NTNU – Trondheim
Norwegian University of
Science and Technology

FPGA Based Real-time Systems Tester

Diaa Jadaan

Embedded Computing Systems
Submission date: July 2015
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

To Father, Mother, Alaa, Israa and Baraa

Summary

Real-time systems can be defined as systems in which correctness depends not only on the logical result of computation, but also on the time at which the results are produced [22]. Therefore, it is important to verify that a real-time system produces the right result at the right time. There are several techniques to implement time analysis of real-time systems, but they often have a great effect on the execution or involve platform dependent tools that are relatively expensive.

The solution this project presents is a low-cost and platform independent system for testing real-time systems. Due to the lack of similar solutions, university labs, like the real-time systems lab of subject TTK4147, use low-cost software-based testing devices with low accuracy and low customizability. This work attempts to solve that problem and provide the students with high quality hands-on experience to increase their understanding of real-time systems, especially that it is design to be compatible with a popular embedded system as a standard System Under Test, which is the Raspberry Pi.

In addition, the platform is fully reconfigurable making it possible to use in various kinds of projects thanks to the added user peripherals and extra GPIO pins available. This makes the tool valuable in hobbyist environments as well.

This work is based on the master thesis of Kyrre Gonsholt [15] where he developed an IP to perform this kind of tests. This report will first give a few details about the previous work and then explain how it was analyzed to correct its malfunctions. Then some key points of the design process will be discussed with special emphasis on protection. After that, the overall structure of the used software will be presented followed by a guide for the usage and development of the real-time tester. Finally, the report will shed light on some of the issues faced during development and suggest a few recommendation for future system development.

This project has successfully met its goals and produced a fully functional real-time testing platform accompanied by a graphical user interface that allows the user to configure and run the tester in addition to performing analysis of the resulting log data. The tester features a resolution of 20ns, a maximum logging frequency of 12.5MHz and it can generate interrupts with a frequency up to 10MHz. These features ensure that this tool will provide labs with a reliable testing solution that is both high-quality and low-cost.

Acknowledgments

This project would not have been possible without the support of many people. My greatest gratitude goes to my supervisor, Amund Skavhaug, for his guidance, valuable feedback and numerous coffee invites throughout this semester. I would also like to thank the staff at the Department of Engineering Cybernetics for extending my deadline when I had unforeseen problems. Also thanks to the people at the Electronics and Prototyping Lab for their help with my PCB assembly. Many thanks to the EMECS program and its coordinator, Prof. Wolfgang Kunz, for this great master's course that allowed me to learn a lot and meet fantastic people.

And finally, thanks to my family back in Syria who provided me for the last two years with the spiritual support that gave me the strength to work hard and finish my master.

Table of Contents

List of Tables

List of Figures

Abbreviations

ALM	Adaptive Logic Module
AS	Active Serial
BTS	Boundary-scan Testing
CPLD	Complex Programmable Logic Device
DIP	Dual Inline Package
DUT	Device Under Test
EOS	electrical overstress
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FPP	Fast Passive Parallel
HPS	Hard Processing System
IP	Intellectual Property
ISP	In-System Programming
.jic	JTAG Indirect Configuration File
LE	Logic Element
LUFA	Lightweight USB Framework for AVR
LUT	Look Up Table
PDN	Power Distribution Network
.pof	Programmer Object File
PS	Passive Serial
QFP	Quad Flat Package
RTS	Real-Time System
SFL	Serial Flash Loader
.sof	SRAM Object File

SUT	System Under Test
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
UART	Universal Asynchronous Receiver/Transmitter
UBGA	Ultra-fine Ball Grid Array
USB	Universal Serial Bus

Chapter 1

Introduction

1.1 Problem description

A real-time system can be defined as "a system where the result correctness depends not only on the logical result, but also at what time the results arrive" [22]. Thus, a real-time system must be verified with respect to time as well as to correct logical behavior.

Various techniques exist to assist system developers to verify the real-time behavior of their systems. One option would be using software techniques, where the execution of the program gets affected by the testing mechanism. An enhanced mechanism is using a logic analyzer to log system reaction and a signal generator for applying the stimuli. The problem with this technique is it must be tailored to the system, which reduces the possibility of reuse and increases development costs.

Professional testing tools exist. However, most of the tools that can provide real-time benchmarking with high accuracy and customizability are usually very expensive and/or platform dependent, which make them an unattractive option in educational environments. For example, the Real-time Systems lab in Technical Cybernetics department at NTNU uses a microcontroller based testing technique that can perform simple response analysis with relatively low accuracy.

The quest of creating a customizable, platform independent and inexpensive real-time testing system at NTNU started with the work of Kyrre Gonsholt[1] in his master thesis [15]. The major outcome of his work was a real-time tester IP as a Verilog module that had excellent results when tested using simulation. The physical implementation, however, had some problems and was not successful.

[1] In electronics department on behalf of Amund Skavhaug

1.2 Prevoius work

As mentioned above, the basis of this master thesis is the work of Kyrre Gonsholt. More details about his work, the used hardware and usage are mentioned next.

1.2.1 Hardware

The real-time tester IP was implemented and tested using the SoCKit development platform from Terasic, figure 1.1.

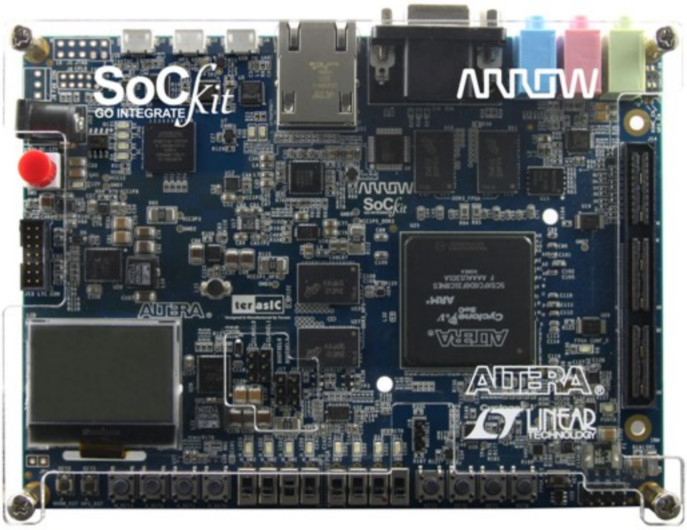


Figure 1.1: Arrow SoCkit from Terasic, source: rocketboards.org [18]

It contains an Altera Cyclone V FPGA with an ARM Cortex-A9 microprocessor, 1GB SDRAM, micro SD card support, a wide variety of peripherals and about 120 GPIO pins available through the HSMC-GPIO daughter board [18].

One of the features of the kit is the Golden System Reference Design (GSRD). The GSRD provides a set of essential hardware and software system components that can be used as a starting point for various custom user designs [1]. It contains a precompiled Linux kernel and file system, precompiled preloaded and bootloader (u-boot) in addition to the Golden Hardware Reference Design (GHRD).

GHRD is a fully configured hardware design that includes all the necessary configuration of the microprocessor and its peripherals so the user only needs to add his/her design as an additional IP block and connect it to the preconfigured blocks. The default configuration of GHRD is shown in figure 1.2.

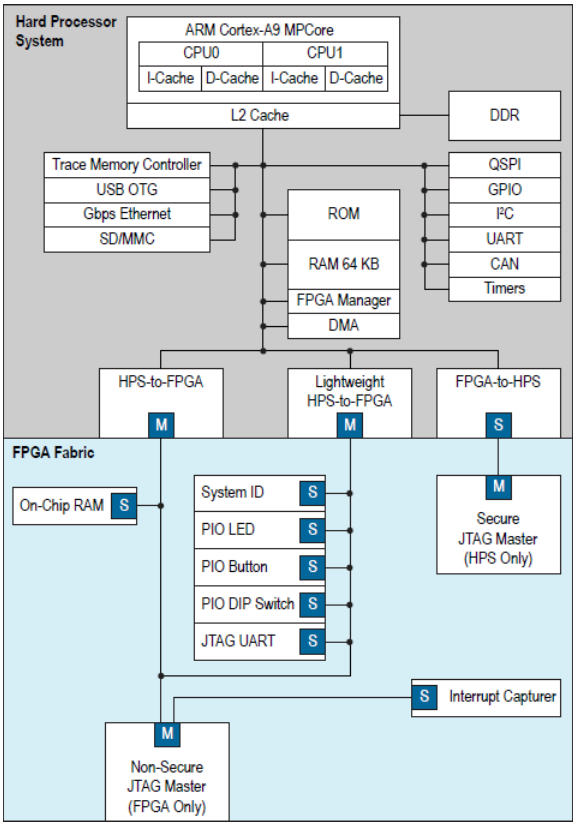


Figure 1.2: GHRD module overview, source: rocketboards.org [19]

The price of this kit is roughly EUR 325 [24], which makes it impractical to purchase a large quantity to be used as an educational platform for real-time systems.

1.2.2 Specifications

The test system has a resolution of 20ns and can perform a new logging every 80ns, which corresponds to a rate of 12.5 MHz. It can maintain an interruption rate of 2.5 MHz with four interrupt lines and 8.33 MHz with one line. The tester may log data with unique time over a period of $n * 2^{43} * 20ns$ with a resolution of $n * 20ns$ where n is clock scaling. That is, a test time of 48.9 hours with a resolution of 20ns.

1.2.3 Software

The RTS-tester uses the ARM processor, which is running Linux, to execute the software that configures and starts the test. The user will need to connect to the system using UART to launch the command line interface and run the program that triggers the test.

If the user needs to change the testing configuration, however, he/she will need to use a PC to modify the source code of the program (which is written in C) according to the new requirements, use a cross-compiler to compile the code into an ARM compatible executable and use the Ethernet port to transfer the executable to the system using SCP protocol. Only after all these steps will the user be able to run the tester using the new configuration.

1.2.4 Completeness

The functionality of the RTS-tester IP was tested and verified using a test-bench written in Verilog to simulate the design behavior using Mentro Graphics Modelsim. More details about the simulation and its result are available in Gonsholt's master thesis report [15].

Although the proper operation of the IP was verified using simulation, testing the implementation on an actual system has failed. Trying to connect the HSMC-GPIO daughter board to the kit and running the test caused damage to the board's circuit.

1.3 Work to be done

The main purpose of this master thesis is to:

- Understand the previous design and its operation.
- Debug and analyze its HDL code.
- Correct the found errors and verify the IP functionality.
- Suggest a design with suitable components for low cost.
- Create a PCB that implements the design.
- Create a user friendly interface to interact with the system.
- Keep the whole system cost below EUR 50.

The ultimate goal is to create a testing platform that can be used in the real-time systems lab as an enhanced replacement for the existing equipment. The platform design is to be compatible with Raspberry Pi Model B as a standard System Under Test (SUT) with the capability to use any other device as a test target. Another desired feature for the created system is to have a user-friendly interface that makes customizing the system an easy task compared to the previous design.

The previous features of the system will increase the students' understanding of real-time systems, especially when they get firsthand experience with such testing techniques. In addition, by providing the PCB design as open-source materials, universities can easily build and even customize the design for their needs.

This tool will also be helpful in hobby environments that usually have tight budgets and desire to have a generic solution. In addition to that, it has been taken into consideration to make the platform usable as a multipurpose tool. By offering the HDL files as open-source and offering extra general-purpose I/O pins for the FPGA and some extra peripherals, users can use the tool as a low-cost FPGA development platform alongside its ordinary function as a real-time tester.

1.4 Structure of the report

This report gives the reader an idea of the design process of the tester in addition to its structure and behaviour along with some background information.

After introducing the problem this project is supposed to tackle and some background information in the Introduction Chapter, Chapter 2 will explain how the previous work was analyzed in order to correct its malfunctions.

Chapter 3 focuses on the main steps for creating the platform. It starts with the FPGA selection and configuration, then describes the system design process followed by some PCB design considerations. It also discusses how the tester can comply with the Raspberry Pi HAT specification as a standard SUT.

Chapter 4 concentrates on the importance of providing proper protection to the system against power supply problems and user mistakes and suggests techniques to overcome these hazards.

Chapter 5 illustrates the software structures of the different parts of the systems along with the interfaces used between them.

Chapter 6 serves as a detailed user guide on configuring the various units of the system including the FPGA, Nios II processor and the microcontroller. It also gives directions on how to use the tester graphical user interface.

Chapter 7 will present the available testing techniques and how the SUT can be programmed in each of them. In addition, it will have an overview of performing result analysis both manually and using the accompanying software.

Chapter 8 sheds light on some issues faced during development and suggests a few recommendations for future system development.

Chapter 2

Validating the tester IP

2.1 Verifying tester IP operation

The first task that needs to be done is making sure that the RTS-tester IP works properly and as expected. At the beginning, several simulations were made using a testbench based on the one Kyrre Gonsholt originally used. The simulations tested multiple configurations of the tester to make sure it is fully functional in all operation modes.

One example of the tester operation is the case where all four interrupt lines are activated and the testbench is set up to properly respond to them. The resulting timing diagrams of this case can be seen in figure 2.1.

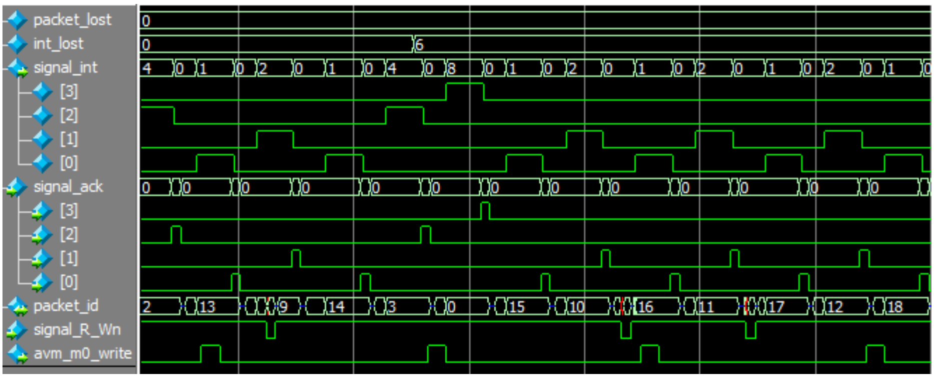


Figure 2.1: RTS-tester simulation result when activating all 4 interrupts

Not all the tests will be listed here. If the reader is interested in the detailed simulation results he/she may refer to the results of the extensive simulations performed by Gonsholt [15].

2.2 Running the design on actual hardware

The next step is finding the reason for the hardware failure when the RTS-tester IP is synthesized in the FPGA and linked to external pins. This failure is particularly strange since the design worked perfectly in simulation.

The first thing to be checked was the Altera Quartus II configuration. Quartus II is the official software from Altera to be used as an IDE to develop HDL code (whether it is Verilog or VHDL), compile it, synthesize it and perform all kinds of time and power analysis of the designs. Along with Quartus, comes a large set of software such as Qsys for building up systems out of IPs and Signal Tap II that functions as a logic analyzer that can be synthesized into the FPGA.

After speaking with Gonsholt, he shared his suspicion that a problem may have happened while fitting the design in the FPGA because of some compiler optimization by Quartus, which may have caused the design to malfunction. This may seem unlikely to happen but it seemed a logical assumption then especially since the author has faced problems before caused by compiler optimizations particularly in microcontrollers.

Quartus has tens of options for optimizing the different processes such as synthesis and fitting (place and route operation). These options may affect the design power consumption, size (occupied logic elements) or speed.

Checking if a software bug caused the problem was done by modifying the optimization parameters and monitoring the internal signals. This task became a lot easier thanks to the Signal Tap II software bundled to Quartus. This software allows the user to make part of the FPGA function as a logic analyzer not only for the I/O pins but also for the internal signals. After changing all the major options that may affect the compilation outcome, no change was observed in the tester signals.

2.3 Examining the design

The only option left to catch the bug that is causing the design to fail is thorough code examination. Although the Verilog code was not well documented, most of the internal signals and variables had meaningful names, which made this task fairly easier.

Code examination could not reveal any obvious bugs in the design. Although the examination was not extremely thorough, it gave more confidence in design correctness, especially that it passed all simulation tests.

Since the design itself is assumed to be correct, the suspicion moved to other elements in the hierarchy. As mentioned before in Section 1.2.1, the design is built by integrating the tester IP into the GHRD that has been used as is. Figure 2.2 shows the tester's place in the hierarchy.

