Institute of Neuroinformatics
UNI - ETH Zurich

Architectura y
Technologia de Computadores
Universidad de Sevilla

Diploma Thesis

Highspeed USB2.0 AER Interfaces

Author:
Raphael Berner

Advisors:
Dr. Tobias Delbruck (ETHZ)
Prof. Anton Civit Balcells (US)
Dr. Alejandro Linares Barranco (US)

14th April 2006

Acknowledgements

First of all, I would like to thank Professor Anton Civit Balcells for accepting me as diploma student at the institute of Architectura y Technologia de Computadores and thereby giving me the opportunity to do this project in Seville and also for giving me a lot of helpful advices and inputs.

I would also sincerely like to thank Dr. Tobi Delbruck of the Institute of Neuroinformatics in Zürich for making this diploma thesis possible and guiding and helping me throughout the whole project.

Many thanks as well to my advisor Dr. Alejandro Linares-Barranco for giving me a lot of guidance and technical advices, as well as for organising our apartment in Seville.

Furthermore I would like to thank Daniel Cacigas Muñiz for his help in everyday Spanish life, Miguel Gutiérrez Páez and Ángel Jiménez Fernández for their help during the design of the PCBs and all members of Architectura y Technologia de Computadores not mentioned here for a great time and their hospitality.

Above all I would like to thank my parents, without whose support it would not have been possible for me to do this project in Seville. Last but not least I would like to thank my girlfriend Patricia who had to be patient sometimes...

Seville, 14th April 2006

Raphael Berner

Abstract

This diploma thesis presents two highspeed USB2.0 AER (*A*ddress *E*vent *R*epresentation) interfaces, allowing real time capturing of AER communication as well as AER event sequencing. Both devices use the Cypress FX2 USB2.0 transceiver.

The USBAERmini2 is a low cost, bus powered interface which is able to monitor and sequence AER events with event rates up to four megaevents per second. It also introduces the capability to capture events synchronously from several devices. It features three parallel AER ports, one for sequencing, one for monitoring and one for passing through the monitored events.

The second device presented is an enhancement of the approved USBAER designed at the Architectura y Technologia de Computadores group at the University of Seville for the CAVIAR project, improving the functionality by replacing the Spartan2 with a Spartan3 FPGA, introducing USB2 highspeed communication, offering a VGA port and Serial AER interfaces.

Additionally, a Java interface is presented, which allows accessing to the described devices from Matlab or with the CaviarViewer application designed by Dr. Tobias Delbruck.

Zusammenfassung

Diese Diplomarbeit präsentiert zwei USB zu AER (*A*ddress *E*vent *R*epresentation) Schnittstellen, welche es ermöglichen, die Kommunikation zweier AER Geräte aufzunehmen und in Echtzeit am Computer zu betrachten, sowie AER Events zu generieren und so AER Geräte zu stimulieren. Beide Schnittstellen verwenden den Cypress FX2 USB2.0 Transceiver zur Anbindung an den Computer.

Das USBAERmini2 wird durch USB mit Spannung versorgt und ist dadurch bestens geeignet für portable Anwendungen. Es ermöglicht das Aufnehmen von AER Events mit bis zu vier Megaevents pro Sekunde und bietet ausserdem die Möglichkeit, mehrere Geräte miteinander zu synchronisieren.

Die zweite vorgestellte Schnittstelle ist eine Weiterentwicklung des bekannten USBAER, und erweitert dessen Funktionalität durch ein grösseres FPGA, ein VGA Anschluss mit bis zu 262144 Farben oder 64 Graustufen und zwei Serial AER Anschlüsse.

Ausserdem wird eine Java Schnittstelle präsentiert, welche den Zugriff auf die Geräte aus Matlab und der CaviarViewer Applikation ermöglicht.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The CAVIAR project

The Institute of Neuroinformatics in Zürich and the Architectura y Technologia de Computadores group in Seville, along with the Institute of Microelectronics in Seville and the University of Oslo are project partners in the CAVIAR (*C*onvolution *A*ddress-Event-Representation *Vi*sion *A*rchitecture for *R*eal-Time) EU project [1]. The goal of the CAVIAR project is to develop AER hardware modules, which enable the construction of a bio-inspired hierarchical structured multi-chip vision processing system. A demonstrator capable of following trajectories of moving objects will be presented.

The main data processing blocks are custom analog VLSI chips, however digital interfaces and devices are required for address space remapping and monitoring the system in various stages.

1.2 Diploma Project Description

With the progress of the CAVIAR project, resolutions of the retina and the convolution chips have increased, causing higher data rates. The available USB1.1 interfaces are therefore not suitable anymore for real-time monitoring of the system, while the PCI-AER interfaces are not usable with laptop computers and are therefore rather inflexible and not suited for presentations.

This diploma thesis presents two new digital AER devices, employing the Cypress FX2 USB2.0 transceiver.

The first device presented is a small, portable and USB bus powered board capable of monitoring an AER connection and sequencing AER events sent by the host computer. It is able to monitor and sequence events simultaneous, which enables the project partners to stimulate and monitor their modules with data recorded from an earlier stage or with artificial data. Several devices can be synchronised for synchron recording of data from various stages of the system.

The second device is an advancement of the proven USBAER developed by the group of Architectura y Technologia de Computadoresat the University of Seville. It adds a bigger FPGA, USB2.0 data-rates, a VGA port and also serial AER interface to the architecture of the USBAER, thereby improving its functionality.

1.3 Outline

This report is outlined as follows. Chapter 2 presents a short introduction to the AER protocol. In chapter 3, the Cypress FX2 USB2.0 transceiver is presented and several advices are given on how to use this device. Chapter 4 describes the first interface, the USBAERmini2 board. Chapter 5 presents the new mapper and serves as introduction for those programming VHDL for its FPGA. Chapter 6 concludes this report. Appendix A is a userguide for the installation and use of the USBAERmini2 and the CaviarViewer application. Appendix B describes the structure of the relevant folders in the subversion repository.

Chapter 2

The Address Event Representation Protocol

2.1 General

The AER (*A*ddress *E*vent *R*epresentation) protocol is an asynchronous communication protocol used for communication between AER building blocks in the CAVIAR project. It defines the communication between neuromorphic devices such as silicon retinas or I&F neuron arrays.

Several protocol versions are proposed in [2]:

- Single sender/single receiver
- Multiple sender/single receiver
- Single sender/multiple receiver

Most of the devices developed for the CAVIAR project rely on the single sender/single receiver protocol (also called P2P), therefore the devices developed during this diploma project implement this version of the protocol. It is described in the following section in detail. Please see [2] for details on the other proposed protocol versions.

2.2 AER P2P Protocol Details

2.2.1 Protocol and Signal Timing

Sender and receiver are connected through 18 signal lines: a request line driven by the sender, an acknowledge line driven by the receiver and 16 data lines driven by the sender.

The AER P2P protocol is four-phase handshake protocol. Figure 2.1 and table 2.1 show the four phases and the timing requirements. Please note that the request and the acknowledge signals are active low.

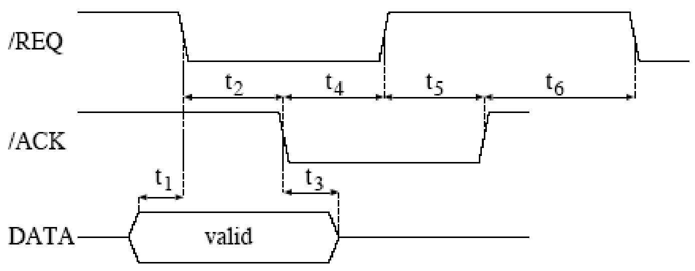


Figure 2.1: AER P2P protocol

	min	max	avg
t_1	$0s$	∞	
t_2	$0s$	∞	$\leq 700ns$
t_3	$0s$	∞	
t_4	$0s$	$100ns$	
t_5	$0s$	$100ns$	
t_6	$0s$	∞	

Table 2.1: AER P2P timing requirements

2.2.2 Physical Layer

The physical layer of the protocol uses ATA 40 pin IDC connectors and cables, which are widely used and therefore easily available and cheap. Please see [2] for pinout information.

Unlike indicated in [2], the CAVIAR project now uses $3.3V$ signals for the AER protocol, $5V$ signals are not supported anymore.

Chapter 3

The Cypress FX2 USB2.0 Transceiver

The Cypress FX2 is a highspeed USB2.0 transceiver with an enhanced 8051 microprocessor. Its main features are shortly described in the following sections. For a more thorough description of the FX2, refer to the datasheet [3] and the technical reference manual [4].

3.1 Architecture

The FX2 consists of the following main building blocks:

- On-chip USB2.0 transceiver operating at Full-Speed (12 Mbits/s) or High-Speed (480 Mbits/s).
- "Smart" Serial Interface Engine (SIE), which implements much of the USB protocol, thereby reducing firmware complexity
- Enhanced 8051 microprocessor
- 8k of on-chip RAM used as program and data RAM
- 4k of endpoint fifo buffers, configurable to be either two, three or four double-, triple- or quad-buffered endpoints.

Figure 3.2 on page 7 shows an overview over the whole architecture.

3.2 The Microprocessor

The microprocessor built in the FX2 shows several enhancements over a standard 8051 processor. It is able to run at up to $48MHz$, and instruction cycles are reduced to an average of four instead of twelve clock cycles per instruction.

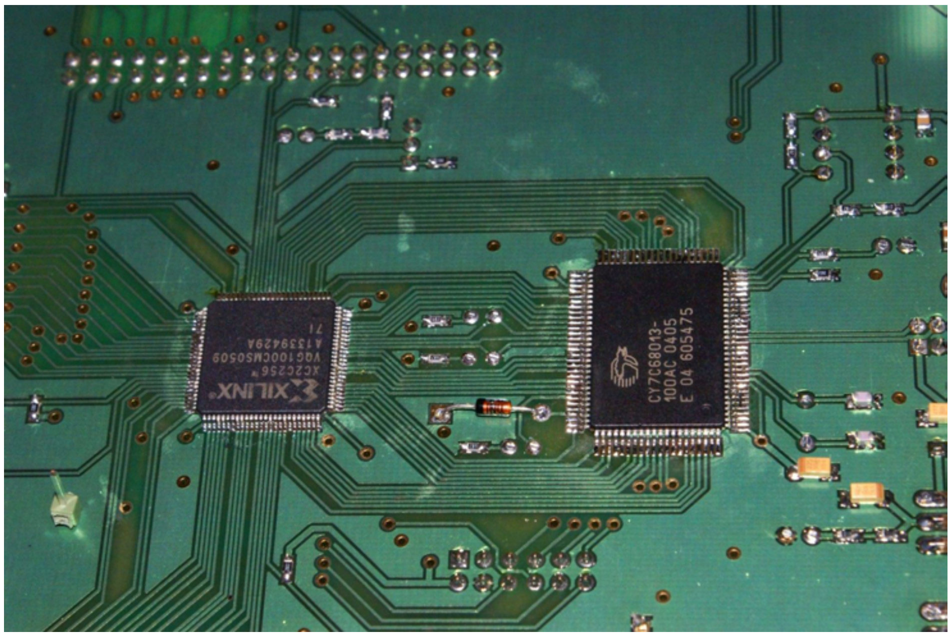


Figure 3.1: A close-up on the Cypress FX2 on the right hand side and the Xilinx Coolrunner 2 used on the USBAERmini2 on the left hand side.

The FX2 adds eight new interrupt sources to the standard 8051. Table 3.1 shows all available interrupts. The FX2 provides a second level of interrupt vectoring called *Autovectoring*, which is used to service the 27 different interrupt sources of USBINT and the 14 different sources for IE4.

It is important to note that the processor normally doesn't take part in high speed USB transfers, it just configures the SIE and the endpoints accordingly and then steps out of the way. It is by far not fast enough to handle the datarates USB2.0 offers.

The processor serves three main purposes:

- High level USB protocol handling like servicing vendor requests.
- Configuring endpoint buffers and fifo interface.
- General purpose system use.

3.3 The Endpoint Fifos

The FX2 can have up to 7 endpoints, listed in table 3.2. EP0, EP1IN and EP1OUT can only be accessed from the CPU, while EP2, EP4, EP6 and EP8 can be accessed from the Fifo Interface (see section 3.4) as well. In fact, this would be the normal

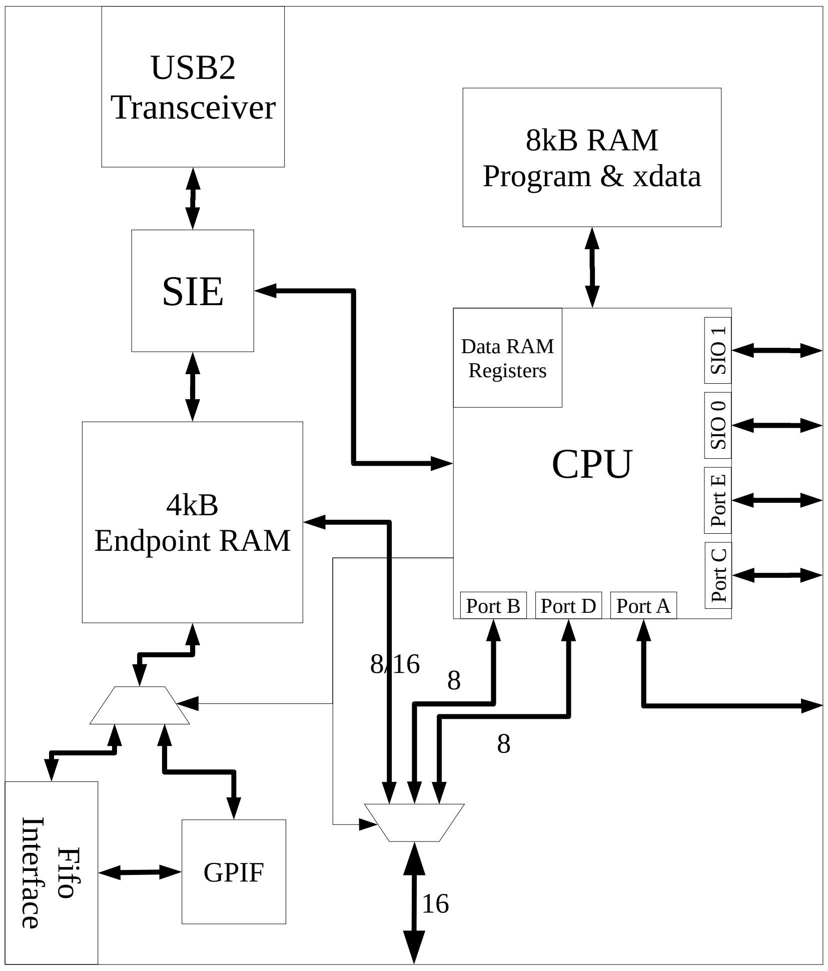


Figure 3.2: Main building blocks of the Cypress FX2

Interrupt	Purpose
WAKEUP	USB Resume Interrupt
INT0	External Interrupt 0
TF0	Timer 0 Overflow
INT1	External Interrupt 1
TF1	Timer 1 Overflow
TF2	Timer 2 Overflow
RI_0 & TI_0	USART0 Rx & Tx
RI_1 & TI_1	USART0 Rx & Tx
USBINT	USB Interrupt
I2CINT	I^2C Bus Interrupt
IE4	GPIF / Fifo / Interrupt Pin 4
IE5	Interrupt Pin 5
IE6	Interrupt Pin 6

Table 3.1: FX2 Interrupts. FX2-only interrupts are printed **bold**

way to access these fifo buffers, since it is not possible to achieve high datarates if the processor is involved in the transfer.

The high bandwidth data moving endpoints EP2 to EP8 can be configured in several ways, as can be seen in table 3.2. However, not all combinations are possible, since the total buffer size is 4kB. For example, if a quad buffered isochronous endpoint of 1024 bytes is desired, no other endpoint can be used. Double buffered means that the fifo interface can fill or empty one buffer, while the other buffer can be processed by the SIE. Triple and quad buffering are similar, giving the possibility to smooth data bursts, therefore improving performance, reducing or even eliminating the time either SIE or fifo interface has to wait for the other. The endpoints are configured through a set of registers, which have to be set to appropriate values by the 8051 firmware.

The endpoints EP2 to EP8 can be (and usually are) configured in either auto-in or auto-out mode, depending on the direction of the endpoint. This means that full buffers are automatically committed from the interface to the SIE and vice-versa, without any involvement of the CPU. This auto mode can also be disabled to give the CPU the possibility to process full buffers when they arrive from the host or before they are sent to the host.

3.4 The Fifo Interface

There two are different possible ways to control and access the high datarate endpoint fifos of the FX2, one requiring external logic and one which doesn't. Both ways are configurable, namely the databus can be chosen to be either 8 or 16 bits wide.

Endpoint	Type	Max. Size (Bytes)	Buffering
EP0	Control	64	single
EP1IN	Bulk or Interrupt	64	single
EP1OUT	Bulk or Interrupt	64	single
EP2	Bulk, Interrupt or Isochronous	1024 (Iso & Int) 512 (Bulk)	double, triple quad
EP4	Bulk, Interrupt or Isochronous	512	double
EP6	Bulk, Interrupt or Isochronous	1024 (Iso & Int) 512 (Bulk)	double, triple quad
EP8	Bulk, Interrupt or Isochronous	512	double

Table 3.2: Available Endpoints in the Cypress FX2, high speed mode

See subsection 3.6.3 for some important information regarding the configuration of these modes.

3.4.1 General Programmable Interface (GPIF)

The FX2 features a programmable state machine, called GPIF, which can be used to control the endpoint fifos, eliminating the need for external control logic. However, due to limited flexibility, the Slave Fifo mode described in the following section has been used for this project. The GPIF is therefore not described further. For more information refer to [4].

3.4.2 Slave Fifo Interface

When using an external Fifo master, the FX2 is configured to act in slave fifo mode. A number of pins are dedicated to control the fifo buffers, see table 3.4.2. The interface can be configured to be either asynchronous or synchronous, clocked either externally or internally.

When the fifo master wants to either read or write from or to one of the fifos, it has to make sure a buffer is available by checking an appropriate flag, set the fifo address, assign the data on the bus (for writing) and then assign either the SLRD and the SLOE pin for reading or SLWR pin for writing.

3.5 ReNumeration

Cypress invented the concept of ReNumeration, where a device enumerates without firmware, a dedicated driver then downloads the firmware to the device, which now disconnects and re-enumerates with a different VID/PID combination. This feature

Pin Name	Purpose
SLRD	Read strobe
SLWR	Write strobe
FifoAddr[0..1]	Endpoint Fifo Selection
PktEnd	Committing Short Packet To SIE
SLOE	Output Enable
FifoFlag[A..D]	Programmable flags to indicate if a fifo is full, empty or filled to a user-programmable level
IFClock	Interface Clock, can be configured as input (from 5 to 48 MHz) or as output (30 or 48 MHz)

Table 3.3: The Pins used to control the Slave Fifo Interface

is very useful during the firmware development, however in a final version it is more convenient to save the firmware on an EEPROM to avoid the necessity of using two different drivers.

