Component price

Module	Component	Price (EUR)	total price(EUR)
Encoder counter			
	CPLD	5	
	Power regulator	0.58	
	X-tal	0.75	
	Connectors	0.80	
			7.60
Motor amplifier			
Alegro A3936	Motor amplifier chip	4.00	
	Connectors	0.80	
	X-tal	0.75	
			6.05
Digital IO	Opamps	2.00	
	Discrete components	3.00	
			5.50
SPI bridge	CPLD	2.00	
	Oscillator	1.00	
	Connectors	1.20	
			4.40

4.4.2 Module Interconnect

The module interconnect adds additional cost to the modules, cost that can be left out in the case that the modules are used only through the parallel port. In table 4.4 the overhead cost can be found.

Table 4.4: Component price interconnect

Component	Price (EUR)
MCP23S17	2.25
Connectors	.50
Total Price (EUR)	2.75

4.5 Chapter Summary

In this chapter the hardware extension modules and interconnect have been implemented. First the design tools have been discussed, following the schematic and PCB design of the modules. Implementation went quite smooth, due to the fact that at each design stage (schematic, PCB) the designs were carefully reviewed by colleagues with some hardware design experience.

After the implementation a cost price calculation is being done over the main components to see how much the extension modules and interconnect cost (financially). On average the module's hardware cost is less then 10 Euros including interconnect, which is an absolute minimum for modular motion control hardware.

5 Measurement Results

In this Chapter the results of the measurements as carried out on the previously designed and implemented hardware, are presented. Each paragraph covers the measurement of one single RTM-USB extension module, the last paragraph combines all these modules in order to test the interconnect. For each module the method is the same: The first part introduces the measurement setup, following a short discussion about which measurements will be carried out and why they are useful for this module. The second part of the paragraph presents the results of the measurements with the aid of some signal plots and measurement data

5.1 Extension Modules

5.1.1 Digital IO

A single Digital IO module (see paragraph 4.2.1) is taken. This module is wired according to figure 5.1, using this setup it is possible to measure the input, output and input after clamping voltages. The used oscilloscope is an **Agilent DSO1024A**the signal generator is a **Philips PM5138A**. Figure 5.2 and 5.3 show two example plots of the measurements. A summary of all the measurements can be found in table 5.1.

Table 5.1: Measured capabilities of the Digital IO extension module

Measured property	Result	Remarks
Max differential output voltage	35,9 Vp-p @ 143,7KHz	Higher voltages can damage the opamps
Max in voltage	18,3 V	This is the clamped input voltage @ 1.5mA
Min input voltage	-17,7V	This voltage is clamped
Max input frequency	1.506 MHz	Beyond this frequency the square wave becomes a sinusoid
Max out frequency	1.01 MHz	As generated by the digital IO board

5.1.2 Encoder Counter

In order to test the encoder counter implementation, simulations were carried out using the Altera Quartus simulator and Mentor Graphics's Modelsim. After these tests the encoder implementation was uploaded to the CPLD and an Maxon Motors quadrature encoder counter was hooked up to the encoder counter input (see fig 5.5 to see an overview of the used test setup). First tests showed weird behaviour (see fig 5.4). The

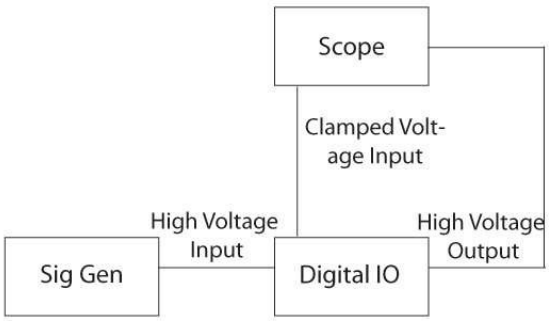


Figure 5.1: Digital IO test setup wiring diagram

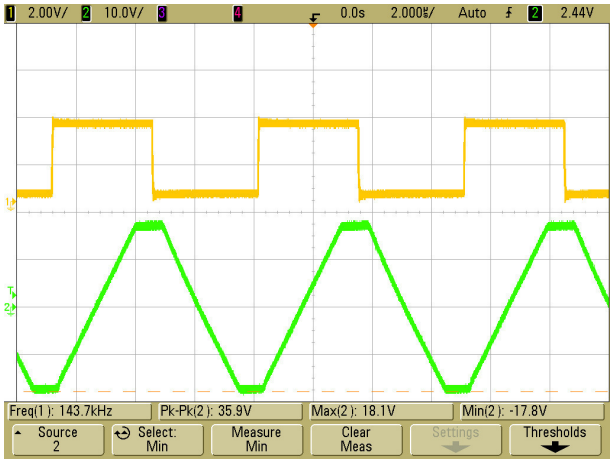


Figure 5.2: Digital IO maximum output frequency at maximum output voltage

input to the motor is a steady voltage (hence the motor should run at constant speed. The previously mentioned figure shows that the count goes backwards at some instance of time and after that it goes continues its normal expected behaviour. It was found that this was simply a mistake in the processing of the incoming read signals from the RTM-USB board. A simple latch, implemented inside the CPLD, solved the latching problem. In order to be able to test the maximum count speed of the encoder counter, a setup was made where another CPLD emulates the encoder (in this case it is possible to set the speed of the encoder using a function generator). The counters are implemented as sampling counters, hence after 25M counts per second a degradation of the counted value can be observed, which is not surprising since this is half the sampling speed (50 Msps) as per the Nyquist theorem data loss will be observed if input signals go faster then half the sampling speed.

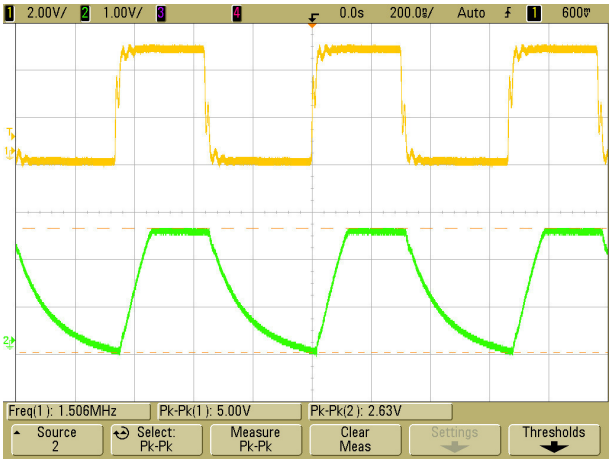


Figure 5.3: Digital IO max input frequency, channel 1 is input channel 2 is output

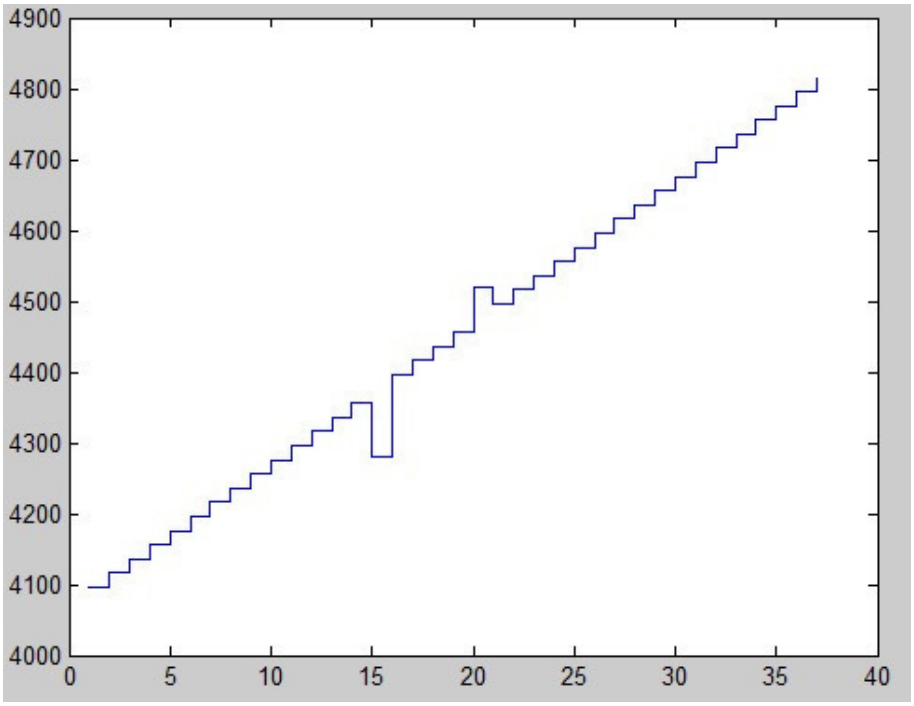


Figure 5.4: Spikes in the readout of the encoder counter, these spikes were caused by latching problems.

5.1.3 Motor Amplifier

The motor amplifier was tested by connecting a Maxon EC 25 Watt motor to the amplifier board. This testing was only performed in order to show that the motor amplifier was functionally correct. To fully test the maximum output power of the amplifier, a test setup was build (see figure 5.6). This setup contains three resistors in a star configuration. Each of these resistors is able to dissipate 50 Watts, while cooled with a fan (to

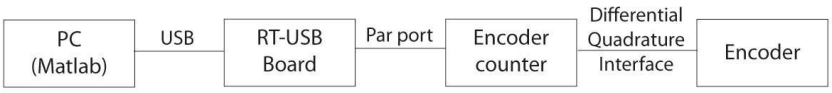


Figure 5.5: Setup to test the encoder counter

make sure the solder between the registers does not melt) for future setups clamps should be used to hold the resistors together, instead of solder. In order to make the amplifier work, a HAL sensor emulator is made, see figure 5.7. This is easily done in a CPLD. Results from this test are that the amplifier is able to supply approximately 10 minutes 150 Watts (3A@50V) (after that the test setup heated up too much and the solder could not be cooled enough anymore). Hence the specifications from the manufacturer are reachable using our own designed setup.

