Design and Development of a Base Station Emulator

Viktor Kallas

April, 2006
Abstract

Pricer AB produce and sell the Pricer Electronic Shelf Label system (ESL). The ESL system automates the work of changing prices, traditionally managed by hand in stores. Price changes can be carried out both on the product labels and in the cash registers simultaneously, which guarantees price integrity for the store. Pricer’s ESL system consists of equipment for wireless communication (infrastructure), electronic shelf labels (ESLs) and the necessary software for their use.

The infrastructure consists of transceivers and base stations constituting the communication link between the software running the ESL system and the labels in the store. Information flows from the software system to the base station, to the transceivers and finally to the labels and backwards in the same manner from the labels to the software system.

Every component in the communication link between the software system and the labels is an error source and together they make a difficult system to debug. If a communication error is detected in either end there are several possibilities where the error could have occurred.

To improve the debugging environment this master thesis has developed a Base Station Emulator, enabling direct communication with a transceiver without the use of an ordinary base station. Instead of the server software, the Base Station Emulator is controlled from TRXLogger, which is a software communication tool supporting generation of transceiver formatted data.

The Base Station Emulator also supports Pulse Position Modulated (PPM) data generation with two different protocols (Pricer Protocol 4 and Pricer Protocol 16) used for IR communication with Pricer’s ESLs.
Abstract

Pricer AB producerar och säljer ett elektroniskt etikettsystem (Electronic Shelf Label (ESL) system). ESL-systemet automatiserar de prisändringar som vanligtvis utförs för hand i affärer. Prisändringar kan göras både på de elektroniska etikettarna och i kassorna samtidigt, vilket garanterar prisintegritet för affären. Pricers ESL-system består av utrustning för trädlös kommunikation (även kallad infrastruktur), elektroniska etiketter och mjukvara för att använda systemet.

Infrastrukturen består av tranceivers och basstationer. Dessa utgör kommunikationslänken mellan mjukvaran och de elektroniska etikettarna ute i affären. Data flödar från mjukvarusystemet till en basstation, vidare till tranceivrar och slutligen i form av IR-ljus till etikettarna. På samma sätt fast omvänt flödar data tillbaka från etikettarna till mjukvarusystemet.

Varje komponent i denna kommunikationslänk utgör en potentiell felkälla, vilket försvårar felsökning och testning. Om ett fel upptäcks i mjukvarusystemet eller i etikettarna så kan felet ha uppstått var som helst i kommunikationslänken.

För att förbättra testmiljön har en basstationsemulator utvecklats i detta examensarbete. Basstationsemulatorn möjliggör direkt kommunikation med tranceivern utan användande av den vanliga basstationen. Istället för systemets mjukvara styrs basstationsemulatorn med TRXLogger som är en kommunikationsmjukvara som kan generera tranceiverformaterat data.

Acknowledgements

First of all I would like to thank my supervisor Rickard Ericson for entrusting me with this thesis and for always taking time to my many questions.

I would also like to thank the head of the R&D department, Kenneth Johansson, for approving the thesis, which must have been a somewhat difficult decision after having worked at Pricer AB for only about 2 weeks. I also want to thank Lars Andersson for guidance and support and my namesake Viktor Kouznetsov for the help with the interface of the base station.

Finally I also want to express my gratitude to my examiner Ingo Sander for the pregnant hours of encouraging, hints and help.

The report was done with fantastic LaTeX.
3.1.3 How the modified BSE should operate 27
3.1.4 Typical usage of modified BSE 27
3.2 Development of modified BSE 27
3.2.1 Structure of modified BSE 28
3.2.2 New components 28
3.2.3 Synthesis of modified BSE 30
3.2.4 Critical parts in the design 30

4 Analysis 31
4.1 Validation and Evaluation 31
4.1.1 Validation 31
4.1.2 Evaluation 34
4.2 Technology analysis 35
4.2.1 FPGA portability 35
4.2.2 System Board portability 36

5 Conclusions 37
5.1 Summary 37
5.2 Conclusions 38
5.3 Future work 38

A Appendix 41
A.1 Base Station Emulator User Guide 41
A.1.1 Programming the board 41
A.1.2 Reset the Base Station Emulator 41
A.1.3 Sending ordinary frames 41
A.1.4 Sending IR-frames 42
A.1.5 Programming the board with new constants 42
A.2 Source code 43
A.3 Pin usage and baud rate constants tables 45
List of Figures

1.1 The Pricer System. Printed with permission from Pricer AB........ 2
1.2 Base Station. Printed with permission from Pricer AB........... 3
1.3 Transceiver. Printed with permission from Pricer AB............ 4
1.4 Electronic Shelf Label. Printed with permission from Pricer AB... 5
1.5 Virtex II System Board with P160 Communications Module plugged in. ................................................................. 9
1.6 P160 Prototype Module. Picture from Avnet Electronics [2]........ 9
1.7 Virtex II FPGA Family Members. Figure from Virtex II Data Sheet [10]. .......................................................... 10
1.8 Virtex II Architecture Overview. Figure from Virtex II Data Sheet [10]. .......................................................... 10
1.9 Virtex II Slice Configuration. Figure from Virtex II Data Sheet [10]. .......................................................... 11
2.1 BSE – interfaces to environment ............................................. 13
2.2 BSE – outline ..................................................................... 14
2.3 BSE – Physical Layout .......................................................... 14
2.4 P160 Prototype Module with add on board ......................... 15
2.5 Add on board schematics ..................................................... 15
2.6 Connecting schematics of Voltage Adder ............................ 16
2.7 Voltage Adder .................................................................. 16
2.8 Data path in FPGA ............................................................ 18
2.9 Incremental development – flowchart ................................. 19
2.10 top_comp ................................................................. 22
3.1 PPM signal. Picture taken from Internet [7] ....................... 26
3.2 Modified BSE ............................................................ 29
3.3 rx_tx_block .............................................................. 29
3.4 rx_tx_ppm_block .......................................................... 30
4.1 Oscilloscope capture of PP4 ............................................. 33
4.2 Oscilloscope capture of PP16 ............................................ 34
4.3 Spartan II FPGA Family Members. Figure from Xilinx homepage. 35
A.1 Printed screen .......................................................... 43
A.2 Printed screen .......................................................... 44
A.3 Printed screen .......................................................... 45
List of Tables

1.1 SRAM configured as an *and* LUT .................................................. 8

2.1 Typical frames sent to TRX .............................................................. 19
2.2 Device utilization summary of BSE on Xilinx Virtex II. Extract
from XST synthesis report ................................................................. 22

3.1 Timestamps with PP4 ................................................................. 26
3.2 Timestamps with PP16 ................................................................. 27
3.3 PPM modes .................................................................................. 28
3.4 Device utilization summary of the modified BSE on Xilinx Virtex
II. Extract from XST synthesis report ............................................. 30

4.1 User case frames ................................................................. 32
4.2 Frames sent and responses .......................................................... 32
4.3 Different TRX baud rates ......................................................... 33
4.4 Evaluation of robustness ............................................................. 34
4.5 Device utilization summary of the modified BSE on Xilinx Spar- 
tan II. Extract from XST synthesis report ........................................ 35

A.1 Pin usage table ............................................................................. 46
A.2 Baud rate constants table with 100MHz oscillator .................. 46
List of Abbreviations

ASIC  Application Specific Integrated Circuit
bps  bits per second
BSE  Base Station Emulator
BSI  Base Station Interface
DSUB  D-subminiature
ESL  Electronic Shelf Label
FEL  Front End LAN
FIFO  First In First Out
FPGA  Field Programmable Gate Array
FSM  Finite State Machine
IP  Intellectual Property
IR  Infrared light
LED  Light Emitting Diode
PPM  Pulse Position Modulation
PSU  Power Supply Unit
RAM  Random Access Memory
RJ45  Registered Jack 45
TRX  Transceiver
TRXD  Transceiver Driver or base station
UART  Universal Asynchronous Receiver Transmitter
VHDL  Very High Speed Integrated Circuit Hardware Description Language
Chapter 1

Introduction

1.1 Overview of Pricer’s Electronic Shelf Label System

Pricer’s ESL system consists of a data server, infrastructure (base stations and transceivers) and electronic shelf labels (ESLs). The ESL system is integrated into the store’s back office system, which is the data system where the store keeps information about stock, price, orders, etc. In the event of a price change, information is sent from the store’s back office system to the checkouts. At the same time the information is sent to the server of the ESL system, from which it is forwarded via a base station to ceiling mounted transceivers where the signal is converted into IR (infrared) light and sent out to the ESLs (see Figure 1.1).

As soon as the signal is received, the price shown on the display of the label is changed, after which an acknowledgement signal is sent back to the stores backoffice system via the transceiver and the Pricer server. All the transactions that are carried out are stored to allow verification or further processing.

