A group design project report submitted for the award of
Master of Engineering

Supervisor: Dr Peter R. Wilson
Examiner:

ELEC6027 VLSI Group Design Project
Team I
Southampton Superchip Automated Tester
by
Alexander Hornung
Xiaolong Li
Pavlos Progias
Romel Torres
Bo Zhou
May 17, 2012

UNIVERSITY OF SOUTHAMPTON

<u>ABSTRACT</u>

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

<u>A group design project report submitted for the award of Master of Engineering</u>

by
Alexander Hornung
Xiaolong Li
Pavlos Progias
Romel Torres
Bo Zhou

Contents

List of Figures

List of Tables

Listings

List of Symbols

w The weight vector

Acknowledgements

Thanks to no one.

To . . .

Chapter 1

Introduction

1.1 Aim and Background

Nano (get started)
mention: D2, Superchip, stuff

1.2 DE2 Board Overview

Nano (get started)

1.3 System Overview

Pavlos (get started)
see slides

Chapter 2

Hardware

2.1 Hardware Overview

The hardware part consists of two main blocks: NIOS System on Programmable Controller and the Tester.

The *NIOS SoPC* coordinates the whole system, writing the data from SD Card into the SRAM for the Tester to read the data from it; retrieving the result that the Tester writes
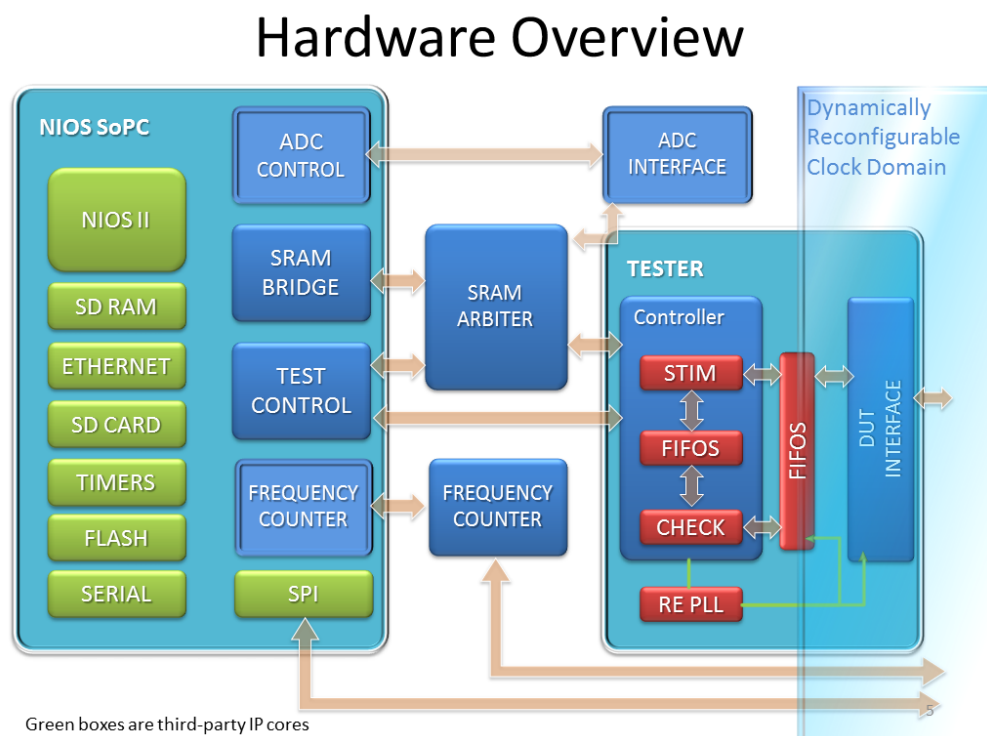
FIGURE 2.1: Hardware Overview.

back to the SRAM and sending the data to the backend for archiving and reviewing for the users. Its tasks include:

- coordinate the data from multiple sources (SRAM, SD Card, Flash, Serial connection, SPI bus, Ethernet, etc.)

- control the testing process (which is executed by the Tester, frequency counter and ADC)

- send results to the backend

- user interface

The *SRAM arbiter* is the interface between the SoPC, tester and the SRAM by Avalon Memory Mapped Interface (Avalon-MM). It handles the SRAM read and write requests from SoPC and tester.

The *Tester* contains several HDL blocks which are responsible for the testing process.

- read the command, request and test vector from SRAM

- generate the test input signals for every pin of the DUT

- monitor the output pins of the DUT and write the result into the memory when the it is valid

- offer a dynamically reconfigurable clock for the DUT and its relative interface

The purpose of the *frequency counter* and the *ADC* is that there is a designated pin on the chip which is the output of a ring oscillator. It is preferable that not only the frequency of this clock output is tested, but that the output waveform is also monitored.

2.2   PCB (interface board)

For the chip tester, two PCBs are needed in this project. One which is called $B2$ has been designed for the virtual designs to be loaded in a slave FPGA. The other is called $B1$ that is developed as an interface for those real chips are about to be tested. Both PCBs are two layers board and designed by using Allegro 16.3 PCB designer tools. The dimensions of $B1$ and $B2$ are $98.93mm * 68.5mm$ and $81.28mm * 68.33mm$ and the minimum wire size is $8mm$.

2.2.1   B1 (DUT testing board)

2.2.1.1   DUT

The superchip under test has the following specific features:

- 6 separate design sites

- 24 digital input pins $[A0 \ldots A23]$, shared between all design sites;

- 4 digital output pins $[Q0 \ldots Q23]$, shared between all design sites;

- 16 separate VDD pins, one dedicated to each design site;

- 1 global GND pin for all design sites and infrastructure circuitry;

- 1 global VDD pin to power the site buffers and I/O pad ring;

- 68-pin JLCC package (with 2 unused pins).

The chip is interfaced with the PCB by a 68WAY PLCC socket. The power is supplied by the 3.3V VDD from the DE2 board. The GND is also connected to the DE2 GND.

The shared I/Os are connected to the powered design site, and disconnected from the other design sites. Hence a controllable power switch is designed with a 4-16 decoder and an integrated power switch array. The $Q1$ is the output of a ring oscillator. The frequency can be digitally tested by the general I/O. However, to examine the analog properties of this output, an analog part including a buffer, 3rd order Butterworth filter and ADC with their own power supply is also designed.

2.2.1.2   Digital Part

Decoder   Although there are 16 design sites to be selected, the data can be shrunk down to 4 bits to save space of the test vector and metadata, since only one site is selected at a time. The decoder we used is $74HC4514$, a $4 - to - 16$ line decoder with latch. The input and output are active high.

Switch Array   The switch array consists of four $TPS2095$ quad power-distribution switches. One could simply use a CMOS to realize a switch. However, with thermal sense, current limit and charge pump, the $TPS2095$ switches are more reliable, stable, smooth (minimum switching current surges) and relatively compact in scale. The operation of a switch is simple: when EN pin is asserted, the OUT pin offers the power with the same voltage in the IN pin. Otherwise the OUT pin is disconnected from the input power. In one $TPS2095$ chip there are 4 such switches, the EN pins of which are all active high.
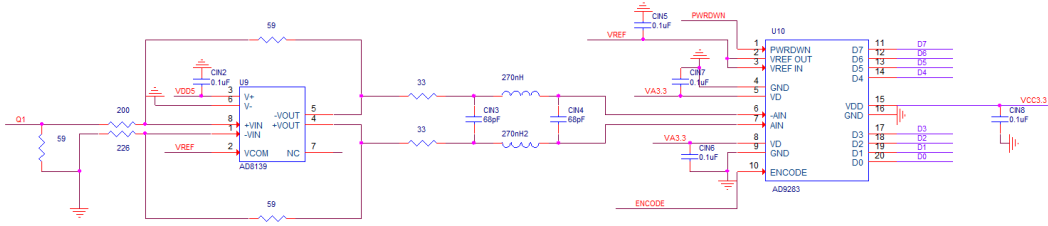
FIGURE 2.2: Analog Part Overview.

2.2.1.3   Analog Part

Buffer   An $AD8139$ differential ADC driver is used as the front buffer of the ADC. It is an ultralow noise, high performance differential amplifier with rail to rail output from Analog Devices. The designed gain of the buffer is 0.295, and the differential gain is approximately 0.14.

ADC   The $8-bit$, $100MSPS$ ADC is used to convert the analog signal. The encode clock is the clock from the DE2 board ($100MHz$ by default). A power-down function select is also connected to the FPGA for shutting down the ADC when its not in use. The ADC signal wires should be placed far away from the other wires for minimum interference; and they should also be placed parallel to each other to have similar length for better signal integrity. However, since the pin density is too high of the HSMC connector we used, it is inevitable that the ADC wires are closed to the other wires. The ADC is placed on the other side of the PCB board for a better result.

Butterworth Filter   To minimize the high frequency noise, a third-order low pass Butterworth filter is designed. With three poles, the attenuation is $-60^{dB}/_{decade}$ on signals higher than the cut-off frequency. The filter topology used here is balanced ladder topology. Automatic design tool Elsie is used for designing the parameters of the RCI circuit. The cut-off frequency is $40MHz$.

Separate Power Supplies   The buffer and ADC are separately powered from the other parts of the board. The $AD8139$s power ranges from $5V$ to $12V$, and the ADC ranges from $2.7V$ to $3.6V$. Two low dropout regulators (LDP), $ADP3335$ and $ADP3333$ are used to offer $5V$ and $3.3V$ voltages respectively. The supply voltage of these two LDP ranges from $2.6V$ to $12V$. Their load currents are up to $500mA$ and $300mA$ respectively.

2.2.2 B2 (Virtual design board)

2.2.2.1 Slave FPGA

Slave FPGA is the core chip of the virtual design board. It can be used for programing designs when they are going to be tested. An Altera Cyclone III FPGA will be implemented as the slave FPGA.

Cyclone III ($EP3C25E144$) belongs to Cyclone III device family. With densities ranging from about $5,000$ to $200,000$ logic elements (LEs) and $0.5Mb$ to $8Mb$ of memory for less than $1/4V$ of static power consumption, Cyclone III devices makes it easier to meet the power budget (**?**).

The reason for choosing Cyclone III is their lowest power, high functionality with the lowest cost. 144 pins are enough for this design. I/O pins on the Cyclone III are grouped together into I/O banks, and each bank has a separate power bus. There are eight I/O banks, as shown in Figure 1 (**?**). All single-ended I/O standards are supported in all banks except HSTL-12 Class II which is only supported in column banks. The same case can be found in all differential I/O standards.

Each I/O banks of Cyclone III has a VREF bus to accommodate voltage-referenced I/O standards. Connect the VREF pin of each group to the appropriate voltage. In this design three voltage levels, which $VCCIO = 3.3V$, $VCCINT = 1.2V$ and $VCCA = 2.5V$, are necessary. Proper bypassing and decoupling technique for the power pins is very important for reliable design operation. In this case, additional decoupling capacitance is needed. The VCCINT, VCCIO and ground pins should add as many as $0.2\mu F$ power-supply decoupling capacitors as possible. So $0.01\mu F$ capacitor for VCCIO and $0.1\mu F$ for VCCINT seems to be appropriate. Note that all the capacitors should be place close enough to the power pins.

In regard of the Cyclone III configuration, the Cyclone II device uses SRAM cells to store configuration data. $EP3C25E144$ can only be configured in Fast Active serial (AS) mode.

Configuration data is loaded into Cyclone III at each DCLK cycle. As soon as the device receives all data, the device releases the open-drain CONF_DONE pin, which is pulled high by an external 10- pull-up resistor. The CONF_DONE pin transits low-to-high to indicate that configuration is complete and initialization of the device can begin. The CONF_DONE pin must have a 10- pull-up resistor for initialization.

Pulling the nCONFIG pin low can begin reconfiguration and this pin must be low for at least $500ns$. The Cyclone III device is reset when nCONFIG is pulled low. Meanwhile, the device will pulls nSTATUS and CONF_DONE low and I/O pins are tri-stated. When nCONFIG returns to high and nSTATUS is released, reconfiguration begins.