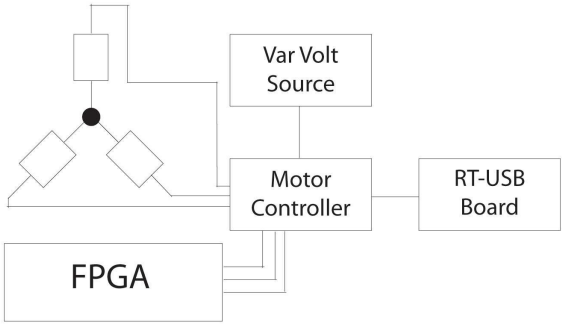


Figure 5.6: Test setup to test the full power delivered by the motor amplifier

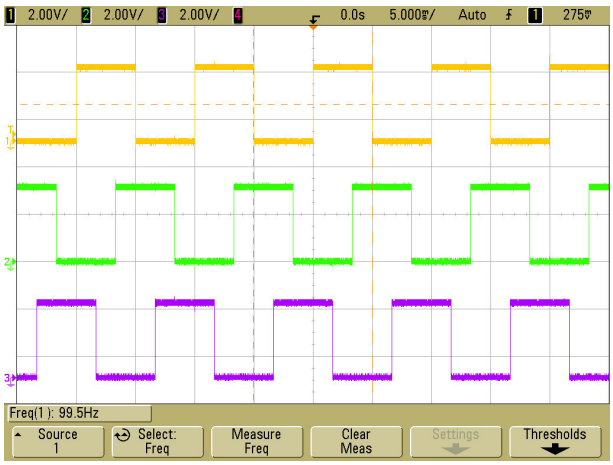


Figure 5.7: BLDC Motor Control Hal sensor input as generated with an FPGA while simulating a BLDC motor

5.2 Interconnect

5.2.1 SPI Bus and SPI Bridge

In order to test the SPI bridge and all the SPI enabled peripherals a setup was made. This setup contained two Maxon EC motors, one encoder counter module (with two encoder counter channels), four digital IO boards (32 channels). The goal of the setup is to test what the loop frequency is, and if the various parts of the SPI protocol work correctly. See figure 5.8 for a schematic overview of the setup and figure 5.8 for a photo of the setup. From figure 5.9 it can be observed that the maximum loop frequency is 1.18 KHz, which is lower then was expected. This is partially due too the switching overhead between sending data to the motor controller's SPI DAC and sending data to the modules. If a parallel DAC was to be used (which needed too much space to implement here) the overhead of switching (approximately 150 micro seconds) would not be necessary. Data would be send in the normal way taking approximately 50 us, hence giving a total time gain of 100 us. Further more the figure shows big gaps between data transfers (about 40 us) these gaps seem to be caused by the slow data transfer between the RTM-USB board and the SPI bridge (the RT2I interconnect). Speedup could be achieved here by using a faster interconnect (memory bus or the LCD interface). These gaps are about 40 us wide and in the total PID loop there are 15 gaps, hence there is about 600 us to save. Taking all these optimizations together, a loop speed of 114 micro seconds could be achieved, which brings the loop frequency to 8,77 KHz. Which is much closer to what was calculated in the first chapters. Due to hardware restrictions and time constraints it was not possible to implement these optimizations.

Note: in all of the following figures which describe the SPI testing, channel 1 is de chip select, channel 2 is the SPI clock, channel 3 is the data from the bridge to the module, channel 4 is the data from the modules to the bridge.

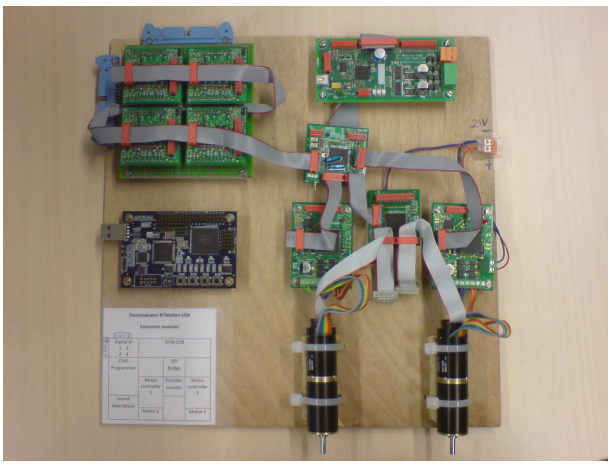


Figure 5.8: Test setup to test the SPI functionality

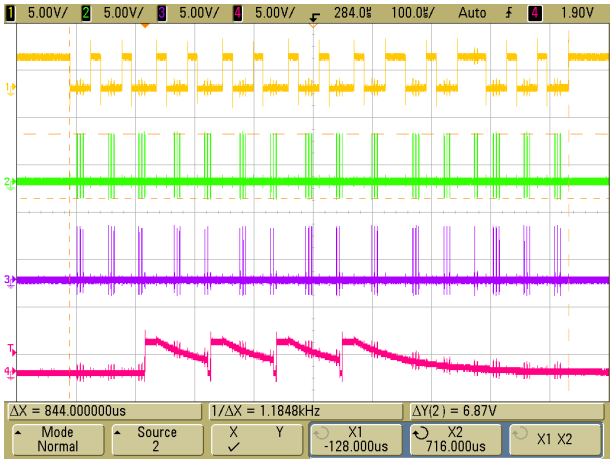


Figure 5.9: SPI Bridge with one complete PID loop

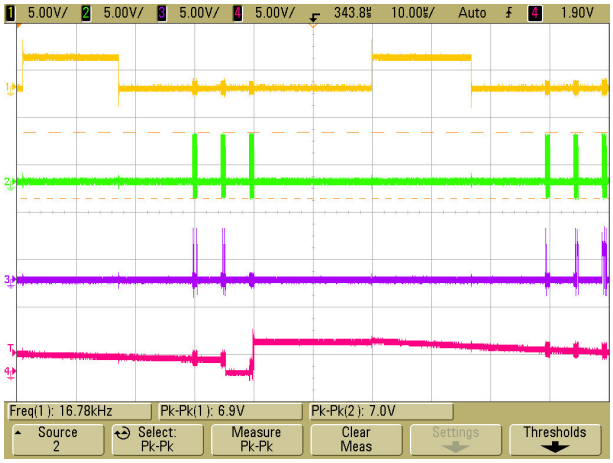


Figure 5.10: SPI Bridge receiving data from the encoder counter extension module

5.3 Chapter Summary

In this chapter the performance of the modules is being measured. All modules function according to the specifications, or better. Some small problems were found, but they were solvable in software. The SPI bridge shows that there is quite a lot of room for optimization. After optimization the SPI bridge should be able to handle 8 times more data traffic.

In the next chapter we will see how this SPI bridge and one of the modules (Encoder Counter) are successfully being integrated in the new version of the RTM-USB board.

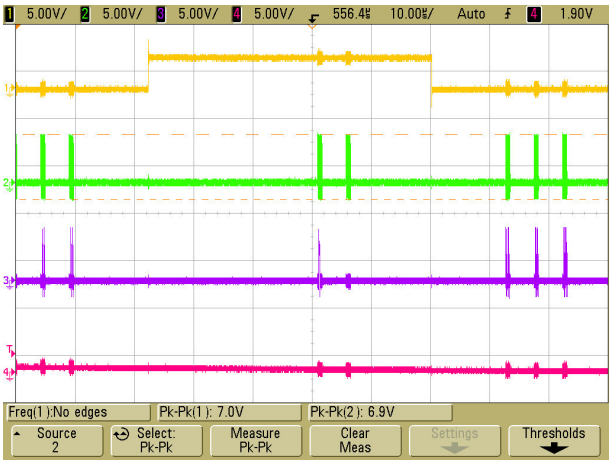


Figure 5.11: SPI Bridge sending data to the DAC on the BLDC motor control extension module

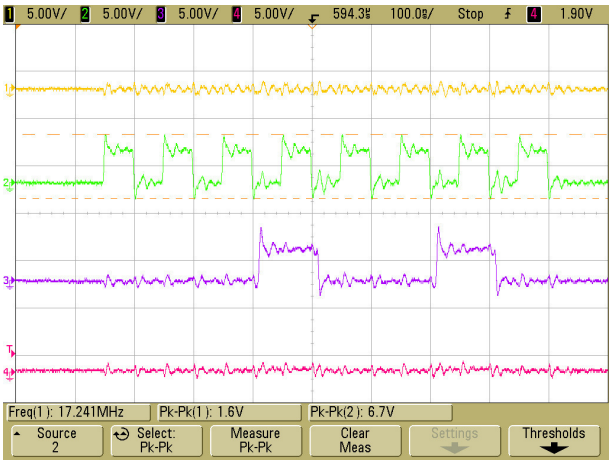


Figure 5.12: SPI Bridge sending one byte to an extension module

6 RTMotion-USB New Version

During the last few month of this thesis a request came in to make several robotic arms for university research. This meant that there was a budget for redesigning the RTM-USB board and putting the main components of this thesis on this new designed RTM-USB board. This chapter will briefly touch the design process of the new board and the outcome of the build boards.

6.1 New RTM-USB Board design process

The main purpose of the redesign of the RTM-USB board (and creating its 3rd generation) was to get the two channel encoder counter and the SPI bridge on the RTM-USB board. In this case it is still possible to attach peripherals (like the motor controller and digital IO) to the RTM-USB board and the most used peripheral, the encoder counter, is on board creating a more space economical implementation. As CPLD a chip was taken which was twice as big as the chip used for the encoder counter such that both the encoder counter and SPI implementation would fit in the same device. The CPLD is connected to the memory interface of the processor on the RTM-USB bus such that in the future a high speed data transfer channel can be setup. Currently the memory bus pins can also be used as General Purpose IO which minimizes the design risk.

For this design process Philips's internal PCB design house, Green house, was asked to implement the design changes according to our specifications.

6.2 New RTM-USB Implementation result

After a few weeks a batch of boards came in (see 6.1) and since all the software was already fully debugged, the only thing that needed to be done was changing the Hardware Abstraction Layer (HAL) on the RTM-USB's processor and combining the CPLD Verilog code of the SPI bridge and Encoder Counter. This whole process took about two days to get the software up and running. Almost no bugs were found after integrating the previously tested and validated hardware and software. This shows the power of the modular design philosophy of the RTM-USB motion control platform.

Finally, after testing the hardware on a functional level, the board was integrated into a robotics arm (see figure 6.2). Several of these arms are being sold to the three technical universities in the Netherlands.

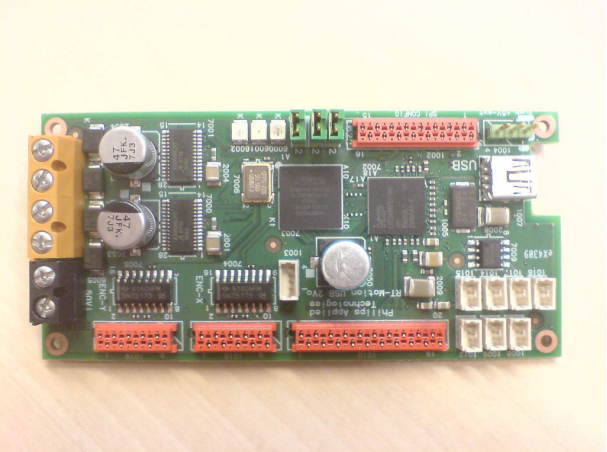


Figure 6.1: The new RTM-USB hardware

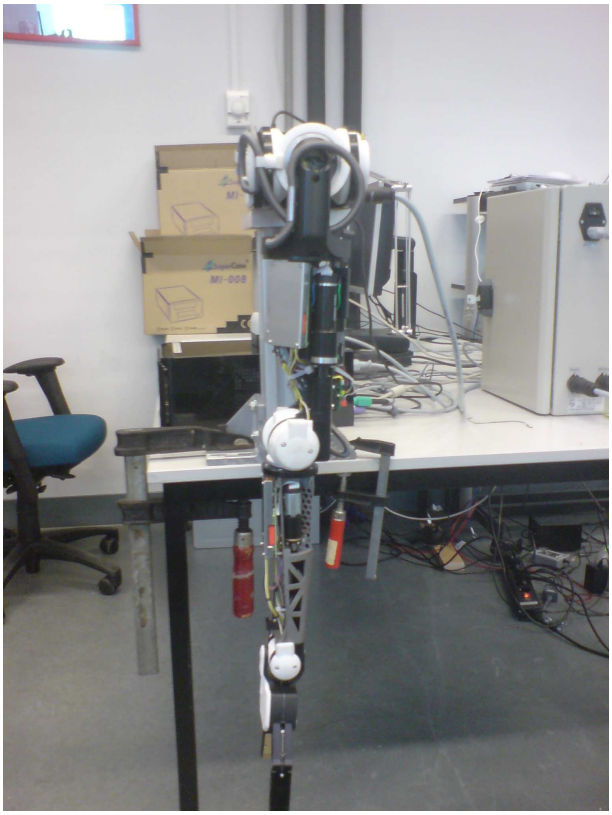


Figure 6.2: Robotics arm where the new RTM-USB hardware is integrated

RTMotion-USB Hardware Future Work And Conclusion

7

Since this thesis is not the final version of the RTM-USB platform, there is a need to look ahead, to see where the platform should be heading for in the future. This chapter gives a summary of the thesis, shows the contributions of this thesis to the world and which work should be done in the future to continue and improve the current design.