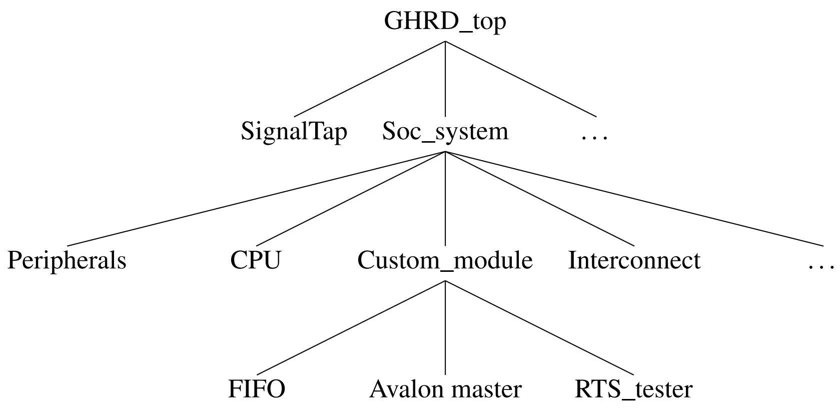
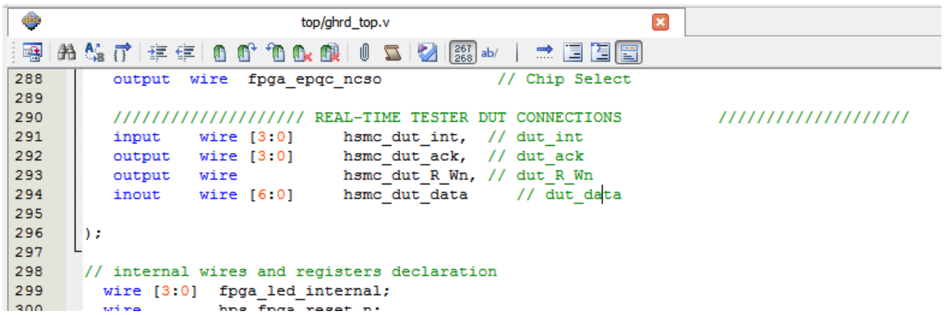


Figure 2.2: Real-time tester design hierarchy

All the components in this structure are part of GHRD except for the Custom_module that is built by Gonsholt. This module consists of an FIFO buffer, Avalon master and the RTS-tester IP. The FIFO buffer is actually an Altera IP that is used for collecting/releasing the data on positive clock edges depending on the control signals. Avalon master module implements Avalon master interface as described in [1] which, in this case, acts as the consumer of the FIFO. Its task is to write data (test result) to the HPS SRAM and it is controlled by the control signals from the top module. It can write to memory either in burst or in normal mode.

By looking at the structure, one can clearly notice that the inputs and outputs of the tester module have to be imported in each level in the hierarchy. This is where the major bug was found. In one of the levels, the assignments of inputs and outputs were inverted causing the system to misbehave and damage any external circuitry connected. Figure 2.3 shows the place of the bug before it was corrected.

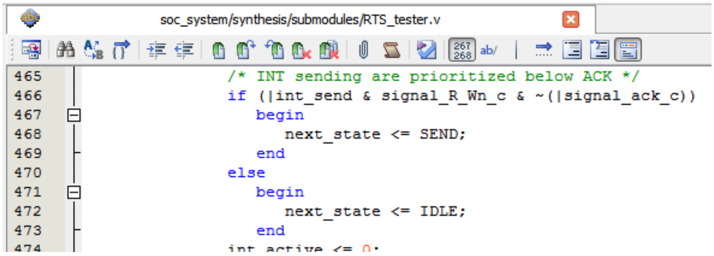


```
    output  wire  fpga_epqc_ncso            // Chip Select

    /////////////////// REAL-TIME TESTER DUT CONNECTIONS         ////////////////////
    input   wire [3:0]     hsmc_dut_int,  // dut_int
    output  wire [3:0]     hsmc_dut_ack,  // dut_ack
    output  wire           hsmc_dut_R_Wn, // dut_R_Wn
    inout   wire [6:0]     hsmc_dut_data     // dut_data

);

// internal wires and registers declaration
  wire [3:0]  fpga_led_internal;
  wire        hps_fpga_reset_n;
```

Figure 2.3: Bug caused by incorrectly importing tester’s I/Os

After correcting the code, another attempt was made to run the tester with one interrupt line activated. SignalTap did not show any improvement in the signals and the module did not issue any interrupts although the clock was running and the internal counters were normally counting.

After getting deeper in code examination, another problem was discovered. Regardless of how many interrupt lines are activated, all acknowledgment lines must be at logic 0 for any interrupt to be issued. This condition was made (according to the code comments) to prioritize acknowledgment over interrupt sending, but probably the mentioned side effect was not noticed. The condition in question is shown in figure 2.4.



```
/* INT sending are prioritized below ACK */
if (|int_send & signal_R_Wn_c & ~(|signal_ack_c))
   begin
      next_state <= SEND;
   end
else
   begin
      next_state <= IDLE;
   end
int_active <= 0;
```

Figure 2.4: An interrupt is only issued when all ACK lines are on logic 0

After the unused ACK lines were manually pulled down, the test was repeated and it was finally successful. SignalTap logic analyzer showed correct waveforms of all the major signals including the used INT and ACK lines. This can be seen in figure 2.5. The used SUT for this experiment was an Atmel microcontroller programmed to interact and respond to the signals issued by the FPGA.
The functionality was confirmed by repeating the test using other configurations like using all interrupt lines, which gave similar results.

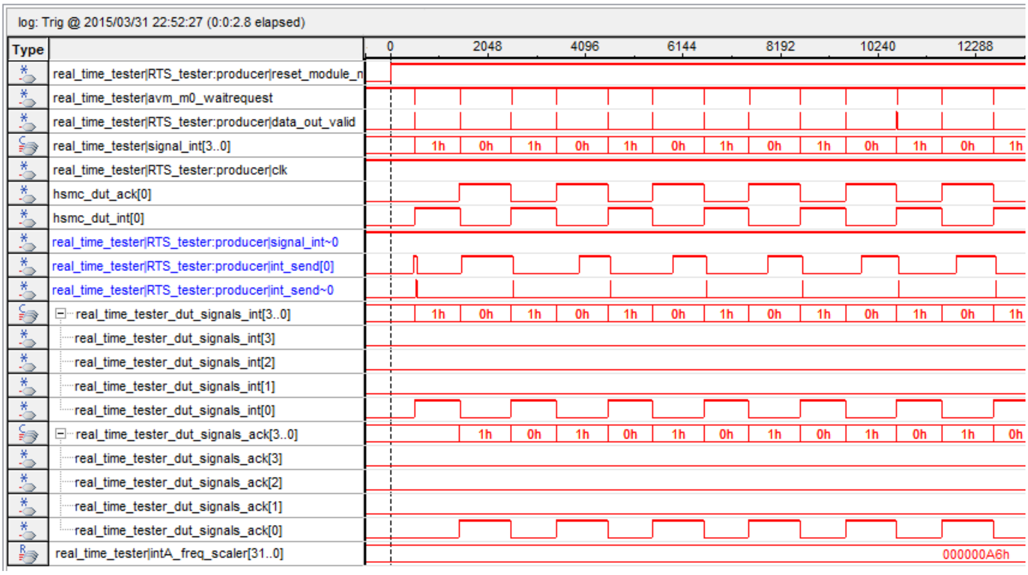

log: Trig @ 2015/03/31 22:52:27 (0:0:2.8 elapsed)
Type
0
2048
4096
6144
8192
10240
12288
real_time_tester|RTS_tester:producer|reset_module_n
real_time_tester|avm_m0_waitrequest
real_time_tester|RTS_tester:producer|data_out_valid
real_time_tester|signal_int[3..0]
real_time_tester|RTS_tester:producer|clk
hsmc_dut_ack[0]
hsmc_dut_int[0]
real_time_tester|RTS_tester:producer|signal_int~0
real_time_tester|RTS_tester:producer|int_send[0]
real_time_tester|RTS_tester:producer|int_send~0
real_time_tester_dut_signals_int[3..0]
real_time_tester_dut_signals_int[3]
real_time_tester_dut_signals_int[2]
real_time_tester_dut_signals_int[1]
real_time_tester_dut_signals_int[0]
real_time_tester_dut_signals_ack[3..0]
real_time_tester_dut_signals_ack[3]
real_time_tester_dut_signals_ack[2]
real_time_tester_dut_signals_ack[1]
real_time_tester_dut_signals_ack[0]
real_time_tester|intA_freq_scaler[31..0]
1h 0h 1h 0h 1h 0h 1h 0h 1h 0h 1h 0h 1h
1h 0h 1h 0h 1h 0h 1h 0h 1h 0h 1h 0h 1h
1h 0h 1h 0h 1h 0h 1h 0h 1h 0h 1h 0h
000000A6h

Figure 2.5: SignalTap II output when using one interrupt line

Chapter 3

Creating the testing platform

So far, the SoCkit from Terasic has been used to work with the tester module. As mentioned in Chapter 1, one of the goals of this work is to create a dedicated platform for testing. This chapter describes the process of designing this platform.

3.1 Choosing an FPGA for the tester

The main component that needs to be selected for this project's target platform is the FPGA. The obvious solution for this is to just use the same device used in SoCkit. This way, the design is guaranteed to work and there is no need to worry about device compatibility issues. The main problem here is cost. The cost of the FPGA is EUR 335 [16] which is even a more expensive than the SoCkit itself.

The next option is to pick a lower end device from the same family that can give good compatibility with a lower cost. The chip that provides the lowest cost and still offers most functionality along with an ARM processor core is 5CSEBA2U19C8SN. It is also from Cyclone V family, it has a single ARM core and more than enough specifications to implement our design. The cost for this chip is EUR 49. Although this is almost our cost limit that we have set earlier, one may argue that since the FPGA is the most costly component, the total system cost may not exceed EUR 70, which is not too much for such a device. While this statement may generally be true, other factors should be considered.

The package of this chip is UBGA-484 (Ultra-fine Ball Grid Array), which has a pin pitch of 0.8 mm and ball diameter of 0.5 mm [3]. This means only 0.3mm is free between two neighbouring balls. Therefore, as figure 3.1 shows, for a trace to pass between two balls its width must be at most 0.1 mm (4 mil) and the drill size for vias can not exceed 0.2mm. This violates the design rules of most of the PCB fabrication houses, thus forcing us to choose a PCB manufacturer with higher end equipment which, of course, will result in a much higher manufacturing cost. Furthermore, having 484 pins in a BGA package will make it almost impossible to design a PCB with less than 6-8 layer especially with vital pins (power, configuration, clock, etc.) scattered all over the chip. These two factors would multiply the platform cost several times even before start to consider assembly cost.

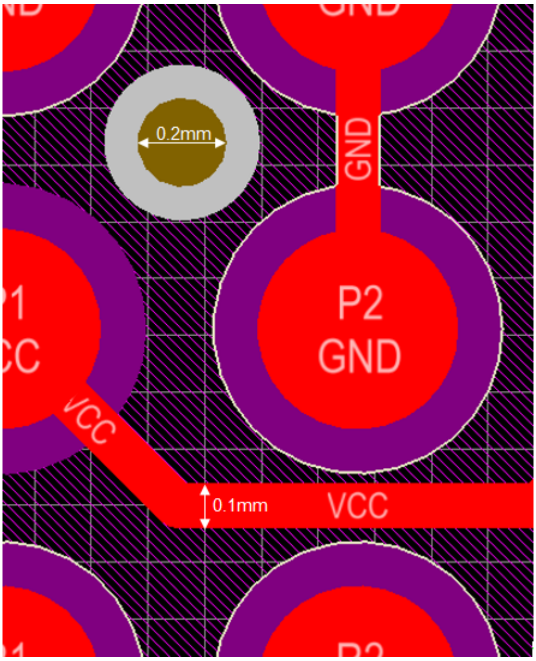


Figure 3.1: UPGA package showing a trace routed between two balls

Taking all that into consideration, the quest now is to find an FPGA with the following characteristics:

- Has enough specs to implement the design.
- Maintains a minimum degree of compatibility with the current chip.
- Satisfy the minimum requirements of Qsys: 12k Logic Elements (LEs) and 128k of embedded memory.
- Comes in a routing friendly package that allows cheap manufacturing and a minimum number of layers.
- Low cost.

Taking all previous points into consideration, an Altera Cyclone IV FPGA with the cost of EUR 23 was found, the EP4CE15E22. It has 15k LEs, 504 kbit of embedded memory and comes in a QFP-144 package. Using a QFP chip allows keeping the PCB design within the standard design rules of most of the manufacturers, thus lowering production cost. Furthermore, it will make using two layers for the design possible.

Figure 3.2 shows QFP-144 and UBGA-484 packages side by side.

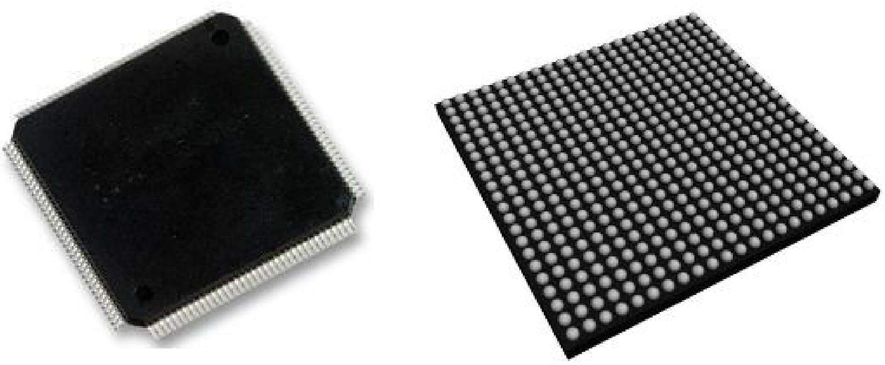


Figure 3.2: QFP-144 (left), and UBGA-484 (right), source: mouser.com

Table 3.1 shows an overview of the selected Cyclone IV FPGA specs. For comparison, Table 3.2 shows an overview of resources available in the SoCkit's Cyclone V FPGA.

Table 3.1: Excerpt from the list of available resources for FPGA (EP4CE15E22)[5]

Specification	**Value**
Logic Elements (LEs)	15k
Memory	504 kbit
FPGA GPIO	81
HPS I / O	N/A
ARM Cortex-A9 MPCore processor	N/A

Picking a chip with lower cost does not come free. As noticed in the tables, the Cyclone IV FPGA does not contain a hard processor core. This is a significant disadvantage because the tester design is based on and integrated with the GHRD. As mentioned in section 1.2.1, GHRD is a hardware design provided by Terasic with their SoCkit that allows users to design their systems by simply adding the proper IPs. Being unable to use GHRD means that a soft processor core, e.g. Altera Nios II, should be used and more time should be invested in migrating the original design to the new hardware. More details about that in the next sections.

Table 3.2: Excerpt from the list of available resources for FPGA (5CSXFC6D6F31C8NES)[6]

Specification	**Value**
Logic Elements (LEs)	110k
Adaptive Logic Modules (ALMs)[1]	41509
Register	166,036
Memory	5570 kbit
FPGA GPIO	288
HPS I / O	181
ARM Cortex-A9 MPCore processor	Dual-core

3.2 FPGA Configuration

3.2.1 Configuration Modes

The chosen FPGA supports the configuration modes shown in table 3.3.

.

Table 3.3: Configuration Features in Cyclone IV Devices

Configuration Scheme	**Configuration Method**	Max Clock (MHx)
Active Serial (AS)	Serial Configuration Device	100
Passive Serial (PS)	External Host with Flash Memory / Download Cable	125
Fast Passive Parallel (FPP)	External Host with Flash Memory	125
JTAG	External Host with Flash Memory / Download Cable	33

JTAG Configuration Scheme

JTAG has been chosen as the main configuration scheme because it provides all the features the other schemes have in addition to being more versatile as we will see later. According to the table, JTAG configuration is slower than the other modes. However, it only takes about 3-4 seconds to completely configure the device in practice. One of the great features of JTAG is Boundary-scan Testing (BTS). The BTS architecture offers the capability to efficiently test pin connections without using physical test probes and capture functional data while the device is normally operating.

[1] Adaptive Logic Module (ALM) is a further development of Look Up Table (LUT), and it is used to instance registers and other logic [7]

In addition, JTAG configuration scheme does not require any external components like PS or FPP. Only a standard Altera download cable, e.g. USB Blaster, is needed. Figure 3.3 shows the connection of a download cable connector to a Cyclone IV device in JTAG mode.

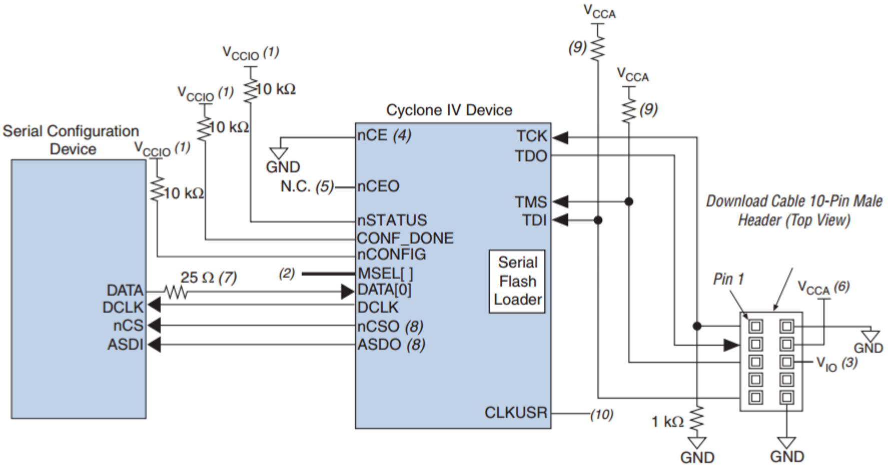


Figure 3.3: Programming the FPGA Using JTAG Interface [2]

Another interesting feature of JTAG is that an unlimited number of devices can be configured/tested at the same time by connecting them in a JTAG chain, i.e. Test Data Out (TDO) pin of one device is connected to Test Data In (TDI) pin of the next device in the chain with Test Mode Select (TMS) and Test Clock (TCK) pins connected to all devices. An example of that is shown in figure 3.4.