3.6 Advices For Working With The FX2

3.6.1 Reserved Pin

All available packages have a pin which is reserved. Do not forget to pull this pin to ground, as this will prevent the oscillator to start.

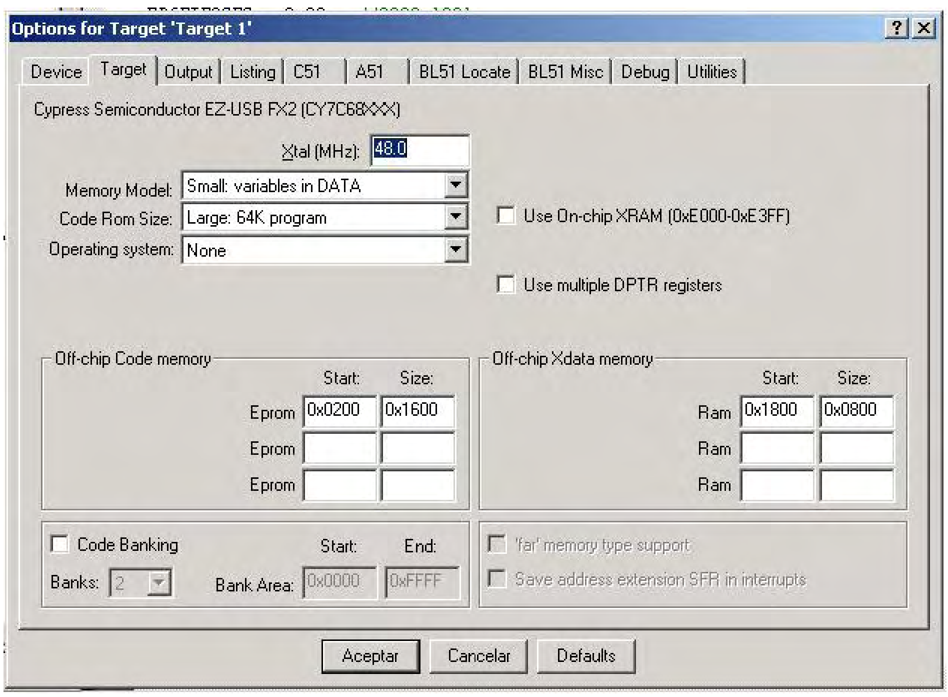
3.6.2 Memory

In contrast to the standard 8051 architecture, code memory and xdata memory share the same physical memory in the FX2. Therefore it is very important to make sure that they don't overlap. The following table shows a possible configuration of the 8kB of shared code/xdata memory, while figure 3.3 shows the Keil memory map configuration window.

Begin	End	Size (Bytes)	Type
0x0000	0x01FF	512	Interrupt Vector Jump Table
0x0200	0x17FF	5632	Code Memory
0x1800	0x1FFF	2048	xdata Memory

3.6.3 Changing From Interface To Port Mode

The register `IFCONFIG` is used to select one of the interface modes described in section 3.4, or the port mode, where the microprocessor ports B and D are brought to the package pins instead of the fifo data bus.



Cypress Semiconductor EZ-USB FX2 (CY7C68XXX)

Xtal (MHz): 48.0

Memory Model: Small: variables in DATA

Code Rom Size: Large: 64K program

Operating system: None

☐ Use On-chip XRAM (0xE000-0xE3FF)

☐ Use multiple DPTR registers

Off-chip Code memory

	Start:	Size:
Eprom	0x0200	0x1600
Eprom		
Eprom		

Off-chip Xdata memory

	Start:	Size:
Ram	0x1800	0x0800
Ram		
Ram		

☐ Code Banking

Banks: 2 Bank Area: Start: 0x0000 End: 0xFFFF

☐ 'far' memory type support

☐ Save address extension SFR in interrupts

Figure 3.3: This figure shows the memory map configuration that is used by the Keil software for the CypressFX2 firmware for both USBAERmini2 and USB2AERmapper.

Changing from port mode to fifo mode could be useful to send configuration bytes or special data on the fifo data lines to the interfacing device. However, it is very important to know that whenever `IFCONIFG` is changed, other configuration registers are reset. Unfortunately Cypress does not indicate which registers are affected, nor does the technical reference manual mention this problem at all. The following list shows affected registers, but without claim for completeness:

- `PINFLAGSAB` and `PINFLAGSCD`
- `OEx`: output enable registers for 8051 ports
- `IOx`: 8051 port state
- `EPxFIFOCFG`: fifo configuration registers

3.6.4 Fifos

When only two high bandwidth endpoints are needed, use EP2 and EP6, as they are the most flexible.

It is very important to disable auto-in or auto-out mode when it is desired that the CPU is involved in handling the endpoint buffers. For example to reset the fifos, first disable auto-in, then reset the fifos, rearm OUT-fifos and finally re-enable auto-in. If auto-in is not disabled while the fifos are accessed, the fifos will be blocked afterwards. The following code section shows how to reset EP2:

```
EP2FIFOCFG = 0x01; // disable auto-in

SYNCDELAY;
FIFORESET = 0x80; // NAK all traffic
SYNCDELAY;
FIFORESET = 0x02; // reset EP2
SYNCDELAY;
FIFORESET = 0x00; // stop NAKing traffic

//rearm EP2 (OUTPKTEND has to be applied twice when
//  EP2 is double-buffered, four times when EP2 is quad-buffered)
SYNCDELAY;
OUTPKTEND = 0x82;
SYNCDELAY;
OUTPKTEND = 0x82;

SYNCDELAY;
EP2FIFOCFG = 0x11; // re-enable auto in
```

3.6.5 Clock Issues

The setup times of the SLRD and the SLWR pins are very tight when working with an internal clock of 48MHz, leaving less than 3ns for the external logic to produce stable outputs after the clock edge. The constrains are somewhat relaxed when applying an external clock.

3.6.6 Firmware Frameworks

The firmware frameworks are a good start for developing the firmware for the FX2. There are however some code sections which are unnecessary for most applications and are therefore only complicating matters. For example the following section can be removed if no external RAM is used (as is always the case for the 56 pin and the 100 pin versions), as it checks if the descriptor tables are in internal or external RAM.

```
if ((WORD)&DeviceDscr & 0xC000)
{
   // first, relocate the descriptors
   IntDescrAddr = INTERNAL_DSCR_ADDR;
   ExtDescrAddr = (WORD)&DeviceDscr;
   DevDescrLen = (WORD)&UserDscr - (WORD)&DeviceDscr + 2;
   for (i = 0; i < DevDescrLen; i++)
     *((BYTE xdata*)IntDescrAddr+i) =*((BYTE xdata*)ExtDescrAddr+i);

   // update all of the descriptor pointers
   pDeviceDscr = IntDescrAddr;
   offset = (WORD)&DeviceDscr - INTERNAL_DSCR_ADDR;
   pDeviceQualDscr -= offset;
   pConfigDscr -= offset;
   pOtherConfigDscr -= offset;
   pHighSpeedConfigDscr -= offset;
   pFullSpeedConfigDscr -= offset;
   pStringDscr -= offset;
}
```

Also the code for suspend mode may cause problems sometimes, so it is best to uncomment it in the beginning and add it later only if necessary.

3.6.7 Firmware Development

The Cypress development kit provides a very useful application called CyConsole, which makes firmware development easier. During firmware development, the firmware is not written to the EEPROM. As the device doesn't have firmware without EEPROM, it enumerates as blank Cypress FX2 device when connected. The firmware can now be downloaded manually with the CyConsole application. The application also offers the possibility to write and read all the available endpoints of the connected devices and is therefore very useful during firmware testing.

To use this application, a Cypress FX2 device without firmware has to be connected to the Cypress driver instead of the USBIO driver. If you want to use the application for testing firmware, in your descriptor table you have to use a VID/PID combination that connects to the Cypress driver, for example VID 04B4 and PID 1004, which is a Cypress sample device.

3.6.8 VID/PID

Be aware the VID and PID are saved in the descriptor table in the order LSB MSB, therefore `dw 0089H` actually stands for 0x8900.

3.6.9 Adding an Interrupt Service Routine

To add interrupt service routines to the FX2 firmware frameworks, the following steps are required:

1. Add the service routine to the main firmware file (not to `fw.c`).
2. Look up the address of the interrupt vector you wish to use in the technical reference manual [4].
3. Add an interrupt jump vector table to the μVision project.

For the USBAERmini2 firmware, which uses the external interrupts zero and one, the jump vector table looks like the following, where the address after `AT` is the one you find in step two:

```
EXTRN CODE (ISR_TSReset)

CSEG    AT      0003H
        LJMP    ISR_TSReset

EXTRN CODE (ISR_MissedEvent)

CSEG    AT      0013H
        LJMP    ISR_MissedEvent
END
```

Chapter 4

The USBAERmini2

The first part of this diploma project was to design an USB2.0 AER Monitor/Sequencer interface with the following features.

- Simultaneous monitoring and sequencing of Address Events
- Pass-through monitored events without introducing a significant amount of delay
- Bus powered
- Easy to use
- Accessible from Matlab and compatible with the CaviarViewer Application developed by Dr. Tobias Delbruck.
- As simple as possible

4.1 The Device

4.1.1 Device Overview

Figures 4.1 and 4.2 show pictures of the final device, while figure 4.3 on page 17 shows a schematic overview. The main elements of the USBAERmini2 are the Cypress FX2, the Xilinx Coolrunner 2 CPLD, three AER ports and an EEPROM.

The 8051 microcontroller in the Cypress FX2 is responsible for the configuration of the endpoints and for receiving and interpreting commands from the host computer. In the future, it could also be used to reprogram the CPLD.

The CPLD is responsible for handshaking with the AER devices and reading and writing events to the fifos in the Cypress FX2. Four finite state machines (FSM) are implemented to achieve this. The CPLD acts as fifo master, controlling the endpoint fifos in the FX2. It doesn't have internal RAM, it can only save one event temporary. So every event monitored has to be written to the fifo before a new event can be received. The CPLD also includes a counter, which is used to

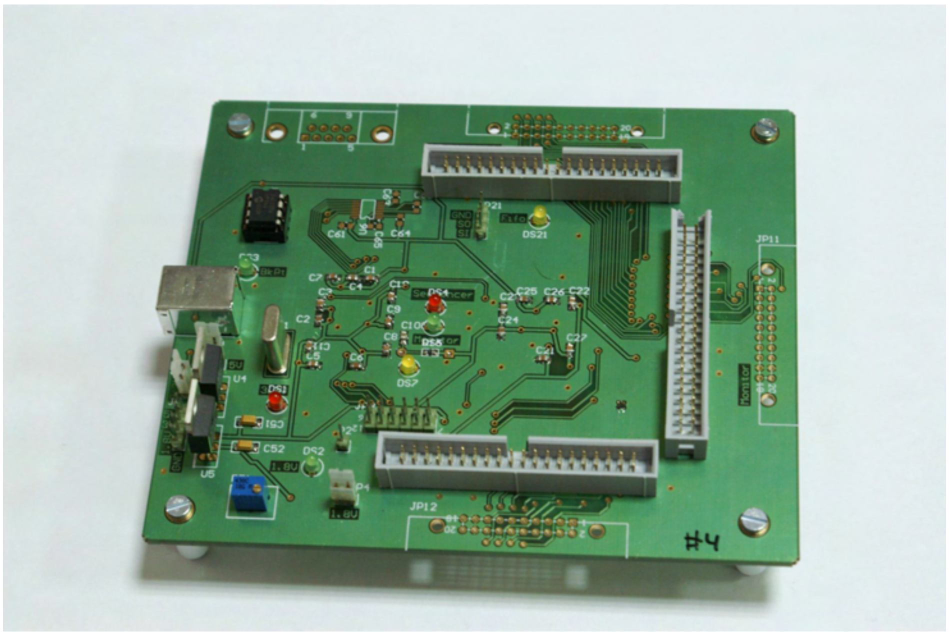


Figure 4.1: The top side of the USBAERmini2

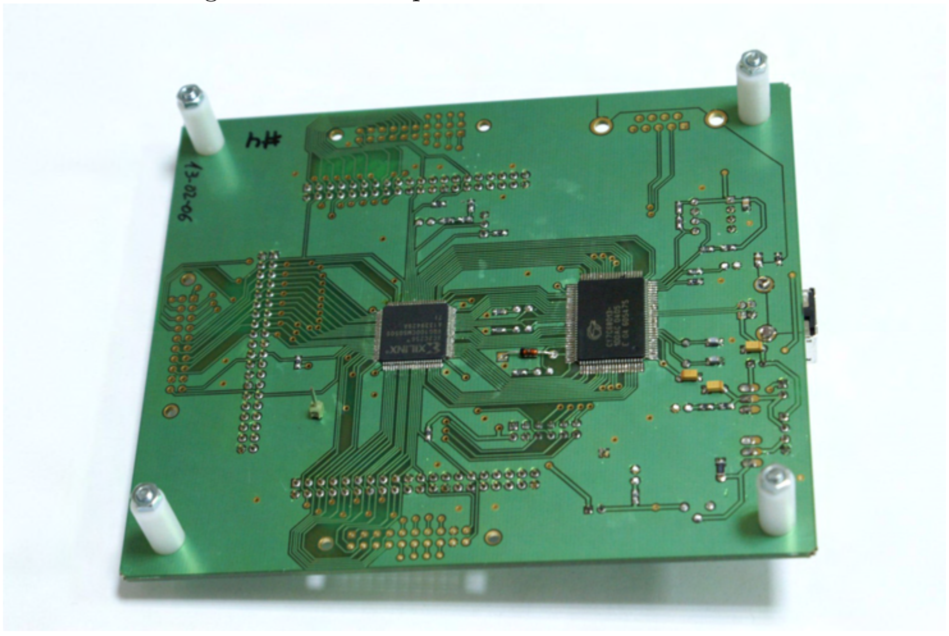


Figure 4.2: The bottom side of the USBAERmini2

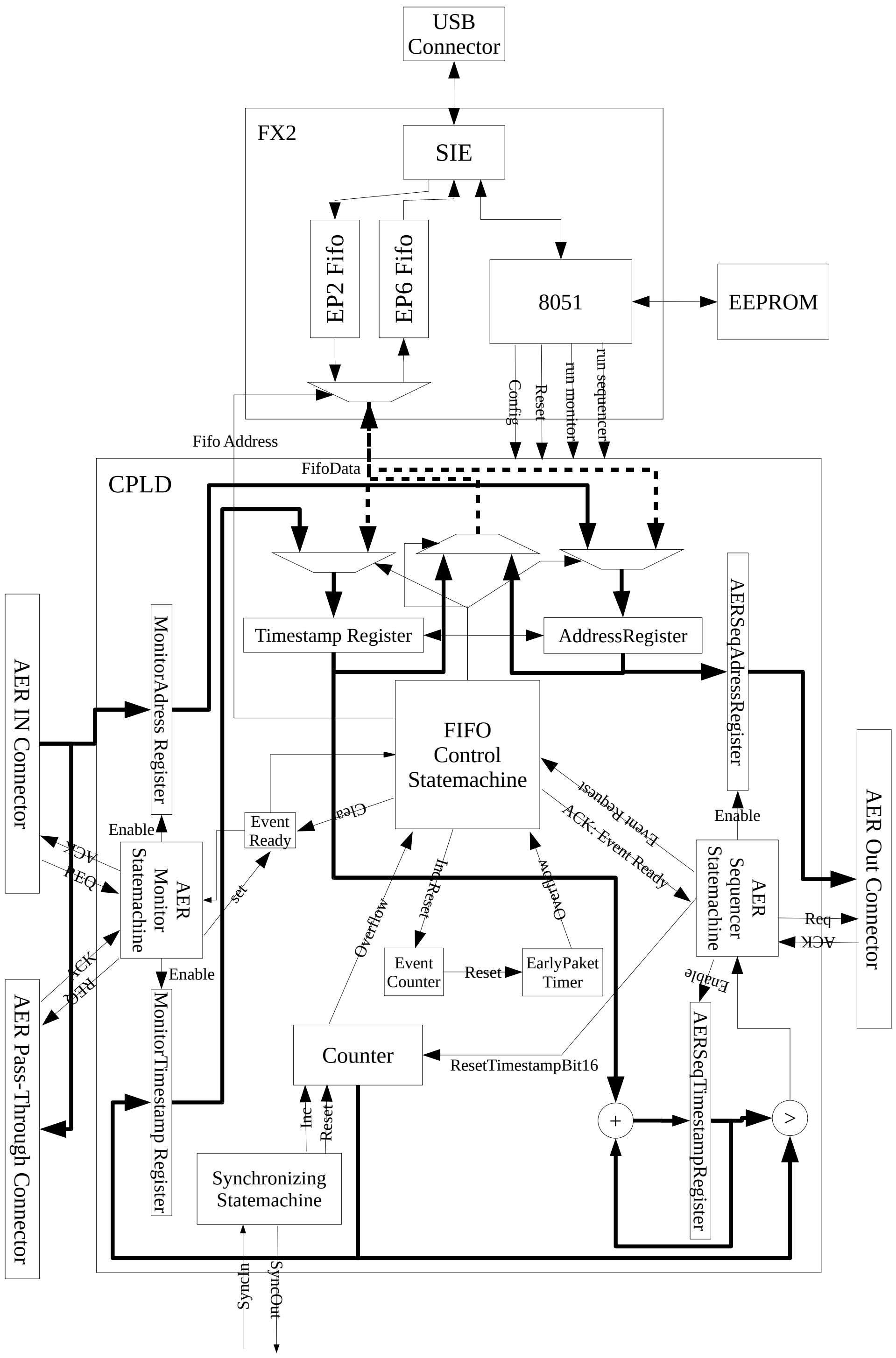


Figure 4.3: USBAERmini2 Device Overview

generate timestamps for the monitor and which serves as reference for the event sequencer. More information on timestamps can be found in 4.1.2.

The device is equipped with three AER ports, each consisting of a CAVIAR AER connector and a Rome AER connector. Each port is dedicated to a single purpose. There is a monitor port, a pass-through port whose address lines are directly connected to the monitor port, and a sequencer port. The reason why a design with three ports has been chosen, and pass-through and sequencer are not shared, is not mainly to be able to use pass-through and sequencer simultaneous (although it is possible), but ease to use. This way, the user doesn't have to set a jumper or send a command to the device to configure the desired mode.

The EEPROM holds the firmware for the 8051 microcontroller, USB descriptor tables and a device name string to identify various devices.

One goal was to make the board jumper-free, which was not achieved completely. The device can detect if a receiver is present on the pass-through port, but only while monitoring is active. When monitoring is inactive, request and acknowledge lines of monitor and pass-through port are connected directly, therefore transmission is blocked if no device is present and monitoring is not active.