Other types of information (discounts, etc.) can also be shown on the label. Some information may also be hidden (stock status, etc.) and only shown when activated with an IR-key (see Figure 1.1).

1.1.1 The data server

The Windows-based server software coordinates the different parts in the ESL system. The server software is connected with a UART (Universal Asynchronous Receiver Transmitter) serial RS232 link to one or more base stations (TRXD) at baud rate 19200 bps. Via this connection the server software communicates with the TRXDs according to Pricer’s TRXD communication interface description [5].

Typical data sent by the server may be requests asking the TRXD to send back some information, or instructions to send or request some data to/from the transceivers (TRX) or the labels.

1.1.2 Infrastructure

The infrastructure consists of base stations (TRXD) and transceivers (TRX).
Figure 1.1: The Pricer System. Printed with permission from Pricer AB.
The TRXD (see Figure 1.2) is connected to the server software and to 1-32 TRXs. The TRX connection is a UART serial RS485 connection at 62500 bps. The TRXD also supplies all 1-32 TRXs with 48V and is therefore also connected to a power supply unit (PSU). The TRXD is responsible both for communication with the TRXs and for communication with the labels.

The TRX (see Figure 1.3) is a ceiling mounted device containing optical components to receive and transmit IR communication to/from the labels. The TRX is also connected to a TRXD. Every TRX has its own identity (ID). A hardware ID is programmed in non-volatile memory during production and a volatile address is set by the TRXD when powered. The ID and address make it possible to communicate with a specific TRX if necessary.

Data between the TRXD and the TRXs is formatted according to Pricer’s *Front End LAN (FEL) communication interface description* [4]. Ordinary data is sent in frames\(^1\). When sending data to the labels, however, the TRXD sends IR-frames which are channeled through the TRXs IR-LEDs to the labels.

### 1.1.3 Electronic Shelf Labels

The Pricer label (ESL) (see Figure 1.4 contains an ASIC controlling the display and the communication and optical components for transmitting and receiving IR-frames.

During production, the ESL is programmed with a unique address and a protocol format identifier. The protocol format identifier is used by the ESL to check that it only accepts frames transmitted with the right format. The address is required for all commands that change any data in the ESL, or request status about the ESL.

Normally, the ESL has the IR receiver turned off, but the IR receiver is activated every second. If the ESL receives a valid frame during this time, it will continue receiving until the transmission stops.

\(^1\)sequence of bytes where different parts of the sequence have different content, e.g. command, address, etc.
The IR-frames are sent with a pulse position protocol (PP4) described in Pricer’s \textit{PP4 protocol specification} \cite{6}. The IR pulses consist of a high frequency IR burst. Several bursts are then combined into frames.

The ESL will reply to updates and status inquiries with an acknowledge signal. For updates the acknowledge signal indicates a successful reception and execution by the label. For inquiries the signal indicates a successful reception and a positive answer to the inquiry i.e a yes.

1.2 Objective of thesis

1.2.1 Base Station Emulator

As explained in Section 1.1 the data path from server software to labels is rather complex. This complicates test and debug work with potential error sources in software, infrastructure and labels. In test and debug work one would ideally only work with one of these parts at the time. Today that is possible with the labels and the server software. A well defined test environment can receive and transmit user defined IR-frames to/from the labels. A software emulator of the base station is used as a test environment for the server software. But for the base station and the transceivers there exist no such test environment.

This thesis is therefore aimed to develop a test and debug unit for one part of the infrastructure, the transceiver. The test and debug unit shall have the interface of the base station and be able to communicate with the transceiver according to the \textit{Front End LAN (FEL) communication description} \cite{4} (from now on the test and debug unit will be referred to as the \textit{Base Station Emulator}).

The server software communication in the ordinary base station will in the Base Station Emulator be replaced by communication with a software module called TRXLogger. TRXLogger supports low level communication with the
transceiver according to the *FEL communication description* [4]. The Base Station Emulator shall accordingly be designed to receive and transmit frames from/to TRXLogger and forward these frames from/to the transceiver.

Since communication with the labels is rather complex, this feature will be left as an optional objective.

Summing up, the Base Station Emulator will enable direct communication between TRXLogger and the transceiver which significantly will improve the test environment for the transceiver.

### 1.2.2 Optional: PPM generation – PP4 and PP16

Of course it would be valuable if the Base Station Emulator (BSE) also could communicate with the labels. This would require the BSE to be able to interpret the incoming frames from TRXLogger and, if addressed to the labels, Pulse Position Modulate (PPM) these frames to IR-frames.

Today Pricer use pulse position protocol PP4 (Pricer Protocol 4) for IR communication, but in an ongoing project future labels are being modified to also handle PP16. If the BSE could also generate PP16 this would of course be valuable when testing out the new labels.

One difficulty when communicating with the labels is the fact that the IR receiver is only activated once every second. This means that IR-frames have to be sent continuously for at least one second to "wake up" the labels. This is called *padding*, and has to be supported by the BSE for communication with the labels.
1.3 Specification of thesis

1.3.1 Base Station Emulator – specification

The aim of this thesis is to design and develop a Base Station Emulator matching the essential functionality of the base station. The BSE shall support the following specifications:

1. Serial UART differential RS232 connection to PC at 19200 bps

2. Base Station interface to TRX (serial UART differential RS485 at 62500 bps, 48V, etc.)

3. Capability to handle frames, i.e. keeping frames together when converting baud rate

4. Configurable baud rates (if baud rates are changed in the future)

5. Full duplex, i.e. ability to send and receive at the same time

6. As technology independent as possible

7. Optional: Capability to Pulse Position Modulate data with protocol PP4

8. Optional: PPM for protocol PP16

9. Optional: Capability to pad IR-frames

1.3.2 Validation and Evaluation

To verify that the above specifications are met, the Base Station Emulator should be validated and evaluated. Validation should verify that the given specifications are met. This should be done both with typical user cases and with constructed test cases.

An evaluation should describe the quality of the BSE. The evaluation should include for instance an analysis of sensitivity and range of input parameters.

1.3.3 Technology Analysis

The BSE will be developed in a development environment given by Pricer. But according to specification 7 the BSE should be as technology independent as possible. Thus the portability should also be investigated. There should also be an investigation to find a suitable technology to fit the BSE on.

1.4 Development Environment

1.4.1 Development Environment – specification

The following hardware and software tools were available at Pricer.
Hardware

- Xilinx Virtex II MicroBlaze Kit
- 48V PSU
- ADM3485E (RS485 Integrated Circuit)

Software

- Xilinx Integrated Software Environment (ISE) 6.2i containing Xilinx Synthesis Tool (XST), etc.
- Xilinx Embedded Development Kit (EDK) 6.2, for development with embedded processor MicroBlaze
- ModelSim XE III 6.0a VHDL simulation tool
- Emacs 22.0.50.1 as text editor

1.4.2 Short introduction to VHDL and FPGA

VHDL

VHDL (Very High Speed Integrated Circuit Hardware Description Language) is a hardware description language. Hardware may be described on different levels of abstraction, from gates and flip-flops to state machines. All VHDL can be simulated with simulation tools but only a subset may be synthesized, e.g. translated to register transfer level on a specific technology such as ASIC, FPGA, PLD and CPLD. The synthesizable subset is also technology dependent, e.g. Xilinx technology does not support exactly the same subset as Altera technology.

FPGA

One technology to synthesize described hardware to is the Field Programmable Gate Array (FPGA). The FPGA is a silicon device with programmable logic. How logic may be programmed will be introduced with an example of a simple FPGA cell: the Look Up Table (LUT).

Consider an SRAM memory with 16 bits. To address these bits you need 4 address signals since $2^4 = 16$. Imagine now that you connect 4 arbitrary signals (A, B, C, D) instead of the address signals. Then you have a function generator with A, B, C and D as inputs and the memory bits as output. Consider now setting up the SRAM memory, f(A,B,C,D), as in Table 1.1.

We have then programmed a function generator or Look Up Table (LUT) of an and function with A,B,C and D as inputs. Thus, the same functionality as an and gate but programmed in a 16 bit SRAM memory.

The LUT example is a demonstration of how logic is programmed. As seen in Section 1.4.4 FPGAs also contain ordinary hardware such as flip-flops, multiplier blocks, muxes, tri-state buffers, RAM, etc.
### Table 1.1: SRAM configured as an and LUT

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>( f(A,B,C,D) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1.4.3 Xilinx Virtex-II MicroBlaze Kit

The Xilinx Virtex II MicroBlaze Kit consists of a MicroBlaze hardware platform and an Embedded Development Kit (EDK).