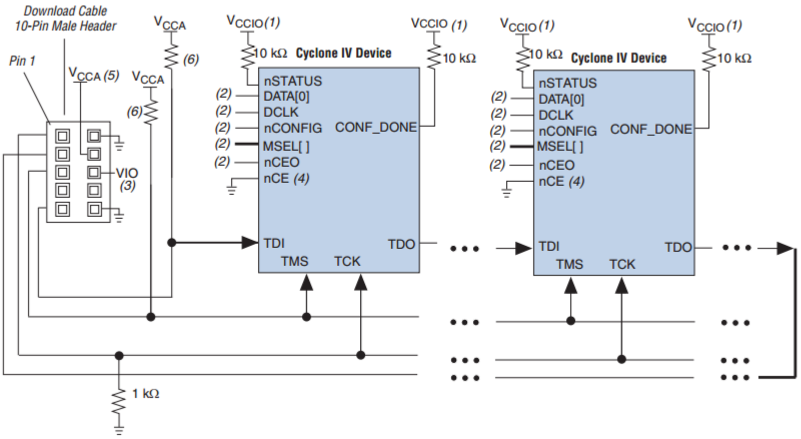


Figure 3.4: JTAG Testing/Configuration of Multiple Devices [2]

In our design, JTAG is also used for programming Nios II processor without the need to reconfigure the whole FPGA. It also serves as a virtual UART interface to provide a console the programmer can use to interact with the processor.

Section 6.1 gives instructions for using the JTAG interface to transfer the HDL design to the FPGA.

AS Configuration Scheme

After programming the FPGA, it stores the configuration data in an on-chip volatile SRAM memory that is erased once the device loses power. A common solution to have permanent configuration is storing the configuration file in an external flash memory that programs the FPGA at every power on.

Altera has a standard line of serial configuration memory chips that use AS interface to be programmed and to connect to the FPGA. For programming the memory, USB Blaster can be connected to it through AS interface and then Quartus II programmer software is used to start the programming procedure. Detailed instructions of how this is done are given in section 6.2.2. Figure 3.5 shows the required connection on the board.

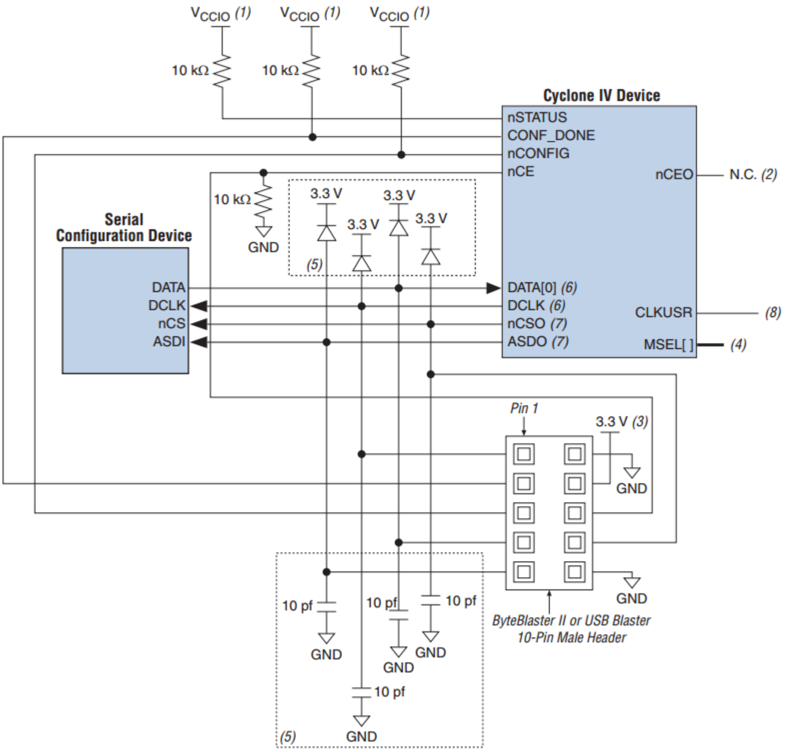


Figure 3.5: In-System Programming of Serial Configuration Device [2]

Programming Serial Configuration Device Using JTAG Interface

USB Blaster can use the JTAG interface of the FPGA to access the AS pins to program the configuration-device thanks to the Serial Flash Loader (SFL) deign embedded in Cyclone IV series FPGAs. In this case, the same connection in figure 3.3 can be used. Detailed instructions of using the Quartus II programmer to perform this operation are found in section 6.2.1.

The Used Configuration Setup

To ensure the functionality of the RTS-tester prototype being built, both AS interface and JTAG interface have been included in the design. One 10-pin connector has been added for each mode to guarantee that the user can configure the design even if one of the interfaces failed. Figure 3.6 shows the used configuration.

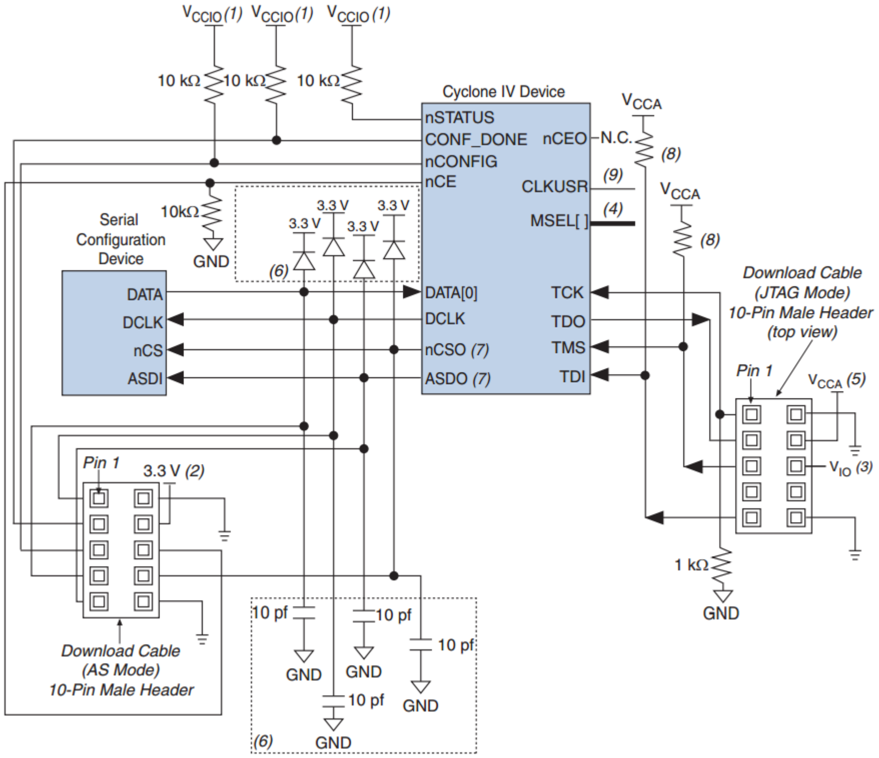


Figure 3.6: Combining JTAG and AS Configuration Schemes [2]

After the tester operation has been verified and both interfaces are confirmed to work, the required components for the AS interface (10-pin connector, diodes and capacitors) can be removed from the design to save space and cost.

Choosing a configuration device

According to the Altera Cyclone IV Configuration and Upgrade Guide [2], the EPC4CE15 FPGA used in the design has a maximum uncompressed configuration file size of about 4 Mbits. However, an EPCQ16 configuration device with 16Mbits of memory has been used to give the user access to more general-purpose memory, especially that the used FPGA has only 504kbits of embedded memory.

The user can easily access the extra memory in his/her design by adding the SFL block to the Qsys system builder. However, caution must be taken not to access the region reserved for storing the configuration file, which usually starts from address 0x0.

Since Altera branded configuration devices are relatively expensive, serial flash memory chips with compatible interfaces from other manufacturers can be used, e.g. S25FL116K serial SPI flash memory from Spansion. Altera's EPCQ16 costs EUR 6.85 on mouser.com while Spansion's chip costs only EUR 0.47.

3.2.2 Configuration tool

USB Blaster, shown in figure 3.7, is the standard download cable for in-circuit reconfiguration of Altera's FPGAs SRAM. In an ideal scenario, USB Blaster hardware would be integrated into the RT-tester's PCB design so no external tool is needed for FPGA configuration. Unfortunately, that was not very doable within the scope of this project.

Figure 3.7: Altera USB-Blaster

The USB Blaster design mainly consists of a CPLD, particularly an Altera MAX series device, along with a USB-Parallel chip from FTDI (FT245BM). Altera has published the USB Blaster schematics, but the HDL code running in the CPLD has not been released. Several attempts to replicate this device exist on the web. Most of them take the approach of reverse-engineering where they try to monitor the control and programming signals issued by the CPLD and attempt to produce HDL code that emulates this behaviour.

A German hardware enthusiast named Kolja Waschk is working on a project to develop a USB JTAG adapter compatible with Altera's USB-Blaster protocol [25]. He managed to produce HDL code that enables a MAX CPLD to function properly in a way similar to an original USB Blaster. He even built a modified version that uses a Cypress microcontroller with USB support to perform both the USB interfacing function and issuing the control signals to the FPGA. The design was reported to function properly in most cases with a few bugs.

For the purpose of this project, the option of using a CPLD to integrate the configuration device functionality into the system is not very preferable as it adds more cost and complexity to our cost-oriented project; whereas using a USB enabled microcontroller is more attractive since a similar device already exists in the design. The main problem with this approach is that Cypress MCU firmware is extremely device specific and heavily uses Cypress libraries.

After spending several days trying to accomplish the task of integrating the USB Blaster into the available hardware using multiple resources, the author decided to leave this task for future work and invest the time in other aspects that can have a better benefit to the project, and use instead an external USB Blaster for configuration purposes. The lack of this feature is not likely to cause any inconvenience to the user either since the RT-tester will probably need to be programmed only once if it is used in a lab or testing environment.

3.3 System Design

3.3.1 Initial Design

The initial design was planned to be similar to figure 3.8.

The SD card is a very important component of the system. It provides a local storage medium to save the test results in a form of a dump file and allows the user to keep track of previous test results. Unfortunately, no proper SD card controller IP for the FPGA was found. Most of the available IPs out there are either commercial or do not support Avalon interface which is needed to communicate with Nios II processor.

3.3.2 Adding a Microcontroller

For the previous reasons, a microcontroller was chosen to perform the task of communicating with the SD card and a device from the AVR family, which is produced by Atmel, was selected. Atmel AVR is a family of 8-bit microcontrollers that covers a wide range of devices with variant features and capabilities and is considered among of the most popular microcontrollers in the world among hobbyists. This popularity helped producing a huge set of open source libraries that cover various needs from LED switching to complex communication protocols. The author has evaluated several libraries that implement SD

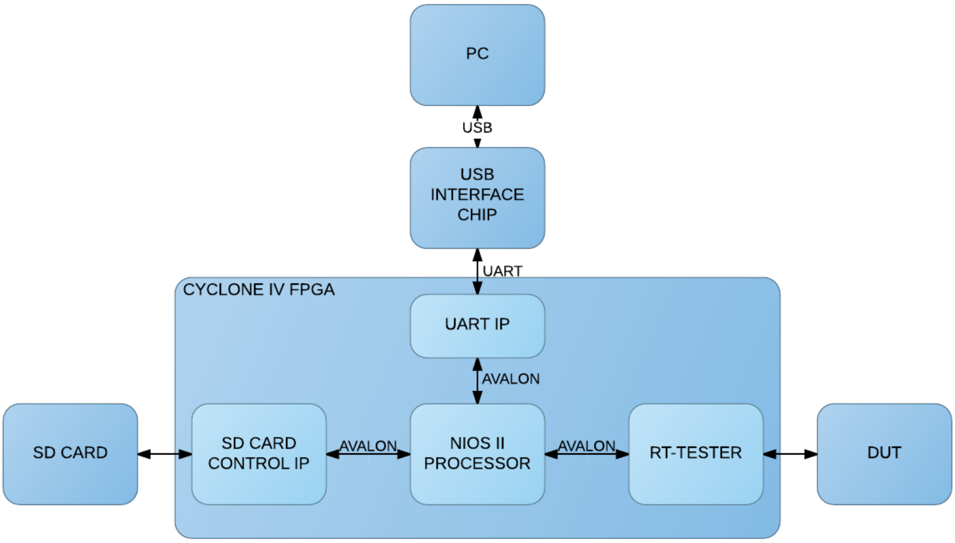


Figure 3.8: Initial design structure

card control over SPI and chose one that has a very convenient command based interface in addition to supporting FAT file system.

UART protocol was chosen to transfer the commands and data between the microcontroller and the FPGA. Figure 3.9 shows the design after adding the microcontroller.

One may argue that an SPI IP with Avalon interface can be used in the FPGA in addition to implementing the SD card control logic and the file system. This indeed can be done, but it would require a non-trivial amount of time and effort with advantages that do not match this cost. Actually, the only advantage, in this case, would be saving the cost of the microcontroller, which may make sense since one of our priorities is cost reduction. However, as we will see next, adding the microcontroller will help reduce the system cost further more.

In case a future developer finds a proper way to use the FPGA for controlling the SD card, the micro-SD card SPI bus is connected to the FPGA through a 74CBTLV3257D bus multiplexer. The microcontroller controls the bus direction using PORTB7 where it sets the selection pin of the multiplexer to '1' to take control over the bus. The schematic of the connection can be found in Appendix A.

3.3.3 USB Interface

As seen in figure 3.9, another UART interface, besides the one we just added for the MCU, is needed by the FPGA to communicate with the USB interface chip. This interface requires special care because it is the data transfer medium to/from the PC, which obviously

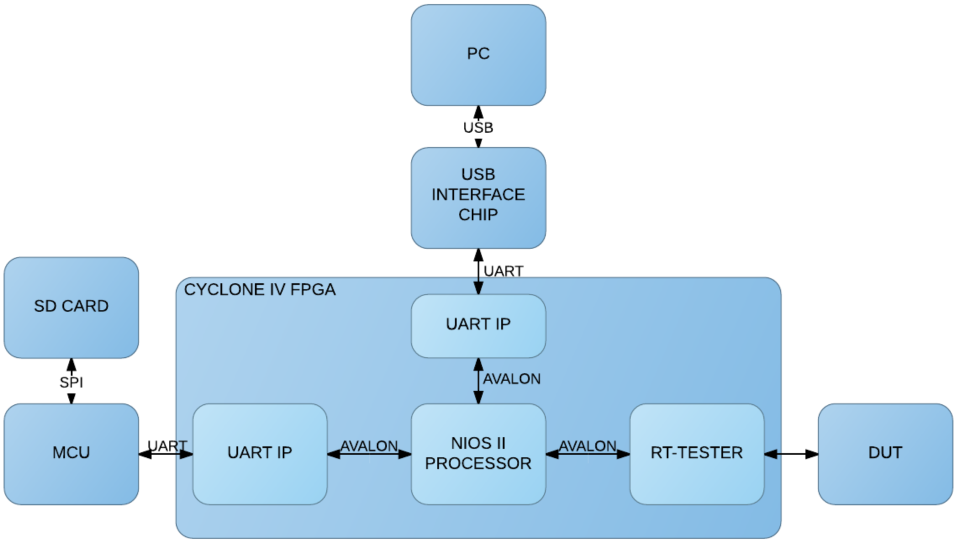


Figure 3.9: RT-tester design after adding a microcontroller to control the SD card

has a greater transfer rate and processing power than Nios II. This means that the PC can finish processing one bunch of data and send a new one way before Nios is prepared for that.

Let us consider a case where Nios is busy with a time-consuming task, like a real-time test operation that may take several seconds, and it starts receiving requests or data from the PC. It is very likely in this case that the receiver buffer overflows and some information gets lost. This illustrates the importance of having an intelligent intermediary device compared to a simple interface chip.

Since we already have a microcontroller in the design, it makes perfect sense to have it supervise the communication with the PC, especially that its only function right now is controlling the SD card. Thankfully, Atmel has several devices in the AVR family that incorporate the USB protocol stack in hardware, which make them a perfect fit for our need. Figure 3.10 shows the system design after applying these proposed adjustments.

The chosen chip is Atmega32u4 which main characteristics are shown in table 3.4.

Having the microcontroller handling the USB-bridge functionality does not only help saving some items from the bill of materials, it will also enhance the functionality and responsiveness of the system. The microcontroller can respond to certain user request without disturbing Nios or waiting for it to be free. For example, user requests related to the SD card, such as browsing and getting test results, can be performed without any need of intervention from Nios. The only disadvantage of removing the dedicated USB-bridge

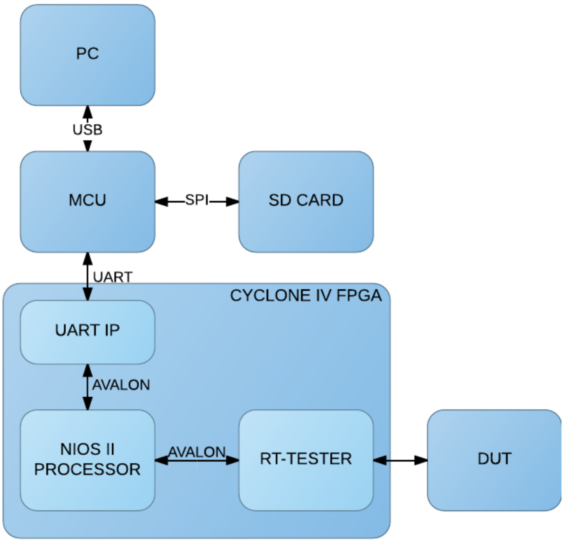


Figure 3.10: RT-tester design after replacing the USB-serial chip with a microcontroller

Table 3.4: Atmega32u4 microcontroller main characteristics [9]

Specification	**Value**
Flash Memory	32K Bytes
RAM	2.5K Bytes
EEPROM	1K Bytes
GPIO	26
Maximum Speed	Up to 16 MIPS
Communication protocols	USB, SPI, USART, I2C
Maximum USB transfer rate	12 Mbit/s

chip is the extra effort needed to set up the microcontroller to operate as a bridge compared to an effortless drop-in solution. However, as mentioned earlier, AVR has a great community support that helped a lot making this task easier.