4.1.2 Timestamp Issues

As bandwidth is crucial, it is desired to use as few bytes per event an as possible. The address consists of two bytes, but the width of the timestamp can be chosen within certain limits. Using only 16 bits is not enough, as only $2^{16} = 65536$ timestamps can be generated, which is equal to $65ms$ when using a tick of $1\mu s$. Using 32 bit timestamps consumes too much bandwidth, as the higher 16 bits will change only rarely. To preserve this bandwidth, a 15 bit counter is used on the device side, another 17 bits (called wrap-add throughout this report) are later added by the host software for monitored events. The 16^{th} bit of the timestamp sent to the host is used to signal an overflow of the counter. This approach has been chosen instead of using only one combination (for example 0xFFFF) to reduce complexity and size of the VHDL implementation.

For the sequencer, a similar approach could be implemented, but so far the sequencer is limited to interspike intervals up to $2^{16} - 1$.

4.1.3 The Functionality Implemented in the CPLD

The CPLD used in this device is a Xilinx Coolrunner 2 with 256 macrocells. Please refer to the datasheet [5] for details. It has an internal EEPROM, so no external components are needed to store the configuration.

The Fifo State Machine

The purpose of this FSM is to control the interaction with the two FX2 endpoint fifos used for this design, which means applying correct signals for reading events to sequence from EP2 and writing monitored events to EP6. When the timestamp

Pin	Name	Dir.	Description
23	Clock	in	
99	Reset	in	Reset (active low)
3,4,6-11,68 70-72,78-81	FifoData[0..15]	inout	Data lines to endpoint Fifos
1	FifoInFull	in	EP6 full (active low)
2	FifoOutEmpty	in	EP2 empty (active low)
13	FifoWrite	out	Write to selected Fifo (active low)
12	FifoRead	out	Read from selected Fifo (active low)
87	FifoOutputEnable	out	Enable Fifo output (active low)
82	FifoPktEnd	out	Commit short packet to SIE
85,86	FifoAddress	out	Select endpoint Fifo
14	SyncIn	in	Synchronisation input
15	SyncOut	out	Synchronisation output
91	RunMonitor	in	Enable monitoring
93	RunSequencer	in	Enable sequencing
95	TimestampTick	in	Select timestamp tick
96	TriggerMode	in	Trigger source (see A.6.1)
94	TimestampMaster	out	If device is acting as timestamp master
92	HostReset Timestamps	in	Set timestamp to zero, also on slave devices
90	Interrupt0	out	Microcontroller Interrupt 0, resets wrap add on host
89	Interrupt1	out	Microcontroller Interrupt 1, used to count missed events
97	PC1	in	Unused pin connected to 8051 port C.1
73,74,76,77	PE[0..3]	in	Unused pins connected to 8051 port E
16	LED	out	Control LED
42	AERMonitorREQ	in	Monitor request line
43	AERMonitorACK	out	Monitor acknowledge line
19,22,24,27-30 32-37,39-41	AERMonitor Address[0..15]	in	Monitor address lines
18	AERSnifREQ	out	Passthrough request line
17	AERSnifACK	in	Passthrough acknowledge line
66	AERSequencerREQ	out	Sequencer request line
67	AERSequencerACK	in	Sequencer acknowledge line
44,46,49,50,52-56,58-61,63-65	AERSequencer Address[0..15]	out	Sequencer address lines

Table 4.1: CPLD IO Pin Description

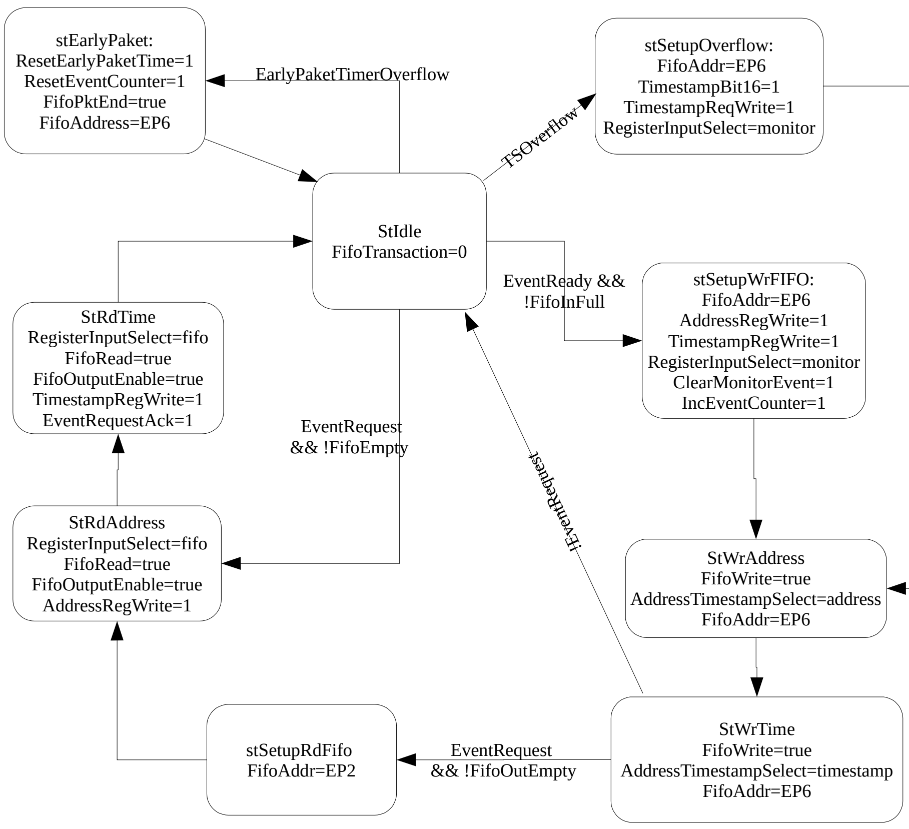


Figure 4.4: The Fifo State Machine, responsible for correctly handling Fifo transactions. Outputs are shown only when they differ from their default value shown in table 4.2. *StX* are the names of the states used in the VHDL code.

counter overflows, the Fifo SM sends an empty wrap event to the host to indicate the wrap-add should be incremented. As soon as the Short Packet Timer overflows, the Fifo SM applies the PacketEnd pin on the Cypress, signalling it to commit a packet to the SIE even if it is not full.

A state diagram of this state machine can be seen in figure 4.4, while table 4.2 shows the ports.

The Monitor State Machine

This FSM is responsible for handshaking with the sender and the potential receiver. It signals the Fifo SM when it has received an event, and can only receive a new event after the Fifo SM has acknowledged this event and sent it to the Fifo.

While waiting for an event, the Monitor SM detects if a receiving device is present by checking the acknowledge line on the receiver side, which is pulled low through a resistor. If a receiver is present, it will pull this line high, telling the

Signal	Direction	Description	Default Value
Clock	in		
Reset	in		
FifoInFull	in	EP6 full (active low)	
FifoOutEmpty	in	EP2 empty (active low)	
FifoWrite	out	Write to selected Fifo (active low)	1
FifoRead	out	Read from selected Fifo (active low)	1
Fifo OutputEnable	out	Enable Fifo output (active low)	1
FifoPktEnd	out	Commit short packet to SIE (active low)	1
FifoAddress	out	Select endpoint Fifo	EP2
AddressRegWrite	out	Write address register	0
TimestampRegWrite	out	Write timestamp register	0
RegisterInput Select	out	Select input to register (fifo or monitor)	0 (Fifo)
AddressTimestamp Select[0..1]	out	Select output to Fifodata lines	'Z'
MonitorEventReady	in	Monitor SM has event saved in its registers	
ClearMonitorEvent	out	Clear EventReady register	0
EventRequest	in	Sequencer SM requests event from fifo	
EventRequestAck	out	Event ready for sequencer	
FifoTransacation	out	Fifo SM not idle	0
IncEventCounter	out	Increment Event Counter	0
ResetEventCounter	out	Reset event counter	0
EarlyPaket TimerOverflow	in	Short packet timer overflow	
ResetEarly PaketTimer	out	Reset short packet timeout because packet has been sent	0
Timestamp Overflow	in	The 15 bit timestamp counter used for the monitor has overflown	
TimestampBit16	out	16^{th} bit of timestamp to send to host, indicates if event is valid or overflow event	0

Table 4.2: Port Description of the Fifo State Machine

Monitor SM its presence.

Depending on the presence of a receiving device, the Monitor SM will handshake only with the sender or with the sender and the receiver. The USBAERmini2 introduces a delay of approximately 100 to 133ns in the communication between sender and receiver.

When the FX2 signals that the EP6 fifo is full, the Monitor SM acknowledges events to the sender or passes them to the receiver, in order not to block the chain. Those events are lost, but every time an event is lost, the Monitor SM increases a four bit counter, whose MSB is connected to the interrupt 1 of the FX2.

Signal	Direction	Description	Default Value
Clock	in		
Reset	in		
Run	in	Enable Monitoring	
AERREQ	in	AER monitor request line (active low)	
AERACK	out	AER monitor acknowledge line (active low)	1
AERSnifREQ	out	AER passthrough request line (active low)	1
AERSnifACK	in	AER passthrough acknowledge line (active low)	
AddressRegWrite	out	Write address register	0
Timestamp RegWrite	out	Write timestamp register	0
EventReady	in	Event ready register, cleared by fifo SM	
Set EventReady	out	Signals the fifo SM that event is stored in registers	0
MissedEvent	out	connected to 8051 interrupt 1 to count missed events	0
FifoFull	in	active when EP6 is full	

Table 4.3: Port Description of the Monitor State Machine

The Sequencer State Machine

This very simple FSM is used to handshake with the receiving AER device and the Fifo SM. Figure 4.6 shows a state diagram.

The Sequencer SM has to deal with long handshake cycles, which could possibly

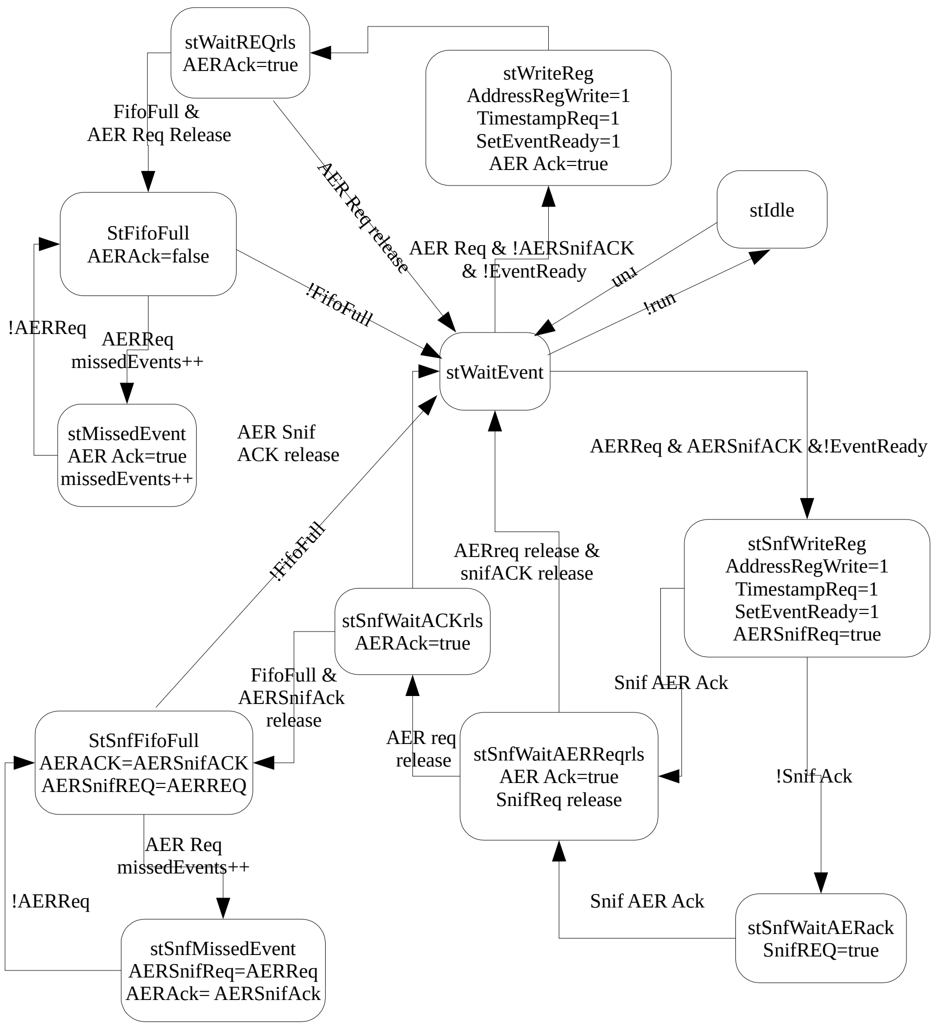


Figure 4.5: The Monitor State Machine. Outputs are shown only when they differ from their default value shown in table 4.3. *StX* are the names of the states used in the VHDL code.

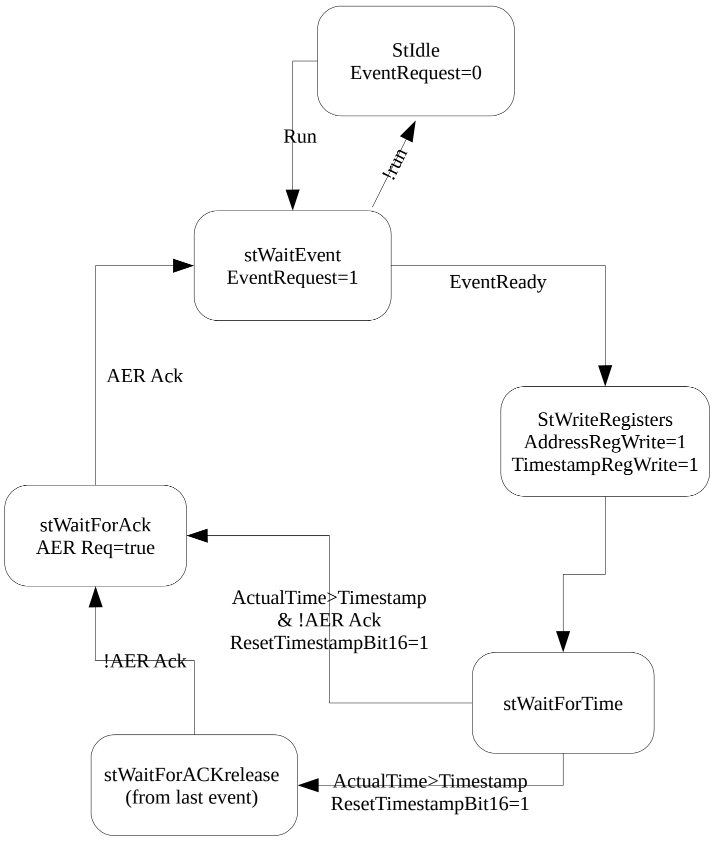


Figure 4.6: The Sequencer State Machine. Outputs are shown only when they differ from their default value shown in table 4.4. *StX* are the names of the states used in the VHDL code.

delay the next event, leading to $65ms$ breaks, as the FSM has to wait until the timestamp counter equals the timestamp of the event to sequence again.

It also has to take into account that the timestamp counter overflows regularly, so a simple “greater than” is not a suitable solution. However, if extended slightly, most of the problems can be solved. The “greater than” solution does not work either if the sum of the timestamp of the last event and the interspike interval (which is the timestamp of the new event) overflows or when the counter overflows between two events. For example if an event is sequenced shortly before an overflow and one almost immediately after this, but the handshake of the first takes so long that the counter overflows in the meantime, a nearly $65ms$ long break would occur. Otherwise, if the sum of the last timestamp and interspike interval overflows, the event will be released as soon as possible, as the counter is already greater than the new timestamp.

To solve these two problems, the CPLD internally uses 17 bits for the sum of the last timestamp and the interspike interval, and 17 bits for the timestamp counter. The 17^{th} bit of the counter is reset every time an event is released. This solves the problem entirely for one event immediately following a very long handshake, but unfortunately not for two or more events.

Therefore, when breaks of $65ms$ are observed in the sequenced spike train, this is due to a long handshake cycle during a timestamp counter overflow, followed by at least two events which were supposed to be sequenced before this overflow.

Signal	Direction	Description	Default Value
Clock	in		
Reset	in		
Run	in	Enable sequencing	
AERREQ	out	AER sequencer request line (active low)	1
AERACK	in	AER sequencer acknowledge line (active low)	
Equal	in	active when actual timestamp is equal or greater than timestamp for next event	
AddressRegWrite	out	Write address register	0
Timestamp RegWrite	out	Write timestamp register	0
ResetTimestamp Bit16	out	Reset timestamp counter bit 16 (see 4.1.2)	0
EventRequest	out	Request event from fifo	0
EventRequestACK	in	Event ready in fifo registers	

Table 4.4: Port Description of Sequencer State Machine

The sequencer implements the AER P2P protocol strictly, the address lines may change as soon as the request line is released, it does not wait until the acknowledge is released as well.

The Synchronisation State Machine

The synchronisation state machine is used to control the timestamp counter according to the actual selected timestamp tick and trigger mode.

In host trigger mode, the state machine starts increasing the timestamp counter as soon as the reset input is inactive, and sends synchronisation signals to potential slave boards. Two timestamp modes can be used, either a tick of one microsecond or a tick of one clockcycle, which is $33\frac{1}{3}ns$.

In slave trigger mode, event acquisition and sequencing starts only when the SI input goes high, i.e. when the master device is active.

As soon as the synchronisation state machine detects a timestamp master on the synchronisation input (i.e. the SI input goes high), it resets the timestamp, signals to the host by assigning interrupt zero and changes to slave mode.

The slave mode is different for the two timestamp ticks. In the slow mode, every increment of the timestamp tick is signalled on the synchronisation line by holding it low for two clockcycles. As soon as the line is held low longer than three clockcycles, the state machine resets the timestamp and switches to master mode (host trigger mode) or disables event acquisition and sequencing (slave trigger mode).

In the fast mode, the synchronisation line is always held high, the slaves increment their timestamp every clockcycle. As soon as the synchronisation line goes low, the slaves reset their timestamps and switch to master mode (host trigger mode) or disables event acquisition and sequencing (slave trigger mode).

The Timestamp Counter

The timestamp counter is basically a 17 bit up-counter, which is increased by the synchronisation state machine. It uses 17 bits to be able to deal with the long handshake problem described in the sequencer paragraph.

There is an overflow output, which is active for one clockcycle every time the 15 lowest bits overflow, to indicate an overflow to the fifo state machine, which then sends a wrap event to the host. This tells the host to increment its wrap-add.

The Short Packet Timer

This timer is needed to ensure that a packet is sent at least every eight milliseconds, even when the current fifo buffer is not full. This is for example to be able to display the events on the host in realtime when the event rate is very low.

A counter resets this timer every time a packet (128 events) is sent. As soon as the short packet timer overflows, the fifo state machine assigns the PacketEnd pin of the Cypress FX2, to commit the current buffer to the SIE.