The MicroBlaze hardware platform consists of three boards: a system board and two expansion boards. The Virtex II System Board is the main board containing the Virtex II FPGA (XC2V1000-4FG456C), DDR memory, two oscillators (24 and 100MHz), RS232 port, PROM, 2.5V and 3.3V supply and many other features. The other two boards are expansion modules that can be plugged into the system board. The P160 Communications Module contains a lot of I/O devices, for instance an RS232 port. The P160 Prototype Module contains prototype area and I/O pins for user devices, which then easily can be integrated with the system board. However, none of these boards contain any RS485 ports.

The EDK includes the MicroBlaze soft processor core (32-bit Harvard Bus RISC architecture), standard peripherals (UARTs, memory interfaces, interrupt controller, etc.) and development tools.

1.4.4 Xilinx Virtex-II FPGA (XC2V1000-4FG456C)

**Virtex II Overview**

Virtex II MicroBlaze Kit includes a one million system gate Virtex II FPGA, XC2V1000-4FG456C (see Figure 1.7. An overview of the FPGA architecture can be seen in Figure 1.8. The major elements in the Virtex II FPGA are:

- I/O Blocks
- Configurable Logic Blocks
- Block SelectRAM memory
Figure 1.5: Virtex II System Board with P160 Communications Module plugged in.

Figure 1.6: P160 Prototype Module. Picture from Avnet Electronics [2].
Table 1: Virtex-II Field-Programmable Gate Array Family Members

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>Array Row x Col.</th>
<th>Slices</th>
<th>Maximum Distributed RAM Kbits</th>
<th>Multiplier Blocks</th>
<th>SelectRAM Blocks</th>
<th>Max RAM (Kbits)</th>
<th>DCMs</th>
<th>Max I/O Pads</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC2V40</td>
<td>40K</td>
<td>8 x 8</td>
<td>256</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>72</td>
<td>4</td>
<td>98</td>
</tr>
<tr>
<td>XC2V60</td>
<td>60K</td>
<td>10 x 8</td>
<td>512</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>144</td>
<td>4</td>
<td>120</td>
</tr>
<tr>
<td>XC2V250</td>
<td>250K</td>
<td>24 x 18</td>
<td>1,536</td>
<td>48</td>
<td>24</td>
<td>24</td>
<td>432</td>
<td>8</td>
<td>200</td>
</tr>
<tr>
<td>XC2V500</td>
<td>500K</td>
<td>32 x 24</td>
<td>3,072</td>
<td>96</td>
<td>32</td>
<td>32</td>
<td>576</td>
<td>8</td>
<td>264</td>
</tr>
<tr>
<td>XC2V1000</td>
<td>1M</td>
<td>40 x 32</td>
<td>6,120</td>
<td>160</td>
<td>40</td>
<td>40</td>
<td>720</td>
<td>8</td>
<td>432</td>
</tr>
<tr>
<td>XC2V1500</td>
<td>1.5M</td>
<td>48 x 48</td>
<td>7,680</td>
<td>240</td>
<td>48</td>
<td>48</td>
<td>894</td>
<td>8</td>
<td>528</td>
</tr>
<tr>
<td>XC2V2000</td>
<td>2M</td>
<td>56 x 48</td>
<td>10,752</td>
<td>336</td>
<td>56</td>
<td>56</td>
<td>1,068</td>
<td>8</td>
<td>624</td>
</tr>
<tr>
<td>XC2V3000</td>
<td>3M</td>
<td>64 x 56</td>
<td>14,336</td>
<td>448</td>
<td>96</td>
<td>96</td>
<td>1,728</td>
<td>12</td>
<td>720</td>
</tr>
<tr>
<td>XC2V4000</td>
<td>4M</td>
<td>80 x 72</td>
<td>23,040</td>
<td>720</td>
<td>120</td>
<td>120</td>
<td>2,160</td>
<td>12</td>
<td>912</td>
</tr>
<tr>
<td>XC2V6000</td>
<td>6M</td>
<td>96 x 88</td>
<td>33,792</td>
<td>1,056</td>
<td>144</td>
<td>144</td>
<td>2,592</td>
<td>12</td>
<td>1,104</td>
</tr>
<tr>
<td>XC2V8000</td>
<td>8M</td>
<td>112 x 104</td>
<td>46,592</td>
<td>1,456</td>
<td>168</td>
<td>168</td>
<td>3,024</td>
<td>12</td>
<td>1,108</td>
</tr>
</tbody>
</table>

Figure 1.7: Virtex II FPGA Family Members. Figure from Virtex II Data Sheet [10].

Figure 1.8: Virtex II Architecture Overview. Figure from Virtex II Data Sheet [10].
- Multiplier blocks
- Digital Clock Managers

There are also programmable routing resources interconnecting all of these elements. All programmable elements, including routing resources, are controlled by values stored in static memory cells. These values are loaded in the memory cells during configuration and can be reloaded to change the functions of the programmable elements.

I/O Blocks
Programmable I/O blocks (IOBs) provide the interface between I/O pins to the FPGA and the internal configurable logic. They can be programmed as input blocks, output blocks with optional tri-state buffer or bidirectional blocks. The IOBs also contain registers configurable as flip-flops or latches.

Configurable Logic Blocks

![Figure 1.9: Virtex II Slice Configuration. Figure from Virtex II Data Sheet [10].](image)

Configurable Logic Blocks (CLBs) are elements for combinatorial and synchronous logic. A CLB include four slices and two tri-state buffers. Each slice is equivalent and contains (see Figure 1.9):

- Two function generators (F and G)
- Two registers
- Large multiplexers
- Other hardware

Note here that it is only the function generators that are configurable logic. The multiplexors, registers and other hardware consist of "ordinary" hardware.
The function generators F and G are configurable as 4-input look-up tables (LUTs) (see Section 1.4.2), as 16-bit shift registers, or as 16-bit distributed SelectRAM memory. The two registers are either edge-triggered D-type flip-flops or level-sensitive latches. Each CLB connects to a switch matrix to access general routing resources.

**Block SelectRAM**

Since it would be a waste to use Configurable Logic Blocks as ordinary RAM memory there are also block RAM resources on the Virtex II FPGA. The block SelectRAM memory resources are 18 Kb of dual-port RAM programmable from 16K x 1 bit to 512 x 36 bits, in various depth and width configurations.

**Multiplier blocks**

There are multiplier blocks available for fast multiplication logic. The multiplier blocks consist of 18-bit x 18-bit dedicated multipliers.

**Digital Clock Manager**

Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for clock distribution. The DCM features delay compensation, clock multiplication and division for generating oscillators with other frequencies and coarse- and fine-grained clock phase shifting to use if there are timing problems due to routing delays.

The clock division feature can divide incoming clock frequencies with 1.5, 2, 2.5, and so forth up to 15 and 16 or with a fraction $\frac{M}{D}$ where $M$ and $D$ are integers from 1–32.
Chapter 2

Design and development of the Base Station Emulator

2.1 Structure of the Base Station Emulator

Figure 2.1: BSE – interfaces to environment

Figure 2.1 is an outline of the whole system with its interfaces. The BSE is outlined in Figure 2.2. The first task was to design the physical layout of the BSE.

Since the Virtex II System Board contains a D-subminiature (DSUB) with RS232 connection the interface to the PC was taken care of. However, the board did not contain any RS485 circuits why the Base Station Interface had to be mounted on the P160 Prototype Module.

Since 48V would be a danger for the Virtex II System Board if a short circuit would occur, the power supply was decided to be physically separated from the board. Instead a new device, the Voltage Adder, was designed with the single functionality to add voltage to the TRX. The resulting physical layout of the BSE can be seen in Figure 2.3.
The following sections will deal with design and development of the different parts of the BSE: the Base Station Interface and the Virtex II FPGA.

### 2.2 The Base Station Interface

#### 2.2.1 P160 Prototype Module

As the Virtex II System Board provides 3.3V, a matching RS485 device was the ADM3485E from Analog Devices [1]. Two RS485 circuits and a RJ45 connector was placed on an add on board together with pull up and pull down resistors to match the interface of the real base station. The add on board was mounted on top of the Xilinx P160 Prototype Module (see Figure 2.4). The connecting scheme can be seen in Figure 2.5.

#### 2.2.2 Voltage Adder

The Voltage Adder forwards communication signals and 48V to the TRX. 48V was supplied from a Power Supply Unit (PSU). The connectors were of RJ45 type (see schematics in Figure 2.6). To protect users from the 48V connections...
Figure 2.4: P160 Prototype Module with add on board

Figure 2.5: Add on board schematics
the construction was embedded in a metal box. This also made it easy to use. The device can be seen in Figure 2.7.