The clock source for initialization is either a $10MHz$ (typical) internal oscillator or an optional CLK pin. In this situation, an external oscillator (A $TXC - 50MHz$ oscillator is going to be introduced later) will be implemented as the clock source. The required clock for initialization in Cyclone III is $3,185$ and the maximum CLK frequency is $133MHz$.

The configuration mode is selected by driving the MSEL pins either low or high. The MSEL pins can be powered by VCCIO and GND. For AS mode, MSEL [1] should be pulled up by connecting to VCCIO. MSEL [0] and MSEL [2] are pulled down by connecting to GND. The MSEL pins have 9 internal pull-down resistors that are always active.

The maximum active master frequency for Cyclone III is $30MHz$ typically and device only work with serial configuration devices that support up to $40MHz$.

In AS mode, Cyclone III reads the configuration data providing by serial configuration (A Spansion SPI Flash is going to be introduced later) via a serial interface. The serial configuration device controls the configuration interface.

There are four pins on the serial configuration devices:

- Serial clock input (DCLK)

- serial data output (DATA)

- As data output (ASDI)

- Active-low chip select (nCS)

Connect these four pins to Cyclone III device pins, as shown in Figure 2.

A 25Ω series resistor must connect a between serial configuration device and the Cyclone III device at the near end of the serial configuration device for DATA [0] when configure the Cyclone III device in the AS mode. The 25Ω resistor works to minimize the driver impedance mismatch with the board trace and reduce the overshoot seen at the Cyclone III device input pin DATA [0].

The maximum trace length between Cyclone III device and the serial configuration device, in another words, the CLK, DATA [0], NCSO and ADSO pins, must less than $10in$. In the B1 PCB designing, only $8mm$ is used.

Cyclone III device uses a $40MHz$ internal oscillator to generate DCLK to controls the entire configuration cycle and provide timing for the serial interface.

By driving the nCSO output pin low, which connect to nCS pin of the configuration device, the Cyclone III device enables the configuration device. DCLK and DATA [1] pins are used to send operation commands and read address signals to the serial configuration device. The configuration device sends data on DATA pin which connects to

the DATA [0] pin of the Cyclone III device. After all the configuration bits are received, cyclone III releases the open-drain CONF_DONE pin with a 10Ω pull-up resistor. The CONF_DONE pin must have an external 10Ω resistor for the device to initialize.

2.2.2.2 Serial Configuration Device

Cyclone III FPGAs are programmable logic devices used for basic logic functions, chip-to-chip connectivity, signal processing, and embedded processing. They can be programmed and configured by a microprocessor, JTAG port, or directly bay a serial PROM or flash. Spansion SPI (Serial Peripheral Interface) flash $S25FL064K$ can configure the FPGA easily at power-up (**?**).

The three stages of the configuration cycle are power-on reset, configuration, and initialization. When the FPGA enters power-on rest (POR), it drives the nSTATUS signal low to indicate it is busy, drives the CONF_DONE signal low to indicate the configuration has not been completed, and tri-states all I/O pins. All pins will be released after POR.

The DCLK generated by the FPGA device control the configuration data transferring. The CONF_DONE pin will be released with pulling high by an external pull-up resistor after all configuration data is transferred to the FPGA. The FPGA enters user mode after internal initialization.

The SPI is a simple four-pin synchronous interface protocol which enables a master device and one or more slave devices to intercommunicate. Four signal wires are:

- Master Out Slave In (MOSI) signal generated by the master (data to slave)

- Master In Slave Out (MISO) signal generated by the slave (data to master)

- Serial Clock (SCK) signal generated by the master to synchronize data transfers

- Slave Select (SS) signal generated by master to select individual slave devices (also known as Chip Select (CS) or Chip Enable (CE))

Figure 3 displays a simple block diagram of the connection between FPGA and SPI flash, as well as the HSMC header and JTAG programming the SPI flash from a host PC.

Figure 4 displays the details of the connection between SPI flash and FPGA. According to the introduction of Cyclone III, we can acquire the whole routing of FPGA, SPI flash and the HSMC header (A Samtec ASP header will be introduced later).

The TMS, TDI, TDO and TCK pins of Cyclone III device are used to operating in the IEEE Std. 1149.1 BST. The TDO pin is powered by VCCIO (3.3V). TDI and TMS are powered by VCCA (2.5V).

The header for JTAG and the header for SPI Flash In-system is use the same HSMC header with 172 pins which is quite enough for these two headers.

2.2.2.3   External Oscillator

As mentioned in the Cyclone III part, an external oscillator is needed as the clock source to provide certain frequency for the FPGA. A $50MHz$ oscillator of TXC will be implemented. The typical clock frequency for Cyclone III device is $30HMz$ and the maximum is $40MHz$. The TXC DEL04 oscillator is a sealed clock crystal oscillator unit with high precision characteristic covering up to wide frequency range ($1MHz$ to $170MHz$), which is appropriate for the FPGA. The supply voltage range is $1.8V \sim 5V$ (**?**).

2.2.2.4   Voltage Regulator

The $B2$ board needs at least three different voltage levels, as mentioned before, $VCCIO = 3.3V$, $VCCINT = 1.2V$ and $VCCA = 2.5V$. The external power source is provided by the main FPGA DE2-115 board at $3.3V$. Hence two voltage regulators are used to transfer $3.3V$ into $1.2V$ and $2.5V$.

1.2V Voltage Regulator $LD1117$ The $LD111712$ is a low drop voltage regulator able to provide up to $800mA$ of output current, available in adjustable version (). The device is supplied in DPAK surface mount package optimize the thermal characteristics even offering a relevant space saving effect. A very common 10 minimum capacitor is needed for stability (**?**).

Figure 5 is the application circuit for 1.2V output.

2.5V Voltage Regulator $TPS78225$ The $TPS78228$ is a low dropout linear voltage regulator designed by TI. The enable pin (EN) is compatible with standard CMOS logic while the low drop output is stable with any capacitor greater than 1. The device requires minimal board space for miniaturized packaging and potentially small output capacitor (**?**).

The enable pin is active high and is compatible with standard and low-voltage CMOS levels. Therefore if the shutdown capacitor is not necessary, enable pin can connect to the IN pin as shown on Figure 6.

2.2.2.5   HSMC Header

The Altera High Speed Mezzanine Card (HSMC) specification defines the electrical and mechanical properties of the HSMC adapter inter face for FPGA-based motherboards. The HSMC connector is based on the Samtec $0.5mm$ pitch, surface-mount QTH/QSH family of connectors (**?**). Two versions can are used in FPGA board. $ASP - 122953 - 01$ Socket for the host boards and $ASP - 122952 - 01$ Header for Mezzanine Cards (slave boards).

Figure 7 is the diagram for HSMC header. The clock-data-recovery differential signals in Bank 1 are the highest frequency signals. Signals between the HSMC connector and the host board FPGA device are intended to be D/C coupled. The JTAG, a system management bus (SMBus), and clock signals are also dedicated in Bank 1. In banks 2 and Bank 3, there are main CMOS/LVDS interface signals, including LVDS/COMS clocks, as well as both $12V$ and $3.3V$ power pins.

The host board is any board with an FPGA connected to one or more HSMC interface. In this project it is $DE2 - 115$ developed board. The interconnect I/O pins available on the HSMC connector can have all possible I/O standard and logic features that can be supported by the host FPGA since FPGAs are configurable devices. However basically they are limited by the wire types on the board.

The HSMC connectors provided the interface between host and slave boards. The "header" part ($ASP - 122952 - 01$) on slave board plugs into the "socket" part on the host board. The host board provides $+12V$ DC and $+3.3V$ DC power to the slave board via the HSMC connector. In addition to power and clock signals, the host board provides access to JTAG, high speed serial I/O, and single-ended or differential I/O via the HSMC connector.

The HSMC connector has a total of 172 pins, including 121 signal pins (120 signal pins + 1 PSNTn pin), 39 power pins, and 12 ground pins. The ground pins are much larger than the power pins and are located between the two rows of signal and power pins. Figure 7 is the modules for HSMC connectors.

The $ASP - 122952 - 01$ header provides 160 total pins and 12 ground plane connection pins down the center. Bank 1 has 40 pins with every third pin removed. Bank 2 and 3 have 60 pins each as no pins are removed. Host boards provide transceivers to Bank 1 which is not used in this project. Single-ended signals are provided to Bank 2 and 3. Typically, the single-ended signals are capable of differential signalling such as LVDS.

The JTAG signals are intended to connect to dedicated JTAG pins on the host FPGA and be part of the JTAG chain. The JTAG signals TCK, TMS and TDI are intended to be output from host board while JTAG TDO should be the input to host board as Figure 4 before.

Tables A.1 and A.2 in the appendix show the pin-outs for the HSMC header on B1 and
B2 boards, respectively.

2.3   Chip Tester (Verilog Module)

Joey (and a bit Alex)

mention: reconfigurable clock domain, synchronizing DC FIFOs

- test controller

- mem_if

- stim

- check

- dut_if (and its pipeline)

- reconfigurable PLL

2.4   SRAM Arbiter (sram_arb_sync.v)

Pavlos

Chapter 3

SoPC

3.1 Avalon Memory Mapped Interface

The Avalon Memory Mapped (Avalon MM) interface is used throughout the project to interconnect blocks with memory elements and the SoPC interconnect. An example Avalon MM waveform is shown in Figure 3.1.
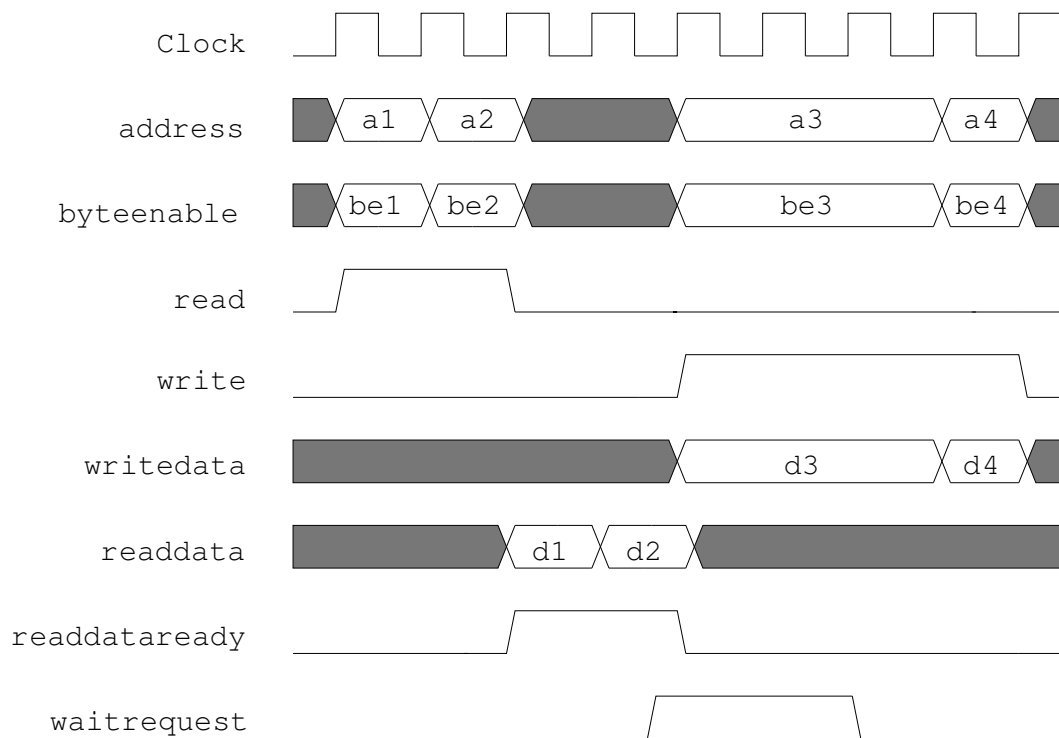
FIGURE 3.1: Example waveforms of an Avalon MM bus

The master can issue both reads and writes, which a slave then replies to. A read is initiated by setting the desired address and byte enables and asserting the `read` signal. The master can continue issuing reads to different addresses as long as the `waitrequest` signal stays low. If the Avalon MM slave asserts the `waitrequest` signal, the master needs to hold the current signals for as long as the `waitrequest` signal is asserted. The Avalon MM slave will reply to the read by asserting the `readdataready` signal and providing the requested read data on the `readdata` bus.