7.1 Thesis Summary

First a study was performed to see which comparable systems are available on the market. Two high end systems (meant for high precision control applications like medical systems and semiconductor), one low end (education) and one middle end (simple control applications) systems were found. With respect to these systems the RTM-USB is targeted between the middle and high end applications (of course whilst respecting the limitations of the low cost nature of the RTM-USB board).

When looking into comparable systems it was also found which kind of extension modules should be created in order to make the RTM-USB board an attractive, competitive board. A list of several extension modules is made and the modules that are mostly needed in control applications (encoder counter, motor controller and digital IO) were selected to be designed and manufactured for this thesis. Additional the possibilities of using FPGAs in motion control applications is investigated. Unfortunately there was no time during the course of this thesis to implement an FPGA module.

Since this RTM-USB prototyping system should be modular it means that there should be an interconnect between the modules. Since the design risk should be minimized, it was chosen to take both a serial and parallel bus to serve as the interconnect. The interconnect for the *serial bus* is divided in three parts, RTM-USB to interconnect interface (RT2I), interconnect interface to interconnect interface (I2I), interconnect to hardware extension module (I2HE). For each of these parts a communication protocol is determined. For the RT2I the LCD interface available on the controller on the RTM-USB board is being used. The I2I interface uses the SPI (Serial Pheripheral Interface), for the I2HE there is not much to choose since it is determined by the chip that is being used to connect to the SPI bus. The *parallel bus* is normal simple GPIO with intelligence only on one side, hence there is not much of protocol that needs to be used here either.

Implementation was done in two steps: The *first step* was performed on the computer (drawing PCBs, Printed Circuit Boards, and writing code in Verilog and C). The *second step* was performed in the lab (soldering and testing the PCBs).
After implementing all the modules, relatively little design mistakes are found (most of the problems find their roots in wrong interpretation of the available data, small mistakes, forgotten components, etc.). Measurements unveil that the designed modules work as good as expected or even better (in case of the encoder counter extension we

were able to implement 2 types of encoders, where initially we were only planning to implement one encoder).

At the final stages of this thesis a need for redesign of the RTM-USB board emerged, since this board was needed in humanoid robotics arms that are sold to universities for research. During this redesign it was possible to take away the biggest bottleneck in the communication (being the RT2I interconnect, where GPIO was replaced with the memory bus interface). Unfortunately it was not possible to get the memory bus up and running during this thesis due to time constrains (the normal slow interface *does* however work).

7.2 Contributions

This thesis contributes in several ways:

- Low cost modular middle- to high-end motion control prototyping is possible now.
- Encoder counters became more cost effective and versatile.
- It is shown that even in cost effective motion control programmable logic (CPLD/FPGA) can be used without compromising the cost aspect.
- The functionality of the RTM-USB motion control board has been increased.
- A new version of the RTM-USB motion control board has been prepared to be part of a series made product which is sold to customers outside Philips.

7.3 Future Improvements

With the redesign of the RTM-USB board (which is currently in its 3rd generation) lots of improvements have already been made, a few more improvement suggestions are being done here. There are several improvements that could highly improve the RTM-USB modular design.

- **More extension modules:** designing and making more modules does not cost too much time however it would make the RTM-USB board more flexible. Examples of these modules would be
  - FPGA module.
  - Digital to Analog and Analog to Digital converter board.
- **Removing bottlenecks:** right now the interconnect interface to interconnect interface (SPI bus) is one of the major bottlenecks. If time and budget permits a more advanced higher speed bus should be used to get more peripherals sharing one bus in order to save cable overhead in the robotics application.
- **Removing bottlenecks:** the memory bus of the new RTM-USB board is not yet up and running, hence this should be implemented in order to get a faster data connection between the RTM-USB's processor and the SPI bridge.

- **Testing:** the modules have been tested on a functional level, but in order to produce them in series they should be evaluated according to a few industrial standard tests like: EMC (Electro Magnetic Compatibility) and CE.
- **Marketing:** right now the RTM-USB platform is only used in selected prototypes or products, mainly internal to Philips. This prototyping platform has the potential to grow into a product that is widely used, since it is very low cost and easy to use.

7.4 Overall Conclusion

When this thesis was started in the fall of 2008 a motion control prototyping system was envisioned which is modular, easily expandable and highly cost efficient.

At the end of this thesis we can conclude that we were able to create such a system. We even reached beyond this vision since it was never intentioned to have the RTM-USB board redesigned and sold outside Philips. Furthermore enough study has been done to increase the use of the RTM-USB system.

During this theses a lot has been learned about the design of electronics hardware for motion control, which was entirely new for me. Inside the Philips Applied Technologies motion control department, there did not exist a concrete background in hardware design. Never the less with my own experience in hardware design, manufacturing and debugging it was possible to extend the RTM-USB motion control platform in a relatively fast and easy way.

We have seen that the strategy of first building a modular prototype and afterwards integrating all the modular parts to one single board, proves itself (i.e. there were no hardware bugs in the 3rd version of the RTM-USB board). For the future we foresee that the RTM-USB board and the extension modules, with some small improvements, can be widely used as Lego-blocks to do easy and fast prototyping of motion control systems.

Motor Amplifiers In Motion Control

In this appendix we take a look into which amplifiers are being used in motion control. First an overview of the different types of motors, which are being used in motion control, is given. Using this overview a choice of amplifiers is presented, stating which motors can be controlled, which kind of interface is available on the amplifier, how much power can be delivered, etc.

A.1 Motors In Motion Control

A variety of motors is being used in motion control. The following list gives an overview of the motors used (in order of importance).

- Direct Current (DC) motor. Low manufacturing cost, easy to control, but still brushes are needed which suffer from mechanical wear and cause large EMC emission.

- Brushless DC (BLDC) motor. The inverse of a DC motor (magnets rotate, coils are standing still), needs a moderate complex controller, does not have brushes which wear out or corresponding EMC problems.

- Stepper motor. Needs a simple controller, moves in discreet steps hence easy to position, however construction is expensive due to the mechanical structure of the motor. Control algorithms are available to increase the accuracy of every stepper motor by a factor two (half stepping) which goes on the expense of extra power usage.

- Linear motor (available in both DC and BLDC versions). Used whenever a linear instead of rotating motion is needed, good performance at both high and low speeds.

- Piezo actuator (high precision stepper motor). Used in applications which require a high precision (nanometer accuracy), needs high voltages to control (>1000 V) at a high frequency (>50KHz).

- Alternating Current (AC) motor. Can be connected to mains supply directly, motor speed is controlled by the frequency of the AC source which makes speed control complex. This type of motors show bad performance at low speeds, which is important for robotics motion control since this type of control uses a lot of stopping and starting.

- Switched reluctance motor. Cheap to design since the rotor is only a aluminium cage and hence no magnets or coils are needed on the rotor however at low speeds we can see high torq ripples.

A.2 Motor Amplifiers

In order to control a mechanical motor electrically, a motor controller is needed. Motor amplifiers can be listed in three categories:

- **Off the shelf:** a plug and play solutions (see table A.1).
- **Single chip:** complete motor driver in one package. Only needs some simple components to function (see table A.2).
- **Custom solution:** Completely self designed amplifier, needed if the other solutions are not usable (due to lack of power and/or high cost) (see table A.3).

For the off the shelf and single chip solutions a few manufacturers and specific product names are mentioned, for the custom solution some topologies and controller strategies are mentioned.

One of the solutions which is particular interesting are the ones from Cirrus Apex. These (SA306 and SA57) are single chip solutions which have an integrated controller for DC and BLDC motors and can deliver up to 400W.

For this last category any output power is possible, it just depends on what kind of switching elements (mosfets, IGBTs, etc) are placed. The good thing about these custom solutions is that moving to a bigger motor only requires changing the switching elements and not a redesign of the whole controlling circuit.

The circuits which are being used for the custom solutions contain a microcontroller/DSP/FPGA running a program that does all the calculations and controls the amplifier part, see table A.4 for some examples.

Table A.1: Off the shelf power amplifier manufacturers

Manufacturer	Power (W)	Current (A)	Voltage (V)	Interface	Motor Type	Setpoint
Thor[18]	50	3	85-264	USB	DC Piezo Stepper	
Arcus[3]	–	–	12-48	USB	Stepper	
Galil[9]	500	4 10	24-80	Ethernet RS232 PCI ISA	Stepper DC servo	
Elmo[7]	160-9600	3 50	11 60	Simple IQ [8]	DC Brush Sine Trapezoid	
Eriks[12]	60-500	3-25	12-40	Analog 0/10 ± 15 0/5 ± 10 RS232 CANopen Sercos[25]	DC DC Brushless AC Stepper	
Trust[?]	25-600	1-6	15-24	analog (± 10V)	DC/BLDC	Current
Baldor[?]	630 135k	270	500	Digital (serial) Analog	DC AC	Current Voltage
Aerotech[?]	10 9.6k	10 30	10 320		DC BLDC	Current Voltage
Maxon[?]	24 1000	2 20	12 50	Can open RS232	DC BLDC	Current Voltage

Table A.2: Single chip amplifier solutions

Manufacturer	Power (W)	Current (A)	Voltage (V)	Interface	Supported motors	Cost[1] (Eur)
Cirrus/Apex	4.5 - 10000	0.010 - 50	38 1200	Analog	DC/BLDC	18- 710
Sanyo	1- 200	0.4- 4	2.5- 50	Analog Digital	DC/BLDC Voice coil Piezo/Stepper	10- 50
Infineon	10- 1000	0- 40	0 42	Analog Digital	DC/SERVO Stepper/ASM	–
National Semi	10- 150	1- 3	4.5- 55	Analog Digital (address bus)	DC Stepper	5- 50
Alegro	10- 150	0.65 3	4- 50	Analog Digital (I2C/Serial)	DC/BLDC Stepper	2 20
ST Micro	0- 150	1- 5.6	8 - 52	Analog Digital	DC/BLDC Stepper	5- 50

Table A.3: Custom solutions

Amplifier Topologies	Motor type
Full H bridge	DC
	BLDC
	Piezo motor
Half bridge	DC (only one direction)
	Stepper motor
DC High side/ low side	Stepper motor
	DC motor (only one direction)
	Three phase reluctance motor
3 phase inverter	AC motor

Table A.4: Motor Controllers

Controller manufacturer	Controller type (bit)	Controller number	Performance (Average CPI)	Device Cost (Eur @100Pcs)	User IO	Pin count	IDE cost
Microchip	16(DSP)	PIC24FJ64GA004	2	3.03	22	28	Free
Atmel	8(RISC)	AT90PWM3	1	2.15	19	32	Free
Texas Instruments	32(DSP)	TMS320F28015	1	3.81	35	100	$495.02
Texas Instruments	32(ARM7)	TMS470R1A64	1.9	5.10	39	60	Free
Texas Instruments	16(RISC)	MSP430F2112	2.5	1.88	24	28	Free
NXP	32(ARM7)	LPC2141	1.9	4.16	45	64	Free

Encoders

B

Encoders are sensors which are mounted on the shaft of a motor. The signals coming out of these encoders give information (direct or derivable) about the direction, speed, position or acceleration. There are many different types of encoders. In this appendix we look closer into which encoders are available in the market and how they function.