![Connecting schematics of Voltage Adder](image1)

Figure 2.6: Connecting schematics of Voltage Adder

![Voltage Adder](image2)

Figure 2.7: Voltage Adder

2.3 Design of the Virtex II FPGA

The Virtex II System Board contains the Virtex II FPGA where the logic of the Base Station Emulator will be designed and developed.

2.3.1 Design decisions

Embedded processor or pure hardware?

When considering the viable alternatives to design the BSE there were two main alternatives to choose between: Either an embedded processor (Xilinx MicroBlaze) with peripheral hardware or a pure hardware solution.
Analyzing the embedded processor solution resulted in the following benefits and drawbacks:

+ **Flexible solution** Easy to change application software if need to modify behavior.

+ **Easy to implement** Many peripheral hardware components available as IP cores (FIFOs, UARTs, etc) and easily added in Xilinx EDK software.

− **Expensive technology** If a soft core processor is used, this means that the BSE can only be instansiated in a chip large enough to contain a processor and peripheral hardware. For instance MicroBlaze require 900-2600 LUTs depending on configuration (see Xilinx homepage [9]). Together with peripheral hardware this will require a rather large and expensive chip.

− **Technology dependent** With MicroBlaze and IP (Intellectual Property) cores one would be heavily dependent on Xilinx technology. If need to change technology to Altera, for instance, the hardware configuration with processor and peripherals would have to be redone, although the application code might still be working.

Analyzing the pure hardware solution resulted in the following benefits and drawbacks:

+ **Technology independent** If only open source hardware is used the design will be as near technology independent as possible (of course RAM instantiations are always technology dependent). If need to change to another technology the same VHDL code can be used.

+ **Small design** With no processor on the chip the design will become smaller which means possibility to implement on cheaper technology.

+ **Control over design** If only open source is used there is full control over hardware. If need to change in the UART protocols or other details not featured by the IP cores, this is an advantage.

− **More work** More work to develop and modify hardware.

As seen in the specification (Section 1.3.1) there is no actual speed or power demands in the design that motivates the use of hardware. UARTs with baud rates 19200 bps and 62500 bps respectively is rather slow communication which could well be managed with a processor with peripheral hardware. However, as the analysis above indicated there is the advantage of technology independency and less use of logic that speaks for the hardware solution. Since this was a decisive design decision the two alternatives were carefully considered. Eventually the pure hardware solution was considered to match the BSE specifications best and therefore this alternative was chosen.

**Design constraints**

Since it was hard to estimate time consumption of design and development, the initial BSE was prioritized to only meet the main objectives. If time left the optional objectives would also be designed and developed.

The following Section will therefore only consider design and development of the main objectives.
2.3.2 Virtex II FPGA design – how the BSE should operate

To meet all specifications of the BSE the design in Figure 2.10 was worked out. Data flows according to the arrows from PC to TRX and vice versa, which can be done in parallel or with full duplex.

The PC sends at 19200 bps taking $520 \mu s$ to send one byte (start bit + 8 bits + stop bit), while the UART to the TRX sends at 62500 bps taking $160 \mu s$ to send one byte. That means that if the PC is sending a frame, bytes are sent one by one with $520 \mu s$ interval, while the UART to the TRX will send one byte and then wait for $520 - 160 = 360 \mu s$ until next byte is received from the PC.

To avoid these gaps the bytes are temporarily stored in a FIFO (First In First Out) until the frame detector signals that the whole frame is received. Then the whole frame can be sent without gaps to the TRX. Two FIFOs instead of one was chosen to enable sending of a new frame while the old one is still being sent out to the TRX.

The UARTs are clocked with frequency dividers, dividing the frequency of the Virtex II System Board oscillator to the matching baud rate. These frequency dividers will be configurable to generate baud rates up to theoretically the board oscillator (division by 1), meeting the specification of configurable baud rates.

In order to keep the BSE as technology independent as possible all parts of the BSE design will be open source.

2.3.3 Typical usage of the BSE

The PC software TRXLogger sends a frame of length up to 50 bytes. The FPGA buffers the frame and sends it to the TRX. If there is a response from the TRX (depending on which frame is sent) it gets buffered in the FPGA and is sent back to the PC. The responding frame from the TRX may contain parameter values, error codes, etc. Typical frames sent to the TRX can be seen in Table 2.1.
<table>
<thead>
<tr>
<th>Frame</th>
<th>Command</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set address</td>
<td>TRXCMD.SETPAR</td>
<td>5 bytes</td>
<td>Set a new value to a TRX parameter address</td>
</tr>
<tr>
<td>Get hardware ID</td>
<td>TRXCMD.GETPAR</td>
<td>5 bytes</td>
<td>Get TRX parameter value hardware ID</td>
</tr>
<tr>
<td>TRX go</td>
<td>TRXCMD.GO</td>
<td>5 bytes</td>
<td>Sets TRX in application mode</td>
</tr>
</tbody>
</table>

Table 2.1: Typical frames sent to TRX

2.4 Development of the Virtex II FPGA

2.4.1 Incremental development

In order to keep down design complexity, an incremental development process, seen in Figure 2.9 was applied throughout the thesis. This means that a primitive model of the final system was designed and developed first. Then, when the primitive construction behaved as expected in simulation and synthesis, functionality was added to the design and started another iteration of the process. This was repeated until the final system behaved as specified.

The first primitive model contained a single UART echoing its input. Then another UART was added giving a two ended system. When the two UARTs behaved as expected the frame detector and FIFOs were developed in parallel, giving the final system.

Figure 2.9: Incremental development – flowchart
2.4.2 Description of Components

Since all components had to be open source there were two alternatives: Find open source components in books or on the web or develop the components yourself. In the end a mixture of the two choices was applied.

The following components were used:

my_pkg
This VHDL package contains type conversion functions.

async_uartrx and async_uartxmt
The UART is standard component which should not require any deeper description. Many open source UARTs were available. The one used here was taken from Cohen [3]. Cohen’s UART is divided in a receiving component, uartrx, and a transmitting component, uartxmt. The async prefix was added since the synchronous reset was modified to asynchronous reset.

Cohen’s UART is simple. 1 start bit, 8 data bits, 1 stop bit and no parity bits. It signals data ready when a byte is received in uartrx and transmit ready when uartxmt is ready to transmit data.

The uartrx component is clocked with a frequency of 16 times the baud rate (to be able to detect start bits). The uartxmt component is clocked with the baud rate frequency. The baud rate is altered by changing the frequency of the clock signals.

The clock signals were generated in the component freq_divider.

freq_divider
The Digital Clock Manager (see Section 1.4.4) was not able to produce clock frequencies as slow as the baud rates used in the UARTs. Therefore a frequency divider (from Sjöholm [8]) was developed generating the baud rates to the two UARTs in the design.

The frequency divider contains a counter trigged by the incoming clock. When a specified number of clocks cycles have been counted the outgoing clock is toggled.

As incoming clock the 100MHz oscillator on the Virtex II System Board was used. To produce the frequency 19200Hz the incoming clock was counted \( \frac{1}{2} \cdot \frac{100 \times 10^6}{19200} = 2604.165 \) clock cycles before the the 19200Hz clock was toggled. Since the number counted in the frequency divider has to be an integer, this number was rounded to 2604 giving the baud rate \( \frac{100 \times 10^6}{2604} = 19201 \) bps and an error of 64 ppm.

frame_detector
The frame_detector detects the end of a frame. This is done by listening to the incoming serial data, the rx signal. If, when a byte has been received, the rx signal is high the following bit length this indicates that there is no new byte coming in and that an end of frame has been detected. The frame detector raises a frame ready signal after detection.
fifo

The fifo component is the FIFO component seen in Figure 2.8. Initially the FIFO was developed using flip-flops as memory, but since block RAM (described in Section 1.4.4) is more efficient with respect to logic utilisation, this was developed later on using the subcomponent dual_port_ram described next.

One could of course argue that using flip-flops instead of block RAM makes the BSE more portable. Other silicon devices such as PLDs and CPLDs seldom contain block RAM resources. Also block RAM must be instantiated using the target technology, i.e. the FIFO design has to be modified to match the target technology.

However, the loss of portability was considered small comparing to the gain in logic utilization, why the block RAM version remained.

Since frames can be up to 50 bytes long the FIFO depth was set to 64 bytes. The FIFO contains two counters keeping track of the front and back address of the FIFO and signals for full and empty.

dual_port_ram

The dual_port_ram component is a template from Xilinx XST User Guide [11]. The template ensures that XST will understand that the component is a dual port RAM with two clocks and synthesize it as such. The two clocks are needed because the PC end will access the FIFO with one clock and the TRX end with another clock.

top_comp

The top_comp component in Figure 2.10 fuses together all of the above components with additional logic to make them work together.