Similarly writes are initiated by the Avalon MM master by setting the desired address, byte enables, data to write and asserting the `write` signal. The master can assume that the write completed without waiting any further unless the `waitrequest` signal is asserted, in which case, similarly to the reads, the master needs to hold the signals steady until `waitrequest` is deasserted.

3.2   Overview

A decision was made to use a hybrid software and hardware approach to tackle the problem so as to simplify the required hardware. To achieve this, a system on programmable chip (SoPC) generated by the Altera QSys software was used.

Figure 3.2 shows an overview of the SoPC module. The system consists of a Nios II/f core and a number of peripherals interconnected via the QSys (Merlin) Network-on-Chip interconnect.
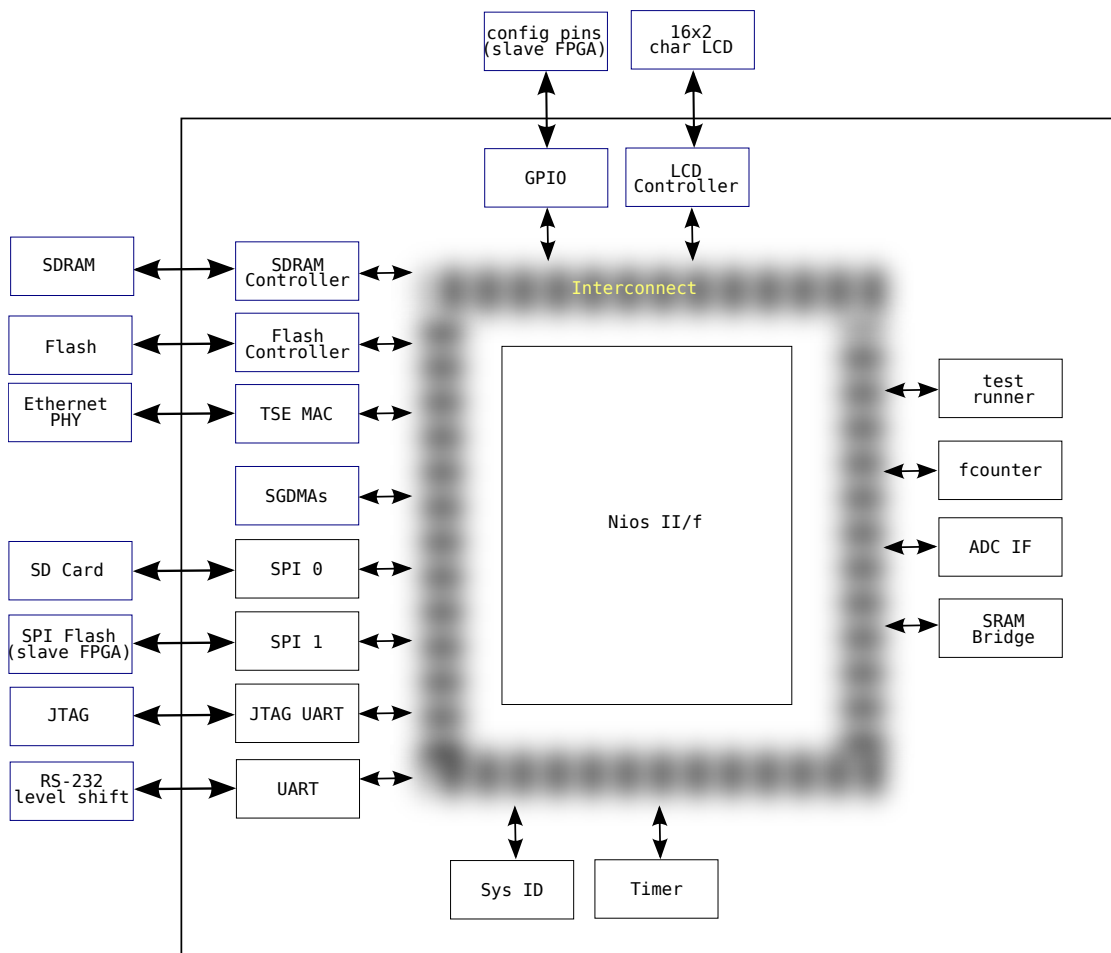
FIGURE 3.2: Overview of the SoPC

The system is clocked by a 100 MHz clock generated by a PLL. Additionally a 10 MHz clock is also generated, which is used to clock the GPIO controller and the LCD controller. The system consists of both third-party and own IP cores.

The GPIO controller provides a General Purpose I/O interface to the operating system. It is connected to the configuration-related pins of the slave FPGA - nCE, nSTATUS, CONF_DONE, nCONFIG.

Two SPI interfaces master controllers are also included in the system. Both run the SPI interface at a safe 10 MHz clock frequency. SPI0 is connected to the SD Card, while SPI1 connects to the SPI Flash on the slave FPGA board.

Serial connectivity is provided by the JTAG UART and UART modules. The JTAG UART provides a serial UART interface over the USB JTAG for use with the custom Nios II software tools on a host PC. The UART module provides a regular serial interface to the on-board RS-232 connector. The controller implements the RX and TX signals as well as transmission control signals CTS and RTS.

The two main memory interfaces are an SDRAM controller interfacing to the on-board 128 MB SDRAM, and a flash controller interfacing to the on-board 8 MB CFI Flash.

An ethernet interface is provided via the Altera Triple-Speed Ethernet (TSE) MAC module. The CPU interfaces to the TSE MAC via two Scatter-Gather DMA controller to maximize throughput. The MAC uses an RGMII interface to connect to the on-board Ethernet PHY. The RX clock to the MAC initially used a 0 degree phase shift. This resulted in a large number of dropped packages and hence TCP rentransmissions. Increasing the phase shift to 90 degrees significantly improved the receive performance. Another large improvement in transmit and receive reliability and throughput was achieved by disabling the statistics counters in the MAC.

The system also includes a timer module for use with the operating system, and a sysid module which provides a programmable ID and the synthesis date of the system to the operating system.

A custom module (SRAM Bridge) is used to interconnect the CPU with the internal SRAM synchronizing arbiter (sram_arb_sync). It is only a bridge exposing a memory range as an external Avalon MM master interface.

Further custom modules interface with other on-chip peripherals. The Test Runner module interfaces the tester and test controller with the system. The ADC interface provides a control interface for the on-chip ADC controller.

A custom frequency counter module takes an external signal bus, and is able to measure the frequency on any of the incoming signals.

Table 3.1 shows the memory map as seen from the CPU.

Address Range	Peripheral
`0x00000000 - 0x07ffffff`	SDRAM
`0x08001000 - 0x08001fff`	TSE MAC SGDMA Descriptor Memory
`0x08002800 - 0x08002fff`	JTAG Controller
`0x08003000 - 0x080033ff`	Triple-Speed Ethernet MAC
`0x08003400 - 0x0800343f`	TSE MAC SGDMA (TX)
`0x08003440 - 0x0800347f`	TSE MAC SGDMA (RX)
`0x08003480 - 0x0800349f`	Timer
`0x080034a0 - 0x080034af`	PLL
`0x080034b0 - 0x080034b7`	JTAG UART
`0x0a000000 - 0x0a7fffff`	CFI Flash
`0x0b000010 - 0x0b00001f`	LCD Controller
`0x0b000200 - 0x0b00020f`	GPIO (nSTATUS)
`0x0b000210 - 0x0b00021f`	GPIO (CONF_DONE)
`0x0b000220 - 0x0b00022f`	GPIO (nCONFIG)
`0x0b000230 - 0x0b00023f`	GPIO (nCE)
`0x0ba00000 - 0x0ba00007`	SYS ID
`0x0ba10000 - 0x0ba1001f`	SPI 0
`0x0ba20000 - 0x0ba2001f`	SPI 0
`0x0c000000 - 0x0c1fffff`	SRAM (SRAM Bridge)
`0x0d000000 - 0x0d0000ff`	Test Runner
`0x0d100000 - 0x0d10001f`	UART
`0x0d200000 - 0x0d2003ff`	Frequency Counter
`0x0d300000 - 0x0d3000ff`	ADC Interface

TABLE 3.1: Memory map of the SoPC as seen from the CPU data master

3.3 Nios II

A Nios II/f is the application processor in the SoPC. The Nios II/f is a 32-bit MIPS-based RISC processor with a branch predictor, barrel shifter, hardware multiplication and division, instruction and data caches and an MMU.

The choice of the Nios II/f was made in particular with the MMU in mind. The lower-end Nios II/e and Nios II/s cores do not support an MMU. By using an MMU it is possible, by using virtual memory, to allocate for example large buffers, of which only the used pages will be backed by real memory. For example a 1 MB buffer will not be fully allocated with backing memory immediately, but only when all its pages are actually used. Another important advantage of an MMU is the memory protection. Faults in userland such as segmentation faults can be caught and recovered from.

To improve the performance of the system with an MMU, a Translation Lookaside Buffer (TLB) of 128 entries was also included.

During the initial stages of development, the CPU was used with a Level 3 debug module, which includes a number of hardware breakpoints and watchpoints. This was later reduced to a Level 1 debug module which only includes a small JTAG controller.

The reset vector of the CPU points to the first location of the CFI Flash, so that the CPU boots up from the non-volatile Flash memory. The exception vectors are located in the SDRAM, where the OS will be located as soon as the bootloader has loaded it.

Initially a small cache size of just 4kB and 8kB (with a line size of 32 bytes) was chosen for the D and I caches respectively. Given the large amount of unused resource on the board a decision was made to increase those sizes to 32kB and 64kB respectively. Table 3.2 shows some benchmark results with both cache sizes. The improvement is fairly significant. In the case of Dhrystone, the complete benchmark fits into the caches with the new larger cache sizes.

Benchmark	Score (small caches)	Score (large caches)
Dhrystone	92165.9	119617.2
BYTEmark Numeric Sort	26.574	33.36
BYTEmark String Sort	1.3667	1.8123
BYTEmark Bitfield	5.6997e+06	5.8613e+06

TABLE 3.2: Benchmark results for small and large caches respectively

3.4 Test Runner

The test runner module provides a memory-mapped interface to control the external tester module. Figure 3.3 shows an overview of the signals of this core.
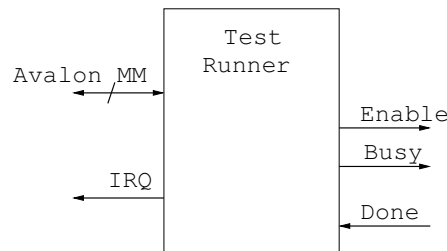
FIGURE 3.3: Black box overview of Test Runner

The Avalon MM Slave interface connects to the SoPC and provides a convenient memory-mapped interface to its internal registers. The interrupt sender interface connects directly to the interrupt controller of the CPU.

The peripheral bus consists of three signals: a `busy` signal which feeds as one of the select signals into the SRAM Arbiter (`sram_arb_sync`), a `enable` signal and a `done` signal which connect to the `tester` module.

Table 3.3 shows the memory map of the core. The system can read an ID from the ID register (which always contains the hexadecimal value 0x0a) to verify that the device is responding. A write to the enable register will assert the peripheral `enable` line for one cycle. A read of the done register returns the value of the `done` peripheral signal.

When a rising edge occurs on the `done` signal, the IRQ register is written and the IRQ line to the CPU is asserted. As soon as the IRQ register is read, it is automatically cleared and the IRQ line is deasserted.

The `busy` signal is asserted between the rising edge of the `enable` signal and the rising edge of the `done` signal.