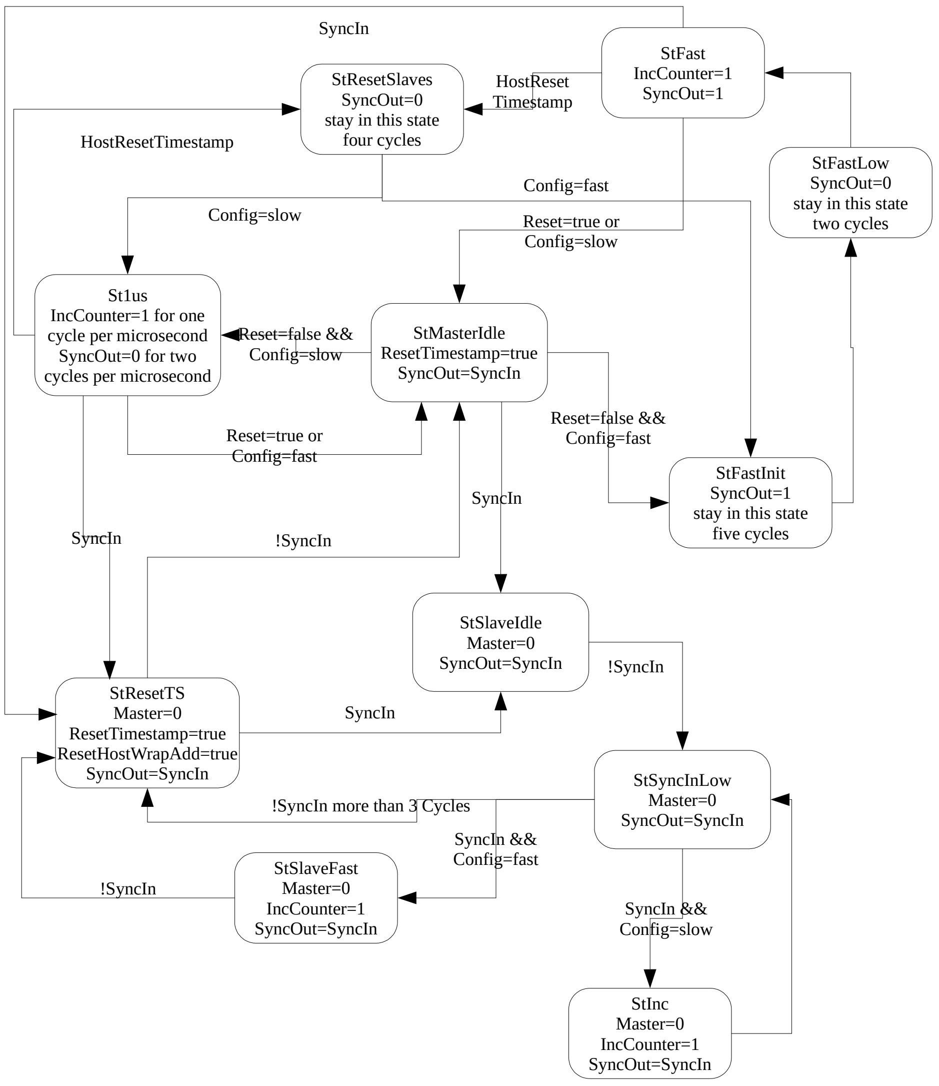


Figure 4.7: The Synchronisation State Machine, responsible for controlling the timestamp counter correctly. Outputs are only shown when they differ from their default value shown in table 4.5. *StX* are the names of the states used in the VHDL code.

Signal	Direction	Description	Default Value
Clock	in		
Reset	in		
Config	in	Sets timestamp tick	
HostReset Timestamp	in	Reset timestamp command from host, reset also slaves (ignored while in slave mode)	
SyncIn	in	Synchronisation input	
SyncOut	out	Synchronisation output	1
Master	out	device is acting in master mode	1
ResetTimestamp	out	Reset timestamp counter (active low)	1
ResetHost WrapAdd	out	Signal host to reset wrap add (microcontroller interrupt 0)	
Increment Counter	out	Increment timestamp counter	

Table 4.5: Port Description of the Synchronisation State Machine

4.1.4 Firmware

The firmware is based on the firmware frameworks supplied by Cypress. It is used to receive commands from the host and signal external timestamp resets back to the host. It is not involved in the communication between CPLD and SIE, because this would drastically reduce performance.

Commands are sent from the host to the device using vendor requests on endpoint zero. A list of all the implemented commands can be found in table 4.6.

Endpoint one is used as status pipe from the device to the host, but so far only one status message is implemented, which signals the host to reset its wrap-add. The message is sent when interrupt zero is assigned, which is the case when the device is in timestamp slave mode and the timestamp master signals it to reset its timestamps or the device switches either from master to slave or from slave to master mode.

Interrupt one is used to count the number of events that have been missed due to the fifo buffers being full. This interrupt is assigned by the CPLD every 16 missed events, and every time it is assigned, a counter is incremented. This counter can be read from the host by sending the corresponding vendor request. The counter is reset every time it is read.

Request Code	Command	Direction	Description	Parameters
0xA2	EEPROM	IN/ OUT	Read or write to EEPROM	value: address length: length
0xA3	RAM	IN/ OUT	Read or write to RAM	value: address length: length
0xB3	EnableAE IN	OUT	Starts event capturing	
0xB4	DisableAE IN	OUT	Stops event capturing	
0xBB	Reset Timestamps	OUT	Reset timestamp counter, also on slave devices	
0xD0	EnableAEOut	OUT	Enables event sequencing	
0xC1	DisableAEOut	OUT	Stops event sequencing	
0xC2	Set Device Name	OUT	Writes new serial number string to EEPROM	Data phase: new string
0xC3	Timestamp Mode	IN/ OUT	Sets or reads timestamp mode	OUT: value: mode
0xC4	Reset Fifos	OUT	Reset EP2 and EP6 Fifos	
0xC6	Enable AE	OUT	Starts event monitoring and sequencing	
0xC7	Disable AE	OUT	Stops event monitoring and sequencing	
0xCB	Timestamp Master	IN	Checks if device is timestamp master	
0xCC	Missed Events	IN	Gets estimation of number of missed events	

Table 4.6: Vendor Requests Implemented by the USBAERmini2 firmware

4.2 Host Software

The host software is based on the java interface of the general purpose USB driver by Thesycon [6]. This java interface provides a high level interface to USB devices, for example threads for reading or writing from or to a pipe.

The advantage of having java classes is that they are easily accessible from Matlab. See subsection A.6.1 for an introduction to the most important accessible java methods and refer to the javadoc [7] for a more complete list.

The USBAERmini2 is accessible as java class *CypressFX2MonitorSequencer*, which is a subclass of *CypressFX2*. The CypressFX2 class provides methods for basic handling of the device like opening, resetting timestamps, getting descriptors, writing firmware to EEPROM, etc. It also provides the functionality to acquire events from a device, but not for sequencing. This functionality is added by the subclass, as well as other functionality like changing operation mode.

Nearly all of the CypressFX2 class has been written by Dr. Tobias Delbruck as base class for a Silicon Retina, only a few adjustments and extensions had to be made.

A thread each is used to send events to the driver for sequencing and grab events from the driver for monitoring. Apart from those threads to send or receive events, the CypressFX2 class holds an additional thread listening for status messages on endpoint one. For the USBAERmini2, only one message is implemented so far, commanding the host to reset its wrap-add.

To get references to a device, the *CypressFX2MonitorSequencerFactory* class is used. This class builds a list of compatible interfaces, which so far are the USBAERmini2 and the USB2AERmapper described in chapter 5.

4.3 Performance

Performance has been tested with an ASUS S-presso host computer with a Pentium4 3GHz CPU and 1 GB of RAM. The performance varies with the host computer, because the device relies on the host computer on emptying the fifos fast enough. The matlab scripts described in section A.7 have been used.

4.3.1 Monitoring

A USBAER has been used as sequencing device. Picture 4.8 shows the test setup for measuring the performance using two monitoring devices simultaneously.

The performance depends highly on the number of buffers and the size of these buffers the USBIO driver uses. Figure 4.9 shows the monitoring performance of one board versus buffer size and number. Based on this test, a the size of the buffers have been chosen to be 8192 bytes, the number of buffers 4.

Table 4.7 shows the resulting monitoring performance. For each test, event capturation was turned on between half a second and a second and each test as

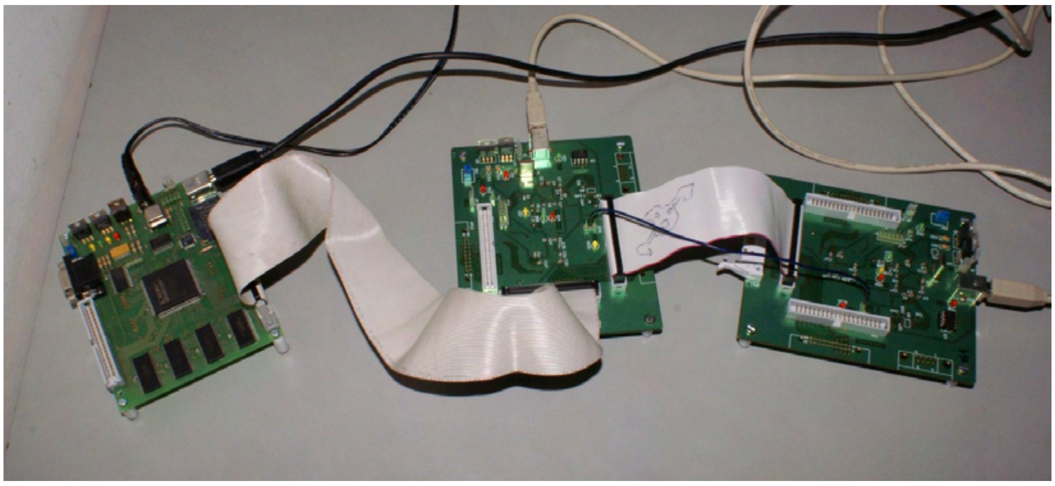


Figure 4.8: Test setup used to measure monitoring performance. On the left the USBAER which is used for sequencing, on the right two USBAERmini2 boards.

been carried out several times. The results were varying slightly, usually in the order of ten percent.

The peak rate of the monitor is five megaevents, which can be reached for a few thousand events.

Test Setup	Megaevents per second per per device	Megabits per second per per device	Megabits per second total
one device	4.5	144	144
two devices	2.5	80	160
three devices	1.65	53	159

Table 4.7: USBAERmini2 monitoring performance

4.3.2 Sequencing

A USBAER has been used as monitoring device.

The sequencer can reach a peak rate of 3.75 megaevents per second for several ten thousand events. The average rate reached while logging with the USBAER mapper was only slightly lower, as can be seen in table 4.8. But as the USBAER mapper can only log half a megaevent, the testing duration was only about $\frac{1}{8}$ of a second. The performance may be slightly lower for longer sequences.

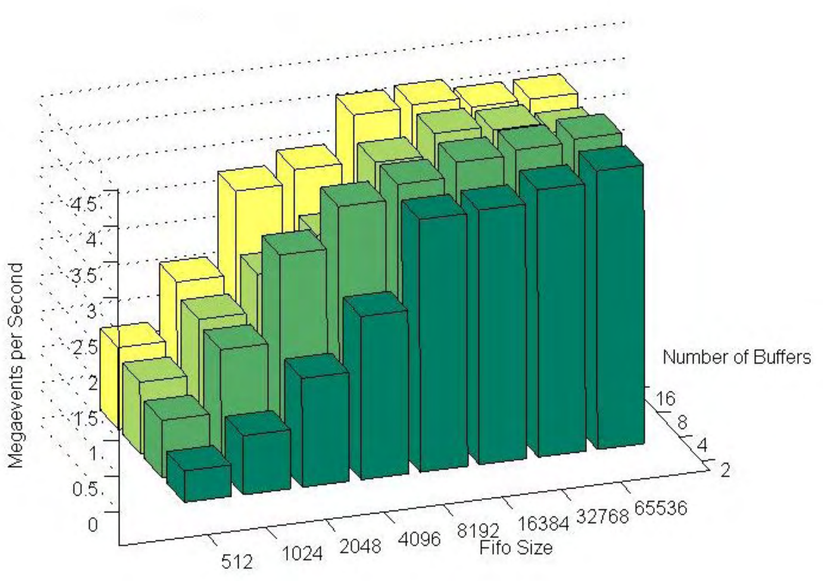


Figure 4.9: Monitoring performance versus buffer size and number of buffers

Test Setup	Megaevents per second	Megabits per second
one device	3.69	118

Table 4.8: USBAERmini2 sequencing performance

4.3.3 Monitoring and Sequencing

Table 4.9 shows the resulting performance for sequencing and monitoring simultaneously. The sequencing device was configured to sequence continuously. Event capturation was turned on between half a second and a second and each test as been carried out several times. The results were varying slightly, usually in the order of ten percent.

4.4 Challenges, Problems

This section describes the most important challenges and problems that had to be faced during this diploma project. The list is not sorted by relevance or order of appearance.

- Due to lack of experience and also lack of communication with the advisors, planning and specification were not complete from the beginning, and some functionality was added to the specifications during the development process.

Test Setup	Megaevents per second per device	Megabits per second, per monitoring device	Megabits per second, total, both directions
one device, acting as sequencer and monitor	2.48	79	158
two devices one monitor, one sequencer	2.7	86	172
three devices one sequencer, two monitors	2.1	67	201

Table 4.9: USBAERmini2 monitoring and sequencing performance

This includes different requirements on timestamp tick from Institute of Neuroinformatics and Architectura y Technologia de Computadores.

- Finding a suitable way to deal with the timestamps was a one of the main problems. The solution had to be of low complexity, allowing long interspike intervals and using no more than two bytes of data sent to the host for each event.
- A similar challenge was to deal with long handshakes, which delay the sequencing state machine as described in subsection 4.1.3.
- The chosen CPLD is rather small for the demanded functionality, therefore the functionality had to be optimised regarding complexity, as compiling the VHDL with respect to size did not meet the timing requirements.
- The chip used turned out to be the Cypress FX2 instead of the FX2LP which was ordered. The FX2 is an older version with less memory and higher power consumption. Fortunately this didn't cause any problems except for the debugger and higher power consumption. The higher power consumption however could lead to problems, because the device now draws more than 100mA during enumeration, which violates the USB specification. Some hosts may therefore disable the device.
- The debugger interface only worked with the development board from Cypress, holding a FX2LP.
- The FX2 firmware was very touchy, small changed sometimes caused the device to stop working, or other functionality would cease to work. But most of the problems were probably due to a misconfiguration of the memory map, which could have caused overlap between code and data memory.

- The setup times on some of the FX2 fifo interface pins are very tight, therefore the clock frequency of the CPLD had to be reduced, because it was impossible to satisfy the setup times. The implementation could not be changed to improve speed, due to the small CPLD.

4.5 Errata

The synchronous resetting of the timestamps on slave devices does not work reliably, when the user requests a reset (for example by pressing the zero key in CaviarViewer).

If an event is received very shortly (a few clockcycles) before an overflow of the counter and therefore has a very high timestamp, it can happen that the wrap event is placed before this event in the fifo. This results in final timestamp which is 2^{15} to big. This could be corrected on the host side by comparing it to the next event's timestamp and subtracting 2^{15} if it is bigger.

4.6 Suggestions to Improve the Design

There are also several (mostly minor) drawbacks of the actual implementation of the USBAERmini2 board. None is very grave, but if it is ever redesigned, the following suggestions should be considered.

- Run the CPLD and the Fifo Interface at 48MHz by the clockout pin of the FX2 as done in the USB2AERmapper. This way, the tight setup constraints of the FX2 fifo interface pins could probably be met.
- Implement a fifo queue for the sequencer, to solve problems when sequencing with a slow receiver just before a timer overflow and speed up communication with the Cypress fifo. This would need a bigger CPLD.
- Implement a sniffer mode which does not affect AER transmission.
- Include overflow packet circuitry for the sequencer, so that bigger than $2^{16}-1$ interspike intervals can be generated.
- Implement suspend mode to agree with USB specification. This would need another voltage regulator for the CPLD core which can either be switched off or does not need a minimum load current of several milliamps.
- Add LEDs which indicate AER activity.
- Add option to either block the transmission or throw away events while fifos are full.
- Rearrange the AER connectors, because they are not placed and oriented very convenient.

- Upgrade to Cypress FX2LP.
- Implement the possibility to program the CPLD from the Cypress, allowing the users to update the CPLD firmware without needing a Xilinx download cable.
- Use either interspike intervals or absolute timestamps for both monitoring and sequencing, not interspike intervals for sequencing and absolute timestamps for monitoring. The best way would probably be to use absolute timestamps, in order to maintain compatibility with the CaviarViewer application. Combined with overflow packet circuitry for the Sequencer, the long handshake problem could be solved easily as well.
- Implement also the SCX version (multi-sender) of the AER protocol.

Chapter 5

The USB2AERmapper

The second part of this diploma thesis describes the design of a new USBAER device, combining the flexibility of the known USBAER designed at Architectura y Technologia de Computadores with the throughput of USB2.0 and adding even more flexibility by providing additional interfaces.

5.1 Architecture Overview

Figure 5.3 shows a block diagram of the architecture and figures 5.1 and 5.2 show a picture of the top and the bottom side of the USB2AERmapper. The device consists of the following main blocks.

- Cypress FX2
- Xilinx Spartan3 FPGA
- Four Silicon Laboratories Cy7C1049CV SRAM modules, total 2MB
- EEPROM to hold the Cypress FX2 firmware.
- Several Interfaces described in section 5.2.

The Cypress FX2 is described in chapter 3.

The FPGA used is a Xilinx Spartan-3 XC3S400 in a PQ208 package, offering 400000 system gates and up to 141 IO pins. Please refer to the datasheet [8] for more information. The FPGA is clocked from the Cypress FX2 CPU clockout pin with $48MHz$, which can be multiplied internally by the Digital Frequency Synthesiser as follows

$$f_{CLKFX} = f_{ClkIn}\frac{ClkFxMultiply}{ClkFxDivide}, \tag{5.1}$$

where $ClkFxMultiply$ is any integer form 2 to 32, while $ClkFxDivide$ is an integer from 1 to 32. Tables 5.1 and 5.2 show the pin assignment of the FPGA. For more details, refer to the schematics in `CAVIAR\wp5\USBAER\INI-AE-Biasgen\doc\`. The

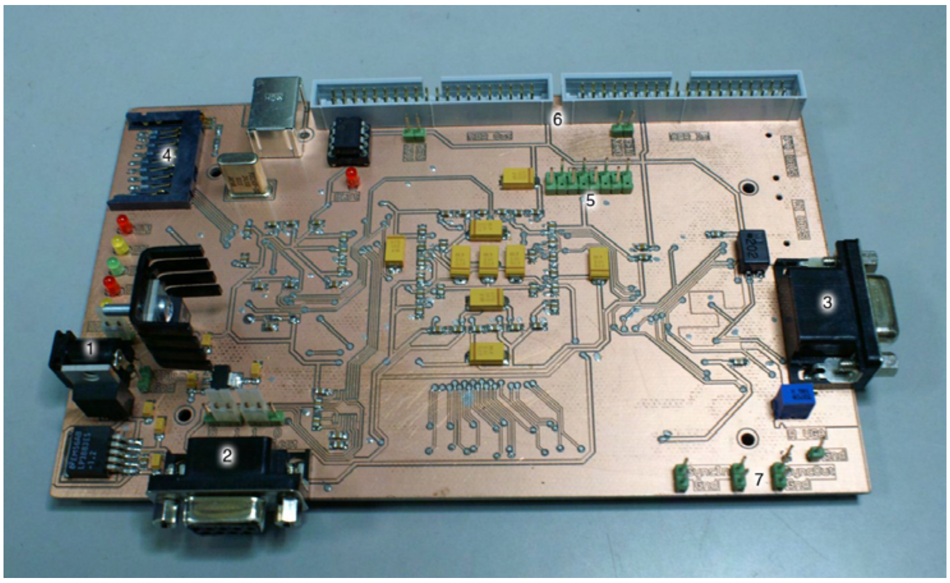


Figure 5.1: Top side of the USB2AERmapper. 1. Power Supply Connector. 2. FX2 Serial Interface Connector, used for debugger. 3. VGA port. 4. MMC card reader. 5. JTAG connector for FPGA. 6. Parallel AER connectors. 7. Synchronisation pins.