A FIFO select signal determines which FIFO to be read from and written to. If low, FIFO1 is being written to and FIFO2 is being read from. When a frame ready pulse is detected the FIFO select signal is toggled so that FIFO2 is being written to and FIFO1 is being read from.

The shift load signal gives a pulse when the FIFO to be read from is not empty and the transmitting UART is ready to transmit. The pulse, connected to FIFO read signal, reads a byte from the FIFO and loads this byte into the transmitting UART.

The write signal to the FIFO is connected to the data ready signal from the receiving UART. When a byte is received the data ready signal goes high as well as the write signal to the current FIFO.

A mux is controlled from the FIFO select signal and makes sure that the right FIFO output is selected to the transmitting UART.

The data path and logic is exactly the same for the PC to TRX direction as for the TRX to PC direction.

2.4.3 Problems during development

- Initially an open source UART from Xilinx was used in development. However, this UART was not very well documented and resulted in unexplainable bugs after synthesis. After several months of trying to improve this UART Cohen’s UART was found and used instead.
Clock division caused trouble with double bytes sent. Initially the frequency divider generated the baud rate clock by dividing an already divided clock with 16. But since there is a small error on the divided clock, due to discretization, the resulting slower baud rate was even more erroneous resulting in bugs such as double bytes sent. This problem was solved by letting each clock output have their own count, making the clocks more exact.

2.4.4 Synthesizing the BSE

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices:</td>
<td>195 out of 5120</td>
<td>3%</td>
</tr>
<tr>
<td>Number of Slice Flip Flops:</td>
<td>178 out of 10240</td>
<td>1%</td>
</tr>
<tr>
<td>Number of 4 input LUTs:</td>
<td>342 out of 10240</td>
<td>3%</td>
</tr>
<tr>
<td>Number of bonded IOBs:</td>
<td>5 out of 324</td>
<td>1%</td>
</tr>
<tr>
<td>Number of BRAM:</td>
<td>4 out of 40</td>
<td>10%</td>
</tr>
<tr>
<td>Number of GCLKs:</td>
<td>1 out of 16</td>
<td>6%</td>
</tr>
</tbody>
</table>

Table 2.2: Device utilization summary of BSE on Xilinx Virtex II. Extract from XST synthesis report.

Looking at the device utilization summary in Table 2.2 it is clear that the Virtex II FPGA is dimensioned for much larger constructions than this one. Of course using an embedded processor would have used much more logic. MicroBlaze uses 900-2600 LUTs depending on configuration. The pure hardware solution is as expected rather small with 342 LUTs. Four block RAMs dimensioned as 64x8 bits are used with a total usage of 2048 bits of block RAM (total on FPGA is 720 Kbits). Only 5 I/O pins are used as expected, rx and tx to
TRX and PC and the reset signal. Only one clock is used since the baud rate clocks are generated in the frequency dividers.

2.4.5 Critical parts in the design

The current BSE is designed to handle single frames. If a sequence of frames is sent with too short space in between, the BSE will not behave properly. What will happen is the following:

Assume that a frame is sent to the BSE from the PC. The BSE receives the frame and starts to send it to the TRX. If the PC starts to send a new frame while the BSE is sending the old frame there are two possibilities. If the sending of the old frame finishes before the new frame is fully received there is no problem. The BSE will start to send out the new frame. But if the new frame is fully received before the old frame has been sent out the BSE will swap anyway and start to send out the new frame and leave the rest of the old frame unsent. The unsent bytes will instead be added to the next incoming frame.

This is a lack of robustness since the BSE will not be protected from erroneous usage. However, it is not specified how the BSE should handle sequences of frames. Assume that the critical behaviour described above is solved. Then the BSE will always be able to send out two frames in sequence properly. If three frames are sent in sequence, the third frame will not be sent at all since the two FIFOs are already busy. The current design is therefore limited to send a maximum of two frames in a sequence.

The specification should therefore be extended to also include desired behaviour of the BSE when sequences of frames are sent. The design should then be modified according to the specified behaviour.

It is also critical to generate a clocks with combinatorics, as with the baud rates. The delay could cause trouble with timing. However this is not an issue in the present BSE where baud rates are slow.
Chapter 3

Design and development of optional objective

When the main objective was implemented and verified the remaining development time was spent on a modification of the main BSE to also support the optional objectives.

The new BSE was initially designed to support all optional objectives, i.e., Pulse Position Modulated data generation with both protocol PP4 and PP16 and capability to pad IR-frames. However, there was only time enough to develop the first two of these.

The optional objectives did only imply a change in the functionality of the FPGA, which means that the Base Station Interface and the Voltage Adder were left unchanged.

3.1 Modified BSE design

3.1.1 Transmission of IR-frames

The transmission of IR-frames differs from ordinary communication with the TRX. An IR-frame contains an IR command and a modulated pulse train. The IR command is sent to the TRX disabling its UART and putting the modulated pulse train through to the IR-LEDs. If the pulse train remains idle until a timeout occurs the TRX goes back to ordinary UART mode.

The modulation of data to pulse trains is done in the base station. The server software sends a frame to be forwarded to the labels. The base station modulates this frame according to Pricer protocol 4 (PP4) and forwards it to the TRX where it is sent out on the IR-LEDs to the labels.

As described in Section 1.1.3 the label’s IR-receiver is activated once every second. To remain activated the label needs to receive an IR-frame during this activation time. This is why IR-frames need to be looped. Otherwise there is little chance of sending while the label is activated. The looping capability is called padding. When an IR-frame is padded it is being sent over and over again in order to wake up the label.
3.1.2 PPM generation

PPM (Pulse Position Modulation) is a serial data transmission protocol. The PPM signal is composed of a reference pulse followed, some clock cycles later, by another pulse (see Figure 3.1). The time between the two pulses gives the value sent by the emitter.

![PPM signal diagram](image)

The advantage of the PPM signal is that the transmission is very reliable: due to the form of the signal an error can be detected very easily. The signal is ideal for use in noisy environments. The disadvantage is that the signal use a huge amount of bandwidth.

PP4

The pulse position modulation protocol used by Pricer is called PP4 (Pricer Protocol 4). When transmitting data, bits are sent in pairs, where each pair is symbolized by a specific time interval called timestamp (see Table 3.1). If a timestamp is received that is not in Table 3.1 an error has occurred.

The transmission time of a frame is dependent on both how many bytes it consists of, as well as the actual content of the bytes. If many bit pairs with large timestamps are used the frame will be longer than if bit pairs with short timestamps are sent.

<table>
<thead>
<tr>
<th>Bit-pair</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>$A\mu s$</td>
</tr>
<tr>
<td>01</td>
<td>$B\mu s$</td>
</tr>
<tr>
<td>10</td>
<td>$C\mu s$</td>
</tr>
<tr>
<td>11</td>
<td>$D\mu s$</td>
</tr>
</tbody>
</table>

Table 3.1: Timestamps with PP4

PP16

In a new revision of the ASIC chip in the label, there will be support for a new variant of the PP4 protocol. The new variant will support bits being sent in groups of four instead of in pairs, thus PP16. This means that there will be a timestamps for each combination of 4 bits as in Table 3.2. In all other respects the protocol will be the same. The idea is to speed up transmission time which in PPM protocols is rather slow.
3.1.3 How the modified BSE should operate

The communication path from TRX to PC should operate as in the initial BSE. The data path from PC to TRX should now also contain hardware to Pulse Position Modulate data.

New modes

If a predefined command is received by the BSE it will enter one of the different modes in Table 3.3.

PPM generation

The modified BSE should support protocols PP4 and PP16 with configurable time constants and also have the ability to pad.

3.1.4 Typical usage of modified BSE

The usage of the modified BSE will differ only in type of frames sent to the TRX. With the first byte matching one of the predefined commands, the rest of the frame may be modulated to an IR-frame and sent out either with PP4 or PP16 and padded or not padded.

3.2 Development of modified BSE

As mentioned before, there was only time to develop the first two optional objectives, i.e. the capability to send occasional PP4 and PP16 frames.