Address	Description
0x0a	IRQ Register
0x7f	ID Register
0x80	Done Register
0x81	Enable Register

TABLE 3.3: Relative memory map of the test runner module

3.5   ADC Interface

The ADC interface has the exact same interfaces as the Test Runner module as shown
on Figure 3.4. The only difference is its memory map, which is slightly different and
shown in Table 3.4.
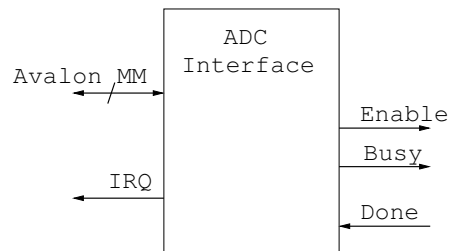
FIGURE 3.4: Black box overview of Test Runner

Address	Description
0x0a	IRQ Register
0x0f	ID Register
0xa0	Done Register
0xb0	Enable Register

TABLE 3.4: Relative memory map of the adc interface module

3.6 Frequency counter

3.6.1 Overview and Specification

The frequency counter module of the Superchip tester is connected to the oscillator found in the superchip designs, to measure its output frequency.

According to specification for the academic year $2011 - 12$, the ring oscillators found in the teams' superchip designs have a basic frequency of $1MHz$ and an offset frequency of the teams' ID number multiplied by $250kHz$ (**?**). For 16 teams, the frequency range would be $1250kHz - 5000kHz$.

3.6.2 Functionality

Since the clock of the FPGA is significantly faster than the oscillator frequency, the frequency counter designed for this project counts the number of the oscillator signal edges over a given number of FPGA clock cycles. The frequency counter consists of three modules, a $24-to-1$ multiplexer, the measurement module and a control module. When the frequency counter is enabled, the processor provides it with the required number of clock cycles. The counter starts counting the rising edges of the input signal and after it stops counting, once the specified number of clock cycles has been reached, it raises an interrupt signal for the data to be read from the processor. While the active-low $nReset$ signal is enabled, the counter ceases all function and all registers and outputs are nulled.

Multiplexer   The purpose of the multiplexer is to connect the measurement module to the correct pin of the chip under test, to achieve better reconfigurability. In order to provide compatibility of the design with chips whose oscillator output can be different, the multiplexer is connected to all design outputs and only connects the one specified by the processor to the measurement module.

Measurement Module   The measurement module of the frequency counter provides the main functionality of the design, measuring the number of input signal periods over a specified time. This module is provided with the target number of clock cycles and the input signal (to be measured). In order to maintain signal integrity, since the input signal is effectively crossing a clock domain, the design contains a synchroniser stage, to synchronise the input signal with the FPGA clock. When the module is enabled, the edge counter increments its value on every rising edge of the input signal. Once the module has gone through the required clock cycles, it stops counting input signal periods and an $count\_overflow$ flag is raised.

Control Module   The frequency counter control module handles the communication and the input setup data from the processor, as well as the the output data from the measurement module. It provides an Avalon Memory Mapped slave interface for communication with the NIOS II processor as well as an Avalon Interrupt signal interface (*IRQ*) and a *busy* signal.

Before starting the frequency measurement procedure the NIOS core writes data to the frequency counter, to specify parameters such as the duration of the measurement. When the *write* signal in the Avalon interface is active, the register specified on the address bus is written with the data on the data bus.

The internal *enable* signal to start the process is activated by the controller when a dedicated register is written with a unity value and kept high for a single clock cycle. As long as the frequency counter is enabled and counts periods of the input wave, the controller keeps the *busy* signal high to inform the NIOS II core that it is processing.

The data stored on the registers of the controller are kept steady and provided to the other modules of the frequency counter. Once the measurement module is done and the *done* flag is enabled, the controller stores the count value to the relevant register and raises the *irq* signal to the processor.

Following an interrupt request, the processor starts reading values from the registers of the frequency counter module. When the *read* signal in the Avalon interface is active, the module outputs the data on the register specified on the address bus to the processor. Meanwhile, when data is read from the registers, the *readdatavalid* signal is high as well. When the data contained in the registers is read after a calculation procedure and an interrupt request to the processor, the *irq* signal is cleared.

3.7   Resource usage

Table 3.5 shows a summary of the FPGA resource usage of the SoPC. The table includes only the largest modules, and a total of all modules. In total the SoPC uses 17,453 logic cells of the 115,200 available on the FPGA.

Particularly interesting is the large number of cells used by the interconnect. About half of these come from the clock crossing adapters to cross from the 100 MHz to the 10 MHz clock domain.

Module	Logic cells (comb)	Logic cells (reg)	DSP Elements	Memory bits
CPU	3852	2822	4	877,056
Interconnect (estimate)	2566	2144	0	0
TSE MAC	2198	2701	0	298,416
SG DMAs	1202	1542	0	2,745
SDRAM Controller	331	338	0	0
SPI0	111	117	0	0
SPI1	110	115	0	0
Timer	130	120	0	0
UART	129	101	0	0
JTAG UART	142	112	0	1024
Test Run-ner	863	1036	0	0
ADC Inter-face	142	140	0	0
Frequency counter	334	298	0	0
Total	13754	11692	4	1,218,729

TABLE 3.5: Resource consumption by SoPC module (only the largest modules are shown), and total (including all modules). Note that the total logic cell usage is not a sum of the combinational and register logic cells, since some logic cells contain both combinational logic and registers. The total number of logic cells used by the SoPC is 17,453

Chapter 4

Embedded Software

_____

4.1 Bootloader and Operating System

To take full advantage of the MMU, Linux was chosen as operating system running on the application processor. uCLinux was the distribution of choice since it is aimed at embedded systems and optimum resource usage. A Linux 3.3-rc6 kernel was used.

Since the Linux kernel can only boot from memory, a bootloader was required to load the Linux kernel image from the non-volatile CFI Flash into SDRAM. The bootloader used is a current development version of U-Boot.

4.1.1 CFI Flash Layout

The CFI Flash was partitioned into several MTD partitions. Table 4.1 shows a map of the MTD partitions and their respective size.

Address Range	Size (kB)	Description
0x00000000 - 0x0003ffff	256	U-Boot
0x00040000 - 0x0004ffff	64	U-Boot configuration
0x00050000 - 0x001fffff	1728	Linux Kernel Image
0x00200000 - 0x0077ffff	5632	Linux Root Filesystem
0x00780000 - 0x007fffff	512	Misc

TABLE 4.1: Partition Layout of CFI Flash Memory

The U-boot bootloader image is located in the first 256 kB, of which around 170 kB are used by u-boot. The CPU reset vector points to this address, so that u-boot always is

loaded on CPU reset. The configuration for U-boot that contains information on how to boot the Linux system is located at the next 64 kB. Only 318 bytes are currently used.

The compressed Linux kernel image is located just after the u-boot configuration. The actual size currently used by the Linux kernel is around 1200 kB. This image is loaded into memory and decompressed by U-boot upon startup.

The root filesystem of the Linux system is located in the largest partition. It is a JFFS2 file system containing the complete uClinux userland. It currently occupies around 4000 kB.

The last partition is currently unused.

4.1.2 Bootloader

The u-boot bootloader is located at the first 256 kB of the CFI Flash. The CPU's reset vector points to it, so that upon startup of the CPU, u-boot is executed.

U-boot prompts the user to interrupt automatic boot during a 5 second countdown. If a user interrupts this process by connecting via UART and pressing a key, the u-boot prompt appears. Using this prompt u-boot's configuration can be modified and stored.

When booting automatically, U-Boot reads the configuration of how to boot from the u-boot configuration on the flash. U-Boot will first read and decompress the Linux kernel into memory and then pass control to it.

A total of 3 small patches including bugfixes were submitted upstream to the U-Boot development community to address a number of issues. 2 of the 3 patches were committed. One of the patches fixed a bug in the memory allocation[1], another implemented some timer functions[2] and the last improved the logic around choosing JTAG UART or UART on the Nios II platform[3] but was not accepted.

4.1.3 OS

A development snapshot of the uClinux distribution and Linux 3.3.0-rc6 kernel is used as the application operating system. uClinux is a distribution aimed at embedded systems with a low footprint. It replaces the GNU glibc with the much smaller uClibc and several core unix tools with the minimalistic busybox toolsuite.

In its current configuration, the kernel uses up around 1.2 MB of space, while the root filesystem weighs in at around 4MB with a JFFS2 filesystem. Both are compressed.

The booted system used only 10 MB of physical memory, leaving almost 120 MB to running programs. No swap/paging space was configured, but since the MMU is enabled, program text pages are paged in on demand and backing physical memory pages are also only allocated on access.

---

[1] http://lists.denx.de/pipermail/u-boot/2012-February/118453.html
[2] http://patchwork.ozlabs.org/patch/142141/
[3] http://patchwork.ozlabs.org/patch/142142/

A patch fixing an issue with SD Card interfacing over SPI was submitted upstream to the Linux MMC team[4].

Due to a bug in the linux kernel in the handling of variable block-size CFI Flash chips, some of the MTD partitions on the CFI Flash are forced to be read-only, even though they are aligned to valid sector boundaries.

Another limitation of the Linux system as is is that it's not possible to use the on-board USB chip ISP1632 in USB device mode, so it's not possible to connect the board running Linux as a peripheral to a host machine.

The boot log of the embedded system including both the u-boot bootloader and the Linux system can be seen in Figure 4.1.

4.1.4 Device Tree

The current generation of Linux kernels can use so-called device tree files to specify the address range, interrupts and several other options of available peripherals. A device tree file is simply a list of all the modules connected to the system and their configuration. It is possible to specify each module's type, memory address range, interrupts, etc. Figure 4.2 shows an excerpt of a devicetree file showing the configuration for an altera SPI master, an SD card connected to it, and the custom test runner module.

The memory mapped address range is specified using the `reg` keyword, the connected interrupt numbers are specified using `interrupts` and `interrupt-parent`. The `compatible` keyword is used to identify the device type so that a given driver with the same compatibility can attach to it.