LUFA (Lightweight USB Framework for AVR) [10] is an open-source library that offers a convenient API for programmers to take advantage to the USB core in AVR microcontrollers. It also comes with some examples for creating a USB bridge that allows the MCU to communicate easily with the PC.

A dedicated PCB was designed using Altium Designer to develop the USB bridge functionality in addition to SD card control. It was also used to connect to the FPGA in the SoCkit to perform firmware development before the final PCB is manufactured. Figure

3.11 shows the board along with its connection to the SoCkit and the SD card reader.

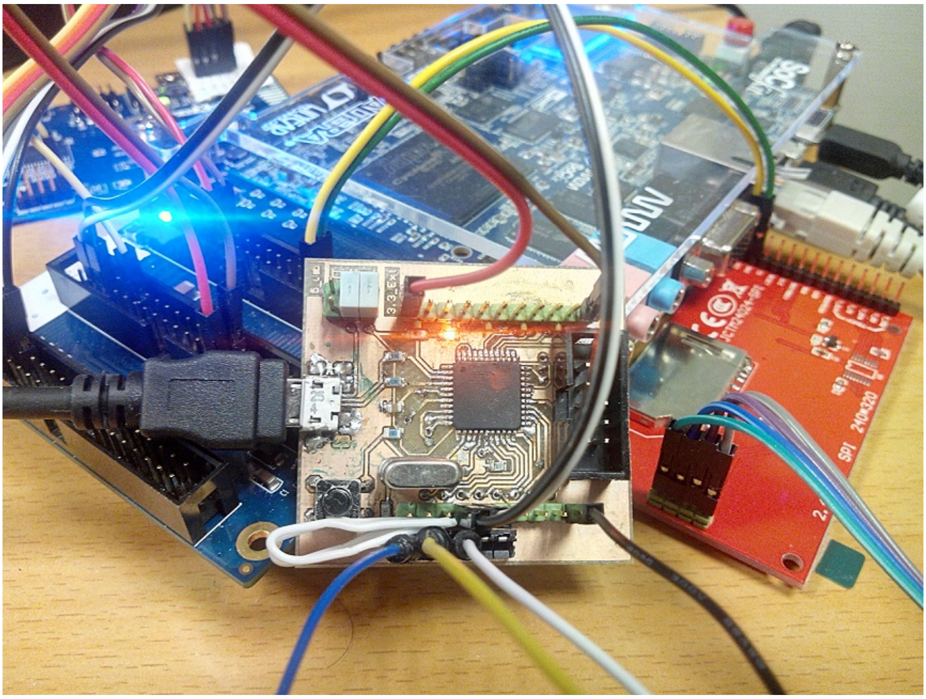


Figure 3.11: Atmega32u4 board used for early firmware development

3.4 Choosing the PCB design software

Eagle is one of the most common PCB design tools among students and hobbyists. This popularity is mainly regarded to the existence of a freeware version that makes it very popular among hobbyists who cannot afford the multi-thousand dollars price tag of the rival software. A good indicator for this popularity is that most open-source electronic design schematics and PCBs are published in Eagle format along with good community support. These were the main reasons why the author has chosen Eagle for most of his electronics designs every time a PCB needs to be created.

Giving this background, the PCB design process for this project started with Eagle as the main tool. However, its shortcomings started to get more noticeable as the design got more complicated. For example, it is not a simple task to create a multi-level hierarchical design for the different parts of the design (e.g. PSU, user I/O, FPGA blocks, etc.).

For these reasons, other software options were evaluated. Most commercial software (such as Orcad and Proteus Ares) were directly ruled out because they are only available as professional packages with a hefty price tag (compared to Eagle that has a free edition). In addition, the available open source tools (e.g. KiCad) are too simple for the task in hand. The only alternative left was Altium Designer which commercially costs $9k, but thankfully NTNU has a license for it which can be used for free by students.

Compared to Eagle, Altium Designer offers several advantages including:

- 2D and 3D modelling of the PCB design
- Functional view of the FPGA as separate blocks (rather than a one big block in Eagle)
- Better hierarchical design support
- Supports creating reusable modules
- Better PCB autoroute functionality
- Faster and more intuitive workflow

3.5 Following Raspberry Pi HAT Specifications

3.5.1 The Raspberry Pi HAT

This project aims to design a real-time testing device with a generic interface to be usable by any external device. However, since the tester is meant to be used mainly in labs for educational purposes, it is beneficial to have an interface compatible with a popular embedded system.

The Raspberry Pi is a credit card sized single-board computer developed with the intention of promoting the teaching of basic computer science in schools [27], and has been chosen to be the standard SUT of the tester. It was preferred over the other options because of its popularity among students and hobbyists and its robust design. Figure 3.12 shows a photo of the Raspberry Pi 2 model B.

After launching Raspberry Pi model B+, the Raspberry Pi Foundation introduced a set of specifications to unify the design and form factor of the add-on boards targeting the Pi. The add-ons that follow those specifications are called HATs (Hardware Attached on Top).

"A HAT is an add-on board for B+ that conforms to a specific set of rules to enhance user experience" [14].

A significant feature of HATs is the inclusion of a system that allows the Pi to identify a connected HAT and automatically configure the GPIOs and drivers for the board.

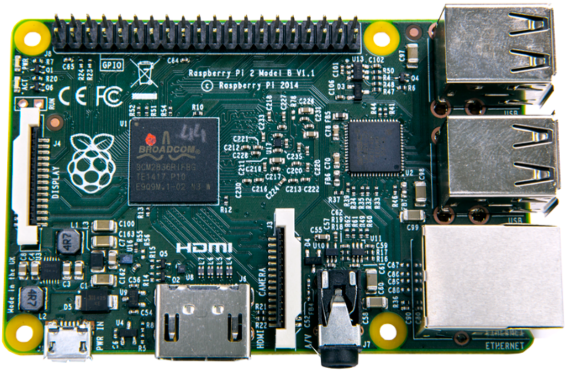


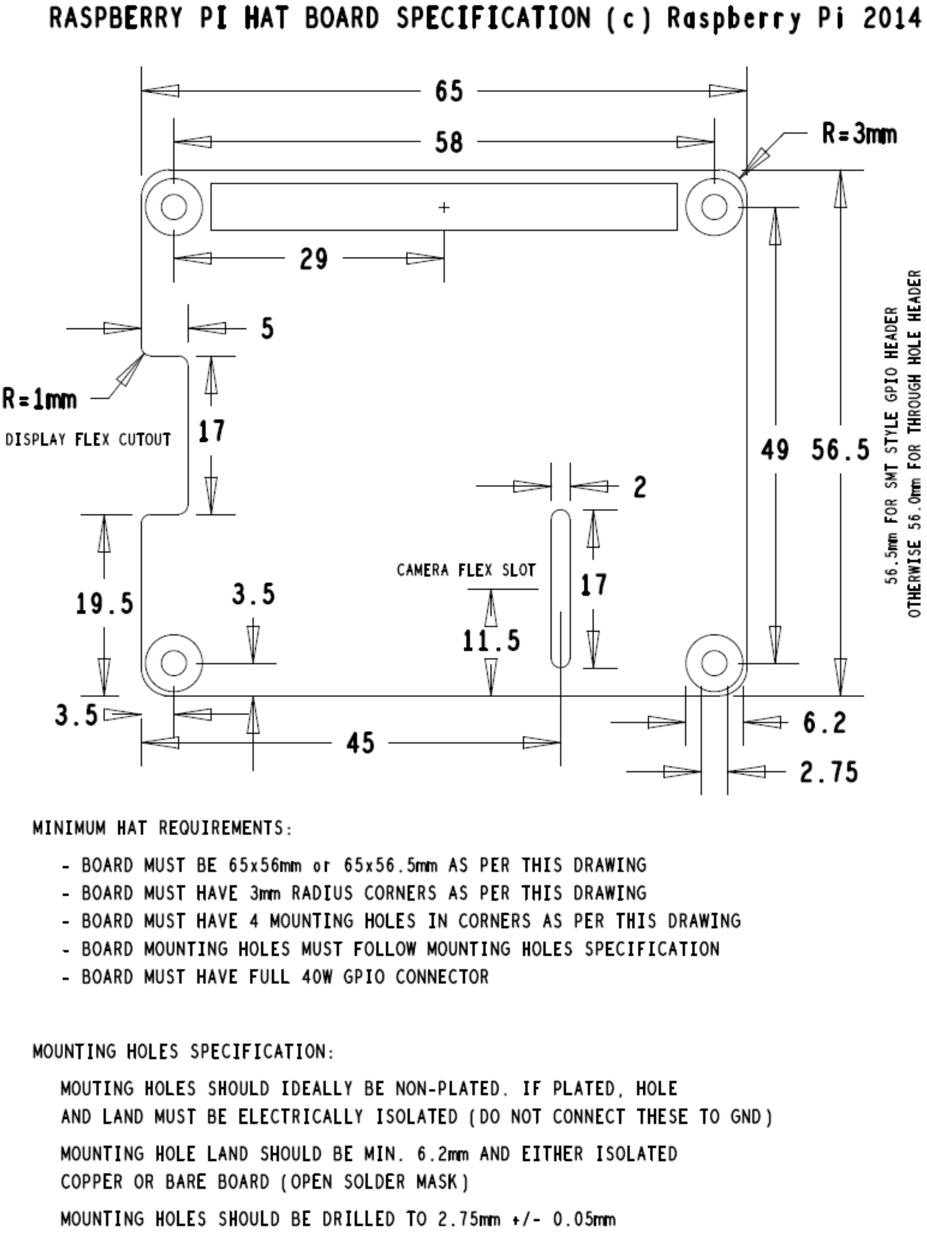
Figure 3.12: Raspberry Pi 2 Model B, source: raspberrypi.org

3.5.2 HAT Specifications

HAT is a rectangular board (65x56mm) that has four mounting holes in the (nicely rounded) corners that align with the mounting holes on the B+, has a 40W GPIO header and supports the special autoconfiguration system that allows automatic GPIO setup and driver setup. The automatic configuration is achieved using 2 dedicated pins (ID_SD and ID_SC) on the 40W B+ GPIO header that are reserved for an I2C EEPROM. The EEPROM holds the board manufacturer information, GPIO setup and a thing called a ’device tree’ fragment - basically a description of the attached hardware that allows Linux to automatically load the required drivers. [14]

Basically, a board is only a HAT if: [12]

1. The ID_SC and ID_SD pins must only be used for attaching a compatible ID EEPROM.
2. If back-powering via the 5V GPIO header pins you must make sure that it is safe to do so even if the Pi 5V supply is also connected.
3. The board must protect against old firmware accidentally driving GPIO6,14,16 at boot time if any of those pins are also driven by the board itself.
4. It has a valid ID EEPROM (including vendor info, GPIO map and valid device tree information).
5. It has a full size 40W GPIO connector.
6. It follows the HAT mechanical specification (figure 3.13).
7. It uses a GPIO connector that spaces the HAT between 10mm and 12mm from the Pi.
8. If back powering via the GPIO connector the HAT must be able to supply a minimum of 1.3A continuously to the Pi.



RASPBERRY PI HAT BOARD SPECIFICATION (c) Raspberry Pi 2014

MINIMUM HAT REQUIREMENTS:

- BOARD MUST BE 65x56mm or 65x56.5mm AS PER THIS DRAWING
- BOARD MUST HAVE 3mm RADIUS CORNERS AS PER THIS DRAWING
- BOARD MUST HAVE 4 MOUNTING HOLES IN CORNERS AS PER THIS DRAWING
- BOARD MOUNTING HOLES MUST FOLLOW MOUNTING HOLES SPECIFICATION
- BOARD MUST HAVE FULL 40W GPIO CONNECTOR

MOUNTING HOLES SPECIFICATION:

MOUTING HOLES SHOULD IDEALLY BE NON-PLATED. IF PLATED, HOLE AND LAND MUST BE ELECTRICALLY ISOLATED (DO NOT CONNECT THESE TO GND)

MOUNTING HOLE LAND SHOULD BE MIN. 6.2mm AND EITHER ISOLATED COPPER OR BARE BOARD (OPEN SOLDER MASK)

MOUNTING HOLES SHOULD BE DRILLED TO 2.75mm +/- 0.05mm

Figure 3.13: Raspberry Pi HAT board Specifications, source: raspberrypi.org

Currently, the tester design follows all the previous requirements except for points 4 and 6. An EEPROM chip is already included in the design and it is connected to ID_SC and ID_SD pins in the Pi header. However, it has not been programmed with valid vendor information, GPIO map or device tree. Detailed information is available to guide the user on what to program on the EEPROM [13]. The EEPROM chip is already connected to the microcontroller's I2C pins and it can be programmed directly using them. Alternatively, an external connection for the I2C bus has been added to the tester in case the user wishes to use an external device for programming the memory.

As for the mechanical specifications, some of them are implemented including the full 40W GPIO connector, two electrically isolated mounting holes with the correct position and dimensions. The other mounting holes and the smaller board size could not be implemented because the large number of components required for the tester functionality did not allow so.

3.5.3 Following the Full Specification

Since the initial tester prototype proved to function properly, many of the components used for prototyping can be safely removed. That includes the components used for programming the serial configuration device over AS interface (as mentioned in section 3.2.1) and SPI bus multiplexer in addition to the extra connectors like the microcontroller's I2C, UART and PortF (which is used for JTAG debugging). Furthermore, if only the pure tester functionality is needed, the additional user peripherals (LEDs and switches) and extra FPGA I/O pins (GPIO_B pins) and their protection circuitry can be removed.

Taking all the mentioned recommendations into consideration, making the tester design completely compliant to the Raspberry Pi HAT specifications will be an extremely easy task.

3.6 Interface to the System Under Test

The interface to the test system is implemented with the following signals:

Interrupt Signal:
Four output connections the tester uses to signal a new interrupt.

Acknowledgment Signal:
Four input connections the tester uses for receiving an acknowledgment signal.

Read/Write signal:
Input signal that controls the direction of the data bus. Used to trigger data logs by software tags.

Databus:
Seven input/output signal for transmitting interrupt identification number or data from software tags.

When the system being tested wants to log data or send an acknowledgment data, it must first add the desired data to the data bus, then lower the Read/!Write signal or raise the acknowledgment signal. Similarly, the Read/!Write signal must be held high in order for the tester to send interrupt signals.

Chapter 4

Protection

4.1 Introduction

A common cause of failure for any electronics product is electrical overstress (EOS). Overstress could be an outcome of numerous scenarios, including supply surges and application of overvoltage. Having effective EOS protection is a primary requirement for product durability.

While both EOS and electrostatic discharge (ESD) are voltage overstress conditions, they differ in the energy involved and time span of the event as shown in table 4.1 [20].

Table 4.1: Comparison of EOS and ESD

Characteristic	**EOS**	**ESD**
Time frame	> 1ms	< 1us
Voltage	smaller values	> 500V
Current	usually more	usually less

ESD protection is out of the scope of this work and only EOS will be discussed.

4.2 Used protection techniques

Having proper protection is particularly important in our system because the tester is proposed to be used in lab environment where it can be handled by students with different backgrounds. The tester will also be connected to external devices which characteristics are not known at design time.

In the next sections, an overview of the possible power related hazards that may affect the circuit will be presented along with the used protection methods to counter their effects.

4.2.1 Reversed Voltage

The first thing to do to protect against unintentional voltage reverse is to make it less likely to happen when providing power from an external source. The simplest way to do that is by using a unidirectional connector like the one shown in figure 4.1, which is mostly suitable to provide power from a battery.

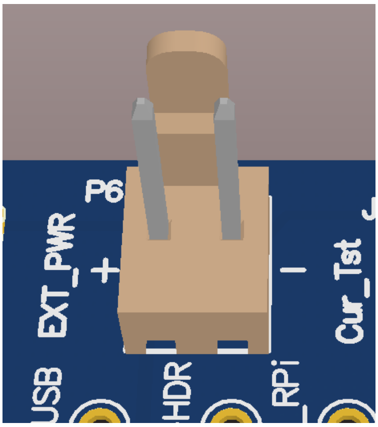


Figure 4.1: Unidirectional battery connector

If the user managed anyway to provide a reversed voltage, the input diodes will make sure no harm is done to the circuit because they will be reverse biased then and will exhibit a large resistance that will almost break the circuit. Figure 4.2 shows the diodes used for the three available inputs. Two Schottky diodes, with forward voltage of 0.4V, are used at each input. Every diode can handle up to 20V of reversed voltage and has a maximum forward current of 1A allowing the circuit to draw up to 2A from each input.

4.2.2 Using multiple power sources simultaneously

If the user connected multiple power sources with different values at the same time, e.g. 3.7V from a battery and 5V from USB, current would start flowing from the USB port to the battery. Luckily, the diodes added in the previous step will provide protection preventing this from happening.