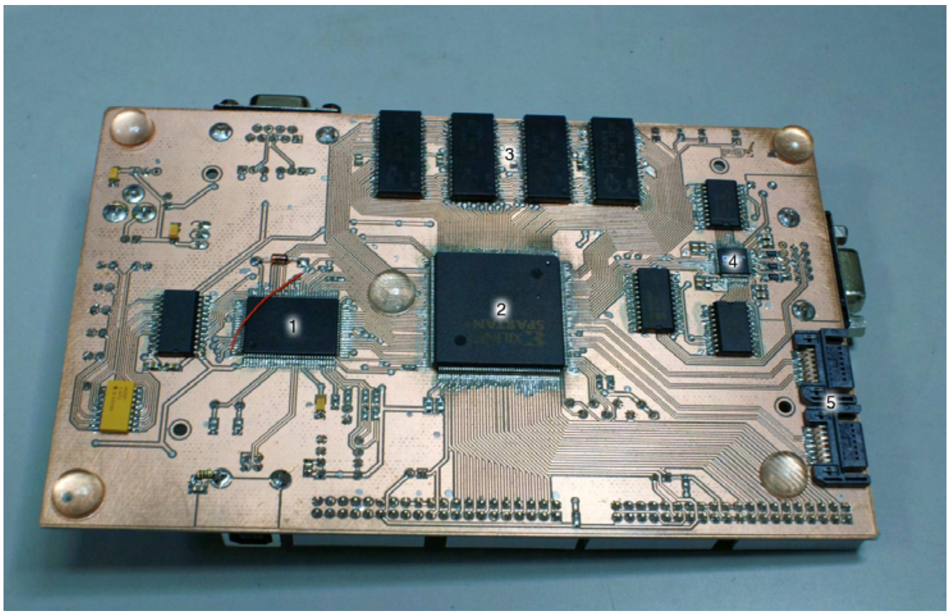


Figure 5.2: The bottom side of the USB2AERmapper. 1. Cypress FX2. 2. Xilinx Spartan3 FPGA. 3. Memory. 4. VGA DAC. 5. Serial AER connectors.

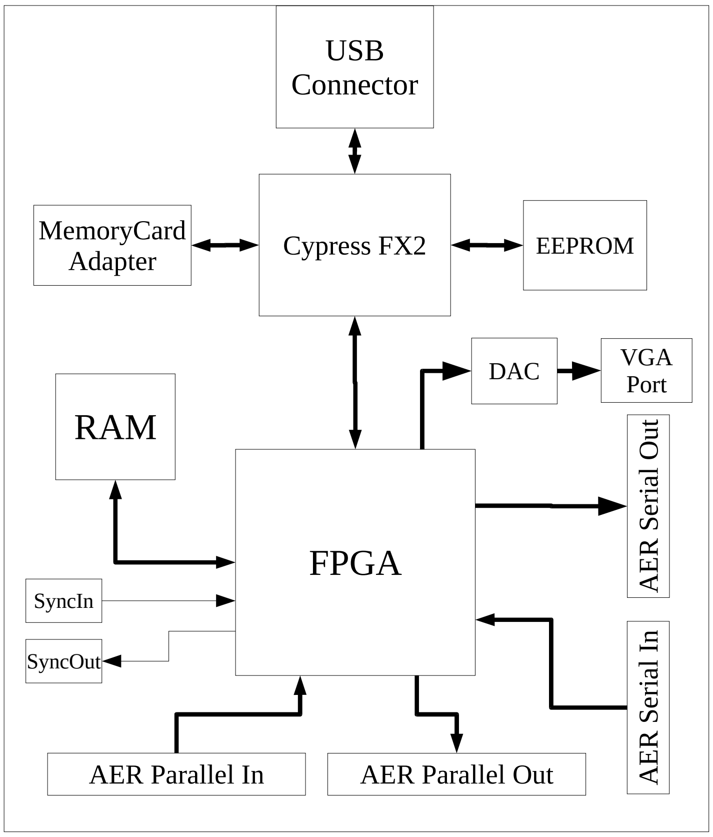


Figure 5.3: Block diagram of the USB2AERmapper

parallel and the serial AER ports make use of the DCI (*D*igital *C*ontrolled *I*mpedance) feature of the Spartan3, which reduces reflections of the signals. In combination with carefully routed impedance matched lines, this improves signal integrity for high speed signals.

The SRAM is connected to the FPGA and uses a wordwidth of 32 bits and 19 address bits, therefore implicating 512kWords.

In contrast to the USBAERmini2, the USB2AERmapper is not powered by the USB bus, it needs power supply input between 6 and 9 Volts.

5.2 Interfaces

The USB2AERmapper provides a variety of interfaces, which can all be used simultaneously [1], thereby offering a great flexibility for future uses of the AER bus. The following list describes the available interfaces.

USB2.0 Provided by the Cypress FX2

Parallel AER IN 40 Pin IDE Connector. 16 address lines, one request line and one acknowledge line. Refer to the CAVIAR AER specification [2] for pin layout.

[1]Assuming the FPGA provides enough logic gates, which depends on the complexity of the design and the VHDL implementation.

Pin	Name	Description
2-71 197-205	Mem_D[0..31] Mem_A[0..18]	Memory data and address lines [1]
204,18,29,63	Mem_WE[0..3]	Memory write enable [1]
46	Mem_OE	Memory output enable
72	PktEnd	Cypress fifo packet commit strobe
74	IFClock	Cypress fifo interface clock
76,77	FifoAddr[0..1]	Cypress fifo address
78	SLOE	Cypress fifo read and output enable strobe
79	Clock	Clock input, driven by Cypress clock output
85	PC4	IO pin connected to 8051 port C.4
81	Busy/PC3	Busy pin, connected to port C.3[2]

[1] See schematic for exact pin locations.
[2] Can be used as IO pin after FPGA programming.

Table 5.1: FPGA IO Pin Description (part 1)

Pin	Name	Description
83	Init/PC2	Init pin, connected to port C.2[2]
85	PC1	IO pin connected to 8051 port C.1
86	EP6full	Cypress endpoint 6 fifo state
87	EP2empty	Cypress endpoint 2 fifo state
90	PC0	IO pin connected to 8051 port C.0
92	DIN/FifoData7	Programming pin, connected to fifo data line 7, port B.7 [2]
93-101	Fifodata[0..6]	Cypress fifo data lines [1]
102	SLWR	Cypress fifo write strobe
106	VRN_3	Bank 3 DCI configuration resistor (110Ω)
107	VRP_3	Bank 3 DCI configuration resistor (110Ω)
108	REQ_OUT	AER parallel out request line
109-130	AEROUT[0..15]	AER parallel out address lines[1]
131	ACK_IN	AER parallel in acknowledge line
132	INT0	Cypress interrupt 0
133	ACK_OUT	AER parallel out acknowledge line
135	REQ_IN	AER parallel in request line
137-156	AEROUT[0..15]	AER parallel out address lines [1]
161	VRN_1	Bank 1 DCI configuration resistor (120Ω)
162	VRP_1	Bank 1 DCI configuration resistor (120Ω)
165	CP1	VGA Buffer write enable[3]
166	VGACLOCK	VGA D/A converter clock [3]
167	Aux1	IO pin connected to 8051 port A.3 [3] Can only be used as input
168,169	SAER_out_Cx	Serial AER out clock
171,172	SAER_out_Dx	Serial AER out data
175	VGA_VSYNC	VGA vertical sync signal [3]
176,178	SAER_in_Dx	Serial AER in data
180,181	SAER_in_Cx	Serial AER in clock
182	VGA_HSYNC	VGA horizontal sync signal [3]
183-190	VGA[2..7]	VGA data lines [1]
191	CP2	VGA Buffer write enable
194	SyncOut	Synchronisation output
196	SyncIn	Synchronisation input

[1] See schematic for exact pin locations.
[2] Can be used as IO pin after FPGA programming.
[3] Connected via buffer, because Bank 1 uses 2.5V supply voltage for LVDS ports.

Table 5.2: FPGA IO Pin Description (part 2)

Parallel AER OUT 40 Pin IDE Connector. 16 address lines, one request line and one acknowledge line. Refer to the CAVIAR AER specification [2] for pin layout.

Serial AER IN Serial ATA Connector. Two differential pairs using LVDS signalling.

Serial AER OUT Serial ATA Connector. Two differential pairs using LVDS signalling.

VGA Analog Devices D/A converter, 6 bits per channel, allowing 262144 colours or 64 grey-levels. Two buffers are used to temporally store the information for the red and the blue channel to reduce the number of pins needed.

MMC MemoryCard Reader Connected to microcontroller.

Synchronisation Pins Can be used to synchronise timestamp counter with other USB2AERmapper or with USBAERmini2, to enable synchron event capturing from several boards. One output and one input pin, and an additional pin which is connected to the input pin, for the creation of daisy chains.

See table 5.3 for a comparison between the interfaces provided by USBAER and USBA2ERmapper.

Interface type	USBAER	USB2AERmapper
USB	1.1 (12Mbit/s)	2.0 (480 Mbit/s)
Parallel AER	yes	yes
Serial AER	no	yes
VGA	yes, without DAC, 1 bit per channel	yes, with DAC, 6 bits per channel
MemoryCard Reader	MMC	MMC
Synchronisation pins	no	one input pin, one output pin

Table 5.3: Comparison of the interfaces provided by USBAER and USB2AERmapper.

5.3 Monitor/Sequencer Implementation

As basic application a functionality similar to the USBAERmini2 is presented, which shows how the communication between Cypress FX2 and FPGA can be implemented.

The implementation is very similar to the USBAERmini2, therefore this section describes only the differences between the implementations. Figure 5.4 shows a block diagram of the implementation.

5.3.1 The FPGA

In contrast to the CPLD of the USBAERmini2, the FPGA is run at 48 MHz, clocked by the ClockOut pin of the FX2 and the Cypress FX2 slave fifo interface is clocked externally by the FPGA, to overcome timing problems described in section 4.4. This also adds the possibility to decrease the interface clock frequency with the digital frequency synthesiser, if timing requirements can not be met.

The main difference to the USBAERmini2 implementation is the additional Configuration State Machine, which is described below. This state machine is necessary because there are less lines connecting the FX2 microcontroller with the FPGA. The Configuration State Machine overcomes this problem by interpreting the FifoData lines as configuration data, when told to do so by an additional input. See figure 5.5 as an illustration on how the FX2 and the FPGA are connected.

Due to the limited number of pins of the FPGA, the fifo databus is only 8 instead of 16 bits wide. Due to this, there is a four-to-one multiplexer instead of a two-to-one multiplexer from the fifo registers to the fifo data bus. When reading from the fifo, the data goes to a 8 bit delay register, whose output is then combined with the actual fifo input to provide a full valid two-byte word to the input of the registers. See figure 5.6 for illustration. Both these changes are not shown in figure 5.4.

The Fifo State Machine has two additional states for reading and writing each, because of the smaller fifo databus width, four cycles are now needed to read or write a complete event. Also an additional output had to be added to control the multiplexer for the fifo databus, selecting either LSB or MSB.

As the USB2AERmapper has only two AER ports instead of three like the USBAERmini2, there is a multiplexer for selecting either the sequencer or the passthrough as output.

The Synchronisation State Machine had to be adjusted to the change of the clock frequency, to ensure the slow timestamp mode is one microsecond and that potential slave USBAERmini2 are synchronised correctly.

Due to the reduced number of connections between FPGA and FX2, only one interrupt sourced from the FPGA is available. Therefore there is no missed events counter, which means that the device blocks transmissions when all fifos are full, until one buffer gets available. Therefore, in the Monitor State Machine, everything referring to the counting of missed events has been deleted, namely the states *stFifoFull*, *stMissedEvent*, *stSnfFifoFull* and *stSnfMissedEvent* and the output *MissedEvent*.

Additionally, the device does NOT recognise if a receiving device is attached when working only in monitor/passthrough mode. Therefore, if there is no receiving device, the request and the acknowledge line of the output have to be connected by

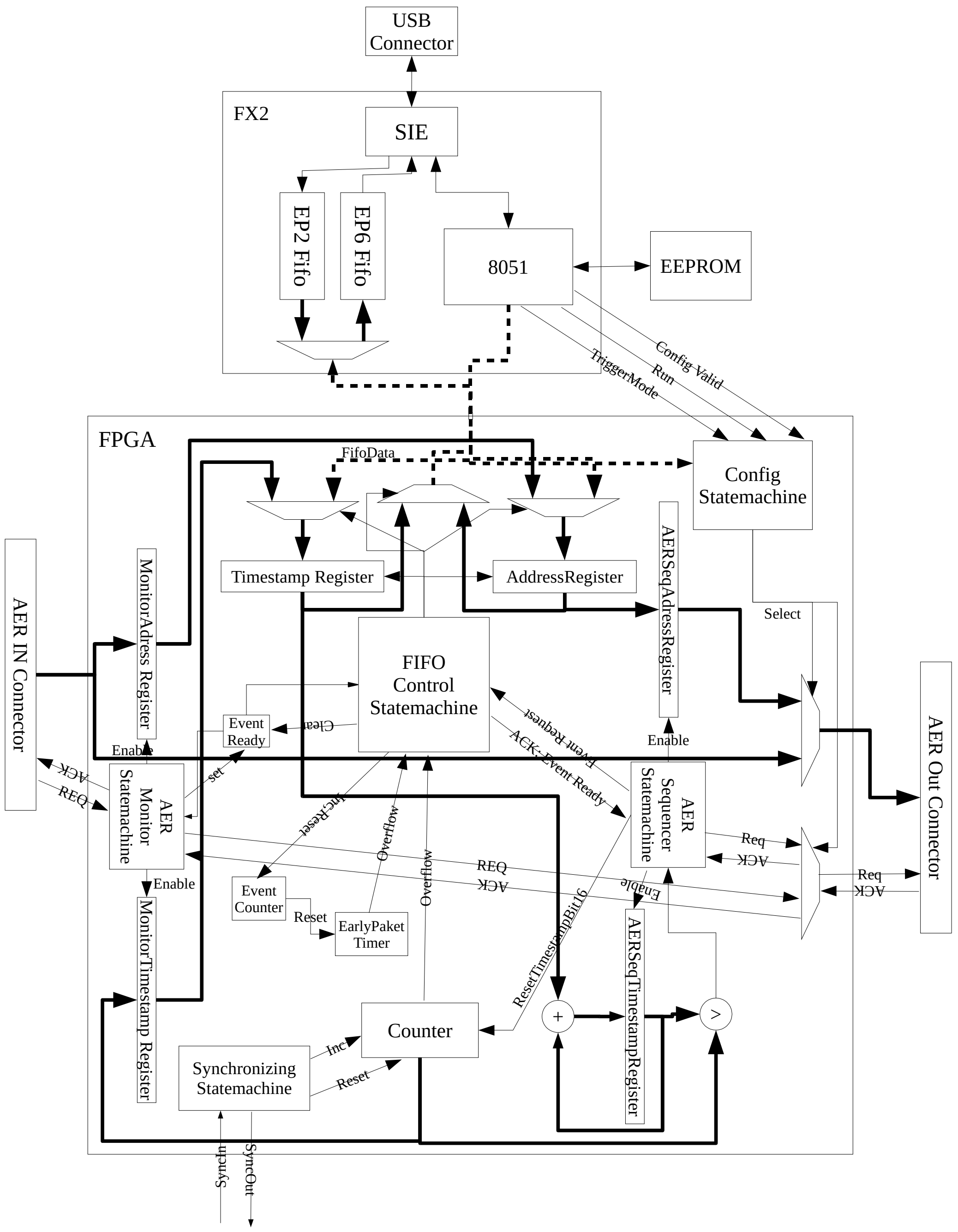


Figure 5.4: USB2AERmapper Monitor/Sequencer Overview. Main differences to USBAERmini2: Configuration state machine, which reads *FifoData* input as configuration byte, when *ConfigValid* is active. Multiplexer for AER out port. FifoData bus width only 8 bits instead of 16. For simplicity, not all signals are showed.

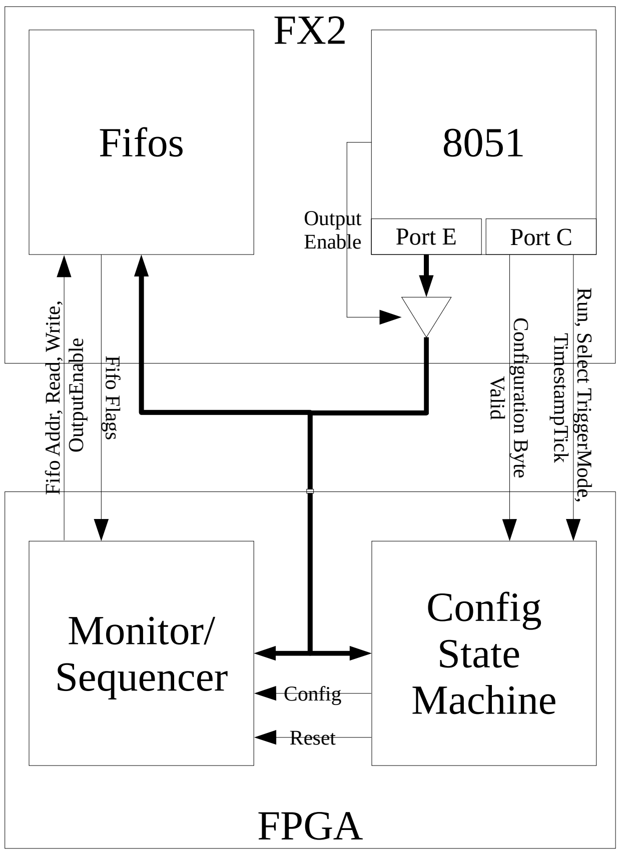


Figure 5.5: Communication between microcontroller and FPGA

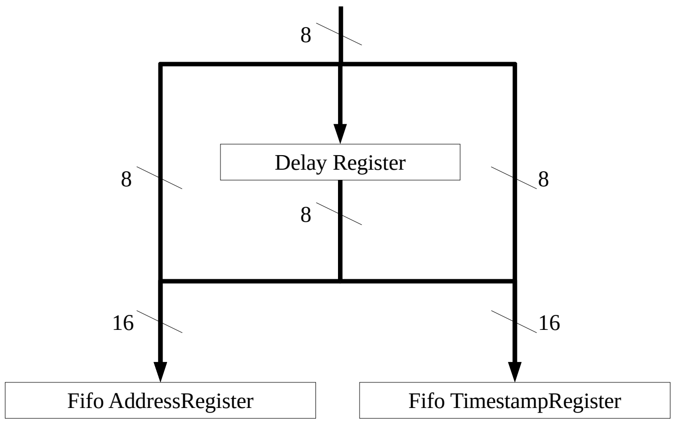


Figure 5.6: 8 bit delay register for the fifo lines

a jumper as shown in figure 5.7.