B.1 Different Types of Encoders

Very often encoders are divided in two main categories:

- Incremental
- Absolute

The *incremental* encoder is an encoder which gives an output pulse every time a marker is passed, multiple pulses can be generated per revolution. If the manufacturer tells the user how many pulses are given per revolution of the encoder (mostly a power of 2: 128, 256,...) the pulses at the output can simply be counted and hence it is clear how many degrees the motor has turned since the counting was started. However we can never know what the position of the motor was at the moment counting was started, simply because the incremental encoder doesn't provide this. This is why *absolute* encoders are invented. These encoders give at power on, the absolute position of the motor shaft (of course with respect to a known zero point on the axis), the typical output is a logic parallel (many wires!) or serial bus. Now the main difference between these encoder types is the implementational complexity (see paragraph B.2), an absolute encoder is way more complex to implement (more electronics, more logic, etc.) as compared to an incremental encoder.
As a kind of hybrid solution between these two types of encoders there exist incremental encoders with an index pulse. This index pulse is given once every revolution of the encoder. Hence in order to know the absolute position of the encoder shaft we need to rotate the motor just as long as to find the index pulse. Ofcourse this solution doesn't solve all problems: it also introduces problems if we start the routines to find the index pulse at the physical limit of the mechanical construction which is controlled by the motor. In this case, rotating the motor might destroy the mechanical construction.

Now how exactly can the position and direction of rotation be found? In case we are using an *absolute* encoder the direction can be easily determined (counting up is one direction, counting down the other) and of course the position is directly found as is explained above.
In the other case, where *incremental* encoders are used, some tricks need to be used

to find out what the direction and position of the motor is. Basically there are two methods:

- Quadrature
- Sin/Cos

A quadrature or sin/cos signal consists of two signals which are 90 degrees out of phase. See figure: add figure

If the picture is read from left to right, signal A comes before B, if the picture is read from right to left B comes before A. Now using some simple logic it is possible to detect which signal comes before the other signal. Hence we can determine the direction in which the encoder is rotating! How these quadrature signals are constructed at the encoder side is discussed in paragraph B.2.

When using the quadrature (square) wave output we have an accuracy in degrees of 360/(number of pulses per revolution), where "number of pulses per revolution" is 512 typically for humanoid robotics applications (with a resolution of $360/512 = 0.7$ degrees) In case higher accuracy is needed, a Sin/Cos encoder could be used. This encoder gives two sinusoidal signals which are shifted by 90 degrees and typically one sinosoid per revolution. Since these signals are analog we can determine the position as accurate as we can sample the analog signal. If we take a 12 bit AD converter we see that the resolution increases to: $360/4096 = 0.088$ degrees which is roughly a 10 times increase in accuracy! Each additional bit doubles the accuracy.

As with everything there is also a catch here, the analog signal has to travel from the encoder to the AD converter. If this signal path is long, the motor introduces a lot of noise and hence the extra accuracy is very likely to degrade.

There exist several ways in which data can be send from an encoder to a processing unit. One is, as we saw earlier, using two simple wires over which a quadrature signal or a sin/cos signal would go, however more robust protocols have been created. The following two are the most popular protocols:

- Endata 2.1/2.2
- HiPerFace

EnDat(Encoder Data) is a protocol defined by the manufacturer Johannes Heidenhain GmbH this protocol let both incremental and absolute encoders communicate over the same interface. Absolute encoders are being read out in serial, solving the "many wires" problem (see above). Furthermore the incremental and quadrature encoders can be used together, the absolute encoder is used to see what the position is of the rotor at startup. The incremental encoder, i.e. a Sin/Cos encoder, can be used to determine the exact location.

HiPerFace (High Performance Interface), by SICK—Stegmann, is basically the same as Sin/Cos (which was invented by the same manufacturer) however a differential RS485 line is added to this bus to send parameters (max motor current, motor voltage, etc.) to and from the encoder.

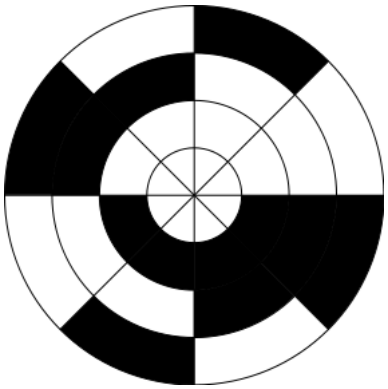


Figure B.1: 3 bit binary absolute encoder disk

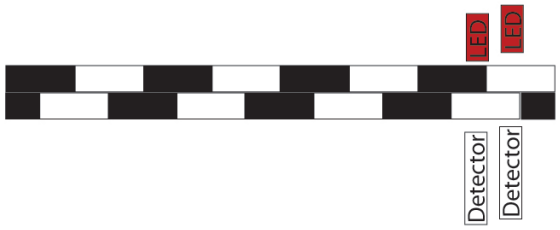


Figure B.2: Track and optical sensors to create a quadrature encoded pattern.

B.2 Physical Implementation of an Encoder

Encoders are devices which should be as simple to build as possible, being a mechanical device, the more parts are used in the design the more likely the encoder will fail early in its lifetime.

Hence most of the time there is an optical connection between the moving and static part of the encoder. Incremental encoders have a comb shaped structure attached to the moving part (see figure: B.2) then a led/optical receiver on the fixed part is used to distinguish between the black and white spots. Absolute encoders work in a similar way. In stead of a comb shaped structure there is a more complex structure like a gray skill encoded structure (see figure: B.1).

If a quadrature signal is needed it can be created by using two tracks which are 90 degrees out of phase and use two detectors to read both tracks (see figure: B.2). There is a very simple optimisation step, the detectors could be placed (multiples of) 90 degrees out of phase. This eliminates the need for a second track, hence saving on space and parts. Besides optical reading it is also possible to create an encoder output signal using a magnet and a hal sensor. This is most convenient if a sine/cosine output is required (using one magnet and two 90 degrees out of phase placed hal sensors yields the correct output).

FPGAs and CPLDs an Introduction

C

This Appendix is written for those who are not familiar to FPGAs and CPLDs. The first part of the chapter discusses a few commonly used terms and abbreviations. Following a discussion of the physical buildup of the devices, emphasis will be put on the differences between FPGAs and CPLDs. Since FPGAs and CPLDs need programming, we will discuss a few programming languages and tools in the last chapter.

C.1 Terminology

FPGA: Field Programmable Gate Array, Contains RAM memory to store its firmware, hence the firmware is lost after a power cycle.
LE: Logic Element, a programmable function on an FPGA, the number of LEs gives a notion of the size of an FPGA.
CPLD: Complex Programmable Logic Device, Contains flash to store its firmware, hence the firmware is **not** lost.
PLD: Both FPGA and CPLD are **P**rogrammable **L**ogic **D**evices (PLD). In case we need to refer to both, we will use PLD as a common name.
PLDs: are devices that contain logic functions (and, or, nor, etc.).
HDL: These logic functions can be programmed using an Hardware Description Language (HDL), paragraph C.3 will discuss in to more detail what an HDL is.
Manufacturers: Xilinx and Altera are the biggest producers of ordinary PLDs, next to these two there are a few who are designing PLDs for niche markets (like radiation hardened devices for aerospace).
Firmware: is the thing that goes into the PLD, some manufacturers will call it a bitstream (Xilinx).
Configuration device: is a device (microprocessor or dedicated chip) which runs an algorithm to program an FPGA. A CPLD does not need this since it has internal flash in which the program is stored.

C.2 Physical structure

C.2.1 FPGA

FPGA's are build around Logic Blocks (LB) (see figure C.1), these LBs are programmed using the firmware. This firmware configures each LB's functionality. This RAM memory is loaded with the firmware on power on by the use of a configuration device or an external processor. FPGA's mostly contain several additional peripherals like: tranceivers (PCI, Ethernet, etc.), Phase Locked Loops, DDR RAM interfaces, etc.

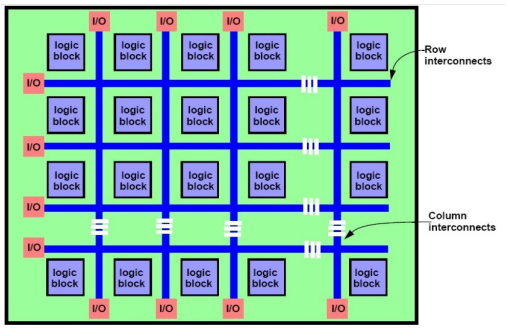


Figure C.1: FPGA Logic Block build up.

C.2.2 CPLD

CPLD's are build up in a similar way as FPGA's however ones loaded with a configuration (in the internal flash) the configuration will not be lost if the power is switched off. Furthermore the logic density on a CPLD is lower then on an FPGA, which simplifies the routing and fitting step in the CPLD design software.

C.3 Programming

As we know there are two main manufacturers Xilinx and Altera, both have free IDEs (Integrated Development environments) which can both handle Verilog and VHDL, the two main hardware description languages, as well as some more obscure or manufacturer dependable languages.

The programming of the devices is being done through JTAG (Joint Test Action Group) this is an industry standard programming and debug interface (basically a huge shift register).

C.4 Usage

In general we can state that all tasks which can be performed by a sequential (micro)processor or microcontroller can also be performed using a PLD. However in the previous paragraph we saw that everything in a PLD runs in parallel. Hence PLDs are mainly used in cases where speed up is needed and the needed speedup can not be performed in a sequential processor. Furthermore there are some processing tasks which require shifting or counting at high speeds (for instance encoder pulse counting or filtering), these tasks put a disproportional amount of work load on a sequential processor. In those cases it is useful to use a PLD, since the internal structure (see paragraph: C.2) alows to perform these counting and shifting tasks at no performance loss for the PLD (keep in mind that the resource loss becomes high if counters and shifting operations are working on wide bit values).

Now the reader would ask the natural question: when to use an FPGA and when to use a CPLD? Here we try to give a simple step by step plan which can easily guide

the user to make a choice between FPGA and CPLD. We should consider de following aspects, these are the most important 3 aspects in choosing between FPGA and CPLD:

1. (Re)configurability
2. HDL design complexity
3. Financially

First we have to wonder if the firmware needs to be changed often. In case it is needed to change the firmware often (when running adaptive algorithms on the PLD) it is most favourable to have an FPGA, since the CPLD's internal flash would suffer from many rewrites.

When the firmware doesn't necessarily needs to be changed often, the HDL complexity comes in to play. For complex designs (basically everything beyond a set of counters or simple filtering) an FPGA will be more beneficial since it will contain more Logic Elements and will be more likely to fit a bigger design, however a configuration device is needed in order to configure the FPGA. Per Logic Element an FPGA (including separate configuration device) becomes cheaper at as little as approximately 2000 Logic Elements.