---

[4]http://comments.gmane.org/gmane.linux.kernel.mmc/12835

```
U-Boot 2011.12-00465-ge37ae40-dirty (May 14 2012 - 20:20:10)

CPU   : Nios-II
SYSID : abcdef00, Tue May 15 16:25:13 2012
BOARD : my_nios2

Hit any key to stop autoboot:  0
## Booting kernel from Legacy Image at ea050000 ...
   Image Name:   Linux-3.3.0-rc6
   Image Type:   NIOS II Linux Kernel Image (gzip compressed)
   Data Size:    1262781 Bytes = 1.2 MiB
   Load Address: c0000000
   Entry Point:  c0000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Linux version 3.3.0-rc6 (alex@alex-pc) (gcc version 4.1.2) #25 We49.47 BogoMIPS (lpj=98944)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
gpiochip_add: registered GPIOs 246 to 246 on device: /sopc@0/bridge@0xb000000/gpio@0x220
gpiochip_add: registered GPIOs 245 to 245 on device: /sopc@0/bridge@0xb000000/gpio@0x200
gpiochip_add: registered GPIOs 244 to 244 on device: /sopc@0/bridge@0xb000000/gpio@0x210
gpiochip_add: registered GPIOs 243 to 243 on device: /sopc@0/bridge@0xb000000/gpio@0x230
JFFS2 version 2.2. (NAND)  2001-2006 Red Hat, Inc.
msgmni has been set to 241
Block layer SCSI generic (bsg) driver version 0.4 loaded (major 254)
io scheduler noop registered
io scheduler deadline registered (default)
ttyAL0 at MMIO 0xd100000 (irq = 11) is a Altera UART
console [ttyAL0] enabled, bootconsole disabled
ttyJ0 at MMIO 0x80034b0 (irq = 2) is a Altera JTAG UART
altera_sysid ba00000.sysid: System creation hash ABCDEF00 timestamp 2012-05-15 16:25:13
Test Runner Module loaded
trunner d000000.trunner: Test Runner device 0, irq 7
DE2LCD Module loaded
de2lcd b000010.de2lcd: DE2 LCD 0
Frequency Counter Module loaded
fcounter d200000.fcounter: Frequency Counter device 0, irq 12
ADC Module loaded
adc d300000.adc: ADC device 0, irq 13
a000000.flash: Found 2 x8 devices at 0x0 in 16-bit bank. Manufacturer ID 0x00703a Chip ID 0x000001
NOR chip too large to fit in mapping. Attempting to cope...
Amd/Fujitsu Extended Query Table at 0x0040
  Amd/Fujitsu Extended Query version 1.3.
number of CFI chips: 1
Reducing visibility of 16384KiB chip to 8192KiB
5 ofpart partitions found on MTD device a000000.flash
Creating 5 MTD partitions on "a000000.flash":
0x000000000000-0x000000040000 : "u-boot"
0x000000040000-0x000000050000 : "u-boot_cfg"
mtd: partition "u-boot_cfg" doesn't end on an erase block -- force read-only
0x000000050000-0x000000210000 : "kernel"
mtd: partition "kernel" doesn't start on an erase block boundary -- force read-only
0x000000200000-0x000000780000 : "rootfs"
0x000000780000-0x000000800000 : "config"
spi_altera ba10000.spi: base eba10000, irq -6
spi_altera ba20000.spi: base eba20000, irq 8
altera_tse-mdio: probed
eth0: Altera TSE MAC at 0xe8003000 irq 4/3
eth0: Reporting available PHYs:
eth0: PHY with ID 0x1410cc2 at 0x10
Clock not ready after 100ms
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
mmc_spi spi32766.0: SD/MMC host mmc0, no DMA, no WP, no poweroff, cd polling
TCP cubic registered
NET: Registered protocol family 17
Freeing unused kernel memory: 3292k freed (0xc0218000 - 0xc054f000)
Preparing GPIO
Starting syslogd
Starting dhcpcd
Starting crond
Welcome to Team I's
    _____ _      _    _____        _
   / ____| |    (_)  |__   __|      | |
  | |    | |__   _ _ __  | | ___ ___| |_   ___ _ __
  | |    | '_ \| | '_ \| |/ _ | __| __|/ _ \ '__|
  | |____| | | | | | |_) | |  __|__ \ |_|  __/ |
   \_____|_| |_|_| .__/|_|\___|___/\__|\___|_|
                 | |
                 |_|


BusyBox v1.18.4 (2012-05-15 19:19:33 BST) hush - the humble shell
Enter 'help' for a list of built-in commands.

root:/>
```

FIGURE 4.1: Boot log of system

```
spi_0: spi@0xba10000 {
  compatible = "ALTR,spi-11.1", "ALTR,spi-1.0";
  reg = < 0x0BA10000 0x00000020 >;
  interrupt-parent = < &cpu >;
  interrupts = < 0 >;
  #address-cells = < 1 >;
  #size-cells = < 0 >;

  mmc_spi@0 {
    compatible = "mmc-spi-slot";
    spi-max-frequency = < 10000000 >;
    reg = < 0x00000000 >;
    voltage-ranges = < 3300 3300 >;
  }; //end mmc_spi@0
}; //end spi@0xba10000 (spi_0)

test_runner_0: trunner@0xd000000 {
  compatible = "trunner,trunner-1.0";
  reg = < 0x0D000000 0x00000100 >;
  interrupt-parent = < &cpu >;
  interrupts = < 7 >;
}; //end trunner@0xd000000 (test_runner_0)
```

FIGURE 4.2: Example excerpt from a devicetree file

4.2   Build Infrastructure

As the project depends on several external resources such as the u-boot bootloader, Linux kernel and uClinux distribution, a build system was developed to make it easy to integrate all these parts into the main project.

The complete build infrastructure consists only of standard unix Makefiles and shell scripts.

All the external resources (uClinux, uClibc, Linux kernel, u-boot) are set up as git submodules. They are pulled in without any changes at a particular revision from their upstream sources.

To allow for custom patches and integrating custom files, two mechanisms are included in the build system. One of them uses regular patches in the `patchq/` directory and applies them in order to each of the submodules. The other mechanism takes the directories in `overlay/` and overlays them on top of the submodules. This allows for a complete separation of the external resources and the custom patches and files. The external resource can be reset to its default state by yet another makefile target, which cleans out all non-versioned files. The patches and overlays can then be applied cleanly. This also makes it trivial to update or downgrade the external resource without the issues associated with patched files.

4.3   Drivers

The SRAM is accessed directly from userland without a driver by simply memory-mapping the region corresponding to the SRAM into the virtual memory of the running application via the `mmap` system call.

GPIOs are controlled via an interface in `/sys` that the Linux GPIO drivers expose. This interface permits setting, clearing and reading pins by reading and writing from what seem regular files.

Four custom Linux drivers were developed for this project for the ADC, frequency counter, test runner/controller and LCD modules. All four drivers share a common skeleton.

All of them use the platform driver framework and detect their devices based on the devicetree entries. The probe function remaps the IO range, sets up the interrupt (if needed) and creates a character device node (which gets exposed in `/dev`). This character device exposes the interface to the userland applications via a combination of `read`, `write`, `poll` and `ioctl` system calls on these nodes.

The drivers interact with the hardware by reading from/writing to specific memory addresses as specified in the register map for each device using the low level `ioread` and `iowrite` routines.

The drivers of the devices using interrupts (frequency counter, ADC, test runner) register an interrupt handler which clears the interrupt register of the device by reading from it and wakes up all threads sleeping on a given wait queue. This allows userland programs to use the `poll` system call to wait for the device to complete its task without polling and instead sleeping on the waitqueue. As soon as the interrupt handler is called, the thread is woken up and execution continues.

All device drivers use a `ioctl` interface to control the driver from userland. The frequency counter and lcd driver in addition also expose a `read` or `write` interface.

4.3.1   Test Runner and ADC Drivers

Both of these drivers are effectively the same with only minor differences such as the address of the registers to match the differing register maps of the devices, and different

names of the created device nodes and the devicetree entries that they match.

The test runner driver creates a device node `/dev/trunner0` while the adc driver creates the device node `/dev/adc0`. Interaction with both is strictly via `ioctl` and `poll` system calls only.

The `poll` interface implemented is described above - `poll` will sleep on a waitqueue until an interrupt wakes it up.

The `ioctl` interface comprises only three messages. The `ADC_IOC_ENABLE` and `TRUNNER_IOC_ENABLE` `ioctls` enable the peripheral by writing to the enable register of the device.

The `ADC_IOC_GET_DONE`, `TRUNNER_IOC_GET_DONE`, `ADC_IOC_GET_MAGIC` and `TRUNNER_IOC_GET_MAGIC` read the done and ID (magic) registers of the devices, respectively.

4.3.2   Frequency counter driver

The frequency counter driver registers a character device node under `/dev/fcounter0` with a `poll`, `ioctl` and `read` system call interface.

The behaviour of `poll` and the `FCOUNTER_IOC_ENABLE` and `FCOUNTER_IOC_GET_MAGIC ioctls` is the same as for the test runner and ADC drivers.

In addition to these, the frequency counter driver provides three additional `ioctls`. The `FCOUNTER_IOC_SET_IPSEL` ioctl writes to the input select register, effectively choosing which of the input pins is multiplexed onto the frequency counting logic. The `FCOUNTER_IOC_SET_CYCLES` ioctl write to the cycle timeout count register. This defines for how many cycles of the host clock the frequency counter will be counting edges on the input signal.

The `FCOUNTER_IOC_GET_COUNT` ioctl and the `read` system calls both read the counted edges register of the device. By knowing the host frequency and the ratio between this edge count and the cycle timeout count, the frequency of the input signal can be calculated.

4.3.3 DE2 LCD Driver

The DE2 LCD character driver registers a device node under `/dev/de2lcd0`. This driver exposes both a `ioctl` and a `write` system call interface.

Writing to the device node using the `write` system call will write the provided text to the offset at which the `write` occurred. The LCD has 2 lines of 16 visible characters, but each line is actually 40 characters long. Hence a write can be a maximum of 80 characters long.

The driver exposes several `ioctl` commands. The `DE2LCD_IOC_CLEAR` `ioctl` clears the LCD display of all characters. The `DE2LCD_IOC_CURSOR_ON` and `DE2LCD_IOC_CURSOR_OFF` commands turn the blinking cursor on and off, respectively. The final command, `DE2LCD_IOC_SET_SHL` enables and disables shifting of the characters on the LCD. The parameter passed to this last command is the time between shifts in ms, or 0 to disable shifting. When a non-zero value is passed in, the driver registers a function to be called regularly by the kernel timers. This function shifts the display left each time it is called. When a zero value is passed in the timer is deactivated again. This effectively allows displaying all 40 characters on each line of the LCD instead of just the 16 visible ones.

4.4   Configuration Format

The configuration format used by the software is an easy to read, easy to write and very lenient format.

Configuration is loaded from an SD Card. The SD Card should have a directory layout as shown in Figure 4.3. The top-level `chiptester` file can be empty and only indicates to the system that this SD Card contains test vectors.

The top level directory should include a subdirectory for each team. Each team needs to provide at least a `team.cfg` file in their subdirectory. They can also provide an arbitrary number of test vector files which need not follow any naming convention (e.g. `foo.bar` is also a valid vector file).

```
.
|- chiptester
|-- team1
|    |-- adder.vec
|    |-- shifter.vec
|    |-- team.cfg
|-- team2
|    |-- divider.vec
|    |-- team.cfg
|-- team3
     |-- multiplier.vec
     |-- team.cfg
```

FIGURE 4.3: Example directory layout

4.4.1   team.cfg

The `team.cfg` file has four different keywords (team, email, academic_year and base_url, all except the team and the base_url can be omitted). These keywords, their arguments and their use are explained in Table 4.2.

```
team: 3
email: foo@bar.com, baz@bar.net
academic_year: 2011/12
base_url: http://192.168.0.10:4567
```

FIGURE 4.4: Example team.cfg

A keyword can be preceded by any number of whitespaces (tabs or spaces). It must be followed by a colon, but it is valid to have an arbitrary number of whitespaces between the keyword and the colon. After the colon any number of whitespaces and newlines can follow before the keyword-specific content (argument). Another example shown in

```
# This is a comment
  team: # This is also a comment
    3
  email:
    foo@bar.com, baz@bar.net

academic_year : 2011/12
base_url      : http://192.168.0.10:4567
```

FIGURE 4.5: Another example team.cfg

Figure 4.5 shows most of these features in use. The parser is however case sensitive.

All characters after a hash (#) are treated as comments.

Keyword	Arguments	Description
`team`	<decimal number>	The team number; saved in the database
`email`	<string>	The email addresses, comma separated, to which to send the results of the test
`academic_year`	<string>	The academic year; saved in the database
`base_url`	<string>	The URL to the Chip Tester web server. This is used internally to send results to the backend and database

TABLE 4.2: Valid team.cfg keywords, their arguments and meaning

4.4.2   Test vector files

The parser used for the test vector files is the same as for the `team.cfg`. As a result, it shares the same leniency - keywords can be prefixed by whitespaces, the colon after a keyword can be prefixed by whitespace, the argument(s) after a keyword, colon, can follow after any number of whitespaces and newlines.

Every line after a keyword that does not contain a keyword by itself is parsed using the same parser as the previous keyword. In other words, every line following a keyword is parsed by the same keyword specific parser unless a new keyword appears.

A special kind of keyword can appear inside lines parsed by the `vectors` line parser. These keywords begin with a dot (.) and are not followed by a colon. These are parsed by a separate parser, but the default line parser is not changed, so that the line following them is still parsed by the `vectors` parser.