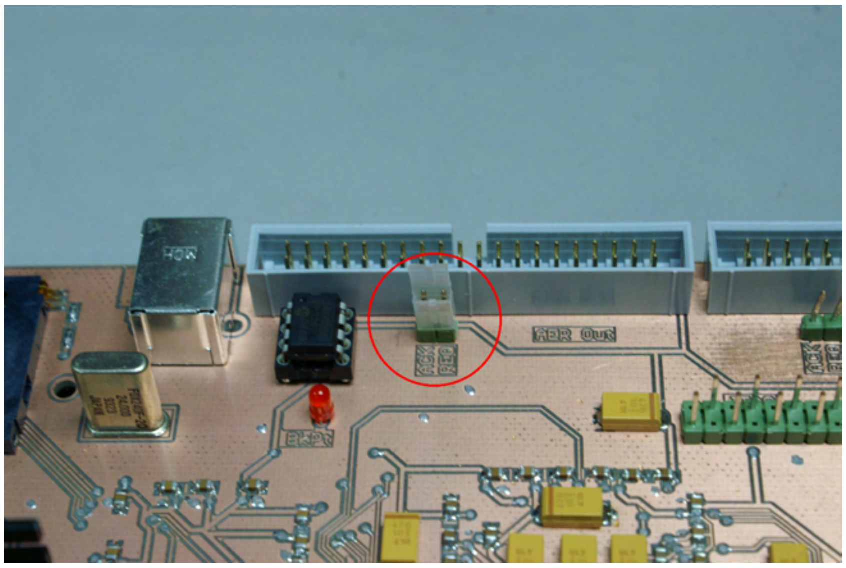


Figure 5.7: Connect request and acknowledge line of the AER out port, when monitoring without receiving device.

The rest of the implementation, including Sequencer State Machine and Short packet circuitry, is not changed at all.

Configuration State Machine

As there are only seven lines between the microprocessor and the FPGA, the configuration state machine is used to read port E of the FX2, whose lines are shared with the fifo data bus, as configuration byte, when it is signalled to do so.

Figure 5.8 shows a state diagram of this state machine, while table 5.4 shows the input and output ports.

5.3.2 The Firmware

Due to the fact that various configuration registers are reset when changing from port mode to slave fifo mode (see subsection 3.6.3), the `IFCONFIG` register is not changed for supplying the configuration byte, instead the fifo data lines are also connected to port E of the microprocessor. This port is unfortunately not bit-addressable, and the connections are crossed to avoid crossing the lines on the PCB. See table 5.5 for details.

To set the configuration byte, the firmware has to make sure that the FPGA doesn't drive the fifo lines, enable port E as output, set it accordingly and then apply the ConfigValid output. This is done by the following code section.

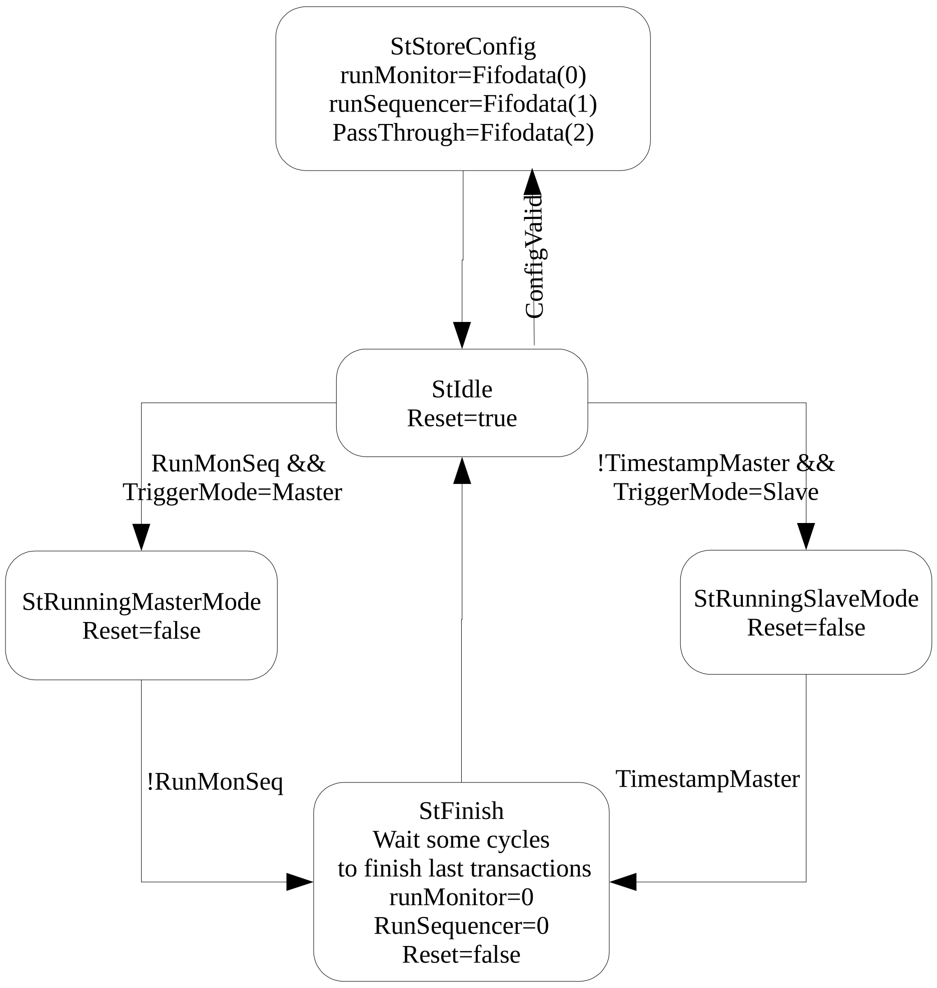


Figure 5.8: The Configuration State Machine. Outputs are shown only when changing. *StX* are the names of the states used in the VHDL code.

Signal	Direction	Description	Default Value (Outputs)
Clock	in		
RunMonSeq	in	Monitor and/or Sequencing is enabled when high	
FifoData[0..7]	in	Configuration Byte	
ConfigValid	in	High when the FifoData Byte is a valid configuration Byte	
Trigger Mode	in	Whether Monitoring/ Sequencing is enabled by host or timestamp master (see A.6.1)	
Timestamp Master	in	Whether device is acting as Timestamp Master	
RunMonitor	out	Enable Monitoring	0
RunSequencer	out	Enable Sequencing	0
PassThrough	out	Selects AER Output: Sequencer or PassThrough	1 (PassThrough)
Reset	out	Resets State Machines and Registers (active low)	0 (active)

Table 5.4: Port Description of the Configuration State Machine

Fifo Line	Port E pin	Configuration Signal
0	7	Run Monitor
1	6	Run Sequencer
2	5	Enable PassThrough
3	4	
4	3	
5	2	
6	1	
7	0	

Table 5.5: Connections from Port E to Fifo Data lines

```
RUN_MONSEQ =0;

OEE=0xFF;
IOE=0x00;
IOE |= SEQUENCER;
IOE |= MONITOR;

CONFIG_VALID=0;
CONFIG_VALID=1;
CONFIG_VALID=0;

OEE=0x00;

RUN_MONSEQ =1;
```

To download the firmware to the FPGA, the firmware first has to change the configuration from slave fifo mode to port mode, as the FPGA programming pin is connected to *FifoData7*. After the initialisation of the programming, the FPGA firmware is sent from the host to the FX2 in chunks of 64 bytes, which are then serialised by the FX2 firmware. The end of the programming file is signalled by the host by setting a flag in the setup data packet. After successful programming, the firmware changes back to slave fifo mode and sets the registers which have been reset due to the change of the `IFCONFIG` register.

Otherwise, the Cypress FX2 firmware of the USB2AERmapper is similar to the firmware of the USBAERmini2.

5.3.3 Host software

The USB2AERmapper is accessed from the host by the CypressFX2Mapper class, which is a subclass of the CypressFX2MonitorSequencer class described in section 4.2. This subclass adds a method for downloading the firmware for the FPGA to the device.

5.3.4 Performance

Average performance is similar to USBAERmini2, as it is not limited by the device, but by the host. Peak performance is slightly higher due to the increased FPGA clock frequency.

5.4 Errata

Please note that the FPGA DIN pin (the pin which is used to program the FPGA), which is supposed to be connected to a pin of port C of the Cypress FX2, is connected to FifoData7 due to an error in the schematics. This is not a grave error, it is still possible to program the FPGA with the microcontroller by changing the `IFCONFIG` register from slave fifo to port mode. Nevertheless, it would be more convenient to correct this error on a future PCB.

5.5 Outlook

Before the PCB is fabricated, the schematics should be changed in a way that the FPGA programming pin (DIN) is connected to a pin of port C instead of *FifoData7*.

To be able to use current USBAER FPGA firmware with the USB2AERmapper, a “Cygnal emulator” has to be created, which emulates the communication with the Cygnal microcontroller of the USBAER. This involves a VHDL module and changes in the Cypress FX2 firmware. With such an emulator, an existing FPGA firmware can easily be ported.

To support current USBAER applications, the CypressFX2Mapper java class has to be extended with methods to send configuration bytes to the FPGA and to write and read the RAM.

Chapter 6

Conclusion

This diploma thesis presented two AER interfaces which provide realtime capturing and sequencing of AER data through the Universal Serial Bus. While one, the USBAERmini2, is buspowered and therefore highly portable, the USB2AERmapper offers high flexibility with an FPGA and several types of interface.

The USBAERmini2 has been successfully used during a CAVIAR workshop which took place in February 2006 at the University of Seville. Various sets of synchronised AER data of the complete system have been captured and can now be used by the project partners to continue their work on the modules.

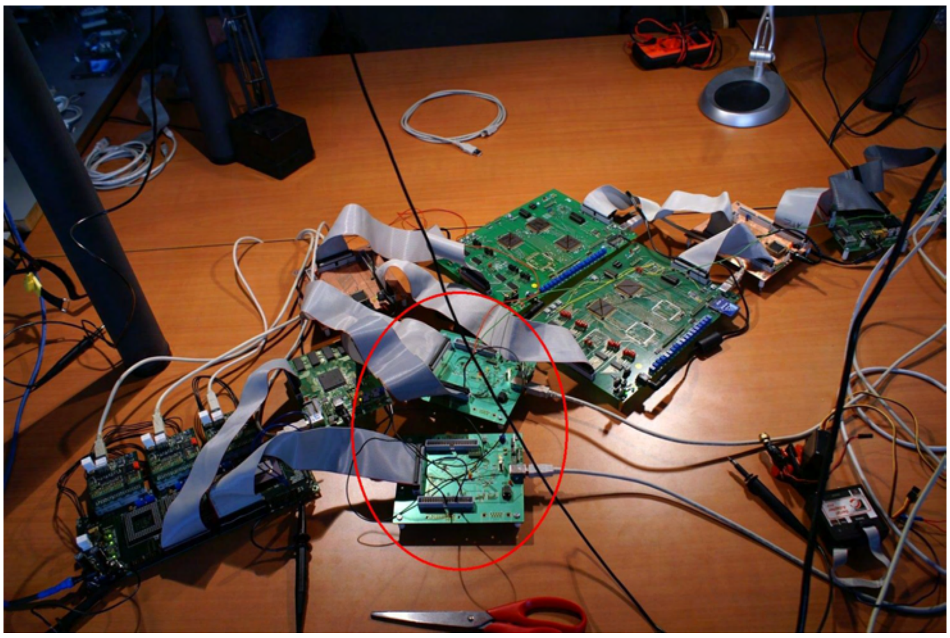


Figure 6.1: Part of the system which was set up during the CAVIAR workshop during February 2006 in Seville. In the centre, two USBAERmini2 can be seen.

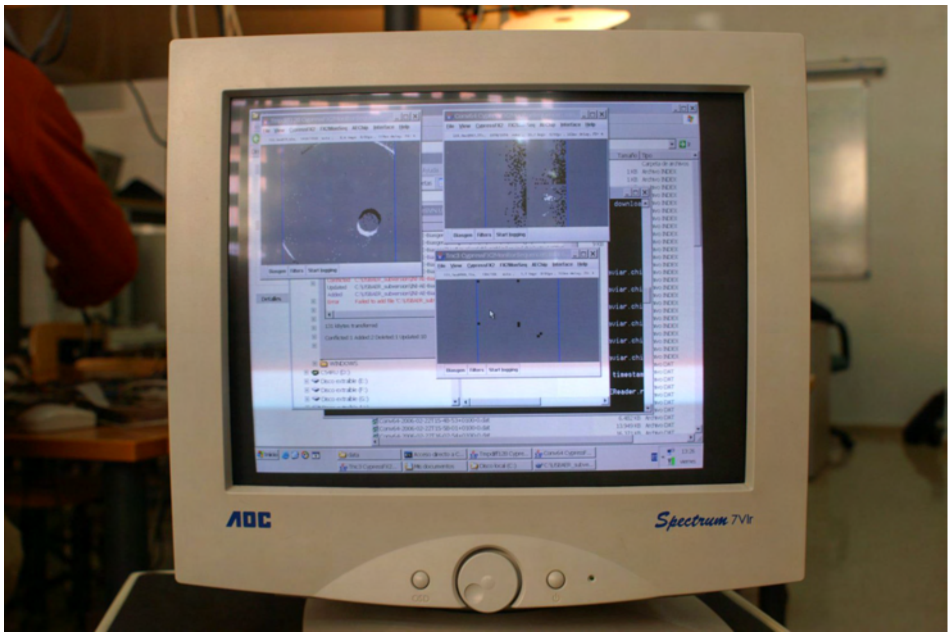


Figure 6.2: Realtime displaying of AER data with three synchronised USBAER-mini2 during CAVIAR workshop. The top left window shows the output of the retina, the top right displays the four convolution chips, while the bottom windows shows the output of the winner-take-all chip.

During the workshop, several participants started working with the board and the USBAERmini2 is now in daily use at the Instituto de Microelectrónica de Sevilla, at the Microelectronics Systems group of the University of Oslo and at the Institute of Neuroinformatics in Zürich.

6.1 Outlook

While the USBAERmini2 is already used by several groups and its design is sufficient for daily use, there are still possible improvements. The most important task is to implement sequencing capabilities into the CaviarViewer application.

In contrast, the USB2AERmapper is not yet ready for daily use. It has to be tested further and then the firmware of the USBAER has to be ported. This will done by the Architectura y Technologia de Computadores group at the University of Seville.

Bibliography

[1] CAVIAR Webpage. http://www.imse.cnm.es/caviar.

[2] P. Häfliger. CAVIAR Hardware Interface Standards, Version 2.0, May 2003.

[3] Cypress Inc. Cypress FX2 Datasheet. http://www.cypress.com.

[4] Cypress Inc. EZ-USB Technical Refernece Manual.

[5] Xilinx Inc. Coolrunner 2 Datasheet. http://www.xilinx.com.

[6] Thesycon Systemsoftware & Consulting GmbH. USBIO Reference Manual. http://www.thesycon.com/usbio/usbioman.pdf.

[7] Dr. Tobias Delbruck, Raphael Berner. CaviarViewer Javadoc. https://svn.ini.unizh.ch/repos/avlsi/CAVIAR/wp5/USBAER/INI-AE-Biasgen/host/java/dist/javadoc.

[8] Xilinx Inc. Spartan-3 Datasheet, 2005. http://www.xilinx.com.

Appendix A

USBAERmini2 User Guide

A.1 Setting Up The Host Computer

A.1.1 Software setup

If not yet installed, install the Java Runtime Environment 1.5. It can be downloaded from `http://www.java.com/en/download/index.jsp`.

Update or check out the code base from subversion. The URL is
`https://svn.ini.unizh.ch/repos/avlsi/CAVIAR/wp5/USBAER/INI-AE-Biasgen/`.

A.1.2 Setting the Environment Variables

This step is not needed if you want to use the USBAERmini2 only with Matlab. But if you plan to use the CaviarViewer, it is absolutely crucial to set up the Windows PATH environment variables, so that Java can use the native code DLLs.

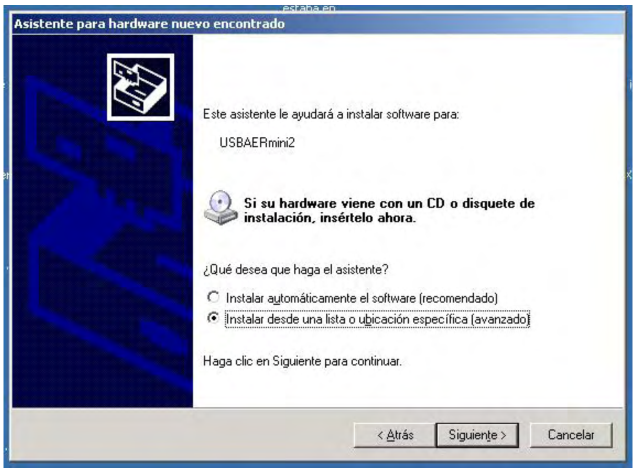
Open the Control Panel "System", click on "Advanced" and then "Environment variables". Now create a new variable called "usb2aemon". Then open a Windows explorer window and navigate to the working copy directory of the code you checked out and copy the complete path to the two following folders, separated by a ";" (only a semicolon, no space!) to this usb2aemon variable.

```
wp5\USBAER\INI-AE-Biasgen\host\java\JNI
wp5\USBAER\INI-AE-Biasgen\host\java\JNI\SiLabsNativeWindows
```

It will look something like this:

```
C:\Documents and Settings\tobi\My Documents\avlsi-svn\CAVIAR\wp5\
USBAER\INI-AE-Biasgen\host\java\JNI;C:\Documents and Settings\tobi\
My Documents\avlsi-svn\CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\java\
JNI\SiLabsNativeWindows
```

Now add this variable to the Windows PATH variable by including it surrounded by "%", as in `...;%usb2aemon%`. PATH (or "path", case doesn't matter) is another environment variable. Don't forget the semicolon ";" between the usb2aemon



Asistente para hardware nuevo encontrado

Este asistente le ayudará a instalar software para:

USBAERmini2

Si su hardware viene con un CD o disquete de instalación, insértelo ahora.