<table>
<thead>
<tr>
<th>Bit-quadruple</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>A\mu s</td>
</tr>
<tr>
<td>0001</td>
<td>B\mu s</td>
</tr>
<tr>
<td>0010</td>
<td>C\mu s</td>
</tr>
<tr>
<td>0011</td>
<td>D\mu s</td>
</tr>
<tr>
<td>0100</td>
<td>E\mu s</td>
</tr>
<tr>
<td>0101</td>
<td>F\mu s</td>
</tr>
<tr>
<td>0110</td>
<td>G\mu s</td>
</tr>
<tr>
<td>0111</td>
<td>H\mu s</td>
</tr>
<tr>
<td>1000</td>
<td>I\mu s</td>
</tr>
<tr>
<td>1001</td>
<td>J\mu s</td>
</tr>
<tr>
<td>1010</td>
<td>K\mu s</td>
</tr>
<tr>
<td>1011</td>
<td>L\mu s</td>
</tr>
<tr>
<td>1100</td>
<td>M\mu s</td>
</tr>
<tr>
<td>1101</td>
<td>N\mu s</td>
</tr>
<tr>
<td>1110</td>
<td>O\mu s</td>
</tr>
<tr>
<td>1111</td>
<td>P\mu s</td>
</tr>
</tbody>
</table>

Table 3.2: Timestamps with PP16
<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRX</td>
<td>Ordinary functionality like in Section 2.3.2</td>
</tr>
<tr>
<td>PP4 tag 0</td>
<td>IR-frame is sent once with protocol PP4</td>
</tr>
<tr>
<td>PP4 tag 1</td>
<td>IR-frame is padded, i.e. sent out continuously with protocol PP4</td>
</tr>
<tr>
<td>PP4 tag 2</td>
<td>IR-frame is sent once with protocol PP4 and padding is turned off</td>
</tr>
<tr>
<td>PP16 tag 0</td>
<td>IR-frame is sent once with protocol PP16</td>
</tr>
<tr>
<td>PP16 tag 1</td>
<td>IR-frame is padded, i.e. sent out continuously with protocol PP16</td>
</tr>
<tr>
<td>PP16 tag 2</td>
<td>IR-frame is sent once with protocol PP16 and padding is turned off</td>
</tr>
</tbody>
</table>

Table 3.3: PPM modes

3.2.1 Structure of modified BSE

In the modified structure of the BSE the data paths from PC to TRX and from TRX to PC have been encapsulated in two new components, rx\_tx\_block and rx\_tx\_ppm\_block (see Figure 3.2). rx\_tx\_block (Figure 3.3) is taken directly from the initial BSE and contains the data path from TRX to PC. rx\_tx\_ppm\_block (Figure 3.4) contains the new functionality for the data path from PC to TRX. To reduce complexity a new abstraction layer has been introduced. rx\_block, tx\_block and fifo\_block constitute a natural division of responsibility within the data path and the fsm component coordinates the system.

3.2.2 New components

config\_pkg

The package config\_pkg replaces the my\_pkg component in the first BSE. This package contains the type conversion functions as well as all configurable constants in the modified BSE.

rx\_block

The rx\_block contains all logic for receiving and interpreting incoming frames. async\_nartrx and frame\_detector work as described in Section 2.4.2. The new component cmd\_interpreter is connected to async\_nartrx and interprets the first incoming byte in every frame. If this byte matches some predefined bytes the cmd\_interpreter signals a new mode to the fsm.

fifo\_block

The fifo\_block contains two fifo\_rams also described in Section 2.4.2 and logic to control these.
The tx_block contain the ppm4, ppm16 and async_uartxmt components. These are the different serial data transmission protocols that the BSE is able to generate. The three components are activated with enable signals and their output is connected to a 3-1 multiplexor giving the right output to the TRX. The three components also have transmission ready signals connected to the fsm and fifo_block.

fsm

The Finite State Machine (FSM) controls the fifo_block and tx_block. Depending on the state the fsm enables the UART or one of the PPM transmitters. If a PPM transmitter is enabled the fsm also coordinates to send the IR-frame command to the TRX first.
3.2.3 Synthesis of modified BSE

Number of slices: \[ \frac{512}{5120} = 10\% \]
Number of Slice Flip Flops: \[ \frac{352}{10240} = 3\% \]
Number of 4 input LUTs: \[ \frac{939}{10240} = 9\% \]
Number of bonded IOBs: \[ \frac{21}{324} = 6\% \]
Number of BRAM: \[ \frac{4}{40} = 10\% \]
Number of GCLKs: \[ \frac{1}{16} = 6\% \]

Table 3.4: Device utilization summary of the modified BSE on Xilinx Virtex II. Extract from XST synthesis report.

As seen in Table 3.4, the modified BSE is very much larger than the initially developed BSE (compare with Table 2.2). It is still smaller than an embedded processor solution though. The number of I/O pins has increased since several debug pins were added in order to be able to monitor the input and output from the FPGA.

3.2.4 Critical parts in the design

The critical part described in Section 2.4.5 remains in the modified BSE.
Chapter 4

Analysis

This chapter will consider the modified BSE.

4.1 Validation and Evaluation

These are the specifications from Section 1.3.1 to be validated and evaluated.

1. Serial UART differential RS232 connection to PC at 19200 bps
2. Base Station interface to TRX (serial UART differential RS485 at 62500 bps, 48V, etc.)
3. Capability to handle frames, i.e. keeping frames together when converting baud rate
4. Configurable baud rates (if baud rates are changed in the future)
5. Full duplex, i.e. ability to send and receive at the same time
6. As technology independent as possible
7. Optional: Capability to Pulse Position Modulate data with protocol PP4
8. Optional: PPM for protocol PP16
9. Optional: Capability to pad IR-frames

4.1.1 Validation

To validate the BSE the following operational tests were performed.

First of all a typical user case was tested on the BSE. Typical frames were sent to see if they had the expected effect. The frames sent are listed in Table 4.1. TRX shows 1 on startup\(^1\). The result of the user case gave an indication of the validity of specification 1, 2 and 3.

To test the behaviour of the BSE a test version of the BSE was developed. In the test version the rx and tx signal going to and from the trx were connected making the outgoing frame to the TRX also become the incoming frame from the TRX. Thus an echo effect was created making it possible to echo UART transmissions.

\(^1\)on a 7 segment display
### Specification 1, 2 and 3

A formal validation of specification 1, 2 and 3 was done by sending frames with lengths from 1 to 65 to the test version of the BSE. The frames sent were built of bytes starting from 0 and going up to the length of the sent frame. For instance if a frame of length 10 was sent it consisted of the following bytes (in hexadecimal form): 00, 01, 02, 03, 04, 05, 06, 07, 08, 09.

<table>
<thead>
<tr>
<th>Frame sent</th>
<th>Response in TRXLogger</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0, ..., 9</td>
<td>0, ..., 9</td>
</tr>
<tr>
<td>0, ..., 19</td>
<td>0, ..., 19</td>
</tr>
<tr>
<td>0, ..., 39</td>
<td>0, ..., 39</td>
</tr>
<tr>
<td>0, ..., 59</td>
<td>0, ..., 59</td>
</tr>
<tr>
<td>0, ..., 62</td>
<td>0, ..., 62</td>
</tr>
<tr>
<td>0, ..., 63</td>
<td>No response</td>
</tr>
<tr>
<td>0, ..., 64</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 4.2: Frames sent and responses

As seen in Table 4.2 communication works fine until a frame of length 64 is sent and there is no response. This depends on the design of the fifo ram component. The FIFO was developed as a circular buffer with a front and back address. When the front and back addresses are equal the FIFO signals empty. When a byte is written the front address is incremented by one. This means that if a frame of length 64 is sent, the front address will swap around and point on the back address and the FIFO will signal empty. Thus, there is no response. If frames longer than 64 bytes are sent this swap around effect will cut the first 64 bytes and send the rest.

### Specification 4

To validate the specification of configurable baud rates the test version of the BSE was used. The test frames in Table 4.2 were sent but with different baud rates programmed. Since TRXLogger could not work with a higher baud rate than 19200 bps the PC baud rate was left constant. The TRX baud rate was tested with higher baud rates. In Table 4.3 "Tested ok" means that frames from Table 4.2 echoed properly.

Thus, specification 4 is validated at least for the baud rates in Table 4.3.
Table 4.3: Different TRX baud rates

<table>
<thead>
<tr>
<th>Baud rate (bps)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>62500</td>
<td>Tested ok</td>
</tr>
<tr>
<td>115200</td>
<td>Tested ok</td>
</tr>
<tr>
<td>230400</td>
<td>Tested ok</td>
</tr>
<tr>
<td>460800</td>
<td>Tested ok</td>
</tr>
</tbody>
</table>

Specification 5 and 6

Specification 5 was automatically validated with the test version of the BSE. When rx and tx to TRX are connected the BSE is sending and receiving at the same time.

Specification 6 will be discussed in Section 4.2.

Specification 7 and 8

The IR-transmissions specified in specification 7 and 8 were validated with oscilloscope and a Pricer device called IR-listener. The IR-listener consists of a IR detector and a counter. When a timestamp is detected the IR-listener registers the time of detection. In this manner timestamps and the time intervals between them can be analyzed and translated back to ordinary data.