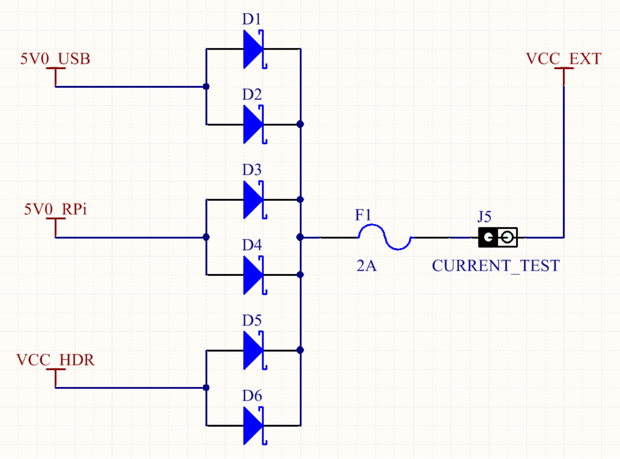


Figure 4.2: Schottky diodes at power input

4.2.3 Over current and short circuit

To provide protection against overcurrent a resettable fuse (polyfuse) is used. The resettable fuse increases its resistance rapidly causing the circuit to breaks when there is a surge of large current passing through it.

Resettable fuses are made up of conductive polymer and the working principal of these resettable fuses is that when the temperature of the device (i.e. the fuse temperature) crosses the threshold limit, they break temporarily. They switch back on when the temperature drops below the threshold (Figure 4.3).

During an over current or short circuit scenario, the current through the fuse increases, this increasing the temperature and in-turn the circuit breaks, thereby protecting the system or device.

4.2.4 Overvoltage

Instead of using dedicated components to provide overvoltage protection, the voltage regulators in the circuit were chosen to have internal overvotlage and overtemperature protection.

4.2.5 Supply noise and voltage spikes

Most ICs suffer performance degradation of some type if there is ripple and/or noise on the power supply pins. A digital IC will incur a reduction in its noise margin and a possible increase in clock jitter [11]. For high performance digital ICs, such as FPGAs, the specified tolerance on the supply (less than 5% in the used FPGA) includes the sum of the

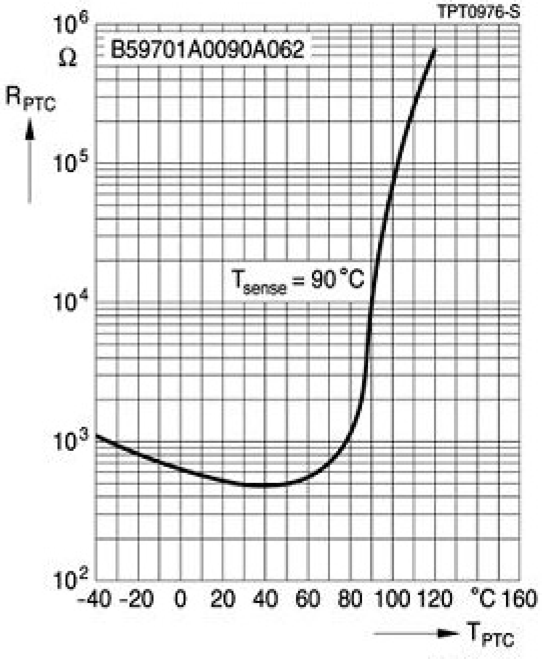


Figure 4.3: Variation of PTC resistance with Temperature, source: hems.de [21]

dc error, ripple, and noise. FPGA will meet specifications if this voltage remains within the tolerance.

Enhancing supply stability is generally done by placing a number of decoupling capacitors near the power pins of the IC. Decoupling capacitors help to stabilize the power distribution bus by supplying current that opposes any change in the bus voltage. Altera has a useful tool that helps PCB designers estimate the number, value, and type of decoupling capacitors needed to develop an efficient PCB decoupling strategy.

The Power Distribution Network (PDN) tool [8] calculates the impedance target of the design (Z_{target}) and plot it against frequency based on the information provided by the user including maximum current consumption in each rail, the nature of the used voltage regulators (linear/switching) and so on. The user's job now is to change the number and value of the used capacitors in each rail until the effective impedance graph is under the target impedance (Z_{eff}) graph all the way until the effective frequency ($f_{effective}$) of the design.

Figure 4.4 shows the plot for the RTS-tester design where we can notice the relation between Z_{eff} (red) and Z_{target} (blue).

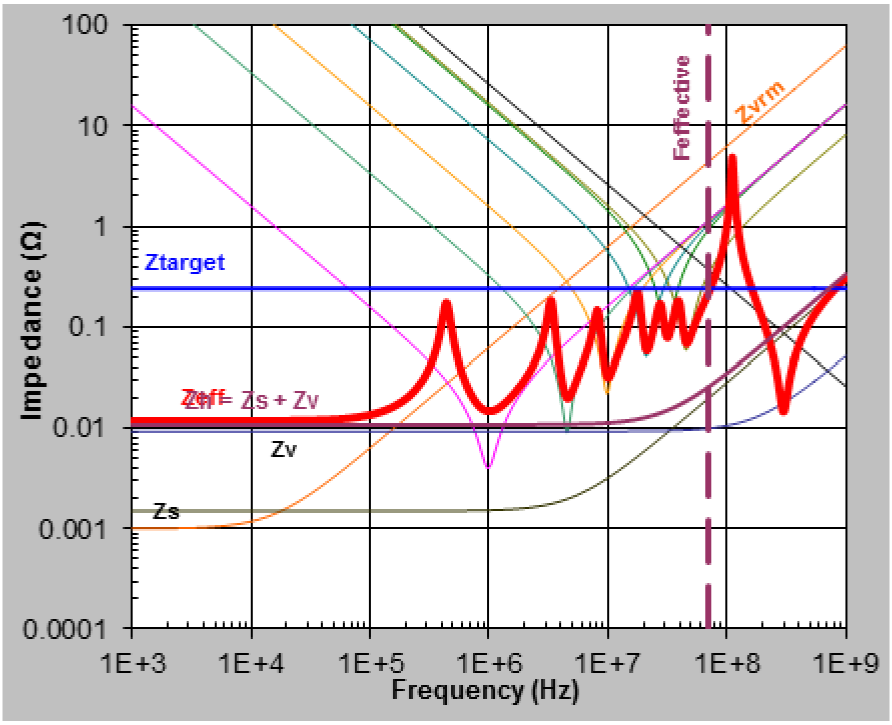


Figure 4.4: A plot generated by PDN tool showing the relation between Z_{eff} and Z_{target}

4.2.6 I/O Overvoltage

Since the tester's I/O pins will be connected to external circuitry with possibly a different power source, those pins should be protected from any overvoltage that may appear on the connection lines. The method used in our design depends on the characteristics of MOS-FET transistors. While this idea is used mostly in level shifting applications, it can also be used to provide protection against overvoltage and voltage spikes.

Figure 4.5 two bus lines connecting the section that needs to be protected with the section with potentially unstable voltage. Each bus line is identical and consists of one discrete N-channel enhancement MOS-FET. The gates (g) has to be connected to the supply voltage of the protected section, the sources (s) to the bus lines of the protected section, and the drains (d) to the bus lines of the unstable section.

Many MOS-FETs have the substrate internally already connected with its source, otherwise it should be done externally. The diode between the drain (d) and substrate is present inside the MOS-FET as n-p junction of drain and substrate.

This structure insures that each source has a maximum output voltage equal to Vdd1. The reader may refer to [17] for more details about the circuit and basic theory of opera-

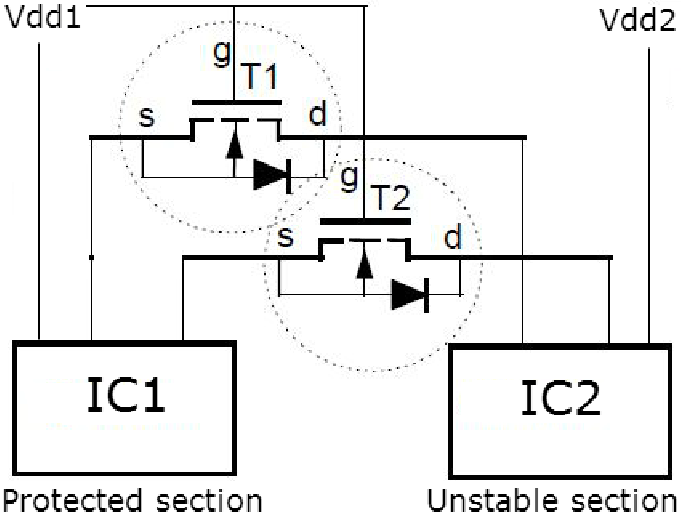


Figure 4.5: Simple circuit illustrating the usage of N-MOS transistors for I/O protection

tion.

Instead of placing a transistor for each I/O pin, the GTL2000 chip from NXP is used. GTL2000 is a 22-bit bi-directional voltage translator operates according to the aforementioned N-MOS based method and it supports voltages from 1.0 V to 5.0 V. This chip was chosen to having a minimum propagation delay in mind. It features a constant 2.5 nS propagation delay which can be easily added to the test results later.

In order to make this voltage translator serve our purpose of overvoltage protection, both gate and drain reference pins (G_{REF} and D_{REF}) must be connected to the tester's 3.3V power rail through a 200kΩ resistor. Sn pins should be connected to the side we want to protect and Dn pins should connect to the side with unstable power. Figure 4.6 shows the pin connections.

4.2.7 I/O Overcurrent

In addition to providing overcurrent protection for the whole circuit, it is also important to take current sinking and sourcing capabilities of the individual FPGA pins into consideration. The importance of this becomes clearer when the user connects an FPGA output (e.g. INT pin) to the DUT where the respective pin is mistakenly configured as output as well resulting in a high probability of a short circuit situation.

The maximum current sinking and sourcing capabilities of the used FPGA pins are 25mA and 40mA respectively. The used I/O standard is 3.3V-LVTTL, which has a current drive characteristic of 4mA according to Altera Cyclone IV datasheet [4]. Whereas, the

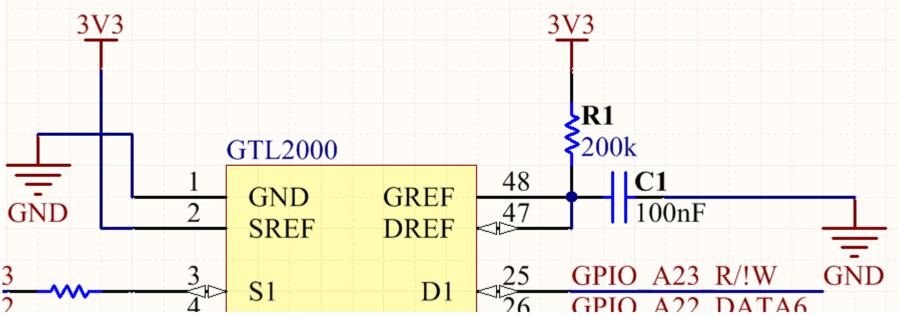


Figure 4.6: Pin connections for GTL2000 based overvoltage protection

current drive capability of the Raspberry Pi, as a standard DUT, range from 2mA to 16mA [26].

Therefore, we need series resistors with values ranging from 767Ω (corresponds to 4mA) to 148Ω (corresponds to 16mA). We should also take into account that the protection chip, GTL2000, has an on-state resistance of 58Ω. Hence, 270Ω resistors were chosen, which would give a maximum current of 10mA for 3.3V.

The Pin Planner tool in Quartus II software allows the user to set the preferred I/O standard to one of several options including LVTTL or LVCMOS along with "Current Strength", which is the maximum current that the pin is allowed to drive. This adds an additional layer of current protection.

Chapter 5

Software

This chapter will have an overview of the software used in Nios II processor, the AVR microcontroller and the PC-based GUI written in C# in addition to the interfaces used for the communications between them.

5.1 The Graphical User Interface

The Graphical User Interface of the tester is created with C# and supports the following functions:

- Performing tests.
- Reading log data to the on-board micro-SD card.
- Reading log data to the PC.
- Configuring the tester (not fully implemented).
- Performing basic analysis of test results.
- Performing time analysis of test results.

The reader may refer to Section 6.5 for a detailed description of these functions and their use. Figure 6.12 in the same chapter shows the GUI and its components.

The GUI consists of several modules each of which is responsible for a dedicated function including serial communications, file management, log data processing, etc. This modularity makes the software easier to modify and develop.

5.2 Microcontroller Software

The microcontroller serves as the interface between the PC and the FPGA in addition to handling SD card storage. Most of the firmware is related to USB interface, SD card interface and SD card file system (FAT16). The rest is a state machine used for handling the commands received from the PC.

5.3 Nios II software

The firmware on Nios can be divided to three main functions:

Main
It keeps waiting for commands from the microcontroller to perform the required function and send back the results.

Start_Test
Called when a "test" command is received. It handles setting up the control register of the tester module with the correct configuration and performing the actual test.

Dump_Memory
Called when a "read memory" command is received. It reads the memory associated with the tester module and arranges the data in packets. After that, it sends the packets to the microcontroller over UART where it can be processed further.

Nios does not differentiate between read-to-PC and read-to-SD commands. The microcontroller simplifies the task to a single read-memory command and it handles the rest afterwards.

5.4 Main Operations

The system uses a simple one-character based commands to facilitate the communication between the units. For example, let us assume ’c’ is the character associated with the desired command. The commands have the form of "c\n" in the MCU=>PC interface and a simple "c" format in the rest of the interfaces.

Next, we will have an idea of how the various commands are handled in the system.

5.4.1 Test Operation

Figure 5.1 shows the commands sequence to perform the test operation. Figure 5.2 shows the actual commands sent.

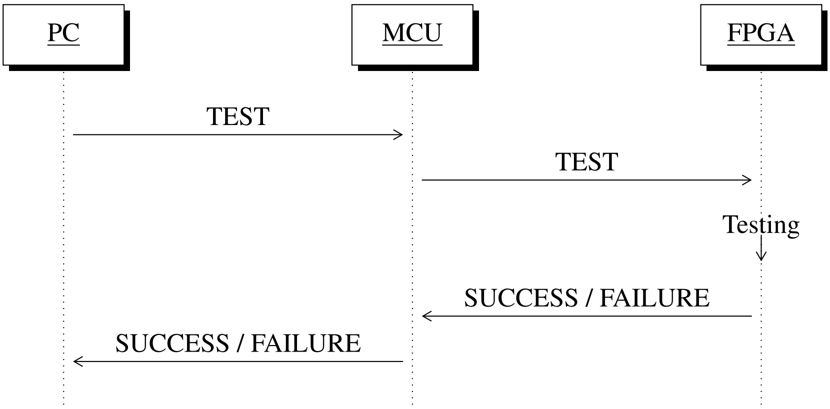


Figure 5.1: Testing Procedure

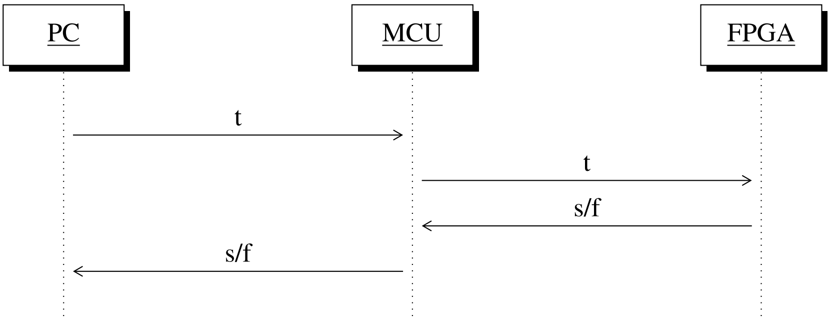


Figure 5.2: Actual Test Commands

5.4.2 Read-to-SD Operation

Figure 5.3 shows the commands sequence to perform the read-to-SD operation. Figure 5.4 shows the actual commands sent.

5.4.3 Read-to-PC Operation

Figure 5.5 shows the commands sequence to perform the read-to-PC operation. Figure 5.6 shows the actual commands sent.