¿Qué desea que haga el asistente?

Instalar automáticamente el software (recomendado)
[✓ Instalar desde una lista o ubicación específica (avanzado)]

Haga clic en Siguiente para continuar.

Figure A.1: Select advanced driver installation.

variable and the other PATH components but DO NOT put a space between the semicolon and the next path.

A.1.3 Driver Installation

After plugging in the device, the Windows New Hardware Wizard will show up. Select "No, not this time" and click on Next. Select the advanced driver installation ("Install from a list or specific location"). Select "Search for the best driver in these locations" and browse to
`CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\driverUSBIO`.

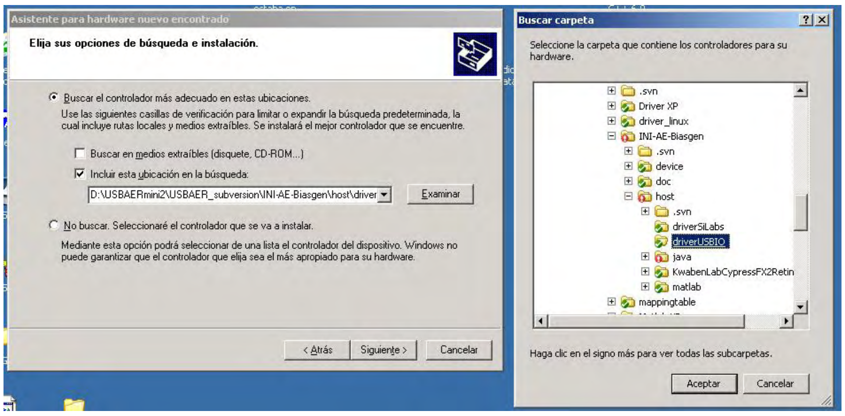
Windows should now be able to install the device correctly. You can check this by starting the device manager, which should now list the device as in figure A.3.

A.2 Setting Up Matlab

Make sure Matlab uses Java Runtime Environment 1.5 by typing `version -java` in the command prompt. If not, create a windows environment variable called `MATLAB_JAVA` which points to the JRE 1.5 home directory.

Start matlab and edit the file librarypath.txt by typing `edit librarypath.txt` in the command prompt. Add the path to the folder
`CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\java\JNI` to this file. If you want to use devices based on the Silicon Labs C8051F320 as well, add also the path to
`CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\java\JNI\SiLabsNativeWindows`.
The file will then look something like this:

```
##
## FILE: librarypath.txt
```



Asistente para hardware nuevo encontrado

Elija sus opciones de búsqueda e instalación.

(●) Buscar el controlador más adecuado en estas ubicaciones.

Use las siguientes casillas de verificación para limitar o expandir la búsqueda predeterminada, la cual incluye rutas locales y medios extraíbles. Se instalará el mejor controlador que se encuentre.

☐ Buscar en medios extraíbles (disquete, CD-ROM...)

☑ Incluir esta ubicación en la búsqueda:

D:\USBAERmini2\USBAER_subversion\INI-AE-Biasgen\host\driver

( ) No buscar. Seleccionaré el controlador que se va a instalar.

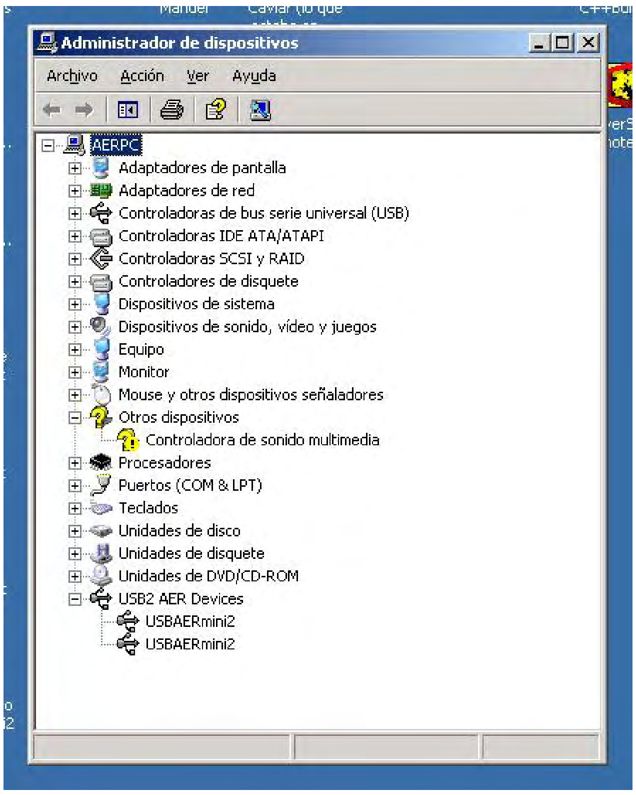
Mediante esta opción podrá seleccionar de una lista el controlador del dispositivo. Windows no puede garantizar que el controlador que elija sea el más apropiado para su hardware.

Buscar carpeta

Seleccione la carpeta que contiene los controladores para su hardware.

Haga clic en el signo más para ver todas las subcarpetas.

Figure A.2: Browse to the driver location.



Administrador de dispositivos

```
AERPC
  Adaptadores de pantalla
  Adaptadores de red
  Controladoras de bus serie universal (USB)
  Controladoras IDE ATA/ATAPI
  Controladoras SCSI y RAID
  Controladores de disquete
  Dispositivos de sistema
  Dispositivos de sonido, vídeo y juegos
  Equipo
  Monitor
  Mouse y otros dispositivos señaladores
  Otros dispositivos
    Controladora de sonido multimedia
  Procesadores
  Puertos (COM & LPT)
  Teclados
  Unidades de disco
  Unidades de disquete
  Unidades de DVD/CD-ROM
  USB2 AER Devices
    USBAERmini2
    USBAERmini2
```

Figure A.3: Windows device manager after successful driver installation.

```
##
## Entries:
##    o path_to_jnifile
##    o [alpha,glnx86,sol2,unix,win32,mac]=path_to_jnifile
##    o $matlabroot/path_to_jnifile
##    o $jre_home/path_to_jnifile
##
$matlabroot/bin/$arch
D:\USBAERmini2\USBAER_subversion\INI-AE-Biasgen\host\java\JNI
```

At this point you have to restart Matlab, otherwise the change in the library-path.txt file does not become active.

Navigate to the folder `CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\matlab\` and run the file `startup.m`. This adds `usb2aemon.jar` and `usbio.jar` to the dynamic matlab path and instantiates the hardware factory, through which the devices can be accessed.

Matlab is now ready to use the USBAERmini2 boards. To verify the installation, connect the sequencer output of a board directly to its monitor input and sequence a few hundred events using the script `aemonseq.m`. See section A.7 for details on this script.

A.2.1 Changing The Java Classes

If you want to develop further functionality for the java classes, download Netbeans 5 from `http://www.netbeans.org/downloads/`. Install it and open the `usb2aemon` project directory, which is `CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\java`.

You also have to download the Netbeans Profiler, which can be found at the same URL as Netbeans 5. Then you have to run the Profiler once by pressing the button shown in figure A.4.

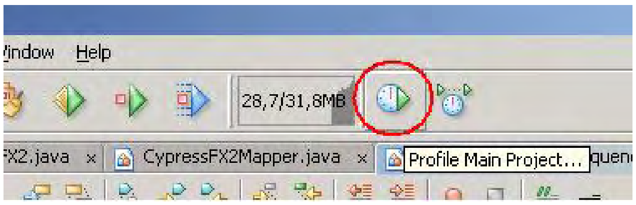
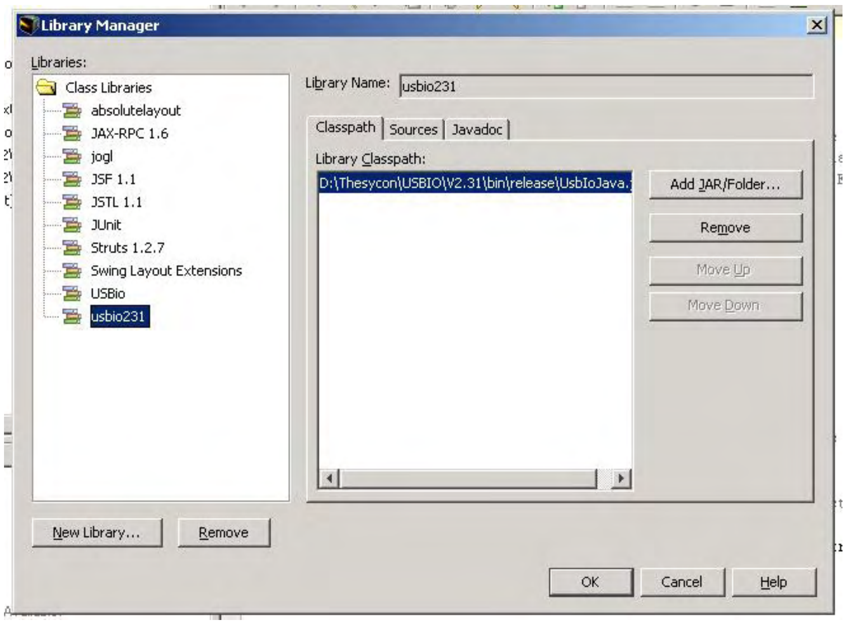


Figure A.4: Netbeans Profiler Button

Start the Library Manager, which can be found in the *Tools* menu. You have to add two libraries, one called *usbio231*, which points to `INI-AE-Biasgen/host/java/jars/UsbIoJava.jar`, and another called *jogl*, which points to `INI-AE-Biasgen/host/java/jars/jogl.jar`.



Library Name: usbio231

Library Classpath:
D:\Thesycon\USBIO\V2.31\bin\release\UsbIoJava.

Figure A.5: Netbeans Library Manager

A.3 Setting Up The Hardware Connections

Connect the USBAERmini2 boards to USB2.0 ports of your computer. The USBAERmini2 does not work with USB1.1. If you wish to capture synchronised data from several boards, connect the synchronisation output pin (labelled SO) from the desired master board to the synchronisation input pins (labelled SI) of the slave boards.

If the device is used in terminal mode, i.e. the monitor port is connected, but the pass-through port is not, it is best to connect request and acknowledge lines of the pass-through port. The device can detect if a receiver is present on the pass-through port, but only while monitoring. When monitoring is inactive, request and acknowledge lines of monitor and pass-through port are connected directly, therefore transmission is blocked if no device is present and monitoring is not active.

A.4 LEDs

DS1/3.3V Power LED. Should always be turned on while connected.

DS2/1.8V Power LED. Should always be turned on while connected.

DS3/BkPt Breakpoint LED. Only active in debugger mode.

DS4/Sequencer Turned on while sequencer is running in host trigger mode.

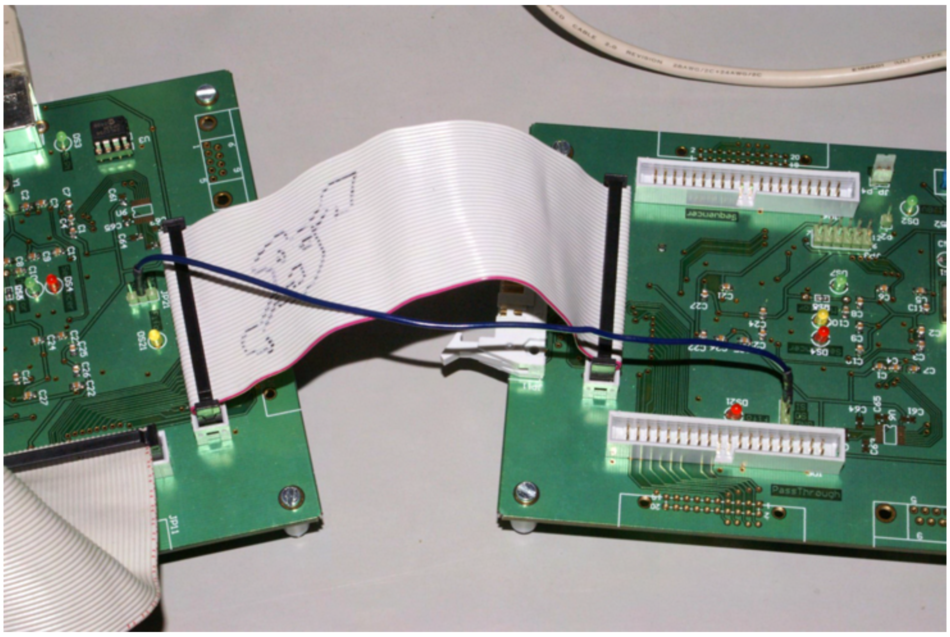


Figure A.6: Two USBAERmini2 boards with synchronisation connection

DS5/Monitor Turned on while monitor is running.

DS7 Heartbeat. Should always be blinking while connected, except while downloading to the EEPROM. If it stops, the microprocessor is stuck and the device has to be reset by unconnecting.

DS21/Fifo Master/Slave. Turned on as long as SI input is low, i.e. when the device is acting as timestamp master.

A.5 The CaviarViewer

The CaviarViewer is a Java application to view real time data form several kinds of AER devices like the USBAERmini2, the USB2AERmapper and the TmpDiff128 retina, to record data from these devices or to view recorded data. Sequencing recorded data is not yet implemented.

A.5.1 Starting and Live Displaying

The CaviarViewer can be started by doubleclicking `CAVIAR\wp5\USBAER\INI-AE-Biasgen\CaviarViewer.cmd`. A window like the one shown in figure A.7 should show up.

Choose the appropriate chip-type from the menu AEChip and then select the interface to display from the Interface menu, see figure A.7. Note that the interfaces

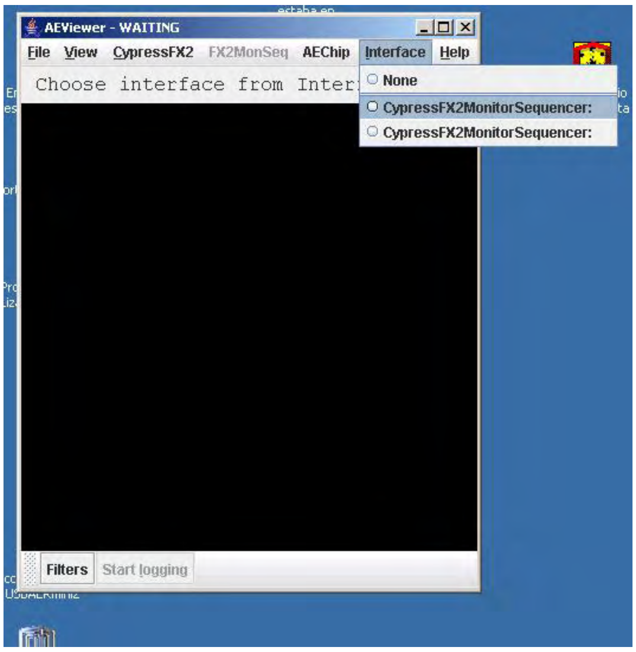

AEViewer - WAITING
File View CypressFX2 FX2MonSeq AEChip Interface Help
Choose interface from Inter
None
CypressFX2MonitorSequencer:
CypressFX2MonitorSequencer:
Filters Start logging

Figure A.7: CaviarViewer Window, selecting an interface

are sorted in the order of the PID, and secondly in the order of their name (serial number), although their name is not displayed. For example the board named *mini 1* is on top of board *mini 2*.

As soon as an interface is selected, event acquisition is started and chiptype, interface type and name are displayed in the titlebar, as can be seen in figure A.8.

If the selected interface is an USBAERmini2, the FX2MonSeq menu will become active and will let you change operation mode and get the number of events missed due to full fifos. See section A.6 for more details.

If the display of the events seems jerky, this may be due to a low event rate, taking a long time to fill the driver's buffers until it passes them to the application. In this case, it is best to reduce the size of the buffers. Figure A.9 shows how to do this.

A.5.2 Recording Events

Recording of events can be started by pressing the "Start logging" button in the lower left corner and stopped by pressing it again.

Synchronized Recording of Multiple Devices

There are two types of synchronisation available.

- Software synchronisation

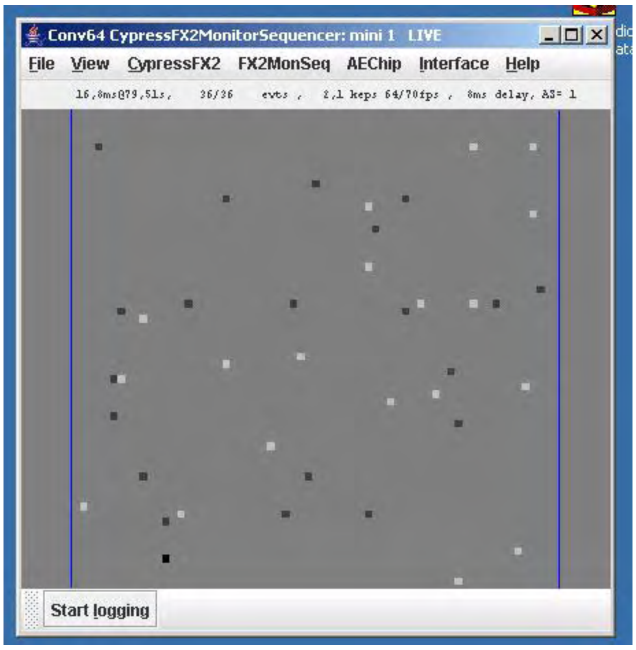

Conv64 CypressFX2MonitorSequencer: mini 1 LIVE
File View CypressFX2 FX2MonSeq AEChip Interface Help
16,8ms@79,51s, 36/36 evts , 2,1 keps 64/70fps , 8ms delay, AS= 1
Start logging

Figure A.8: CaviarViewer Window, LIVE view. Chip-type and interface are displayed in the titlebar

- Hardware (electrical) synchronisation

Software synchronisation resets the timestamps on the devices by sending a vendor request to each device in turn. This is not absolutely synchronous, but does not require the synchronisation pins to be connected. The devices are reset by pressing the zero key in one window.

The electrical synchronisation is more accurate, because all the devices are reset from the master device.To enable electrical synchronisation, connect the SO pin of the desired master device to the SI pins of the slave devices and check the *Electrical Synchronisation* checkbox in Menu → File. Then press the zero key *in the window of the master device*. Unfortunately, the resetting does not work reliable at the moment, so check if the time really is reset in all the windows.

A.5.3 Playing Back Recorded Files

You can view either single data files (.dat) or synchronised sets of files (.index). You can drag and drop either type of file onto a fresh CaviarViewer AEViewer window. Or you can select the file using menu item File/Open... (shortcut "o"). If you want to select an .index file, then you need to change the file type in the file chooser; due to a bug in the graphics it only shows you one choice and doesn't show the Open button until you hover over it. This is a byproduct of using a fast native "heavyweight" Canvas to render the events.