Table 4.3 describes all normal keywords while table 4.4 describes all dot commands. Figure 4.6 shows an example test vector file.

```
design: 1-bit shift left
clock:     A23
frequency: 1
trigger:
          Q23, Q22
pindef:
       A3, A2, A1, A0, Q3, Q2, Q1, Q0
vectors:
       0001 0010
       0001 T 0010
       0010 0100
.measure adc
       0011 0110
       0100 0100
       0100 1000
.measure frequency Q1
       0101 1010
       0110 1100
       0110 110X
.measure frequency Q2
       0111 1110
       1000 0000
       1000 1000
       1001 0010
       0100 W5 0111
       0001 T 1010
```

FIGURE 4.6: Example test vector file

This example file contains test vectors for a design called "1-bit shift left". The design will be tested at 1 MHz, and this 1 MHz clock will be connected to pin A23. The triggered test vectors will complete when Q23 and Q22 are asserted. The pins used in the

test vectors are A3, A2, A1 and A0 for the input test vectors, and Q3, Q2, Q1, Q0 for the results.

Most test vectors are simple 1-cycle test cases where the result one cycle after applying the input should match the given output. It also contains two triggered test cases which complete whenever the trigger condition of both Q23 and Q22 being asserted is met, or alternatively, after the default timeout of 32 cycles. Another test checks that the output 5 clock cycles after applying the vector `0100` is `0111`.

This file also contains three measure commands. The first activates the ADC module and stores the captured result to the backend. The following two measure the frequency on pins Q1 and Q2 respectively. The default timeout of 2^{24} cycles is used so that with the 100 MHz system clock the measurement completes in 0.16 seconds, giving a resolution down in the tens of Hz.

Keyword	Arguments	Description
`design`	\<string\>	A string describing the design to be tested (e.g. "4-bit adder")
`frequency`	\<decimal number\>	The frequency at which the device is tested. Minimum is 1, maximum is 100. The unit is MHz. Default is 10 MHz.
`clock`	\<pins\>	A list of (input) pins to which the clock signal is connected.
`trigger`	\<pins\>	A list of (output) pins which need to be asserted (all of them) for a triggered test to complete without a timeout
`pindef`	\<pins\>	An list of input and output pins for which vectors and expected results are provided in the vectors section
vectors	\<binary digits or X\>	A list of binary digits making up a test vector and expected result. first digit will be applied or checked against the first pin in the `pindef` and so on. An X can be specified on an output pin, meaning that that pin is ignored when checking the result of that particular test vector
	or T[\<number\>]	`T` specifies that this is a triggered test case which only completes after either the triggers specified with `trigger` are matched or a timeout, which by default is 32 cycles but can be overriden by providing a cycle number immediately after the T.
	or W\<number\>	`W` specifies that this is a fixed latency test case, whose result will be checked after the number of cycles specified. If neither T nor W are specified, then the implicit default is equivalent to `W1`

TABLE 4.3: Valid keywords, their arguments and meaning

4.5   confrd

The confrd program is the main application running on the embedded Linux system. It is written in C for performance and memory footprint reasons. It parses configuration structures such as the one outlined in the previous section, parsing each file and controlling the peripherals such as the test runner, frequency counter and ADC module.

The confrd program can also be compiled on a normal Linux host system. In this case its only use is to verify the syntax of configuration files. When the confrd program is invoked without the `-w` argument it goes through the provided directory structure and just parses all the files, verifying their syntax. If the `-p` flag is specified, then it will also print out a summary of everything it would store to SRAM for use with the tester

Keyword	Arguments	Description
`.measure adc`	(none)	The ADC measure command will enable the ADC module after the previous vector has finished and send the results of the capture to the server
`.measure frequency`	\<pin number\> \<timeout\>	This command enables the frequency counter on a given pin number (either just a decimal number, or a decimal number prefixed by Q) for a given number of cycles and stores the result to the server

TABLE 4.4: Valid dot commands

```
Usage: confrd [options] <configuration directory>
Valid options are:
 -p
        Print out the data written to SRAM in a human readable form
        on screen.
 -s <file>
        Write an SRAM initialization file containing the data generated
        using the configuration file(s).
 -v
        Marks this as a virtual design. Only affects remote logging.
 -w
        Write to actual SRAM and start the test runner after reading
        a file.
```

FIGURE 4.7: Usage message of the confrd program

module.

The parser understands and implements the configuration formats specified in the previous section. It was designed with performance in mind, and, as a result, it can parse up to 60000 lines per second on the 100 MHz Nios II system.

In its normal operating mode, with the `-w` flag, the operation is as follows. The program will go through each subdirectory in the directory specified on the command line and look for a `team.cfg` file in each. First this `team.cfg` file is parsed, which contains the address of the Chip Tester backend. Using the information in this file and the URL to the backend, confrd will first create a result entry in the database by using the HTTP API. The id of the newly created object, as returned from the backend, is stored in memory. This id is used to identify subsequent queries for the same result.

It will then parse each of the vector files in the same directory, one after another. For each of the vector files it will create a design result entry in the database via the HTTP

API. The returned id is used to identify further HTTP requests storing the test vector
and measurement results.

Most keywords generate a request memory structure for use by the tester hardware which
is written to a SDRAM memory location of the same size as the SRAM. In the previous
configuration example, the following memory structures/requests would be generated:

- The `team` line in the `team.cfg` generates a change target request

- The `design` line does not generate any request - it only stores the design name to
  send it to the backend/database.

- The `frequency` line generates a PLL Reconfiguration request. The factors (multi-
  plication, division, post-loop counter) for a given frequency are taken from a header
  file which includes pre-calculated values for each integer frequency between 1 and
  100 MHz. These factors are generated by the `pllfreq.rb` support tool described
  in a later section.

- The `clock` line generates a DICMD request with the bitmask generated by the
  individual pins as payload, and the type set to DICMD_SETUP_MUXES.

- The `trigger` line generates a DICMD request with the bitmask generated by the
  individual pins as payload, and the type set to DICMD_TRGMASK.

- The `pindef` line generates a change bitmask request

- Each `vector` line generates a test vector request

Whenever the SRAM staging area is (almost) full, the file has been completely read,
or a `measure` command appears, the SRAM staging area is written to actual SRAM,
followed by a mem end request (to denote end of the test vectors), and the test module
is activated. confrd then sleeps until the test hardware is done using the driver's `poll`
interface. As soon as it is done, it resets the staging area and either continues parsing
test vectors or executes the `measure` command, as appropriate.

Since the DUT interface holds the last input vector until a new one is processed, this
ensures that `measure` commands are executed with a known state on the input pins.

Each time the tester, frequency counter or ADC modules complete, their result is sent
to the backend/database via the HTTP API, identified by the design result id retrieved
from the backend at the start of a vector file.

Any syntax or other error that occurs after the `team.cfg` file has been parsed will be logged remotely to the backend. All errors and some status messages indicating the current progress are printed to the character LCD using the DE2 LCD Driver.

The throughput of the complete process is determined by the upload speed. Both the file parsing and the actual testing on hardware are extremely fast, as is the SRAM access. Without uploading, 60000 tests complete in roughly 10 seconds. With uploading this extends to several minutes.

For verification purposes of the test hardware, the confrd program has an additional flag `-s` which writes what it would normally write to SRAM into a memory initialization file that can be used in the `tester` testbench to initialize the SRAM model. This allows for verification of the `tester` module using actual test vectors as they are generated by the software.

The confrd tool relies on two external libraries; libjansson to encode and decode JSON structures and libCURL to provide a convenient HTTP interface. Both of these libraries are used for the HTTP API accesses to the backend.

4.6 SPI Flash Programmer and Model

The Spansion S25FL064K0SMFI011 memory flash present in the second FPGA has to be programmed through SPI, to program the memory the `spi flash.c` uses the SPI kernel framework present in Linux since kernel version 2.6.27.

The commands that the flash can perform (to see the full set of commands please refer to the data-sheet of the flash memory) that were implemented in the `spi flash.c` are presented in the table 4.5.

Byte	code
0x06	write_enable(int fd)
0x04	write_disable(int fd)
0x50	write_enable_sreg(int fd)
0x05	read_sreg1(int fd, uint8_t *sreg1)
0x35	read_sreg2(int fd, uint8_t *sreg2)
0x01	write_sreg(int fd, uint8_t sreg1, uint8_t sreg2)
0xC7	chip_erase(int fd)
0x03	read_data(int fd, uint32_t addr, size_t len, uint8_t *buffer)
0x9F	read_jedecID(int fd, read_jedec_id *jedec_data)It reads the JEDEC assigned Manufacturer ID byte and two Device ID bytes, Memo
0x4B	read_unique_id(int fd, uint64_t *unique_id)
0x90	read_manufacturer_id(int fd, manufacturer_read *mf_data)
0x02	page_program(int fd, uint32_t addr, uint8_t *data, size_t len)
0xD8	block_64_eraser(int fd, uint32_t addr)
0x52	block_32_eraser(int fd, uint32_t addr)
0x20	sector_eraser(int fd, uint32_t addr)

TABLE 4.5: SPI commands

Using this commands to communicate with the flash through the SPI, the function

```
static int prog_flash_from_file(int spi_fd, int erase_sectors, const char *file_in)
```

is used to program the flash, on the second FPGA, reading from a file (provided by the user when uploading files to the system).

4.6.1 SPI Flash Model

To verify the correct operation of the SPI Flash Programmer, a simple SPI Flash Model was written independently for use with the programmer. It implements the same commands shown in the table 4.5 and the SPI flash programmer communicates to it as if it were a real flash plugged in. This model was written in order to provide verification for the SPI Flash Programmer without having the real flash connected, therefore lowering the debug times when the real board arrives.

4.7   vconfig

The `vconfig.sh` script is executed via an `flock`ed cron job every 10 minutes. By using `flock` there will never be more than one instance running at a time, even if the execution takes longer than 10 minutes.

The `vconfig.sh` script is used to download virtual design packages from the backend, program the slave FPGA and run a battery of tests.

The script will look for an SD Card that contains a file called `vconfig.sh`. This file needs to contain just one line, giving the address of the Chip Tester backend from which it can download configurations. The format is as shown below. It differs from the regular configuration format used for the rest of the system as it is used directly from a shell script without any parsing - it is only "sourced".

```
BASE_URL="http://192.168.0.10:4567"
```

If this file is not present (for example because a different or no SD Card is inserted), `vconfig.sh` will not do anything.

If however the file is present and the URL points to a valid ChipTester backend, it will try to download a virtual design via the HTTP API. If none is available for download it will exit immediately. Otherwise it will download the file to a temporary location in `/tmp`.

Once the file is downloaded it will try to decompress trying all of the following archive formats:

- ZIP archive

- TARball without compression

- TARball with gzip compression

- TARball with bzip2 compression

- TARball with LZMA compression

If all attempts fail, the script will give up and log the error remotely to the backend using the external `rlog` tool mentioned in the next section.

Next the syntax of the decompressed files is verified by running the `confrd` tool without the `-w` flag. If a syntax error occurs it will be logged remotely and the process will be aborted.

After the initial validation of the downloaded files, the configuration of the slave FPGA begins. Using the GPIO interface (described in the previous Drivers section), the `nCE` and `nCONFIG` pins are tied low. Tying `nCONFIG` low will cause the slave FPGA to lose all its configuration and enter a reset state, tri-stating all its I/O pins.

While the FPGA is in this reset state, the SPI Flash programmer `spi_flash` is used to write the `fpga.rbf` contained in the configuration archive to the SPI Configuration Flash on the slave FPGA board. If the process fails, the error is logged remotely and the script exits.

If the configuration was written successfully, the `nCONFIG` pin is asserted, causing the FPGA to begin reconfiguration from the newly written flash. After waiting 10 seconds for the FPGA to finish reconfiguration, the FPGA's `nSTATUS` pin is checked to determine whether an error occured during reconfiguration. If so, the error is logged and the operation is aborted.

Otherwise the `confrd` program is invoked in its normal mode with the `-vw` parameter, causing it to perform all the tests in the provided test cases and logging the results remotely as explained in the previous section about confrd. The `-v` flag only changes a flag it sends to the backend when saving the results - it is only used to mark the result as having been performed on a "virtual" design.