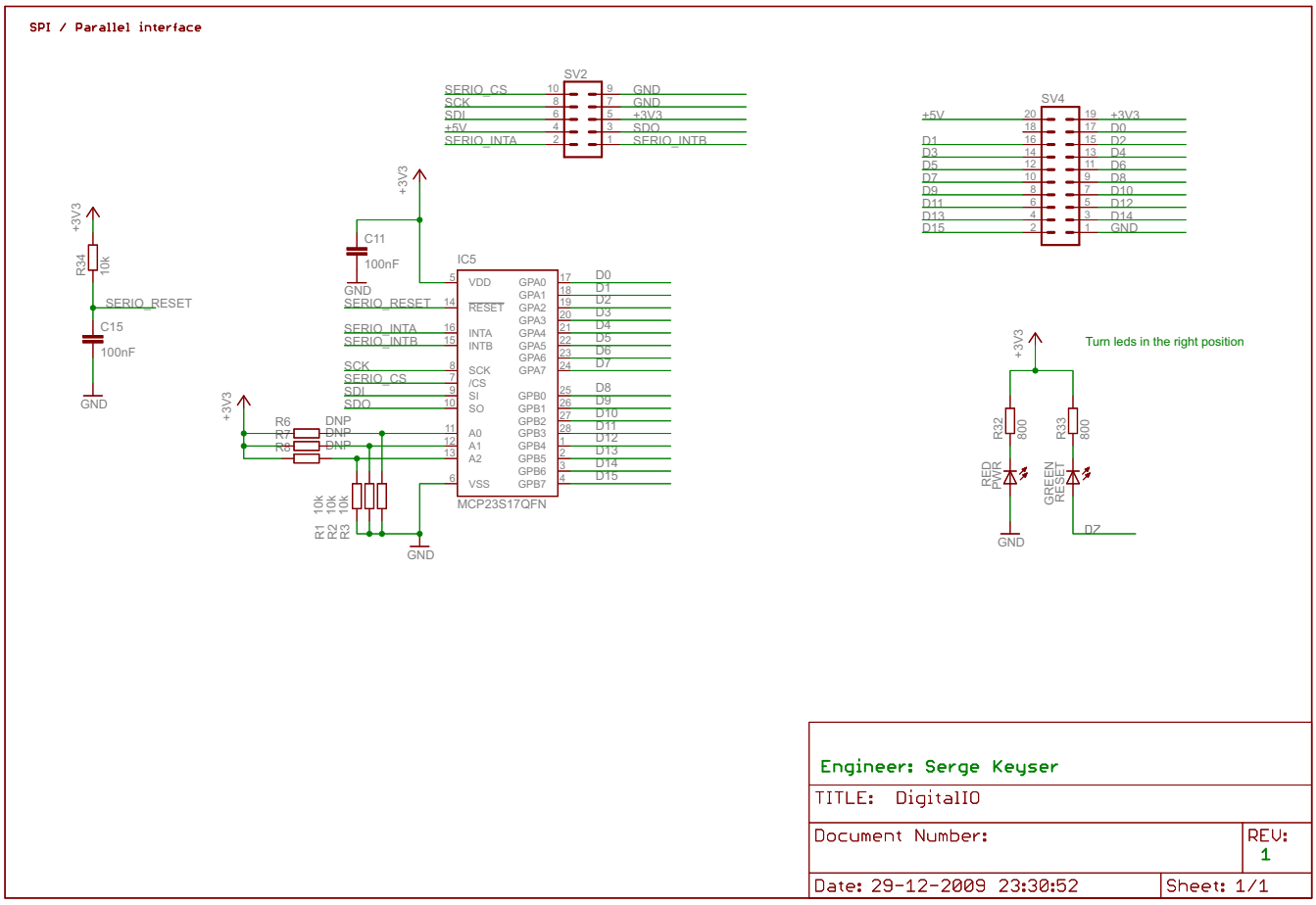
This discussion is a little over simplified, however it gives a starting point to choose an FPGA or CPLD for those who are not familiar to PLDs to make a choice between FPGA or CPLD, in fact the above reasoning is what the author of this thesis always uses in order to start making a choice.

Schematics

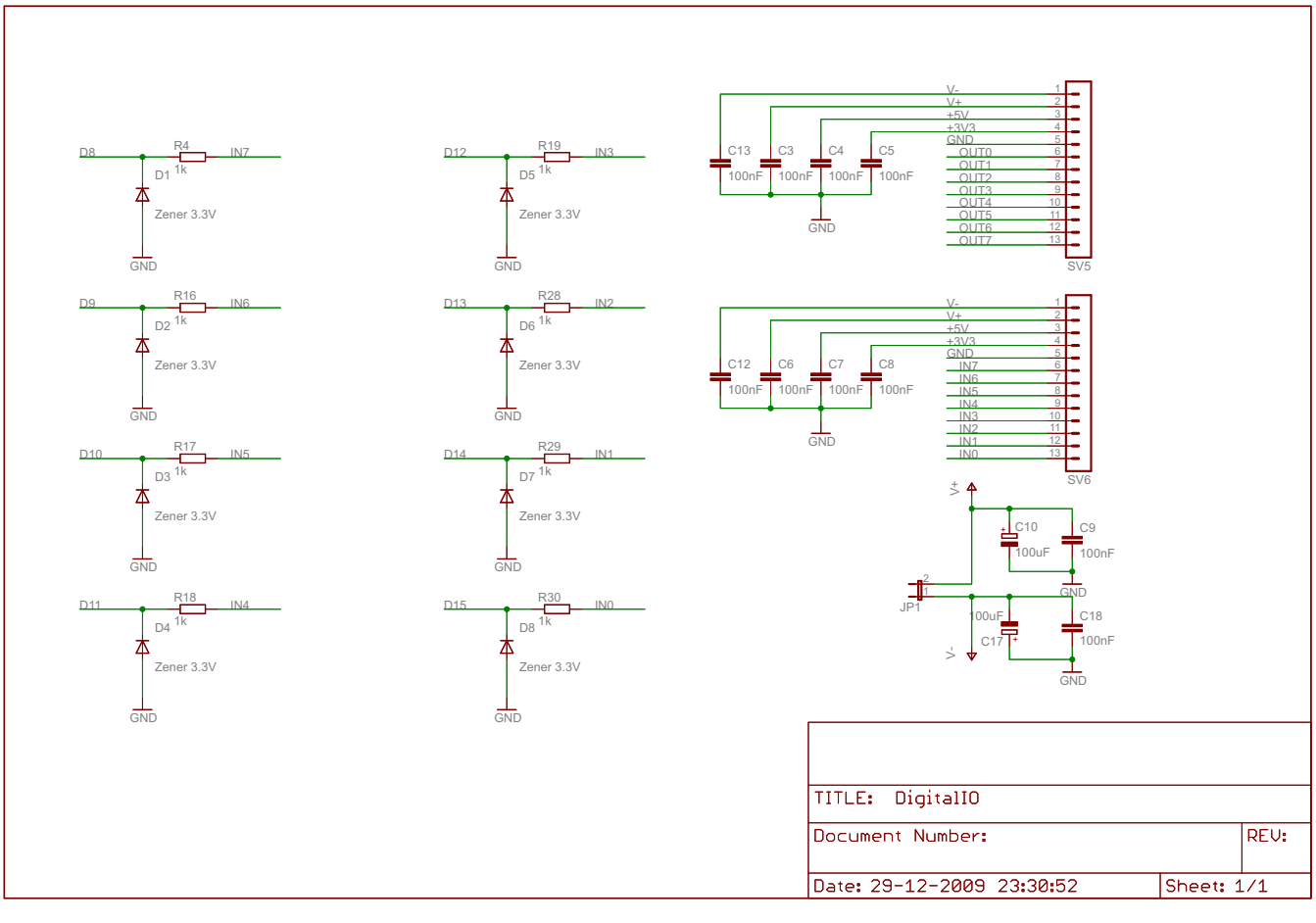
D

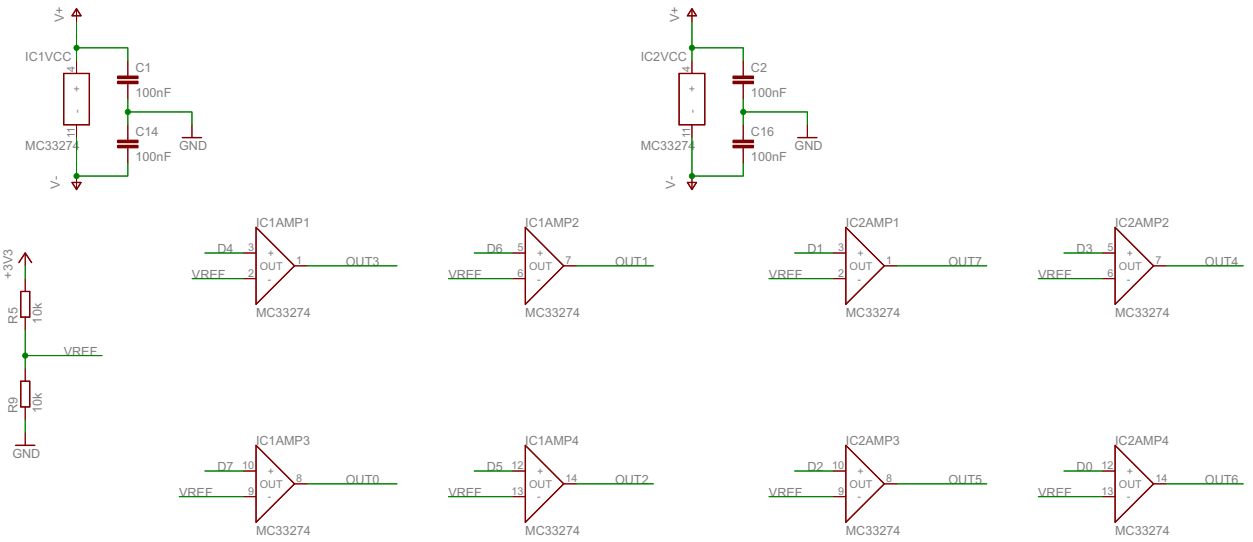
This appendix contains all the schematics of the designed modules.