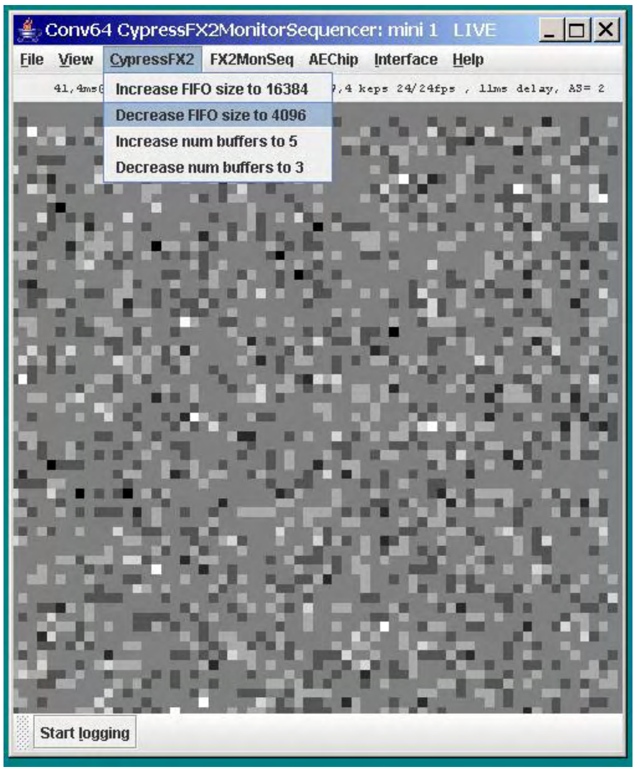

Conv64 CypressFX2MonitorSequencer: mini 1 LIVE
File View CypressFX2 FX2MonSeq AEChip Interface Help
Increase FIFO size to 16384
Decrease FIFO size to 4096
Increase num buffers to 5
Decrease num buffers to 3
Start logging

Figure A.9: Reducing the driver buffer size in the CaviarViewer

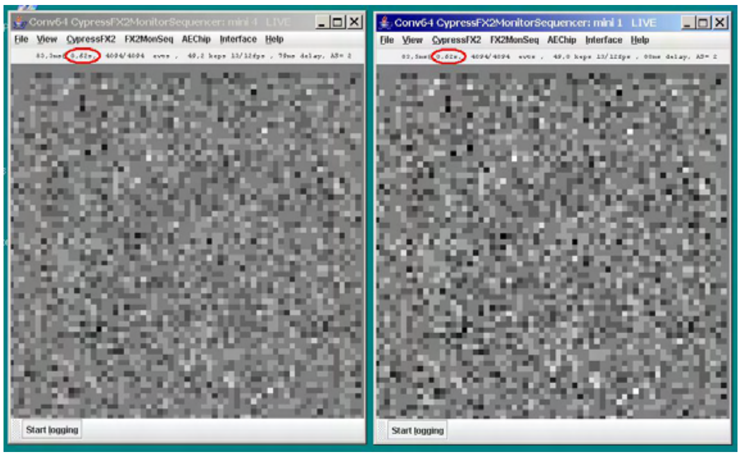

Conv64 CypressFX2MonitorSequencer: mini 4 LIVE
File View CypressFX2 FX2MonSeq AEChip Interface Help
Start logging
Conv64 CypressFX2MonitorSequencer: mini 1 LIVE
File View CypressFX2 FX2MonSeq AEChip Interface Help
Start logging

Figure A.10: Check if the time is reset in all windows.

Examine the menus for help; almost all menu items have single-key shortcuts. (e.g. r=rewind, f=faster, s=slower, etc).

A.6 Accessing The Java Classes From Matlab

Java classes and methods can be accessed directly from Matlab without any wrapper functions. For example the following calls instantiate a hardware factory and get a reference to the first USBAERmini2 device.

```
>> factory = ch.unizh.ini.caviar.hardwareinterface.usb.
CypressFX2MonitorSequencerFactory.instance();
>> usb0=factory.getInterface(0)
```

Note that the instantiation of the factory is not necessary, as this is already done by `startup.m`. See section A.7 for details.

A.6.1 Important Java Methods

Please see the javadoc [7] for more details and a complete list of methods.

CypressFX2MonitorSequencerFactory.listDevices()

List the available USBAERmini2 devices. Ignore the following errors, they are due to rapidly opening and closing the devices. *error binding to pipe for EP1 for device status: Error code 0x00000006: Windows system error code. can't set pipe parameters: Error code 0xE0001000: Operation failed.*

CypressFX2MonitorSequencerFactory.getInterface(int interfaceNumber)

This method returns a reference to a USBAERmini2 device. Parameter is the interface number, which has to be chosen from the output of `factory.listDevices`.

CypressFX2MonitorSequencer.open()

Opens the device.

CypressFX2MonitorSequencer.setOperationMode(int mode)

Sets operation mode, which includes timestamp tick and trigger mode. Valid modes are shown in table A.1.

In host trigger mode, event acquisition and sequencing is started and stopped when the commands from the host are received through USB. With this mode, it is not possible to start several devices synchronously.

In slave trigger mode, event acquisition and sequencing is started and stopped when the master device starts or stops. Therefore, slave mode works only when the SI pin is connected to the SO pin of the master device and the master device

Mode	Tick	Trigger
0	1us	Master (Host)
1	0.033us	Master (Host)
2	1us	Slave
3	0.033us	Slave

Table A.1: USBAERmini2 operation modes

receives the start or stop command from the host. This mode is recommended to use for the slave devices when using `multi_monitor.m` or `multi_monitor_seq.m`. These matlab scripts are described in section A.7.

CypressFX2MonitorSequencer.getOperationMode()

This method returns the timestamp tick and displays the operation mode.

CypressFX2MonitorSequencer.setContinuousSequencingEnabled(bool)

Disables or enables continuous event sequencing. If enabled, the AEWriter thread rewinds at the end of the packet of events it has to send and sends these events over and over again. If disabled, the AEWriter thread send the events only once. The default state is that events are sequenced only once.

This method can be used in combination with the scripts `aemonseq.m` and `multi_monitor_seq.m`. The scripts `aeseq.m` and `aeseq_cont.m` already call this method with the suitable parameter.

CypressFX2.setAEReaderFifoSize(int size)

Sets the buffer size of the event-capturing thread. In general, a buffer size of at least 8kB leads to the highest possible event rates, however, when the CaviarViewer is used to monitor a low eventrate connection, smaller buffer sizes are needed to produce suitable frame rates.

CypressFX2MonitorSequencer.getNumMissedEvents()

This method returns an estimation of the number of events the device has missed because of the Cypress FX2 fifos being full when the host does not collect the events fast enough.

CypressFX2MonitorSequencer.writeMonitorSequencerFirmware()

This methods writes the firmware, which is saved in the usb2aemon.jar files as well, to the EEPROM. Use this function when a new firmware version is available.

startup.m	adds classes to the dynamic path and instantiates the hardware factory
aemon.m	monitoring a single device
aemonseq.m	monitoring and sequencing with a single device
aeseq.m	sequencing from a single device
aeseq_cont.m	continuous sequencing from a single device
aeseq_cont_stop.m	stops continuous sequencing
multi_monitor.m	monitoring from multiple devices
multi_monitor_seq.m	sequencing from one device and monitoring from multiple devices

Table A.2: Matlab scripts

CypressFX2MonitorSequencer.setDeviceName(String name)

Sets a new serial number. Parameter is the new serial number string. Be advised that after you plug in a device with a new serial number, the Windows New Hardware Installation Wizard will show up. The string length is limited to eight characters.

A.7 Matlab Scripts

Table A.2 gives an overview over the matlab scripts available in `CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\matlab\monitor_sequencer`.

A.7.1 Important Note on Sequencing and Monitoring

IMPORTANT! Be advised that the sequencer needs *interspike intervals* for correct sequencing of spike trains, but the monitor returns *absolute* timestamps. Therefore recorded spike trains have to be processed (usually by the Matlab function `diff`) before they can be sent to the sequencer again.

A.7.2 startup.m

This script adds the path to the jar-files to the dynamic matlab path and instantiates the hardware factory, through which the devices can be accessed. This script has to be called at startup. Please note that this script is located in the parent folder, i.e. in `CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\matlab\`.

A.7.3 aemon.m

`[inaddr,ints,tick]=aemon(usbinterface,monitortime)`

This is a script to monitor events with one USBAERmini2 device.

Parameters

usbinterface Reference to USBAERmini2 device. Get a reference to a USBAER-mini2 device using for example `usb0=factory.getInterface(0)`.

monitortime How long the monitoring is active. This time is measured in seconds.

Returns

inaddr Address array returned from device.

ints Timestamps array returned from device.

tick Timestamp tick used in timestamps vector.

A.7.4 aemonseq.m

`[inaddr,ints,tick]=aemonseq(usbinterface,addr,ts,monitortime)`
Script to sequence and monitor events with one USBAERmini2 device.

Parameters

usbinterface Reference to USBAERmini2 device. Get a reference to a USBAER-mini2 device using for example `usb0=factory.getInterface(0)`.

addr Array of addresses to be sent to device.

ts Array of interspike intervals. Note that no interspike interval should be bigger than $2^{16}-1$. Also note that you have to set them according to the timestamp tick used on the device.

monitortime How long the monitoring is active. This time is measured in seconds.

Returns

inaddr Address array returned from device.

ints Timestamps array returned from device.

tick Timestamp tick used in timestamps vector.

A.7.5 aeseq.m

`aeseq(usbinterface,addresses,timestamps)`
Function to sequence events with one USBAERmini2 device. The device will stop sequencing when it has sequenced all events. Use `aeseq_cont.m` if you want to sequence continuously.

Parameters

usbinterface Reference to USBAERmini2 device. Get a reference to a USBAERmini2 device using for example `usb0=factory.getInterface(0)`.

addr Array of addresses to be sent to device.

ts Array of interspike intervals. Note that no interspike interval should be bigger than $2^{16}-1$. Also note that you have to set them according to the timestamp tick used on the device.

A.7.6 aeseq_cont.m

`aeseq_cont(usbinterface,addr,ts)`
Function to continuously sequence events with one USBAERmini2 device. The device will rewind if it reaches the end of the arrays. This function is non-blocking. Call `aeseq_cont_stop.m` to stop sequencing. Use `aeseq.m` if you don't want to sequence continuously.

Parameters

usbinterface Reference to USBAERmini2 device.

addr Array of addresses to be sent to device.

ts Array of interspike intervals. Note that no interspike interval should be bigger than $2^{16}-1$. Also note that you have to set them according to the timestamp tick used on the device.

A.7.7 aeseq_cont_stop.m

`aeseq_cont_stop(usbinterface)`
Stops a continuous sequencing device and releases it, so it can be accessed again from other processes.

Parameters

usbinterface Reference to the sequencing USBAERmini2 device.

A.7.8 multi_monitor.m

`[addr,isi,timestamps,tick]=multi_monitor(devices,monitortime)`

Function to monitor events with one or more USBAERmini2 devices.

Parameters

devices Array of references to monitor devices.

monitortime How long the monitoring is active. Time in seconds.

Returns

addr Cell array of address arrays. Same order as in monitors array.

isi Cell array of interspike intervals in nanoseconds.

timestamps Cell array of timestamps, unprocessed.

tick Timestamp tick used in timestamps arrays.

Make sure all the monitoring devices use the same timestamp tick! For synchronising the monitoring devices, connect SO pin of the desired master device to the SI pins of the slave devices, and set operation mode accordingly using `usbinterface.setOperationMode(mode)`. See A.6 for more details.

A.7.9 multi_monitor_seq.m

```
[addr,isi,timestamps,tick]=
      multi_monitor_seq(sequencer,monitors,addr,ts,monitortime)
```

Function to sequence events with a USBAERmini2 device and monitor with one or more other USBAERmini2 devices.

Parameters

sequencer Reference to sequencer device.

monitors Array of references to monitor devices.

addr Array of addresses to be sent to device.

ts Array of interspike intervals. Note that no interspike interval should be bigger than $2^{16} - 1$. Also note that you have to set them according to the timestamp tick used on the sequencer device.

monitortime How long the monitoring is active. Time in seconds.

Returns

addr Cell array of address arrays. Same order as in monitors array.

isi Cell array of interspike intervals in nanoseconds.

timestamps Cell array of timestamps, unprocessed.

tick Timestamp tick used in timestamps arrays.

Make sure all the monitoring devices use the same timestamp tick! For synchronising the monitoring devices, connect SO pin of the desired master device to the SI pins of the slave devices, and set operation mode accordingly using `usbinterface.setOperationMode(mode)`. Use one of the monitoring devices as master and use the sequencing device in slave mode and don't forget to connect it's SI pin. If you use it master mode, it will start sequencing a few milliseconds before the other devices start monitoring.

See A.6 for more details on the operation modes.

A.8 Importing Recorded Data To Matlab

To import data into Matlab that was recorded with the CaviarViewer, use the script `loadaerdat.m` located in `CAVIAR\wp5\USBAER\INI-AE-Biasgen\host\matlab\`. A window will pop up where you can navigate to the desired file. The function returns two vectors, one containing the addresses, one containing the timestamps. Remember that the timestamp vector represents absolute timestamps, which are not suited directly for sequencing! Create a suitable vector with the following Matlab command.

```
isi = [timestamps(1); diff(timestamps)];
```

A.9 Simultaneous Sequencing with Matlab and Using CaviarViewer

There is one important thing that has to be aware of if simultaneous sequencing and viewing the other devices with CaviarViewer is desired.

The interface with the highest serial number should be chosen as sequencing device, because the sequencing device has to be locked while sequencing continuously, the interface menu of CaviarViewer is built only up to the interface number in front of the sequencing device. Available devices with higher serial number than the sequencer are not displayed.

Ignore the errors *CypressFX2MonitorSequencer.openUsbIo(): getDeviceDescriptor: Error code 0xE000100: Operation failed.* They are due to the driver trying to access the locked device.

A.10 Updating Firmware

The Cypress firmware can be updated from Matlab with the java method `CypressFX2MonitorSequencer.writeMonitorSequencerFirmware()` or from the CaviarViewer by launching the CypressFX2 EEPROM utility in the CypressFX2 menu.

To update the CPLD firmware, a Xilinx download cable and the Xilinx Impact application are required. The Impact project file is stored in the repository under `CAVIAR\wp5\USBAER\USBAERmini2\CPLD\Xilinx project\USBAERmini2\USBAERmini2.ipf`. Please note that this project file stores the location of the firmware file with an absolute path, you will have to edit it with a text editor to change the path to your checkout directory.

The current PCB unfortunately doesn't have printed information on how to connect the Xilinx JTAG download cable. The JTAG connector is JP3, the pin layout is the following:

Pin	Description
7	VCC
8	GND
9	TCK
10	TDO
11	TDI
12	TMS

A.11 Using A Device Without Firmware

This section describes how to handle a device without EEPROM or with an EEPROM that has not been programmed yet. This would be the case for a newly built device.

Plug the device into your computer and follow the driver installation instructions described in subsection A.1.3. After successful driver installation, the device will show up as *CypressFX2Blank* in the device manager. Now start the CaviarViewer application. When the CaviarViewer application finds a blank FX2 device, it will automatically download the firmware for the TmpDiff128 retina to its RAM. Therefore you will be prompted again to install the driver. When the device is successfully installed as TmpDiff128 device, you can run the CypressFX2EEPROM utility from the CypressFX2 menu of the CaviarViewer application.

Press *Scan for device* to open the blank device, and then press *EEPROM Monitor/Sequencer firmware* to download the USBAERmini2 firmware to the device.

To download the CPLD code to the device, follow the instructions in section A.10. To download the CPLD configuration, the device doesn't need FX2 firmware, but it has to be plugged into a USB port to be powered.

A.12 Power Issues

The USBAERmini2 was actually designed to be used with the Cypress FX2LP, which is an improved version of the FX2 and needs less power. As the FX2 draws more than 100mA from the USB bus during enumeration, some hosts may disable the device, as this is not within the USB specifications. If this is the case and the USBAERmini2 is not recognised, please try another USB port on your computer.

A.13 Using USBAERmini2 Simultaneous With Other Devices

Note that so far it is NOT possible to use the USBAERmini2 and the TmpDiff128 retina which uses the Cypress FX2LP simultaneous on the same computer. You must not have a retina plugged in when you want to use a USBAERmini2. Using the USBAERmini2 with a TmpDiff128 that uses the SiLabs USB chip is no problem though.

It is also no problem to use a USBAERmini2 and a USB2AERmapper simultaneous on the same computer.

Appendix B

Repository Folder Structure

All the files associated with this master project are located in the following subversion repository: `https://svn.ini.unizh.ch/repos/avlsi/CAVIAR/wp5/USBAER/`. The files are distributed over three folders, described in the following sections.

B.1 INI-AE-Biasgen

This folder holds everything concerning the host computer. This includes the driver, java classes, matlab scripts and CaviarViewer application. Additionally, this folder hosts files regarding a retina device.

device Contains files regarding the TmpDiff128 retina.

doc Contains several documentation files, including this report.

host This folder holds all files regarding the host.

- **driverUSBIO** This folder contains the windows driver for both devices described in this report.
- **java** Holds all files concerning the java implementation. This includes source code and javadoc documentation.
- **matlab** Holds all the matlab scripts described in section A.7 in a subfolder called monitor_sequencer.

CaviarViewer.cmd Starts the CaviarViewer application.

CypressFX2EEProm.cmd Starts the utility to download Cypress firmware to the EEPROM.

B.2 USB2AERmapper

This folder holds all files regarding the USB2AERmapper device.

CyUSBAER2_PCB This folder holds the Altium PCB and schematics files.

data_sheets Holds data-sheets of components used for the USB2AERmapper.

Firmware_Cypress Holds the sourcecode and μVision project files for the Cypress FX2 firmware.

FPGA Holds files concerning FPGA implementation.

- **MapperProject_ISE6** Xilinx ISE6.1 project files.
- **sourcecode** VHDL sourcecode for the Monitor/Sequencer implementation
- **template** Template for future FPGA implementations. Consists of VHDL top level file and constraints file.

B.3 USBAERmini2

All files regarding the USBAERmini2 device are located in this folder.

CPLD Holds files concerning the CPLD implementation.

- **Xilinx project** Holds sourcecode and project files.
  - **sourcecode** Holds the VHDL sourcecode.
  - **USBAERmini2** Xilinx ISE 7.1i project files.
- **small-board-block-diagram.pdf/.sxd** Pdf and OpenOffice files containing a block diagram for the USBAERmini2.
- **state-machines.pdf/.sxd** Pdf and OpenOffice files containing state diagrams for the state machines.

Firmware_Cypress Holds the sourcecode and μVision project files for the Cypress FX2 firmware.

firmware_loader Windows driver which connects to PID 8800 (USBAERmini2 without firmware) and downloads firmware to the device. The device would then ReNumerate with the new firmware. Obsolete as firmware is stored in EEPROM.

PCB_def Contains Altium PCB and schematics files regarding an improved PCB, which is considerably smaller than the current PCB. Note that this PCB has not been fabricated.

PCB_developement_board Contains Altium PCB and schematics files regarding the current PCB, which can be seen in figures 4.1 and 4.2.