4.8 Test and support programs and scripts

4.8.1 de2lcd

The `de2lcd` program is a standalone program using the DE2 LCD driver to control the character LCD. It was used to test the driver and controller for the DE2 LCD, but can also be used as a standalone tool to write to the LCD. It is used from some scripts to write text to the LCD.

```
Usage: de2lcd <options >
Valid options are:
 -c
        Clear LCD.
 -n
        Enable blinking cursor.
 -f
        Disable blinking cursor.
 -s <miliseconds >
        Enable left shift every <miliseconds > ms
 -t
        Test.
 -w <Text >
        Writes <Text > starting at first character of LCD
```

FIGURE 4.8: Usage message of the de2lcd program

4.8.2 fcounter

The `fcounter` program is a standalone program interfacing with the frequency counter module via the fcounter driver. It can show the id number of the device, set the number of timeout cycles (the number of host clock cycles over which the edges of the input signal are counted), select on which input line to measure the frequency and enable the frequency counter. After enabling it, it will only exit after the frequency counter is done. Once it is done, the number of counted edges can be printed.

```
Usage: fcounter <options >
Valid options are:
 -m
        Get magic number.
 -e
        Enable.
 -a
        Print edge count.
 -c <cycles >
        Set timeout <cycles >.
 -s <sel >
        Select <sel > line as input.
```

FIGURE 4.9: Usage message of the fcounter program

4.8.3 rlog

The `rlog` program is a standalone program that takes the URL of the ChipTester back-
end and can log a string remotely to the backend. It is used from several scripts, such as
the `vconfig.sh` script to log errors remotely. Additionally the log level can be specified.
The level can be one of "debug", "info", "warn" or "err".

```
Usage: rlog [-l <level>] -b <base_url> <message>
```

FIGURE 4.10: Usage message of the rlog program

4.8.4 umount2

The `umount2` tool is a simple implementation around the `umount` system call. It was
written because the `umount` tool integrated in busybox failed to unmount on the Linux
3.3 kernel used for this project. As it was felt that the effort to debug the problem with
the `umount` in busybox would outweigh the effort of writing the tool from scratch, the
latter aproach was chosen.

`umount2` simply takes either the path of the device that is mounted or the path a device
is mounted on and unmounts it.

4.8.5 automount.sh

The `automount.sh` tool is invoked by mdev (a busybox tool included in the uClinux
distribution) whenever an SD Card is inserted and removed from the system. It simply
mounts the sd card file system, and if the mounting is successful and the sd card contains
a file named `chiptester` in its root, it will execute the `confrd` program to run the tests
contained on the sd card.

On removal of an SD Card, any running instance of the `confrd` program is killed and
the filesystem unmounted using the aforementioned `umount2` tool.

4.8.6 rc

The `rc` script is the main script executed at startup. This script is based on a default
script shipped with uClinux, but with a number of custom modifications.

It sets the hostname, configures the network via DHCP and sets up temporary memory file systems for several paths such as `/tmp`, `/media` and `/var/log`. The script also starts the syslogd logging daemon and the cron daemon to execute periodic tasks.

It also sets up the GPIO pins to be accessible, and also sets their direction (whether they are input or output pins).

As a final step it uses the `de2lcd` program to write a greeting message to the LCD and also prints a greeting message to the (serial) console.

4.8.7   pllfreq.rb

The `pllfreq.rb` tool is a ruby program that, taking a base frequency (i.e. input frequency to an Altera Cyclone IV PLL), generates a C header file that contains values for the loop multiplier, divider and post-scale counters to achieve every integer output frequency between 1MHz and 100 MHz. The output header file is also annotated with the error between the intended frequency and the actual achieved frequency. The calculation is done while maintaing valid values for the multiplier and divider so that the VCO frequency stays in the valid range.

The generated header file is used by the `confrd` tool to determine which vectors to pass to the test controller to set the desired frequency in the reconfigurable clock domain. Figure 4.11 shows a small excerpt from this file.

```
{ .m = 32,      .n = 5, .c = 5 },   /* 64 MHz, err: 0.0 */
{ .m = 26,      .n = 5, .c = 4 },   /* 65 MHz, err: 0.0 */
{ .m = 33,      .n = 5, .c = 5 },   /* 66 MHz, err: 0.0 */
{ .m = 47,      .n = 5, .c = 7 },   /* 67 MHz, err: 0.1428571428571388 */
```

FIGURE 4.11: Excerpt from the header file generated by the `pllfreq.rb` script

Chapter 5

Backend Software

5.1 Overview

In order to store and show the results sent from the **FPGA** in a database a web server was developed; this server provides the user an interface where they can see and manage the results, upload designs and configuration files to the **chip tester** and administrate the database where the results have been stored. Basically the web server "serves" the content to the user through the Internet to a web page, where the users (administrators or students) can check the results of their designs or manage the results stored in the database. The users load their designs into the web server and then whenever the FPGA sends a "download configuration files request" to the server, it sends the files to the FPGA (if available), also the server provides the **FPGA** the methods for storing in the database all the results obtained after a test has been done.

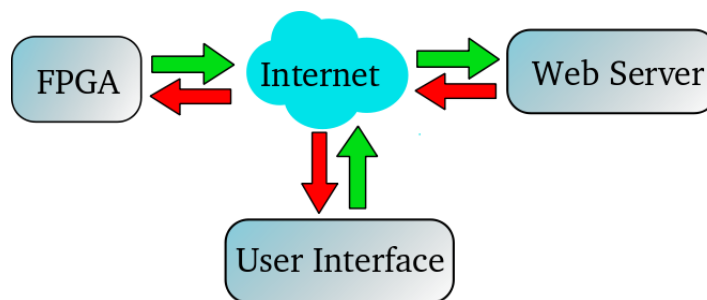
FIGURE 5.1: Interaction between elements in the system

Nowadays almost every programming language provides libraries to develop web servers; for this application we required a robust, easy to use, programming language. Java, Python, PhP, Ruby, Perl are the most common languages used to develop servers. We decided to use Ruby with its **web framework** Sinatra because Ruby is a general-purpose

object-oriented programming language easy to learn and powerful, Sinatra is a Domain Specific Language (DSL) for quickly creating web applications in ruby.

WEBrick is an HTTP server built as part of Ruby 1.8 and it is the default HTTP server when developing an application in ruby, not to mention the core of the application itself, nevertheless, WEBbrick has a major drawback: It does a reverse DNS lookup for every incoming HTTP request on the IP address that makes the request. This means that if the FPGA has no entry for a reverse DNS the server has to wait until the lookup times out (the default timeout in WEBbrick is 5 seconds). This feature can be "easily" changed modifying some native code in the WEBbrick library, nevertheless, it has to be changed whenever the environment is set up again. The solution to this problem is to use Unicorn, Unicorn is an HTTP server that has been designed to be fast and easily configurable, therefore it has the reverse DNS lookup disabled by default and performs the calls from the FPGA in milliseconds.

5.2   Server

```
+server
    +public
        +css
        +img
    +uploads
    +utils
    +views
    +adc_data
-config.ru
-config.yml
-database.rb
-email.rb
-Gemfile
-init.rb
-Rakefile
-s.css
-server.rb
-s.scss
-unicorn.conf.rb
-plot.rb
```

FIGURE 5.2: Folder structure of the server

Inside the folder called **server** are all the files regarding to our back-end application. In the folder **public** are the stylesheets for the web page styling, and the images displayed in the page, in **uploads** are all the files uploaded in the server by the users, in **utils** are some utility scripts written in ruby such as secure password generation, **views** contains all the views in the page that the different users can have access to and in the folder **adc_data** are all the measurements taken by the adc in the tester and uploaded on the server by the FPGA.

5.2.1 Database

Datamapper is an ORM (Object-Relational Mapping) that maps objects written in ruby to any supported database, thus giving freedom to the administrators of the system to use any database system (such as MySQL, SQLite, Postgres,. . .) to choose the database the administrator simply has to change the configuration parameters in the file **config.yml** (found in the server folder) as shown in the figure 5.3 where the administrator has set up the database system with MySql.

```
database:
    type: mysql
    user: root
    password: chip1234
    database_name: ChipTester
    address: localhost
```

FIGURE 5.3: Database system selection using the config.yml file

The figure shows an example of the class Admin written in ruby, this class is mapped into the database by datamapper (the fields created by the ORM in the database are shown in the table 5.1.

```
class Admin
  include DataMapper::Resource
  include BCrypt
  property :email, String, :key => true #Natural primary key
  property :permission, Integer, :required =>true
  property :password_hash, BCryptHash, :required => true
```

FIGURE 5.4: Class admin that will be mapped by the ORM (Datamapper) into the database

The Tables in the database are presented below. In the table Admin 5.1 are stored the administrators of the database. The email field is the email used to log in, the Permission field is the level of authority that each administrator has. It can be either 0 or 1 (where 0 means that it can add more administrators to the system and 1 means that can only manage the database) and the password_hash field is a hash related to the user password (more in the server security section)

Email	Permission	Password_hash
String	Integer	BCryptHash

TABLE 5.1: Admin table in the database

The initial administrator and master Administrator is configured in the archive config.yml as shown in the figure 5.5. For security reasons the password of the initial administrator must be encrypted using the script **pass.rb** located in server/utils. Whenever the server is run by the first time, or the database has been created again this user will be mapped into the database, granting always access to the administrator.

```
#Master email address and password
admin:
    username: prw@soton.ac.uk
    password: $2a$10$KVIbrolv6wwL0CoHYt.L2.EQab4kGVTWdmRgz2YRV4SEe56PQErNG
    permission: 0
#Maximum permission in the databse
```

FIGURE 5.5: Initial Master Admin selection using the config.yml file

The File Upload table 5.2 in the database stores the configuration files that has been uploaded to the server by the students, the file name is a hash of the contents of the uploaded file, this is to prevent students from uploading the same design.

Id	Email	Team	File_name	Is_valid	Created_at	Updated_at
Serial	String	Integer	String	Boolean	Date	Date

TABLE 5.2: File Upload table in the database

The Log entry table 5.3 stores all the logs sent from the FPGA to the server, this table is displayed to the user in the web page. The type of messages that can be stored are "Debug", "Info", "Warning" and "Error", if there has been an error in the vector file uploaded to test the results, the line where the error was found is stored, giving information to the students about their mistake.

Id	Type	Message	File	Line	Created_at	Updated_at
Serial	Integer	String	String	Integer	Date	Date

TABLE 5.3: Log Entry table in the database

The result table 5.4 has an overview of the design uploaded into the FPGA. This table is displayed to the user in the overview view.

Id	Team	Academic Year	Email	Created_at	Updated_at	Virtual	Email sent
Serial	Integer	String	String	Date	Date	Boolean	Boolean

TABLE 5.4: Result table in the database

The table Design Results 5.5 belongs to the Results table and it has specific information related to the test done in the design.

Id	Created_at	Updated_at	Triggers	File_name	Clock_Frequency	Design_name
Id	Date	Date	String	String	String	String

TABLE 5.5: Design Result table in the database

The table Test Vector Result 5.6 belongs to the table Design Result and it has more detailed information (for example the expected inputs vectors and the obtained results) inherent to the design tested on the FPGA.

Id	Type	Input_vector	Expected_result	Actual_result	Cycle_count
Serial	Integer	String	String	String	Integer

Trigger_timeout	Has_run	Fail	Created_at	Updated_at
Boolean	Boolean	Boolean	Date	Date

TABLE 5.6: Test Vector Result table in the database

The Frequency Measurement table 5.7 belongs to the Design result table and has data related to the measurement of the frequency of the digital oscillator on the chip. The frequency field is a float with the value of the measured frequency and the pin field is the defined pin used to measure the frequency.

Id	Frequency	Pin	Created_at	Updated_at
Serial	Float	String	Date	Date

TABLE 5.7: Frequency Measurement table in the database

The ADC Measurement table 5.8 belongs to the Design result table and has data related to the sampling of the digital signal coming from the digital oscillator on the chip.