1. Page 1,2 Digital IO
2. Page 3,4 Encoder Counter
3. Page 5 Motor Controller
4. Page 6 SPI Bridge

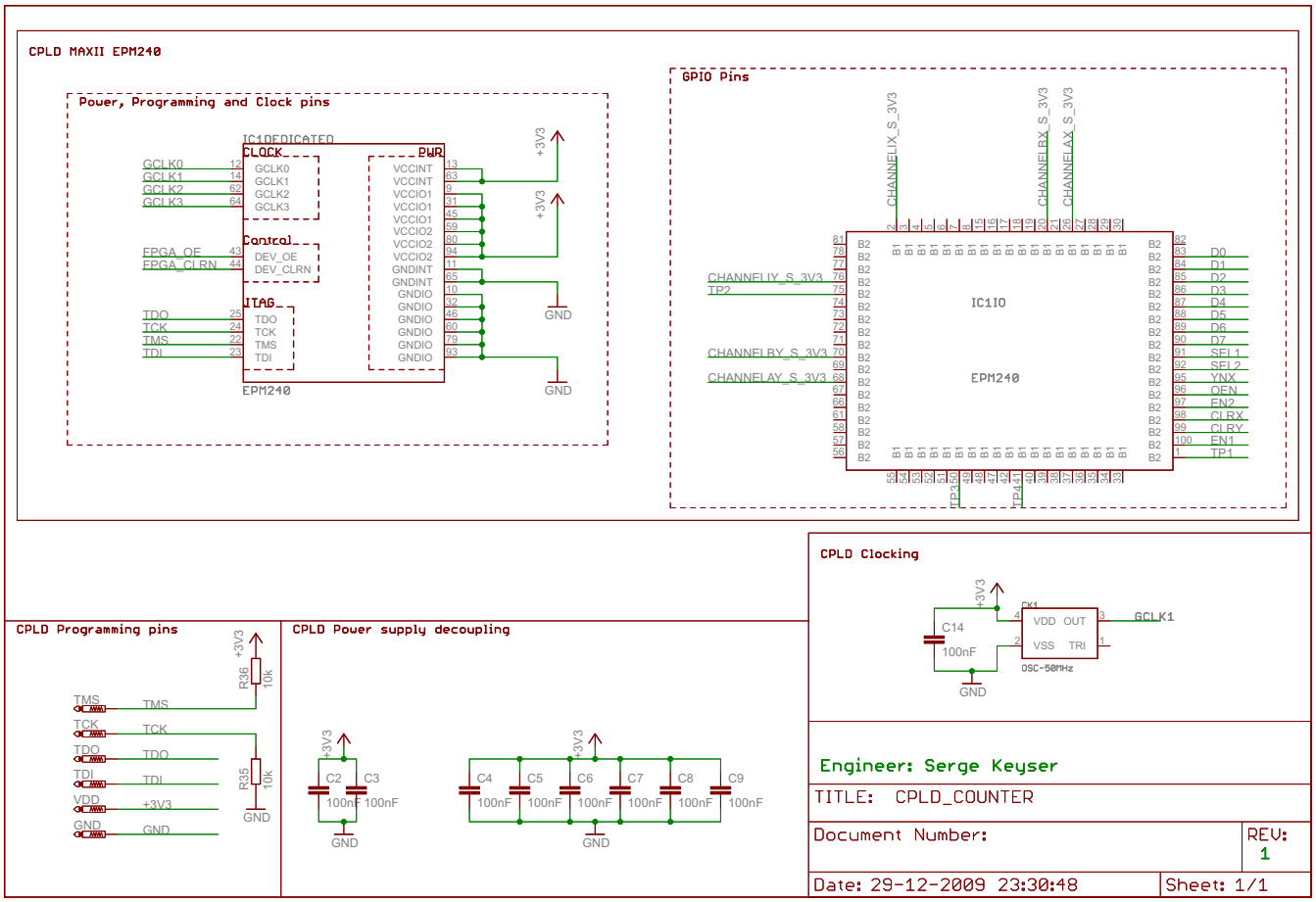


28-1-2011 13:57:26 E:\Thesis\Philips\RTUSB_design\BrushlessDC\DigitalIO\DigitalIO.sch (Sheet: 1/1)