5.4.4 Configuration Command

Figure 5.7 shows the commands sequence to perform the tester configuration. Figure 5.8 shows the actual commands sent.
The configuration data is sent in the following order:

- DATA[0]: INT A PERIOD (us)

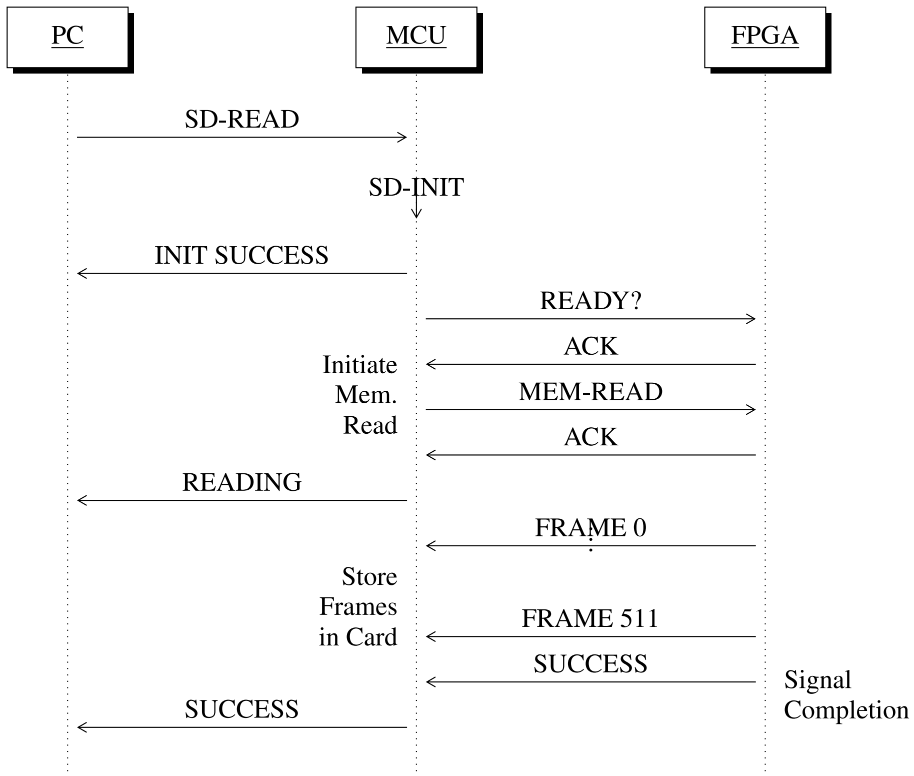


Figure 5.3: Read-to-SD Operation

- DATA[1]: INT B PERIOD (us)
- DATA[2]: INT C PERIOD (us)
- DATA[3]: INT D PERIOD (us)
- DATA[4]: BURST WRITE
- DATA[5]: TIME SCALER
- DATA[6]: HOLD TIME (ns)
- DATA[7]: TEST DURATION (ms)

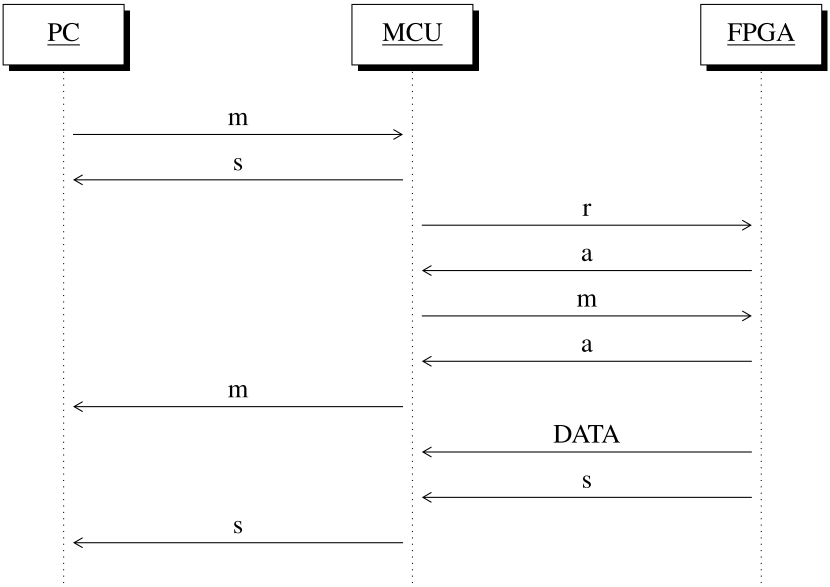


Figure 5.4: Actual Commands for Reading to SD card

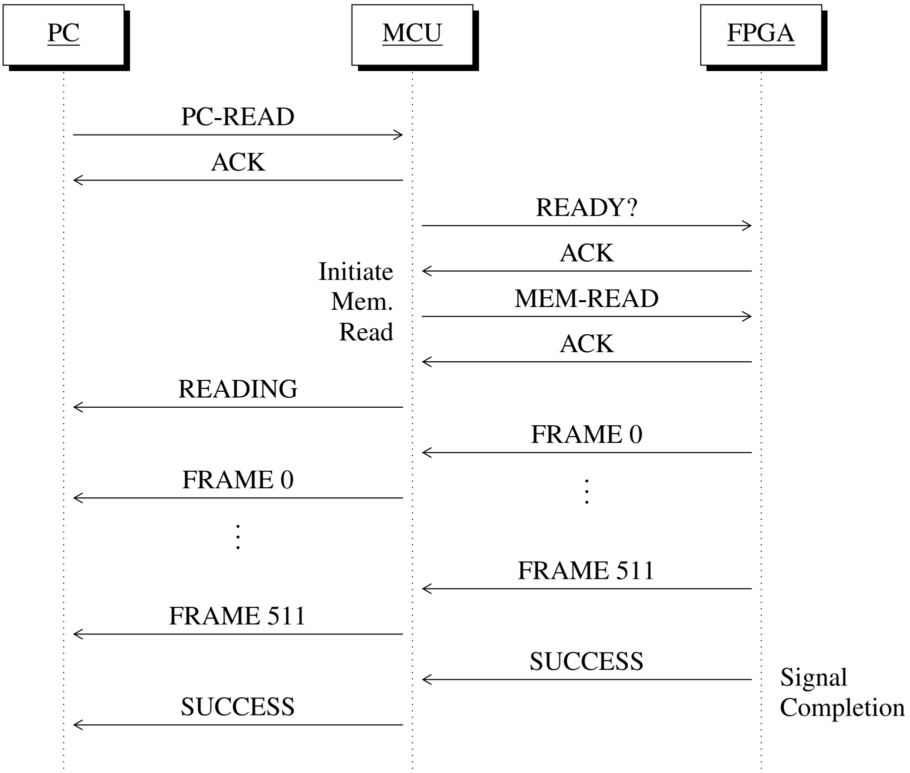


Figure 5.5: Read-to-PC Operation

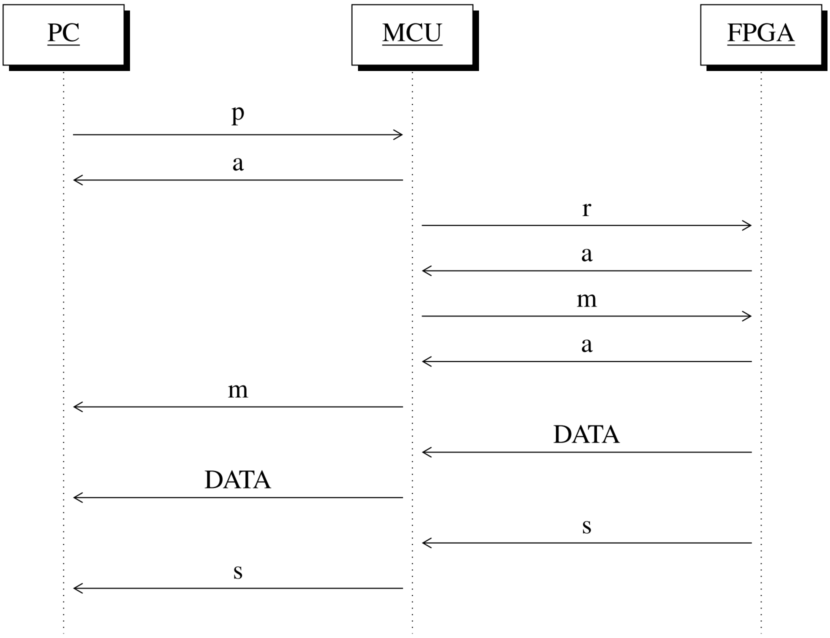


Figure 5.6: Actual Commands for Reading to PC

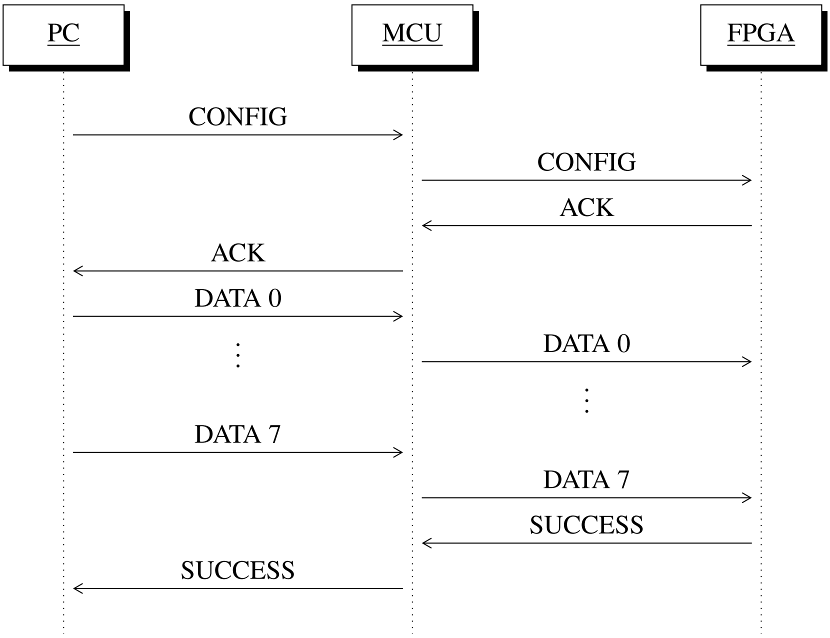


Figure 5.7: Tester Configuration

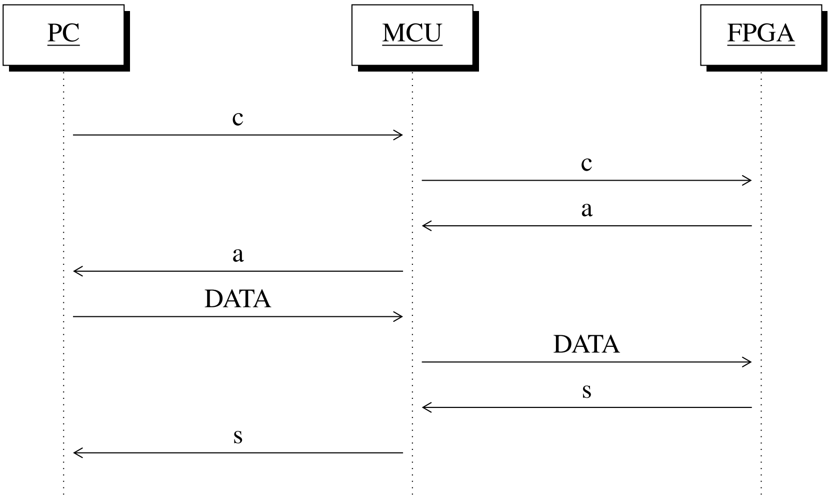


Figure 5.8: Actual Configuration Commands

Chapter 6

Using The Tester

This chapter will cover all the necessary aspects required to fully utilize and configure the tester.

6.1 Configuring the FPGA

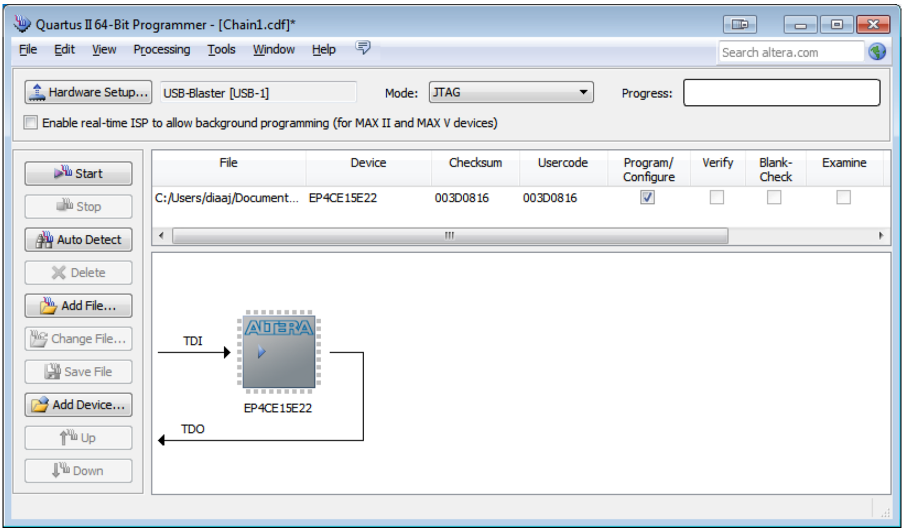
When compiling a project, Quartus II Compiler's Assembler module automatically generates an SRAM Object File (.sof) that contains the data for configuring SRAM-based FPGAs. This file can be directly used to program the FPGA using the following steps:

1. Launch Quartus II Programmer software.
2. Click on "Hardware Setup..." and choose the connected download cable (e.g. USB Blaster).
3. Click on "Add File..." and choose the .sof file from the project directory.
4. Check "Program/Configure" option next to the device name.
5. When your programmer window looks like Figure 6.1 click the "Start" button.
6. The progress bar should show a successful operation.

Note: If after adding the file the programmer failed to detect the device, try "Auto Detect" first, select the device from the list and then use "Change File" option to choose the programming file.

6.2 Programming the Configuration Device

The configuration device is a flash memory that can be used to retain the FPGA configuration file after power reset. When powered, the FPGA will program itself automatically over AS interface. The configuration device can be programmed either using JTAG through the FPGA or through AS interface.



Hardware Setup... USB-Blaster [USB-1]   Mode: JTAG   Progress:

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
C:/Users/diaaj/Document...	EP4CE15E22	003D0816	003D0816	☑	□	□	□

Figure 6.1: Quartus II Programmer window after completing the setup

6.2.1 Using JTAG Interface

The serial configuration device cannot be programmed using the .sof file generated by default. This file needs to be converted to a JTAG Indirect Configuration File (.jic) first.

Generate JIC File

The following steps should be followed:

1. Start Quartus II software.
2. From Quartus II, start "Convert Programming File" utility from "File" menu as seen in Figure 6.2.
3. Setup the conversion parameters (Figure 6.3):
   - Select "Programming file Type" as "JTAG Indirect Configuration File (.jic)"
   - Select "Configuration Device" as "EPCQ16" (or the correct one if another chip is used).
   - Select "Mode" as "Active Serial"
   - Enter "File name" for the .jic file to be generated
4. Select the "SOF Data" line on the bottom panel and click "Add File" (Figure 6.4).
5. Browse and select the .sof file containing the hardware design.
6. Select the "Flash Loader" line on the bottom panel and click "Add Device" (Figure 6.5).

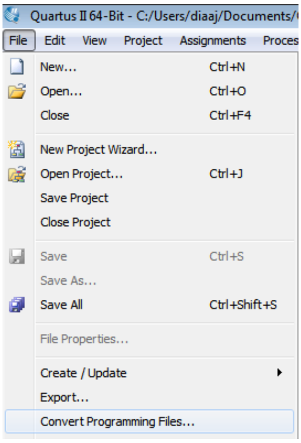
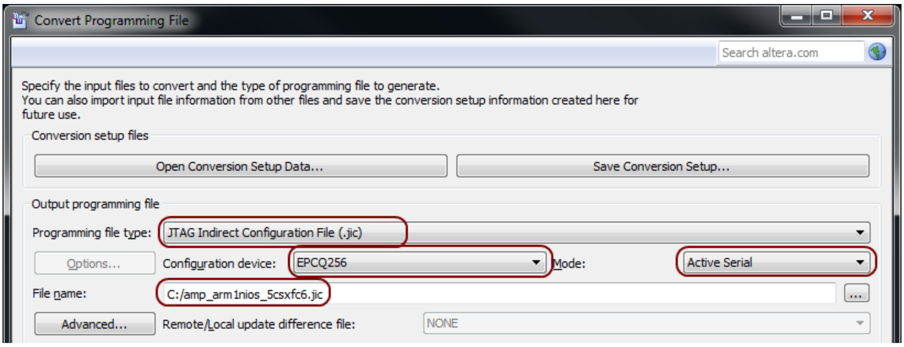


Figure 6.2: Starting the File Converter from Quartus II Programmer



Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

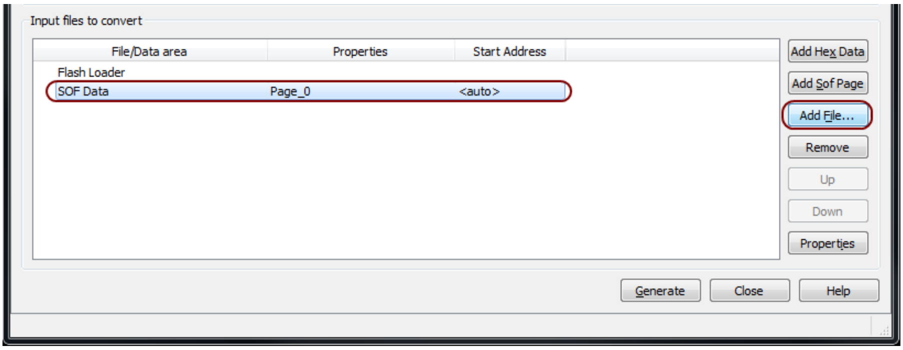
Configuration device: EPCQ256

Mode: Active Serial

File name: C:/amp_arm1nios_5csxfc6.jic

Remote/Local update difference file: NONE

Figure 6.3: Setting up the conversion parameters

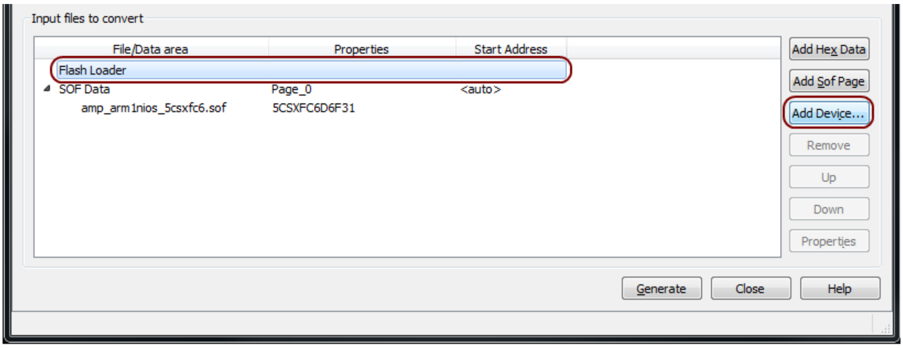


Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>

Figure 6.4: Adding the .sof file

7. Select the correct device from the list. In our case, this is Cyclone IV EP4CE15.



Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>
amp_arm1nios_5csxfc6.sof	5CSXFC6D6F31	

Figure 6.5: Choosing the correct target device

8. Click "Generate" button.

9. A completion message is displayed. Click "OK" to close it.

Note: A Conversion Setup Data file has been created so the configuration can be set automatically. The file name is "sop_to_jic_conversion(jtag).cof" and it is located in the Quartus project directory. It can be used by clicking "Open Conversion Setup Data..." button in the Converter window.