Id	Created at	Updated at
Serial	Date	Date

TABLE 5.8: ADC Measurement table in the database

5.3   API

The way the FPGA communicates to the server is through the Hypertext Transfer Protocol (HTTP). In the client-server model HTTP works as a request-response protocol. The FPGA and the user web browser act as clients, while the application running on the computer hosting the web site is the server. The FPGA and web browser submit HTTP request messages to the server. The server stores content, provides resources such as configuration files, sends emails, manages the database on behalf of the clients. HTTP defines 9 methods indicating the action to be performed on a resource: HEAD, GET, POST, PUT, DELETE, TRACE, OPTION, CONNECT and PATCH. To develop this application we have used only three of this methods **GET** which retrieves information from the server, **POST** which sends data to the server as part of the request and **DELETE** which deletes resources. The figure 5.6 shows an example of the get method, both in the server and in the browser.

In order to communicate from the FPGA to the server a data exchange language is needed; the most popular data exchange language is "XML", nevertheless, adding the xml parsing libraries in the μClinux would slow down the parsing data process due to the complexity of the xml libraries. JSON, is a lightweight data exchange library supported in C (for the FPGA) and ruby (for the Server) therefore it was perfect for our application. The figure 5.7 shows an example of a message in the JSON format that would be sent from the FPGA to the server.
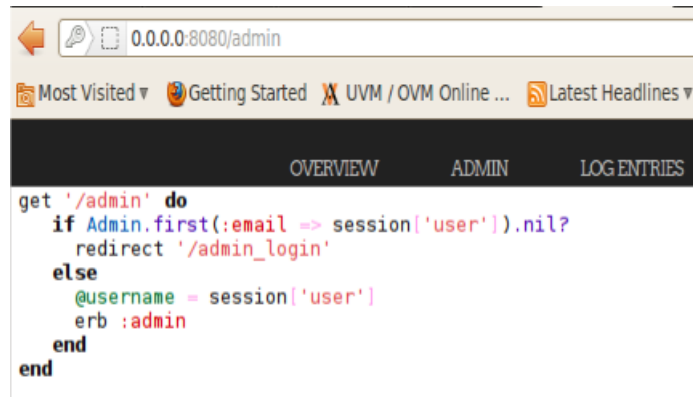
FIGURE 5.6: Example of the HTTP method GET in the browser and in the server

FIGURE 5.7: Example of a JSON message

The table 5.9 shows the "URL" the FPGA uses to connect to the server, and a brief description of what each command does in the server. :result_id, :design_id, and :name are parameters attached in the "URL" that ruby uses as information.
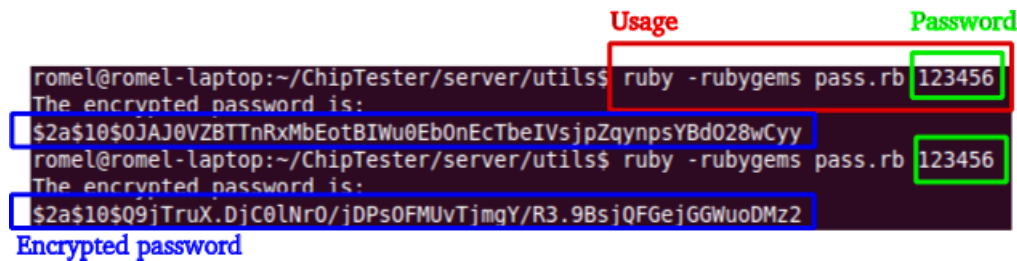
5.3.1 Security

Whenever in an application is stored any user password security must be added. For instance a user could have the same password for our application than for their email, thus, if an attacker gains control over our database and the passwords are unprotected the consequences would be terrible. The solution to harden the security in the application is to create a digital "fingerprint" or hash of the password and store it in the database because this process is not reversible (in other words you cannot go back from hash to password). Ruby offers a library to manage secure passwords storage called BCrypt, usually hashing algorithms are designed to be quick, therefore, attackers can generate more possible passwords in less time (as computing power increases more and more). Nevertheless, Bcrypt has been designed to be computational costly.

The figure 5.8 shows the use of the script found in the folder **/utils** "pass.rb" to encrypt the initial passsword of the database. As can be seen the two encrypted passwords are different, this is because everytime Bcrypts encrypts a password adds some random data (called salts) wich increase the protection of the password against attackers. Also note

Type	Link	Command
post	/api/result	Stores the results sent from the fpga in the table Results in the database
post	/api/result/:result_id/design	Stores in the table Design_Results associated to the result (in the table Results) identified with the id: result_id (results summarized)
post	/api/result/:result_id/design/:design_id /measurement/frequency	Stores in the table FrequencyMeasurement associated to the design_result identified with the id design_id (information related to the frequency of the oscillator in the design)
post	/api/result/:result_id/design/:design_id /measurement/adc	Stores in the table AdcMeasurement associated to the design_result identified with the id design_id (information related to the sampled oscillator in the design)
post	/api/result/:result_id/design/:design_id /vector	Stores in the table Test_vector associated to the design_result identified with the id design_id (more information about the results of the test)
post	/api/done/:result_id	It sends the email to the students when everything is done
get	/api/vdesign	Download a configuration file into the fpga
delete	/api/received/:name	Informs the Server that the file has been received properly and can be erased.

TABLE 5.9: API for the communication between the FPGA and the server

that both encrypted passwords start with "$2a$10$..." where "2a" is the version of the hashing algorithm used to create the hash and "10" is the cost factor used to generate the password. This cost factor was designed to cope with "Moore's Law" due to the fact that computers get faster and faster so an atacker can attempt to decipher the password in less time with more powerful hardware. So, increasing the cost factor also increases the time the hash takes to generate,

FIGURE 5.8: Encryption of the initial admin password

5.4 E-Mails

Once the FPGA has finished testing the chip and sent the results to the server, it sends a request to the server to send the results by email. The configuration of the mail account is shown in the figure 5.9

FIGURE 5.9: Configuration for sending out emails from the server

This configuration can be adapted to any email account. In the folder **views** is a file called "email_body.erb". This file has the template body of the email to be sent and it can be changed easily whenever the contents of the email must be changed.

5.5 Remote Reconfiguration

5.6 User Interface

In order to provide an interface between the Superchip Tester, the database and the user, a webpage has been set up to provide the required functionality. The website provides access to the results database, but also allows the user to upload configuration files for the device.

The design of the webpage is based on a website template found on CSS Templates, altered to provide a look and functionality suitable for interface to the chip tester.

The webpage is laid out in a user-friendly and simple manner, with a navigation bar at the top of the page through which the user can browse the pages of the chip tester

interface. The four entries in the navigation bar are *Overview, Admin, Log Entries* and *Upload Files.* In the main part of the screen, the content of the current view is displayed.

FIGURE 5.10: The ChipTester webpage showing an overview of the database.

In figure 5.10, a screenshot of the website showing the first navigation option, *Overview.* This view takes the user to an overview of the database where the superchip test results are stored. The table displays information on completed tests such as the number of the team, information regarding when the test was run and the relevant academic year and the result of the test.

Clicking on the result of an entry takes the user to a more detailed view of that specific run. In this view the user can also see the measured frequency of the oscillator of the design and the name of the design. If the design has failed the test, clicking on the failed result of the test takes the user to another view, where they can monitor the failed test in more detail. This page provides information on the failed test, to help identify the reason of the failure. In specific, the user can see information about the test run, including the input vector to the design and a comparison between the expected and the actual result.

The second option of the navigation bar, *Admin* prompts the user to enter their administrator credentials and takes them to the administrator view page. From this page, an administrator can reset the database, e-mail the results to a specified e-mail address, go to a detailed view of the database or add a new administrator to the system.

more stuff about manage database

Choosing the third option on the navigation bar, *Log Entries*, take the user to yet another webpage that I can't explain what it does.

The last option of the navigation bar, *Upload Files* allows the user to upload a configuration file to the server. The user is asked to enter their e-mail and their team number and a file to upload.

Chapter 6

Some other yet-unnamed section

6.1 Detailed resource usage on FPGA

stuff.

6.2 Rough bill of materials (or more like, cost of each board)

more stuff.

Appendix A

Pinout Tables

Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name
39	ENCODE	48	A0	50	A1
54	A2	56	A3	60	A4
62	A5	66	A6	68	A7
72	A8	74	A9	78	A10
80	A11	96	A12	98	A13
102	A14	104	A15	108	A16
110	A17	114	A18	116	A19
120	A20	122	A21	126	A22
128	A23	132	Q0	134	Q1
138	Q2	140	Q3	144	Q4
146	Q5	150	Q6	152	Q7
156	Q8	158	Q9	157	Q10
155	Q11	151	D0	149	D1
145	D2	143	D3	139	D4
137	D5	133	D6	131	D7
127	Q12	125	Q13	121	Q14
119	Q15	115	Q16	113	Q17
109	Q18	107	Q19	103	Q20
101	Q21	97	Q22	95	Q23
91	SEL1	89	SEL2	85	SEL3
83	SEL4	79	PWRDWN	45	VCCIO
51	VCCIO	57	VCCIO	63	VCCIO
69	VCCIO	75	VCCIO	81	VCCIO
87	VCCIO	93	VCCIO	99	VCCIO
105	VCCIO	111	VCCIO	117	VCCIO
123	VCCIO	129	VCCIO	135	VCCIO
141	VCCIO	147	VCCIO	153	VCCIO
159	VCCIO	161	GND	162	GND
163	GND	164	GND	165	GND
166	GND	167	GND	168	GND
169	GND	170	GND	171	GND
		172	GND		

TABLE A.1: Pin-outs for the HSMC header on B1 board.

Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name
35	TCK	36	TMS	37	TDO
38	TDI	39	Ded_Clock	41	CONF_DONE
42	nSTATUS	43	nCONFIG	44	nCE
48	IO1	50	IO2	54	IO3
56	IO4	60	IO5	62	IO6
66	IO7	68	IO8	72	IO9
74	IO10	78	IO11	80	IO12
84	MOSI	86	MISO	90	nCS
92	SCK	96	IO13	98	IO14
102	IO15	104	IO16	108	IO17
110	IO18	114	IO19	116	IO20
120	IO21	122	IO22	126	IO23
128	IO24	132	IO25	134	IO26
138	IO27	140	IO28	144	IO29
146	IO30	150	IO31	152	IO32
156	IO33	158	IO34	157	IO35
155	IO36	151	IO37	149	IO38
145	IO39	143	IO40	139	IO41
137	IO42	133	IO43	131	IO44
127	IO45	125	IO46	121	IO47
119	IO48	115	IO49	113	IO50
109	IO51	107	IO52	103	IO53
101	IO54	97	IO55	95	IO56
91	IO57	89	IO58	85	IO59
83	IO60	79	IO61	77	IO62
73	IO63	45	VCCIO	51	VCCIO
57	VCCIO	63	VCCIO	69	VCCIO
75	VCCIO	81	VCCIO	87	VCCIO
93	VCCIO	99	VCCIO	105	VCCIO
111	VCCIO	117	VCCIO	123	VCCIO
129	VCCIO	135	VCCIO	141	VCCIO
147	VCCIO	153	VCCIO	159	VCCIO
161	GND	162	GND	163	GND
164	GND	165	GND	166	GND
167	GND	168	GND	169	GND
170	GND	171	GND	172	GND

TABLE A.2: Pin-outs for the HSMC header on B2 board.

Bibliography

Altera Corporation. High Speed Mezzanine Card specification, June 2009. Revision 17.

Altera Corporation. *Cyclone III Device Handbook*, volume 1. December 2011.

Spansion. Connecting Spansion SPI Serial Flash to Configure Altera FPGAs, 16 November 2011. Revision 4.

ST Microelectronics. LD1117xx datasheet, Adjustable and fixed low drop positive voltage regulator, 13 February 2012. Datasheet.

Texas Instruments. TPS782 150mA, Ultra-low Quiescent Current, Low-Dropout Linear Regulator Datasheet, September 2008.

The University of Southampton. D2 Specification 2011-12, 2011.

TXC. Oscillators 7X5mm SMD CMOS CXO 7W Series Datasheet.