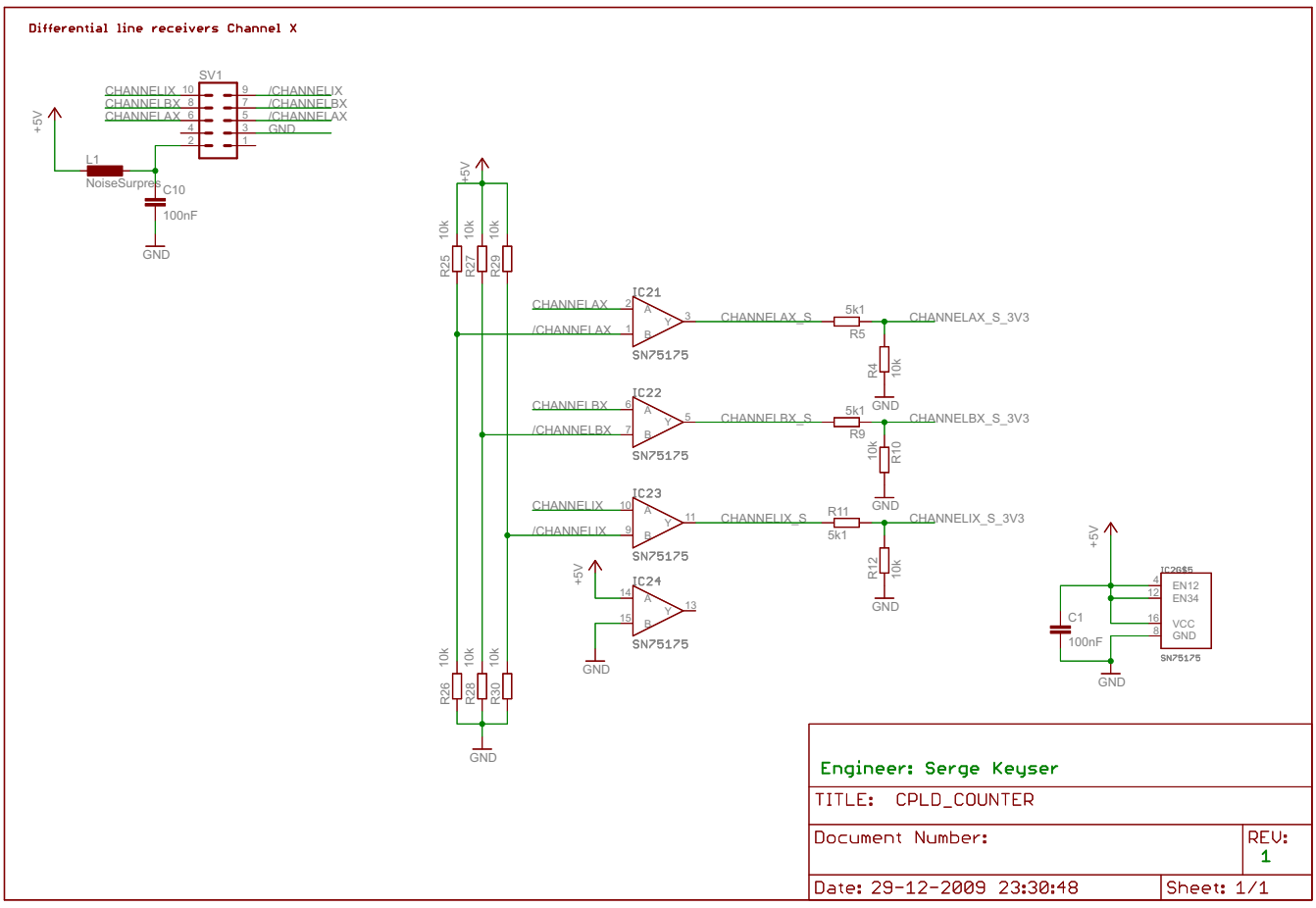


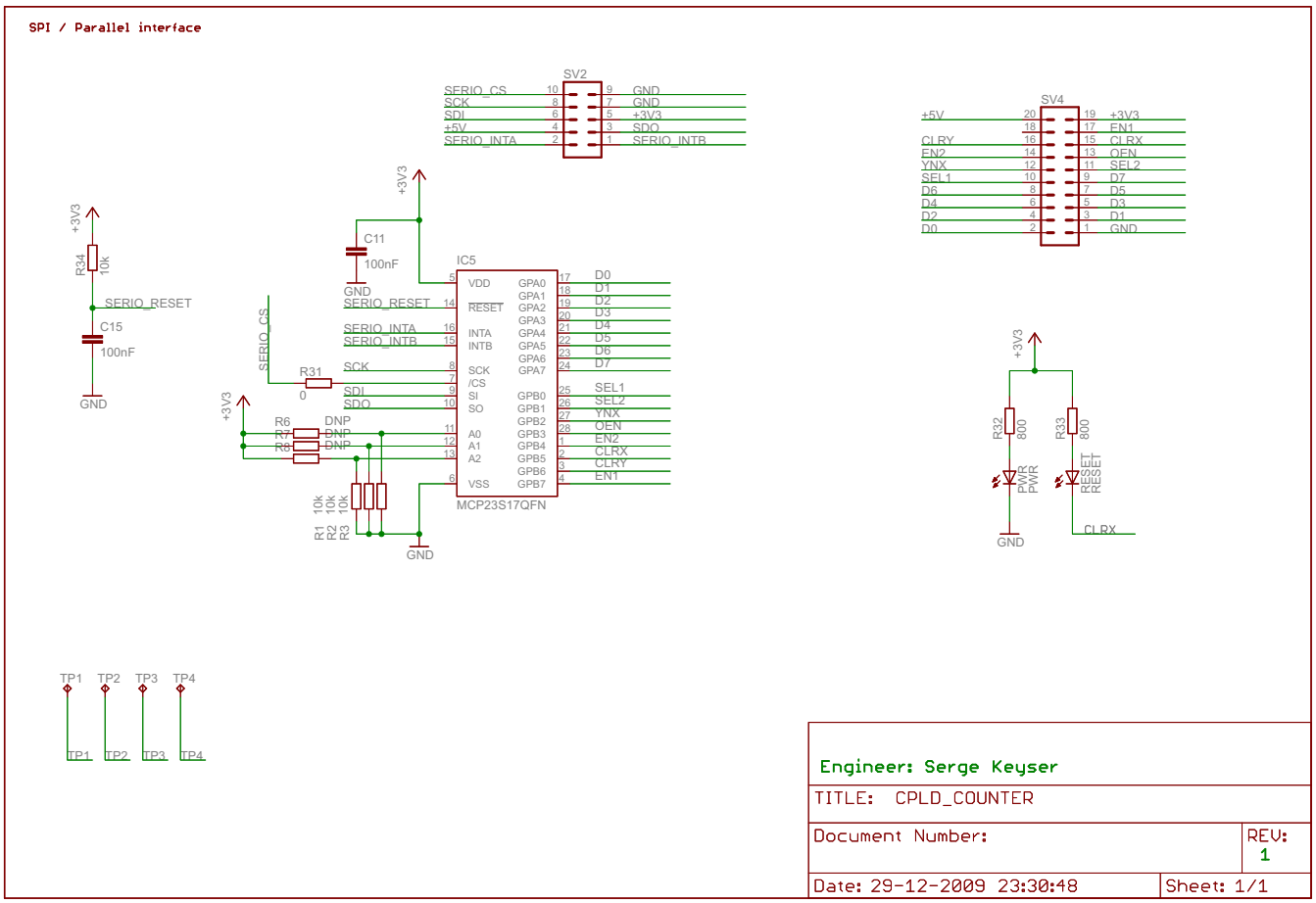


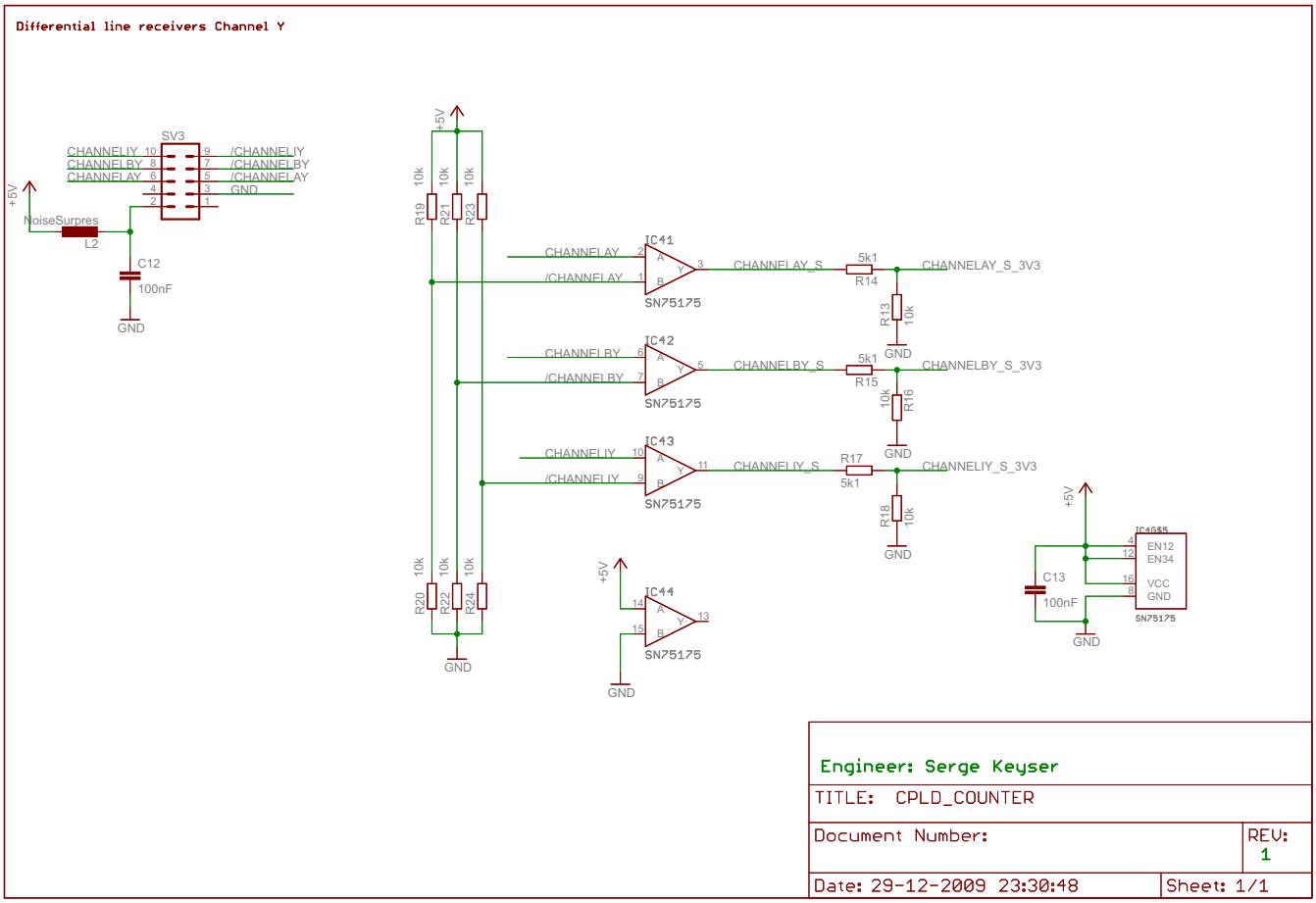
TITLE: DigitalIO	
Document Number:	REV:
Date: 29-12-2009 23:30:52	Sheet: 1/1

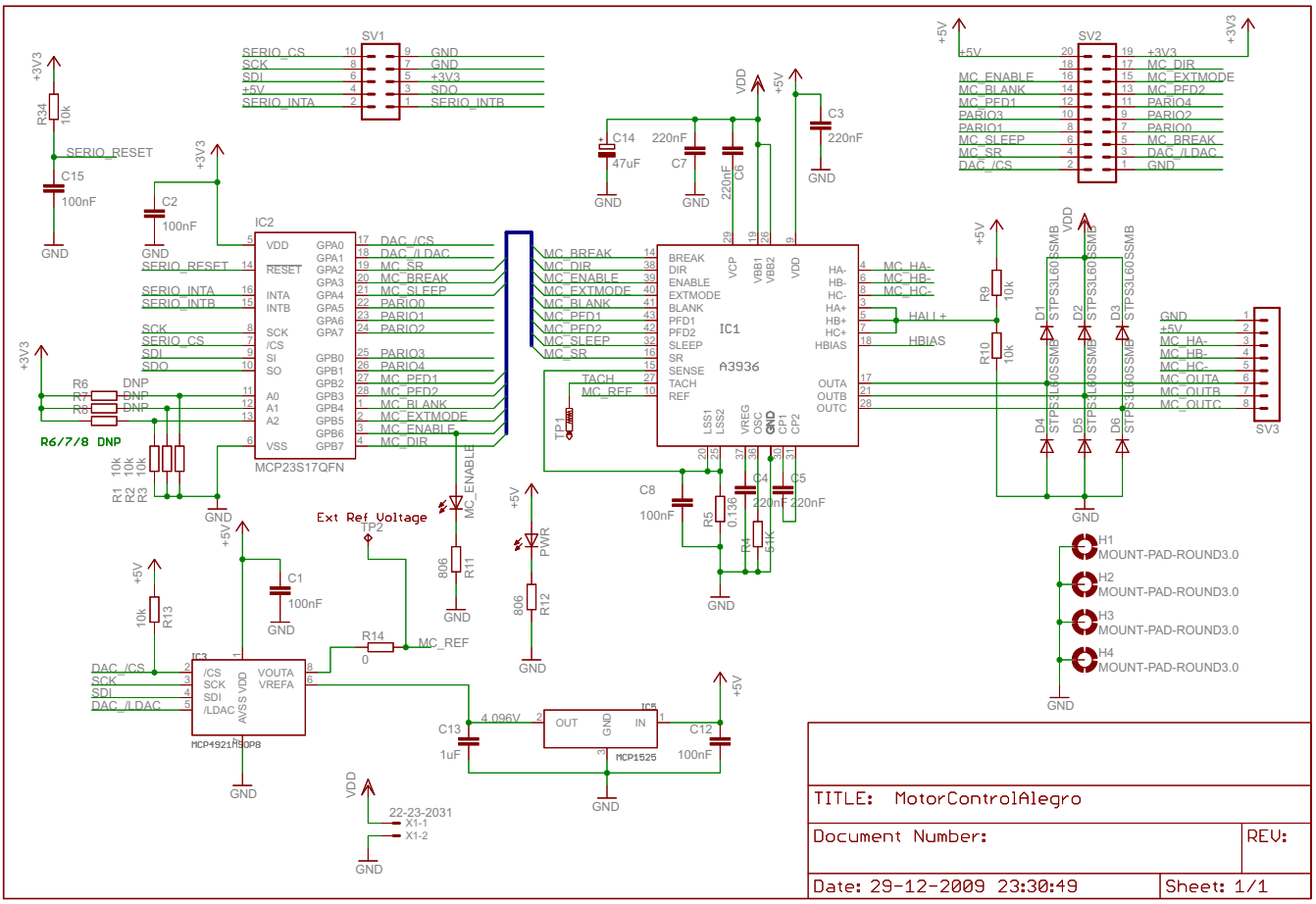


28-1-2011 13:58:58 E:\Thesis\Philips\RTUSB_design\BrushlessDC\EncoderCounter\CPLD_COUNTER.sch (Sheet: 1/1)

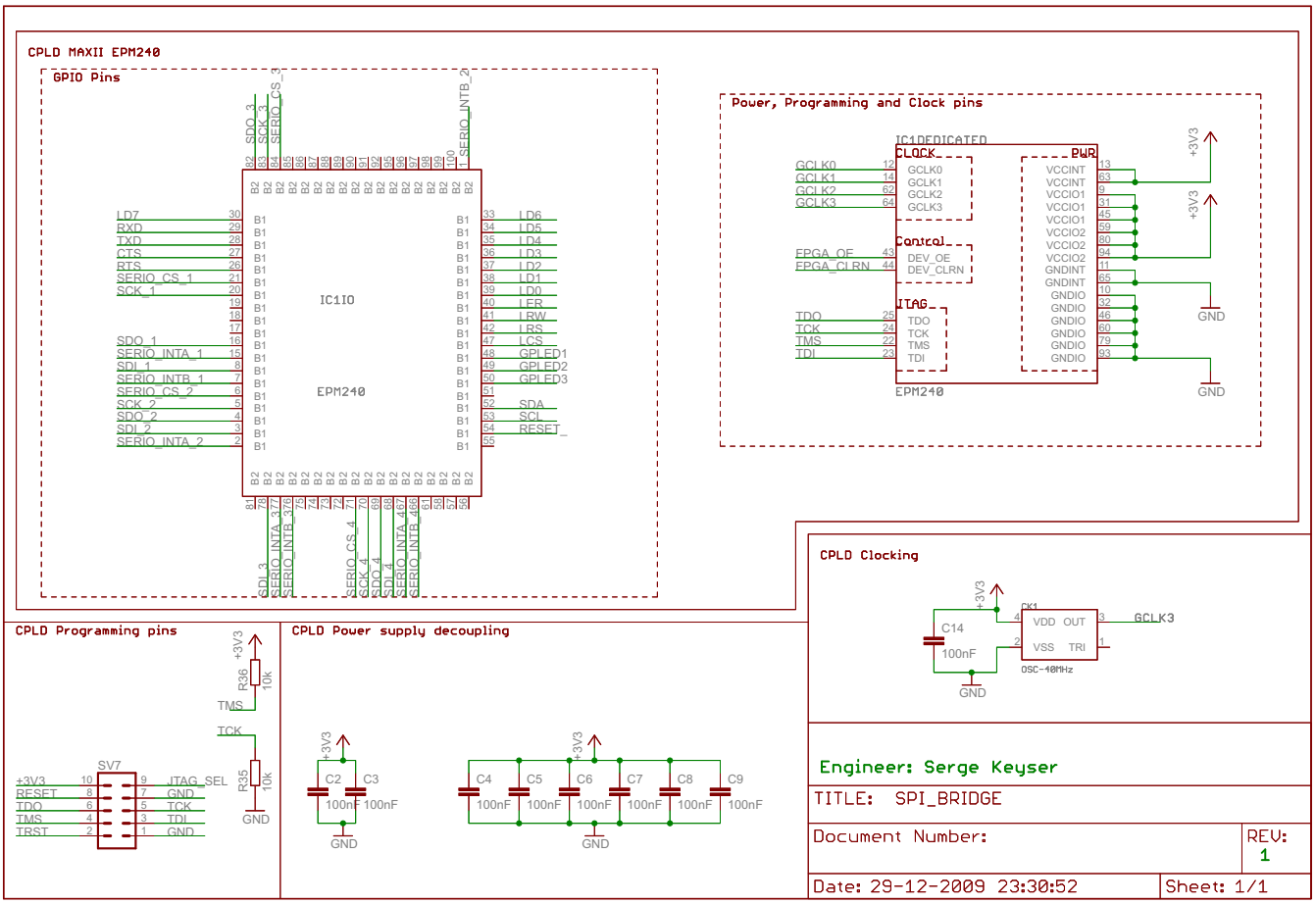



SPI / Parallel interface
SV2
SERIO_CS
SCK
SDI
+5V
SERIO_INTA
GND
GND
+3V3
SDO
SERIO_INTB
SV4
+5V
CLRY
EN2
YNX
SEL1
D6
D4
D2
D0
+3V3
EN1
CLRX
OEN
SEL2
D7
D5
D3
D1
GND
+3V3
R34
10k
SERIO_RESET
C15
100nF
GND
C11
100nF
GND
SERIO_RESET
SERIO_INTA
SERIO_INTB
SERIO_CS
R31
0
SCK
SDI
SDO
+3V3
R6
R7
R8
DNP
DNP
DNP
R1 10k
R2 10k
R3 10k
GND
IC5
VDD
RESET
INTA
INTB
SCK
/CS
SI
SO
A0
A1
A2
VSS
GPA0
GPA1
GPA2
GPA3
GPA4
GPA5
GPA6
GPA7
GPB0
GPB1
GPB2
GPB3
GPB4
GPB5
GPB6
GPB7
D0
D1
D2
D3
D4
D5
D6
D7
SEL1
SEL2
YNX
OEN
EN2
CLRX
CLRY
EN1
MCP23S17QFN
+3V3
R32
800
R33
800
PWR
PWR
RESET
RESET
GND
CLRX
TP1
TP2
TP3
TP4
TP1
TP2
TP3
TP4
Engineer: Serge Keyser
TITLE: CPLD_COUNTER
Document Number:
REV: 1
Date: 29-12-2009 23:30:48
Sheet: 1/1