Program the Device

1. Start Quartus II Programmer.

2. Click on "Hardware Setup..." and choose the connected download cable (e.g. USB Blaster).

3. Set the "Mode" field to "JTAG".

4. Click on "Add File..." and choose the generated .jic file from the project directory.

5. Check "Program/Configure" option next to the device name.

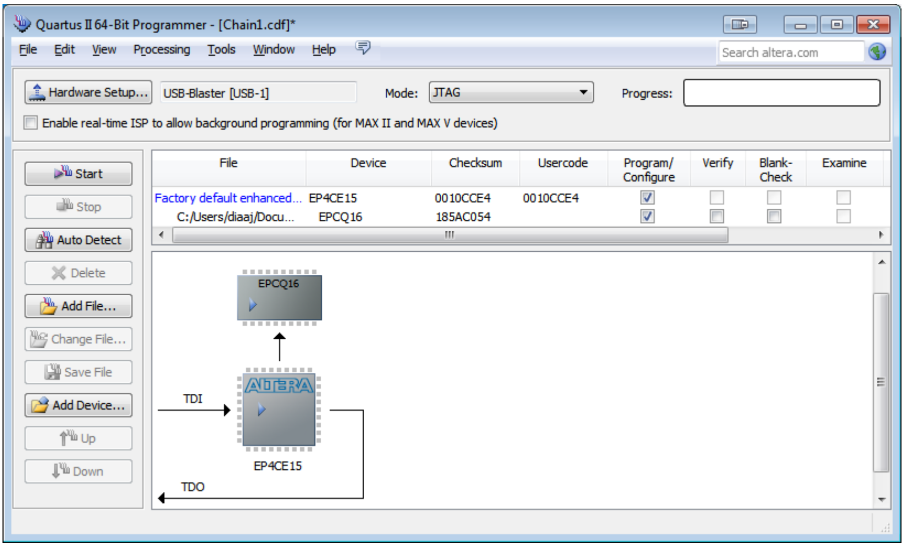
6. When your programmer window looks like Figure 6.6 click the "Start" button.

7. The progress bar should show a successful operation.

Note: If after adding the file the programmer failed to detect the device, try "Auto Detect" first, select the device from the list and then use "Change File" option to choose the programming file.

6.2.2 Using AS Interface

To program the serial configuration device through AS interface, the .sof file generated by default needs to be converted to a Programmer Object File (.pof) first.



Quartus II 64-Bit Programmer - [Chain1.cdf]*

Hardware Setup... USB-Blaster [USB-1] Mode: JTAG Progress:

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
Factory default enhanced...	EP4CE15	0010CCE4	0010CCE4	☑	□	□	□
C:/Users/diaaj/Docu...	EPCQ16	185AC054		☑	□	□	□

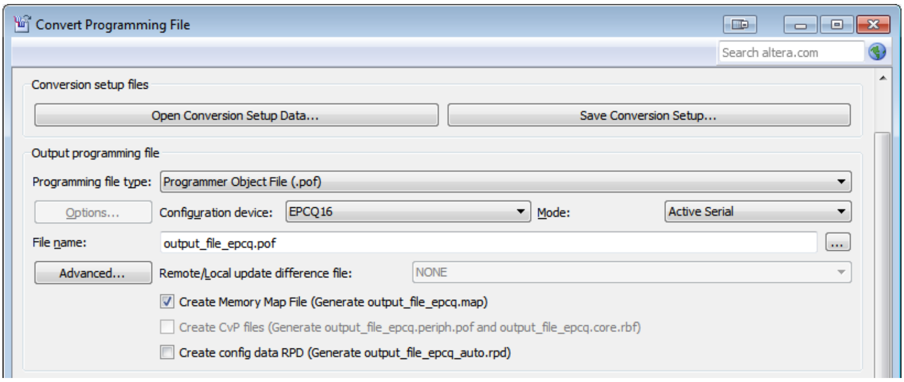
Figure 6.6: Quartus II Programmer window after completing the setup

Generate POF File

The following steps should be followed:

1. Start Quartus II software.
2. From Quartus II, start "Convert Programming File" utility from "File" menu as seen in Figure 6.2.
3. Setup the conversion parameters (Figure 6.7):
   - Select "Programming file Type" as "Programmer Object File (.pof)"
   - Select "Configuration Device" as "EPCQ16" (or the correct one if another chip is used).
   - Select "Mode" as "Active Serial"
   - Enter "File name" for the .pof file to be generated
4. Select the "SOF Data" line on the bottom panel and click "Add File" (Figure 6.8).
5. Browse and select the .sof file containing the hardware design.
6. Click "Generate" button.
7. A completion message is displayed. Click "OK" to close it.

Note: A Conversion Setup Data file has been created so the configuration can be set automatically. The file name is "sop_to_pof_conversion(AS).cof" and it is located in the Quartus project directory. It can be used by clicking "Open Conversion Setup Data..." button in the Converter window.



Programming file type: Programmer Object File (.pof)

Configuration device: EPCQ16

Mode: Active Serial

File name: output_file_epcq.pof

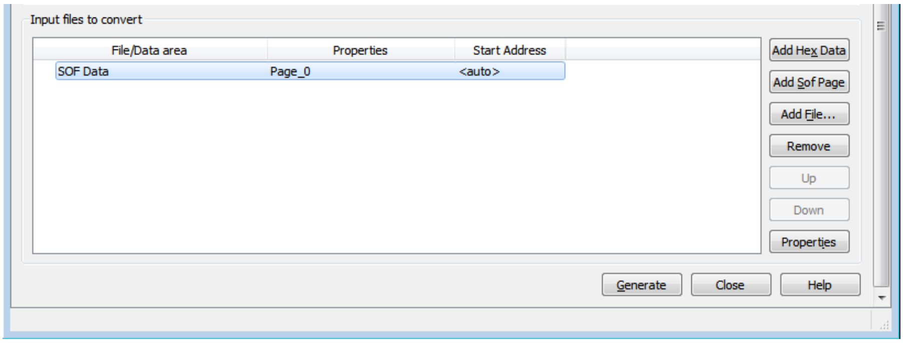
Remote/Local update difference file: NONE

☑ Create Memory Map File (Generate output_file_epcq.map)

□ Create CvP files (Generate output_file_epcq.periph.pof and output_file_epcq.core.rbf)

□ Create config data RPD (Generate output_file_epcq_auto.rpd)

Figure 6.7: Setting up the conversion parameters



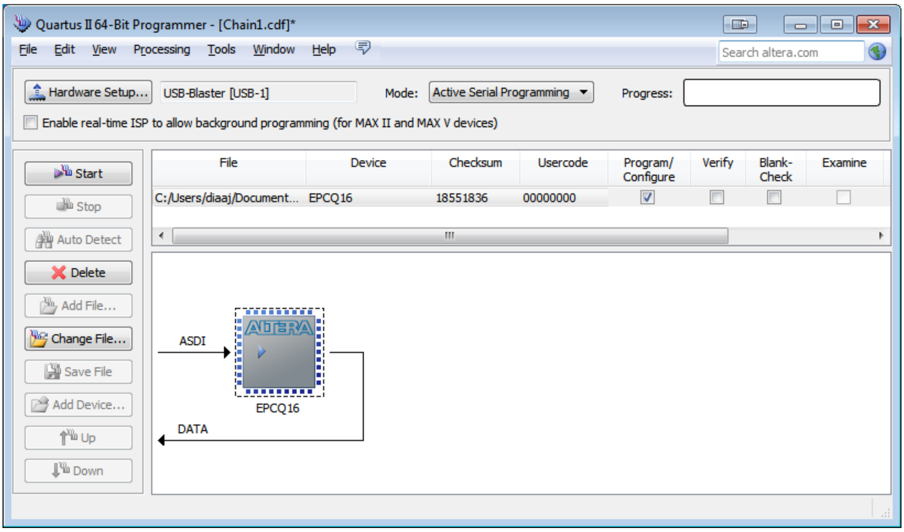
File/Data area	Properties	Start Address
SOF Data	Page_0	<auto>

Figure 6.8: Adding the .sof file

Program the Device

1. Start Quartus II Programmer.
2. Click on "Hardware Setup..." and choose the connected download cable (e.g. USB Blaster).
3. Set the "Mode" field to "Active Serial Programming".
4. Click on "Add File..." and choose the generated .pof file from the project directory.
5. Check "Program/Configure" option next to the device name.
6. When your programmer window looks like Figure 6.9 click the "Start" button.
7. The progress bar should show a successful operation.

Note: If after adding the file the programmer failed to detect the device, try "Auto Detect" first, select the device from the list and then use "Change File" option to choose the programming file.



Quartus II 64-Bit Programmer - [Chain1.cdf]*

Hardware Setup... USB-Blaster [USB-1]

Mode: Active Serial Programming

Progress:

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
C:/Users/diaaj/Document...	EPCQ16	18551836	00000000	☑	□	□	□

Figure 6.9: Quartus II Programmer window after completing the setup

6.3 Programming Nios II Processor

If further development to the firmware is desired the Nios II Embedded Design Suite needs to be installed. Nios II EDS is installed automatically with Quartus II software. Alternatively, it can be obtained from Altera website as a stand alone package.

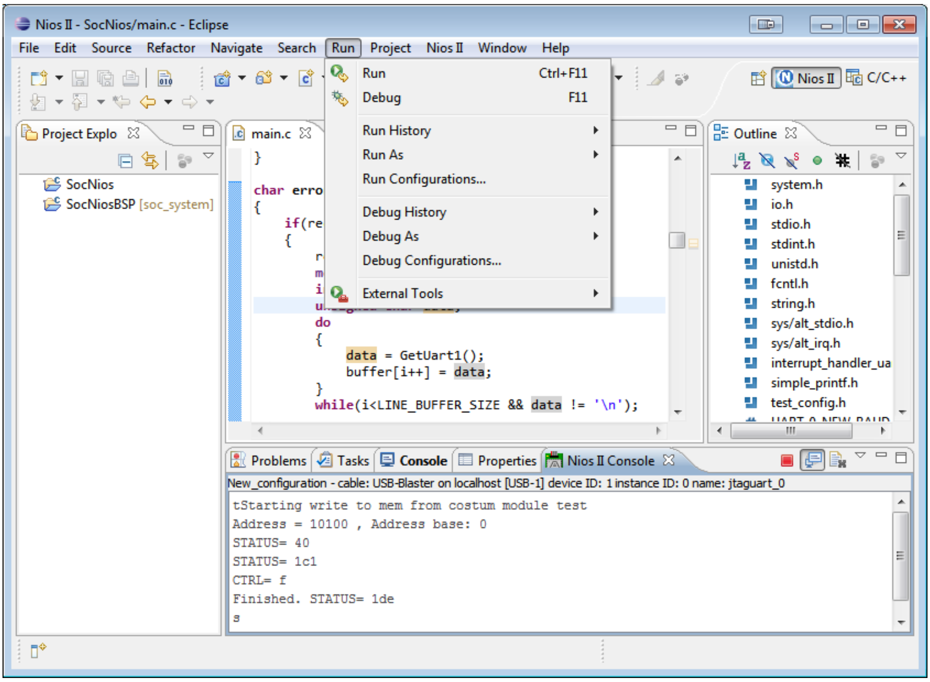
The following steps need to be followed the first time you run the software:

1. Start Nios II EDS software.
2. It will ask for the workspace location. Browse to "/Project location/ software/ soc_system_RT_CIV-workspace".
3. Make sure USB Blaster is connected to the JTAG connector on the tester.
4. Do the required modifications to the code.
5. Choose the "Run" option from "Run" menu as shown in Figure 6.10.

A progress bar should appear indicating the operation progress. Figure 6.10 shows the Run option and how the window should look like. If the operation was unsuccessful, the user is advised to check "Run Configuration" to make sure the EDS has detected the programming cable. It can be opened from "Run" menu.

6.4 Programming the Microcontroller

Programming the AVR Atmega32U4 microcontroller can be done using the 6-pin In-System Programming (ISP) connector on the left of the board. This is a standard program-



```
}

char erro
{
    if(re
    {
        r
        m
        i
        u
        do
        {
            data = GetUart1();
            buffer[i++] = data;
        }
        while(i<LINE_BUFFER_SIZE && data != '\n');
```

New_configuration - cable: USB-Blaster on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0

```
tStarting write to mem from costum module test
Address = 10100 , Address base: 0
STATUS= 40
STATUS= 1c1
CTRL= f
Finished. STATUS= 1de
s
```

Figure 6.10: Nios II Embedded Design Suite

ming connector for Atmel AVR devices and all compatible programmers should support by default. It is worth mentioning that this connector can be used only to program the device, i.e. device debugging is not supported by the ISP protocol.

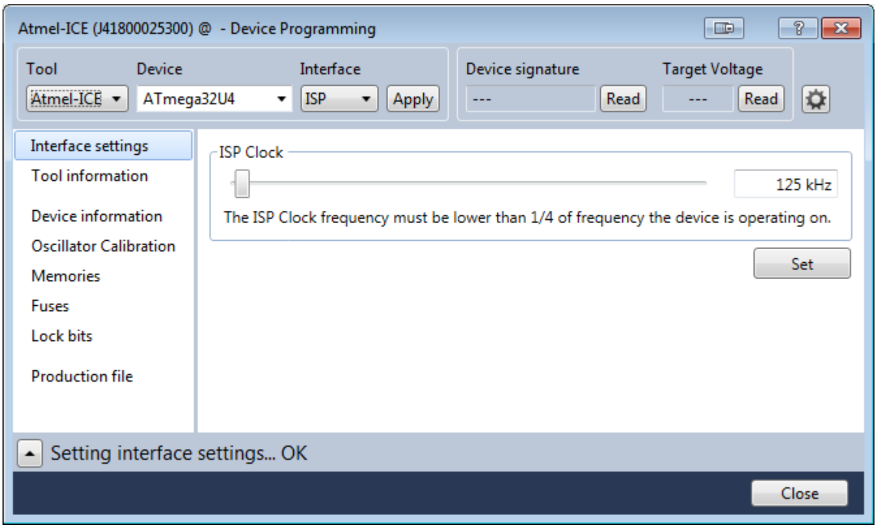
ISP was chosen over JTAG for programming mainly because of the smaller footprint of the ISP connector compared to JTAG, which helped saving valuable PCB space. This choice should be suitable for the application since development is assumed to be done on an external platform and only the final working firmware needs to be transferred to the tester. If debugging is urgently needed, however, the JTAG pins of the microcontroller have been exposed just next to it in the form of PortF pins. A few wires can be used to make a temporary JTAG connector. The author used this trick to debug the microcontroller's firmware when a last minute bug was discovered.

The following steps should be followed to program the AVR device using Atmel ICE programmer:

1. Launch Atmel Studio software.

2. From "File->Open->Project/Solution.." browse to the "USB_Interface" project directory and choose "USB_Interface.atsln" file.

3. After the solution opens launch the "Device Programmer" from the "Tools" menu.



Atmel-ICE (J41800025300) @ - Device Programming

Tool: Atmel-ICE

Device: ATmega32U4

Interface: ISP

Device signature: ---

Target Voltage: ---

ISP Clock: 125 kHz

The ISP Clock frequency must be lower than 1/4 of frequency the device is operating on.