With the IR-listener and oscilloscope it was verified that the BSE could send IR-frames according to PP4 and PP16. In the oscilloscope printouts (Figure 4.1 and Figure 4.2) an IR-frame with 6 bytes is sent to the BSE (the lower signal) and modulated with PP4 and PP16 respectively (the upper signal). First comes the command byte and then the pulse train. Note also that the PP16 transmission contains less pulses than the PP4 transmission.

Figure 4.1: Oscilloscope capture of PP4
4.1.2 Evaluation

An evaluation should give some measure of the quality of the BSE. Quality for a hardware device was here translated to robustness, meaning that "noise" or erroneous utilisation does not affect device, and reliability, meaning that a specific input should always produce the same output.

Evaluation of robustness was done by sending "noise" to the test version of the BSE. The "noise" consisted of 4 x 20 bytes frames sent with baud rate 9600 bps filling up the FIFO with random data. After that the frames in Table 4.2 were sent and the responses compared. As seen in Table 4.4 something happened in the first sent frame after the "noise". An extra byte was transmitted. The rest of the frames tested ok.

This behaviour originates from the problem described in Section 2.4.5. If frames are sent too close together in time this will result in fragments of frames in the FIFOs. As seen in Table 4.4 the BSE behaviour is restored after one proper frame transmission.

<table>
<thead>
<tr>
<th>Frame sent</th>
<th>Response in TRXLogger</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3, 0</td>
</tr>
<tr>
<td>0, ..., 9</td>
<td>0, ..., 9</td>
</tr>
<tr>
<td>0, ..., 19</td>
<td>0, ..., 19</td>
</tr>
<tr>
<td>0, ..., 39</td>
<td>0, ..., 39</td>
</tr>
<tr>
<td>0, ..., 59</td>
<td>0, ..., 59</td>
</tr>
<tr>
<td>0, ..., 62</td>
<td>0, ..., 62</td>
</tr>
<tr>
<td>0, ..., 63</td>
<td>No response</td>
</tr>
<tr>
<td>0, ..., 64</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 4.4: Evaluation of robustness

Evaluation of reliability was meant to be tested with a bit error ratio test, giving a percentage of erroneous transmissions. However, this involved developing software with ability to send frames continuously, which was not prioritized (instead time was spent on developing support for IR-transmission).
4.2 Technology analysis

According to specification 6 the BSE should be as technology independent as possible. This Section will discuss the portability of the FPGA and the system board.

4.2.1 FPGA portability

All components are described in open source VHDL. Even the RAM component is open source, which means that all hardware can be modified if needed.

However the RAM component may have to be replaced if another technology is used. Altera or Lattice will probably not synthesize the RAM component to block RAM. It should, however, not be difficult to fit another RAM component into the BSE.

As seen in the device utilization summary (Table 3.4) only a small part (about 9%) of the Virtex II FPGA has been utilized, indicating that a much smaller chip could be used. Since Xilinx ISE can target all Xilinx devices in synthesis, an iterative matching process was conducted to find the smallest chip able to contain the BSE.

The best match was made with Spartan II FPGA (xc2s50) with 50000 gates (compared to 1000000 gates with Virtex II). On this device around 70% of the FPGA was used when programmed (see Figure 4.5). Spartan II also contains 32kbits of block RAM. Since the BSE uses $64 \times 8 \times 4 = 2048$ bits of RAM there is no need to modify the RAMs either.

<table>
<thead>
<tr>
<th>Device</th>
<th>Logic Cells</th>
<th>System Gates (Logic and RAM)</th>
<th>CLB Array (R x C)</th>
<th>Total CLBs</th>
<th>Maximum Available User I/O</th>
<th>Total Distributed RAM Bits</th>
<th>Total Block RAM Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC2S15</td>
<td>432</td>
<td>15,000</td>
<td>6 x 12</td>
<td>96</td>
<td>86</td>
<td>6,144</td>
<td>16K</td>
</tr>
<tr>
<td>XC2S50</td>
<td>972</td>
<td>30,000</td>
<td>12 x 18</td>
<td>216</td>
<td>92</td>
<td>19,624</td>
<td>24K</td>
</tr>
<tr>
<td>XC2S60</td>
<td>1,728</td>
<td>50,000</td>
<td>16 x 24</td>
<td>304</td>
<td>176</td>
<td>24,576</td>
<td>32K</td>
</tr>
<tr>
<td>XC2S100</td>
<td>2,700</td>
<td>100,000</td>
<td>20 x 30</td>
<td>600</td>
<td>176</td>
<td>36,400</td>
<td>40K</td>
</tr>
<tr>
<td>XC2S150</td>
<td>9,888</td>
<td>150,000</td>
<td>24 x 96</td>
<td>864</td>
<td>200</td>
<td>55,296</td>
<td>48K</td>
</tr>
<tr>
<td>XC2S200</td>
<td>5,292</td>
<td>200,000</td>
<td>28 x 42</td>
<td>1,176</td>
<td>284</td>
<td>75,284</td>
<td>56K</td>
</tr>
</tbody>
</table>

Notes:
1. All usage I/O counts do not include the four global clock input I/Os. See details in Table 3.4 rows 1-2.

A rough price comparison on Xilinx homepage showed that a Spartan II FPGA would cost around one fifth of the Virtex II FPGA.

Table 4.5: Device utilization summary of the modified BSE on Xilinx Spartan II. Extract from XST synthesis report.
4.2.2 System Board portability

If the Virtex II System Board would be replaced one would have to consider the following:

- The BSE uses the on board 100MHz oscillator. If another device would be used there may not be such an oscillator available. If another oscillator is used the frequency dividers will have to be given new constants to generate the right baud rates.

- The Base Station Interface is adapted for the P160 Prototype Module made for the Virtex II System Board. If another board would be used one would probably have to solve the Base Station Interface differently using other I/O pins. Also the RS485 circuits (ADM3485E) are made for 3.3V. One would also need a board supplying 3.3V to fit these.

- The PC interface uses a RS232 connection. This would also have to be available if another board would be used.
Chapter 5

Conclusions

5.1 Summary

This master thesis has developed a Base Station Emulator with the following specifications:

- Serial UART differential RS232 connection to PC at 19200 bps
- Base Station interface to TRX (serial UART differential RS485 at 62500 bps, 48V, etc.)
- Capability to handle frames, i.e. keeping frames together when converting baud rate
- Full duplex, i.e. ability to send and receive at the same time
- Capability to Pulse Position Modulate data with protocol PP4
- Capability to Pulse Position Modulate data with protocol PP16
- Capability to reprogram
  - Baud rate for UART to PC (currently set to 19200 bps)
  - Baud rate for UART to TRX (currently set to 62500 bps)
  - FIFO depths (currently set to 64 bytes)
  - IR command sent to TRX (set to TRXCMD_FRAME)
  - Separation time between IR command and IR modulated frame
  - Pulse width for PP4 and PP16
  - Burst width (time duration of burst) for PP4 and PP16
  - Timestamp for all symbols in PP4 and PP16

The Base Station Emulator is developed in open source VHDL which means that the design is as near technology independent as possible. The design can be instantiated in silicon devices large enough to contain the design (FPGAs, ASICs, etc). An investigation showed that among Xilinx devices, the Spartan II device (with 50000 system gates) was most suitable (see Section 4.2).
The Base Station Emulator is however limited to only handle occasional frames. If a sequence of frames is sent there is no guarantee that the BSE will handle them properly (see Section 4.1.2).

Finally, there was no time to develop padding functionality, i.e. capability to send IR-frames continuously. Therefore the BSE is not capable of communicating with the Electronic Shelf Labels.

5.2 Conclusions

The major effort in this thesis was put into development. This resulted in much functionality where two of the optional objectives were met. Less time was spent on evaluation and improvement of the existing design. This, on the other hand, resulted in lack of robustness when sending sequences of frames. This was a trade off which was difficult to plan ahead. But in a future project this balance should be considered carefully.

One could of course also discuss the technical decisions made during the thesis. Maybe an embedded processor solution would have been more suitable since it would be easier to change application code and/or add peripheral hardware if needed. Also the FIFO solution with two FIFOs could have been developed more efficiently so that the BSE would be able to handle sequences of frames. In a similar project these conclusions might help to make more experienced decisions.

5.3 Future work

First of all the handling of sequences of frames should be investigated, specified and developed. A first improvement would be to make sure that two frames close together in time always are handled properly. There should also be protection against incoming frames when both FIFOs are in use.

Next one would have to discuss how to develop the padding functionality. One solution would be to do it in software. If the above functionality is developed one could send frames continuously to the BSE and the BSE would make sure that they are sent properly. Of course there may be frames that are denied access to the BSE since the FIFOs are in use, but since padding means sending the same frame over and over again, this is not a problem.

One should also develop TRXLogger further so that sequences of frames can be used. That would enable building of more realistic test cases instead of just sending occasional frames.