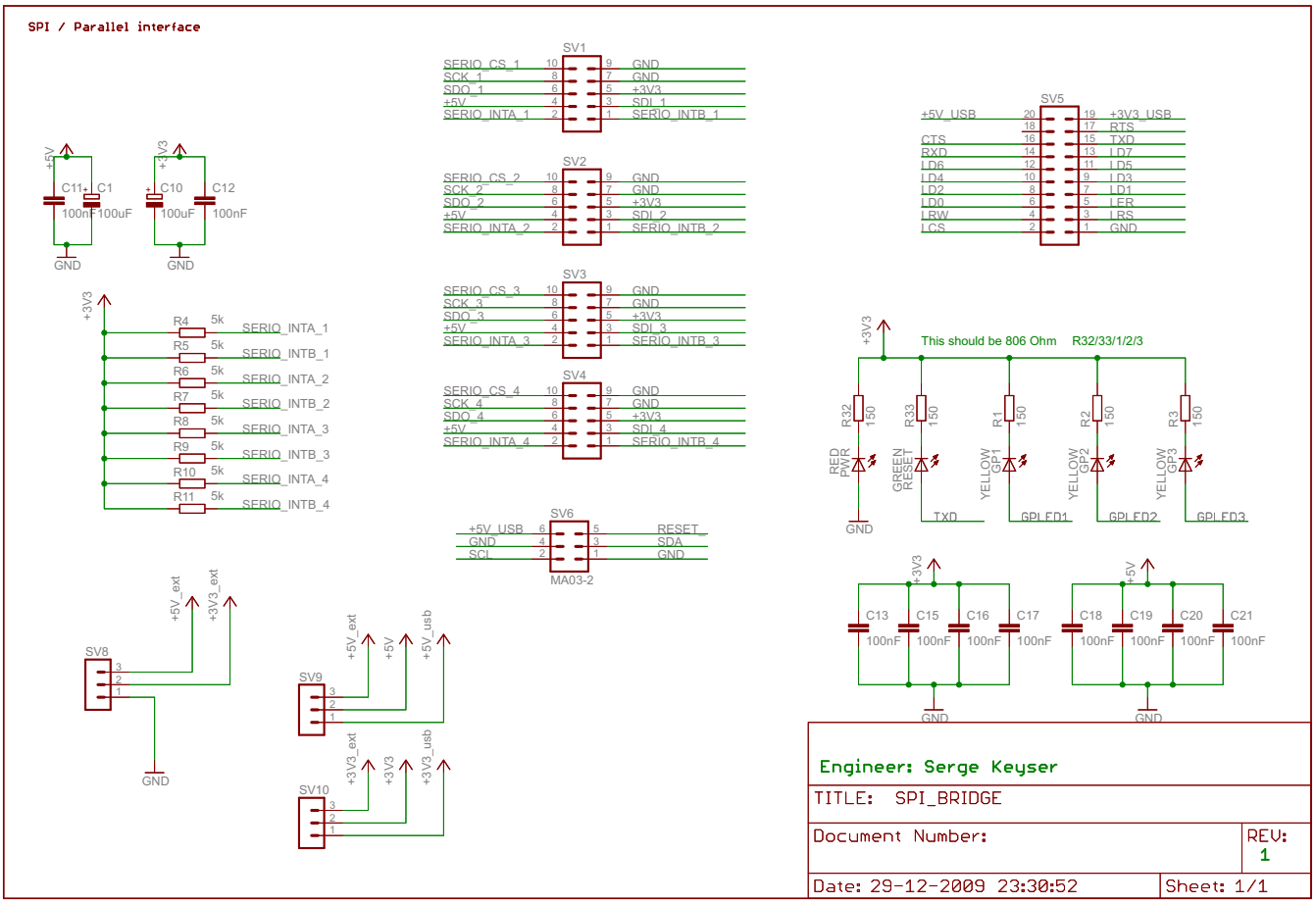

Differential line receivers Channel Y
SV3
CHANNELIY
CHANNELBY
CHANNELAY
/CHANNELIY
/CHANNELBY
/CHANNELAY
GND
+5V
NoiseSurpres
L2
C12
100nF
GND
+5V
R19 10k
R21 10k
R23 10k
IC41
CHANNELAY
/CHANNELAY
SN75175
CHANNELAY_S
5k1
R14
CHANNELAY_S_3V3
R13 10k
GND
IC42
CHANNELBY
/CHANNELBY
SN75175
CHANNELBY_S
5k1
R15
CHANNELBY_S_3V3
10k
R16
GND
IC43
CHANNELIY
/CHANNELIY
SN75175
CHANNELIY_S
R17
5k1
CHANNELIY_S_3V3
R18 10k
GND
R20 10k
R22 10k
R24 10k
GND
+5V
IC44
SN75175
GND
+5V
IC4655
EN12
EN34
VCC
GND
SN75175
C13
100nF
GND
Engineer: Serge Keyser
TITLE: CPLD_COUNTER
Document Number:
REV: 1
Date: 29-12-2009 23:30:48
Sheet: 1/1



28-1-2011 14:00:28 E:\Thesis\Philips\RTUSB_design\BrushlessDC\IntegratedMCAlegro\MotorControlAlegro.sch (Sheet: 1/1)



28-1-2011 14:00:59 E:\Thesis\Philips\RTUSB_design\BrushlessDC\SPI_BRIDGE\SPI_BRIDGE.sch (Sheet: 1/1)



Bibliography

[1] NICEe 4000, *http://www.boschrexroth.com.*

[2] Altera, *Avalon Interface Specifications*, (2008).

[3] Arcus, *http://www.arcus-technology.com/.*

[4] D. Montesinos-Miracle S. Galceran-Arellano O.Gomis-Bellmunt A.Sudria-Andreu, *A new low-cost motion control educational equipment*, 2007 European Conference on Power Electronics and Applications, Sept 2007, pp. 1–6.

[5] R.Hofmann B.Drerup, *Next Generation Coreconnect Processor*, 15th Annual IEEE Intl ASIC/SOC Conf., September 2002, pp. 221–225.

[6] D. Bertozzi L. Benini, *Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip*, IEEE circuits and systems magazine, May 2004, pp. 18–31.

[7] Elmo Motion Control, *http://www.elmomc.com/.*

[8] ———, *http://www.elmomc.com/products/elmo-motion-control-simplIQ-tech.htm.*

[9] Galil Motion Control, *http://www.galilmc.com/.*

[10] dSpace, *http://www.dspace.com.*

[11] E-blocks, *http://www.matrixmultimedia.com.*

[12] Eriks, *http://mechatronica.eu.*

[13] Wiki Pedia / Hash Functions, *http://en.wikipedia.org/wiki/Hash_function*, (2009).

[14] D. Carrica M.A. Funes S.A. Gonzalez, *Novel Stepper Motor Controller Based on FPGA Hardware Implementation*, Transactions on Mechatronics Vol 8 Number 1, March 2003, pp. 120–124.

[15] Y. Kung G.Shu, *Development of a FPGA-based Motion Control IC for Robot Arm*, ICIT 2005. IEEE International Conference on Industrial Technology, December 2005, pp. 1397–1402.

[16] National Instruments, *Creating Custom Motion Control and Drive Electronics with an FPGA-based COTS System*, (2006).

[17] D. Kim, *An implementation of fuzzy logic controller on the reconfigurable FPGA system*, IEEE Trans. Ind. Electron, June 2000, pp. 703–715.

[18] Thor Labs, *http://www.thorlabs.com/.*

[19] A. Ehliar D. Liu, *An FPGA Based Open Source Network-on-Chip Architecture*, International Conference on Field Programmable Logic and Applications, Augustus 2007, pp. 800–803.

[20] C. Lin C. Hung C. Liu, *Position Sensorless Control for Four-Switch Three-Phase Brushless DC Motor Drives*, Proceeding of the 2006 IEEE Industry Applications Conference Forty-First IAS Annual Meeting (IAS'06), October 2006, pp. 2049–2053.

[21] P. Martin, *Design of a Virtual Component Neutral Network-on-Chip Transaction Layer*, Proceedings of the conference on Design, Automation and Test in Europe - Volume 1, March 2005, pp. 336–337.

[22] X. Shao D. Sun J.K. Mills, *A New Motion Control Hardware Architecture with FPGA-Based IC Design for Robotic Manipulators*, Proceedings of the 2006 IEEE International Conference on Robotics and Automation, May 2006, pp. 3520–3525.

[23] A.Kapoor N.Simaan P.Kazanzides, *A System for Speed and Torque Control of DC Motors with Application to Small Snake Robots*, IEEE/APS Mechatronics and Robotics, 2004.

[24] K. Goossens J.Dielissen Andrei Radulescu, *Aethereal Network on Chip: Concepts, Architectures,and Implementations*, IEEE Design and Test of Computers, September 2005, pp. 414–421.

[25] E. Schemm, *SERCOS to link with Ethernet for its third generation*, (2004).

[26] Silicore, *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, (2002).

[27] Circuit Cellar/ simple motion control system, *http://www.circuitcellar.com/flash2002/First/abstractM250.htm*, (2002).

[28] Philips Applied Technologies, *http://apptech.philips.com*, (2007-2008).

[29] Lab View, *http://www.ni.com/labview/*.

[30] Y.F. Chan M. Moallem W. Wang, *Efficient Implementation of PID Control Algorithm using FPGA Technology*, 43rd IEEE Conference on Decision and Control, December 2004, pp. 4885–4890.

[31] FB1122 FPGA with altera cyclone FPGA, *EtherCAT piggyback controller boards*, (2008).

[32] T. Ya Z. Runjing H. Xiaoxia, *Application of FPGA in Direct Current Motor Servo System*, Proceedings of the 27th Chinese Control Conference, July 2008, pp. 261–265.

[33] G. Young, *Motion Control and Mixed-Signal FPGAs*, (2007).