Finally one could also investigate development of other features on the BSE such as error correcting codes.
Bibliography


Appendix A

Appendix

A.1 Base Station Emulator User Guide

NB! The Base Station Emulator is programmed to handle frames of length 1-63 bytes. Frames longer than 63 bytes will not be sent out properly.

A.1.1 Programming the board

The Virtex II System Board is equipped with a non volatile PROM which has been programmed with the construction code. With all jumpers M0-M3 closed on J1 the board will be programmed with the PROM content when powered. If jumper M0 and M2 are opened the board will have to be programmed with the JTAG connection. A LED (DS5 with text ”DONE”) is lit when the FPGA has been programmed.

A.1.2 Reset the Base Station Emulator

The Base Station Emulator is asynchronously reset with SW5 (switch 5)(D7). It is important to reset the emulator before use, in order to set all signals in a defined mode.

A.1.3 Sending ordinary frames

1. Open TRXLogger.
2. Connect to Virtex II System Board at 19200 bps.
3. Click ”Low Level Com”.
4. Click ”Update” button and choose trxcmd.cmd. Now there are several predefined frames to choose from the scroll menu. You can also edit the frame as you like.
5. Remember to calculate checksum before clicking ”Send”.

41
A.1.4 Sending IR-frames

1. Open TRXLogger.
2. Connect to Virtex II System Board at 19200 bps.
3. Click "Low Level Com".
4. Click "Update" button and choose trxcmd.cmd.
5. Start with sending the "Set address" frame.
6. Continue by sending the "Execute program" frame.
7. Continue by sending the "Set transtit mask" frame.
8. Now the TRX is ready to send IR-frames. Choose "IR-frame with PP4" or "IR-frame with PP16". The first byte in these frames is a symbol recognized by the Base Station Emulator with the effect that the rest of the frame will be modulated with PP4 or PP16 respectively. Of course you can also edit the rest of the frame as you like. Remember that the maximum frame length is 63 bytes, i.e. 62 bytes IR-frame. NB! You don’t need a checksum when sending IR-frames to the TRX.

A.1.5 Programming the board with new constants

The configurable parameters of the Base Station Emulator are

- Baud rate for UART to PC
- Baud rate for UART to TRX
- Symbol bytes for recognizing IR-frames
- FIFO depths
- IR command sent to TRX
- Separation time between IR command and IR modulated frame
- Pulse width (carrier frequency) for PP4 and PP16
- Burst width (time duration of burst) for PP4 and PP16
- Burst separation time for all symbols in PP4 and PP16

The constants are gathered in config_pkg.vhd. To change these constants follow these instructions.

1. Open Xilinx ISE 6/Project Navigator.
2. Adjust system clock to November 2005 (the Xilinx software is an evaluation version).
3. Click "Open project" and choose emulator.npl in /Exjobb/BASE_STATION_EMULATOR/VirtexII_BSE_Project.
4. Open `config.pkg` source file and edit constants (VHDL uses `--` for comments) and save file.

5. Be sure that the top component `emulator.vhd` is selected and double click "Configure Device(iMPACT)" from the "Processes for Source" menu. The Base Station Emulator will now be rebuilt with the new constants and a new software, iMPACT, will appear in a new window (see Figure A.1).

6. Be sure that the Virtex II System Board is powered and the JTAG cable connected. Choose "Boundary Scan Mode".

7. Choose "Automatically connect to..."

8. If everything is in order the software will find 2 devices, the PROM and the FPGA and ask which file to program the devices with. Choose `emulator.mcs` for the PROM (you will not program the PROM so this is just a dummy) and `emulator.bit` for the FPGA (there will be a warning about a change in a clock, ignore this and continue) (see Figure A.2).

9. Right click on the FPGA device and choose "Program..."

10. Uncheck the "Verify" option if selected and press ok. The board will now be programmed with the modified Base Station Emulator (see Figure A.3).

A.2 Source code

The source code of the Base Station Emulator is placed in `/Exjobb/BASE_STATION_EMULATOR/Source_code` and consists of
Figure A.2: Printed screen

- config_pkg.vhd
- async_uartrx.vhd
- async_uartxmt.vhd
- freq_div.vhd
- cmd_interpreter.vhd
- frame_detector.vhd
- rx_block.vhd
- fifo_block.vhd
- dual_port_ram.vhd
- fsm.vhd
- ppm4.vhd
- ppm16.vhd
- tx_block.vhd
- rx_tx_block.vhd
- rx_tx_ppm_block.vhd
A.3 Pin usage and baud rate constants tables

LVDS is short for LVDS TRANSMIT or GPIO which is found on the Virtex II System Board.
### Table A.1: Pin usage table

<table>
<thead>
<tr>
<th>Signal</th>
<th>FPGA Pin</th>
<th>Placement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pc_rx</td>
<td>b7</td>
<td>internal</td>
<td>data from PC to BSE</td>
</tr>
<tr>
<td>pc_tx</td>
<td>a7</td>
<td>internal</td>
<td>data from BSE to PC</td>
</tr>
<tr>
<td>resetn</td>
<td>d7</td>
<td>SW5</td>
<td>asynchronous reset (active low)</td>
</tr>
<tr>
<td>clk</td>
<td>b11</td>
<td>internal</td>
<td>100MHz oscillator</td>
</tr>
<tr>
<td>trx.tx</td>
<td>e11</td>
<td>internal</td>
<td>data from BSE to TRX</td>
</tr>
<tr>
<td>trx.rx</td>
<td>v8</td>
<td>internal</td>
<td>data from TRX to BSE</td>
</tr>
<tr>
<td>trx.tx_debug</td>
<td>h2</td>
<td>LVDS 1</td>
<td>trx.tx debug pin</td>
</tr>
<tr>
<td>trx.rx_debug</td>
<td>h1</td>
<td>LVDS 2</td>
<td>trx.rx debug pin</td>
</tr>
<tr>
<td>pc_tx_debug</td>
<td>j2</td>
<td>LVDS 3</td>
<td>pc_tx debug pin</td>
</tr>
<tr>
<td>pc_rx_debug</td>
<td>j1</td>
<td>LVDS 4</td>
<td>pc_rx debug pin</td>
</tr>
<tr>
<td>debug_bit1</td>
<td>k2</td>
<td>LVDS 5</td>
<td>NC</td>
</tr>
<tr>
<td>debug_bit2</td>
<td>k1</td>
<td>LVDS 6</td>
<td>NC</td>
</tr>
<tr>
<td>debug_bit3</td>
<td>e4</td>
<td>LVDS 7</td>
<td>NC</td>
</tr>
<tr>
<td>debug_bit4</td>
<td>e3</td>
<td>LVDS 8</td>
<td>NC</td>
</tr>
<tr>
<td>gnd</td>
<td>-</td>
<td>LVDS 9</td>
<td>-</td>
</tr>
<tr>
<td>gnd</td>
<td>-</td>
<td>LVDS 10</td>
<td>-</td>
</tr>
<tr>
<td>debug_byte[7]</td>
<td>f4</td>
<td>LVDS 11</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[6]</td>
<td>f3</td>
<td>LVDS 12</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[5]</td>
<td>g4</td>
<td>LVDS 13</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[4]</td>
<td>g3</td>
<td>LVDS 14</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[3]</td>
<td>h4</td>
<td>LVDS 15</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[2]</td>
<td>h3</td>
<td>LVDS 16</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[1]</td>
<td>j4</td>
<td>LVDS 17</td>
<td>NC</td>
</tr>
<tr>
<td>debug_byte[0]</td>
<td>j3</td>
<td>LVDS 18</td>
<td>NC</td>
</tr>
<tr>
<td>gnd</td>
<td>-</td>
<td>LVDS 19</td>
<td>-</td>
</tr>
<tr>
<td>gnd</td>
<td>-</td>
<td>LVDS 20</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table A.2: Baud rate constants table with 100MHz oscillator

<table>
<thead>
<tr>
<th>Baud rate (bps)</th>
<th>1 * baud rate</th>
<th>16 * baud rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>5208</td>
<td>326</td>
</tr>
<tr>
<td>19200</td>
<td>2604</td>
<td>163</td>
</tr>
<tr>
<td>38400</td>
<td>1302</td>
<td>81</td>
</tr>
<tr>
<td>57600</td>
<td>868</td>
<td>54</td>
</tr>
<tr>
<td>62500</td>
<td>800</td>
<td>50</td>
</tr>
<tr>
<td>115200</td>
<td>434</td>
<td>27</td>
</tr>
<tr>
<td>230400</td>
<td>217</td>
<td>13</td>
</tr>
<tr>
<td>460800</td>
<td>108</td>
<td>7</td>
</tr>
</tbody>
</table>

Table A.2: Baud rate constants table with 100MHz oscillator