Setting interface settings... OK

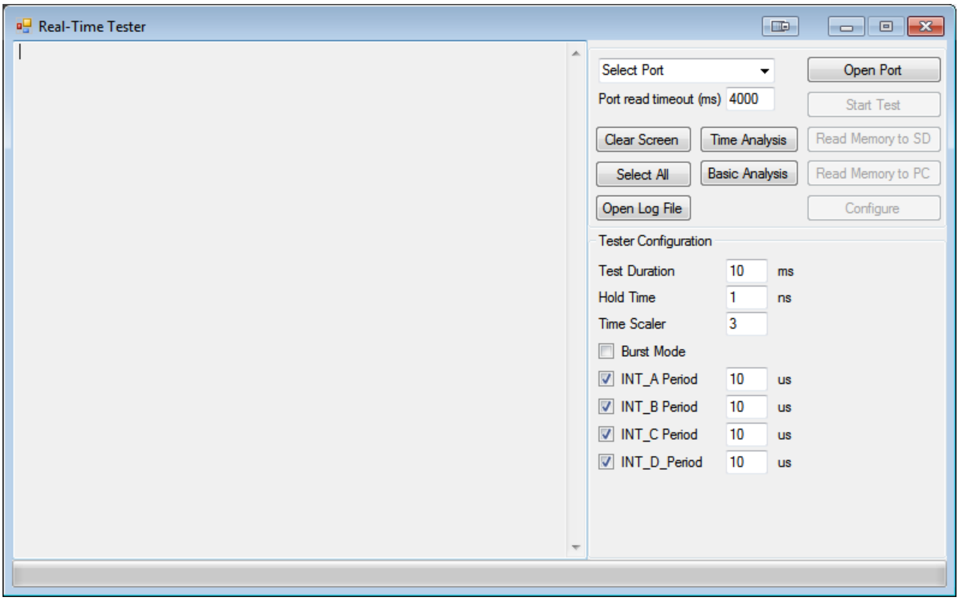
Figure 6.11: Setting up Atmel Device Programmer

4. Setup the programming parameters: (Figure 6.11)
    - Tool: Atmel-ICE
    - Device: ATmega32U4
    - Interface: ISP
    - ISP Clock: Any number below 250kHz
    - Click "Set" to set the chosen ISP clock.
5. Click "Apply".
6. Click "Read" button next to "Device Signature" field.
7. If the device signature is displayed and no error message is shown proceed. Otherwise, check that the programmer is connected and the device is powered on. The correct "Target Voltage" should be displayed as well.
8. Close the "Device Programmer" window.
9. From "Debug" menu choose "Start Without Debugging".
10. The firmware now should be transferred to the microcontroller.

Note: If JTAG is used the user can follow a similar procedure with choosing "JTAG" as the interface in the "Device Programmer" window.

6.5 Using the Graphical User Interface

After launching the Real-Time Tester GUI it will look similar to Figure 6.12. If no port is selected, the buttons related to controlling the tester will be deactivated. The user can use the timeout after which the software will stop waiting for an answer from the tester.



Port read timeout (ms) 4000

Test Duration 10 ms

Hold Time 1 ns

Time Scaler 3

□ Burst Mode

☑ INT_A Period 10 us

☑ INT_B Period 10 us

☑ INT_C Period 10 us

☑ INT_D_Period 10 us

Figure 6.12: Real-time Tester Graphical User Interface

6.5.1 Performing a Test

1. Make sure the correct port is open.
2. Click "Start Test".
3. A message saying "Test successful" should be printed.

6.5.2 Reading Test Results

1. Make sure the correct port is open.
2. Click "Read Memory to SD" or "Read Memory to PC".
3. Messages in the status text-box (left of the window) will inform you of the operation status and the log file storage location. The progress bar will show the progress of the current operation.
4. The message ""Memory read successfully" should be printed if the operation ends successfully.

6.5.3 Analyzing Test Results

The software offers the possibility to perform Basic Analysis and Time Analysis of the collected data.

1. Select the log to be analyzed by clicking "Open Log File". The default log file extension is ".rts".

2. Choose either "Basic Analysis" or "Time Analysis" to start the analysis.

3. Analysis output will be printed on the screen in a similar manner to Figure 7.6 and Figure 7.7.

6.5.4 Configuring the Tester

The Tester can be easily configured by setting the desired parameters in the interface and clicking "Configure". A success message will be displayed when the tester is configured.

Chapter 7

Testing and Result Analysis

This chapter will present the available testing techniques and how the SUT can be programmed in each of them. Finally, it will have an overview of performing result analysis both manually and using the accompanying software.

7.1 Available Testing Techniques

7.1.1 Software Tags:

This technique is implemented using data lines and read/!write signal in the module where SUT lowers read/!write signal after adding the desired data on the data lines. This operation takes place where the user has placed a mark in the code. How this software tag is implemented depends on the system architecture and programming language selected by the user. Figure 7.1 shows the desired behavior of the signal lines using software labels.

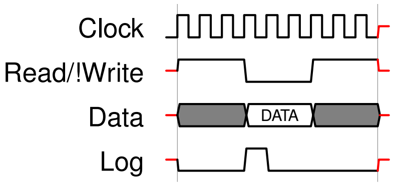


Figure 7.1: Desired behavior of logging with software tags

7.1.2 Interrupt to acknowledgment

Implemented by sending an interrupt signal on one of the interrupt lines; then the tester waits for a response on the respective acknowledgment line. If SUT want to write back the identification number, it places it out on the data lines before the acknowledgment signal is raised. Figure 7.2 shows the desired behavior.

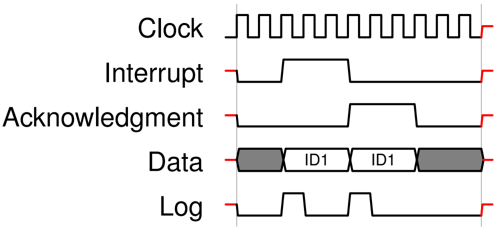


Figure 7.2: Desired behavior of interrupt to acknowledgment tests

7.1.3 Interrupt to thread trigger

This gives an idea of the time taken from issuing an interrupt to running a thread that is waiting for this interrupt. In this case, SUT must place the same identification number it received on the bus and initialize a data-logging at the start of the thread that is triggered by the operating system in response to the interrupt. Figure 7.3 demonstrates this.

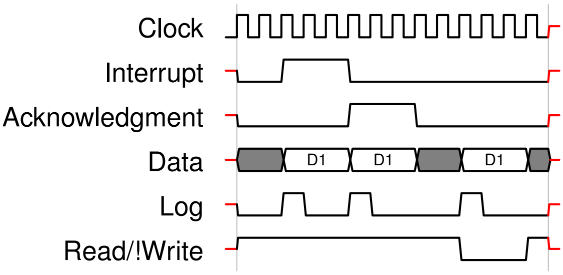


Figure 7.3: Desired behavior of interrupt activated thread test

7.2 Programming SUT to interact with the tester

The tester can be used to implement many different tests. The following sections give an example of how to carry out tests of real-time systems as suggested in [15].

7.2.1 A to B Tests and Execution Logging

This type of testing is the most flexible of the test methods. It requires connecting the read/!write signal and one or more data lines. The pseudo-code in Listing 7.1 shows how a software label can be created in C to minimize the overhead during runtime; code 7.2 is the equivalent function call. In the code below it is assumed that PORTB is connected to the data lines and that the most significant bit (the 8th bit) is the read/!write signal, which is set to 1. Using a macro instead of a function call avoids processor jump in assembly code in order to implement a data logging. Using macros will increase the program size, but the proposed macro is so short that it will make a very little difference and this penalty is insignificant compared to the overhead produced by a function call.

A to B testing is performed by placing two labels, or more, in the code and a logging operation is performed every time the execution arriving at one of these points. When the test is completed the results will be analyzed and the user will be able to figure out how long the program has taken between points. A practical example of A to B tests is how much time does the processor take for context switching, synchronization of threads or critical sections in addition to estimation of worst-case execution time.

Listing 7.1: Example of using the macro software label

```
#define TAG (data) PORTB = data; \
PORTB &= ~(0x80); \
PORTB |= 0x80;
```

Listing 7.2: Example of a function that places the software label

```
Void tag (data) {
PORTB = data;
PORTB &= ~(0x80);
PORTB |= 0x80;
}
```

7.2.2 Interrupt to the Acknowledgment and Interrupt to Thread

By connecting the desired number of acknowledgment and interrupt signal pairs and using pseudo code from Listing 7.3, the user can obtain information about the response time of a system, which is the time taken between interrupt reception and the system starting to deal with it. In the example code below, data lines are not being used. The average response time can be found by analyzing the tester log data.

If the user includes the data lines, he/she can also see how long the system takes to move from an interrupt to a thread waiting for it. Listing 7.4 shows an example of a code to perform that. An important factor to consider here is that it can take variable times to conduct context switching between threads or between an interrupt service routine (ISR) and a thread because some processors include shadow registers or other techniques to minimize switching time.

Listing 7.3: Example of ISR for the interrupt-to-acknowledgment test

```
void ISR () {
PORTA |= 0x1;        // Acknowledgment = 1
PORTA &= ~(0x1)        // Acknowledgment = 0
}
```

Listing 7.4: Example of logging in a thread for an interrupt-to-thread test

```
void ISR () {
DDRB = 0;              // Set the data direction to read
data = PORTB;          // Read the global variable data
PORTA |= 0x1;          // Acknowledgment = 1
PORTA &= ~(0x1);       // Acknowledgment = 0
signal(thread);        // Signal to a thread
}

Void Thread () {
DDRB = 1;              // Set the data direction to read
PORTB = data;          // Read the global variable data
PORTB &= ~(0x80);      // Toggle r/!w line to start logging
PORTB |= 0x80;
}
```

7.3 Analysing Test Results

7.3.1 Data Logging Format

Data logging is initiated upon receipt of acknowledgments, requests for data logs from SUT or sending interrupts from the tester. The following information is included in each log:

Start of frame (4 bits)
Is always b0101 so the user can see that the data is written to memory correctly.

Data type (2bits)
Indicates if this log was because of an acknowledgment, data log or interrupt. Data types are, in order of priority, 2 for acknowledgments, 3 data logs and 1 for interrupts.

Active interrupt signal (4 bits)
The active interrupt signals during logging.

Active acknowledgment signal (4 bits)
The active acknowledgment signals during logging.

Data (7 bits)
Data or identification number located on the data lines.

Timestamp (43 bits)
Time Counter value when data logging took place.

For example, this is a line from a data log generated by the tester

$$56C1500000011936$$

When converted to binary, it will look like this:

0101 01 1011 0000 0101010 00000000000000000000000000010001100100110110

By applying the rules above to analyze this bit string we find that:

0101
Start of frame.

01
Type: data log is caused by an interrupt.

1011
Status of interrupt lines: all interrupt lines are high except for INT2.

0000
Status of acknowledgment lines: no acknowledgment is present.

0101010
Interrupt identification number is 42.

00000000000000000000000000010001100100110110
Timestamp: this log is recorded at time 71990.

7.3.2 Example of analyzing actual log data

Table 7.1 shows a few lines extracted from an actual log with their binary counterpart organized according to the format mentioned in the previous section. It is worth mentioning that the time mentioned in the Timestamp column may not correspond to actual clock cycles. This depends on the Time Scaler configuration of the tester. This will be explained with a practical example in the next section.

We notice in the table that the interrupt signals are initially issued from time 165 until 216 and the value of the data type field is 01 indicating an interrupt-issued event. The first acknowledgment is received at time 639 from ACK0. At this point, the data type field changes to 10_2 (2) indicating and acknowledgment event even though INT0 line has changed as well. This is because acknowledgment reception has more priority than interrupt signal change.

The analysis can continue in the same manner to extract more information about the test from log data and eventually a full timing diagram to describe the test can be built. Figure 7.4 shows the timing diagram constructed based on the log data in the table.

7.3.3 Comparing log data to the actual signals

SignalTap II was used to capture the test shown in figure 7.5. Figure 7.6 shows part of the analysis of the respective dump data generated by the RTS-tester GUI software. The timing mentioned next is actually the number of clock cycle since the start of the test. Since we are using a 50MHz clock oscillator (i.e. 20ns period), each number should be multiplied with 20 to get the actual time in ns.

Table 7.1: Sample log data

	Raw Hex	Type	INT Lines	ACK Lines	DATA Lines	Timestamp
0	54400000000000a5	01	0001	0000	0000000	165
1	54c00000000000b6	01	0011	0000	0000000	182
2	55c00000000000c7	01	0111	0000	0000000	199
3	57c00000000000d8	01	1111	0000	0000000	216
4	5b8408000000027f	10	1110	0001	0000001	639
5	57c008000000029f	01	1111	0000	0000001	671
6	5b480800000003f5	10	1101	0010	0000001	1013
7	5a50080000000439	10	1001	0100	0000001	1081
8	56c0080000000483	01	1011	0000	0000001	1155
9	58e0080000000489	10	0011	1000	0000001	1161
10	55c00800000005e2	01	0111	0000	0000001	1506

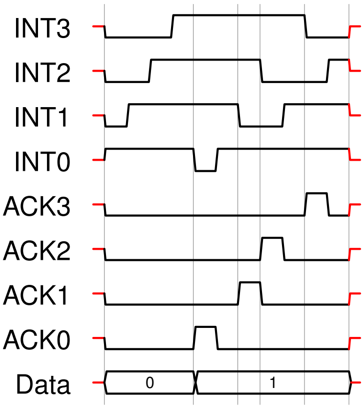
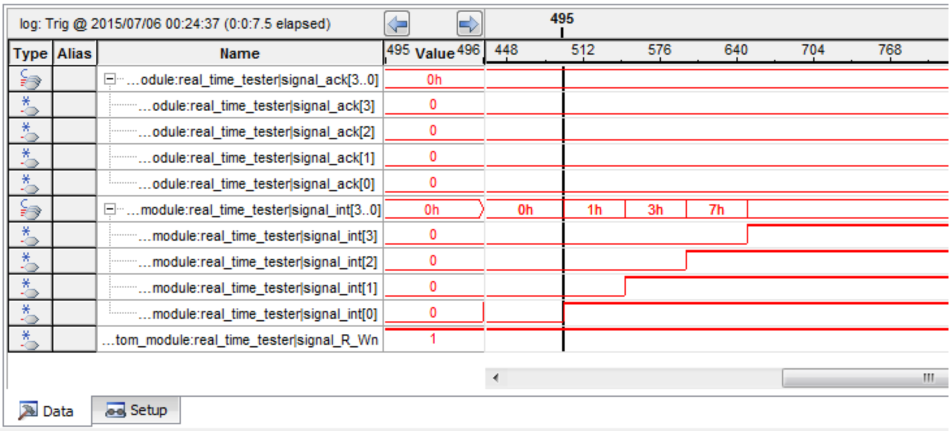


Figure 7.4: Timing diagram built based on extracted log data

Figure 7.5 shows that the first interrupt to be issued is INT0 at clock 495. Line 0 of the analysis output in figure 7.6 shows that the interrupt bus has the value of 0001 at time 165, meaning INT0 was activated at this moment. According to the configuration shown in the same figure, the time scaler value is '3'. This means that the scaling timer will count once every three clock cycles. Therefore, the timestamp 165 corresponds to 495=165*3, which is the same information we had from the logic analyzer.

7.3.4 Getting response duration information

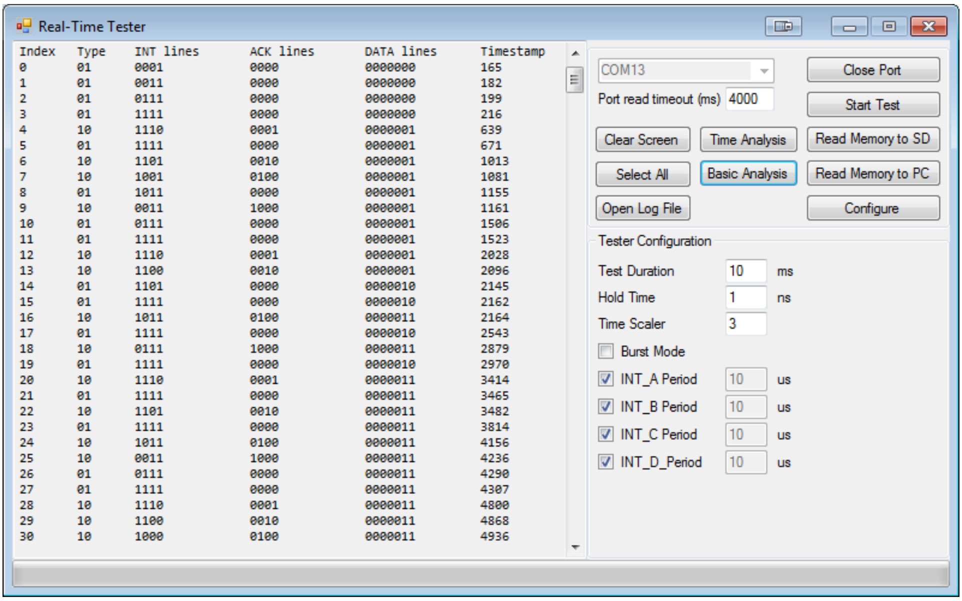
Using the information presented earlier, one can easily deduce the response time for every interrupt issued. The tester software can also be used to automatically calculate this data and print it for each interrupt line. Figure 7.7 shows this information for the log example



log: Trig @ 2015/07/06 00:24:37 (0:0:7.5 elapsed)

Type	Alias	Name	Value
		...odule:real_time_tester|signal_ack[3..0]	0h
		...odule:real_time_tester|signal_ack[3]	0
		...odule:real_time_tester|signal_ack[2]	0
		...odule:real_time_tester|signal_ack[1]	0
		...odule:real_time_tester|signal_ack[0]	0
		...module:real_time_tester|signal_int[3..0]	0h
		...module:real_time_tester|signal_int[3]	0
		...module:real_time_tester|signal_int[2]	0
		...module:real_time_tester|signal_int[1]	0
		...module:real_time_tester|signal_int[0]	0
		...tom_module:real_time_tester|signal_R_Wn	1

Figure 7.5: Tester signals captured by SignaTap II



Real-Time Tester

```
Index  Type   INT lines       ACK lines       DATA lines      Timestamp
0      01     0001            0000            0000000         165
1      01     0011            0000            0000000         182
2      01     0111            0000            0000000         199
3      01     1111            0000            0000000         216
4      10     1110            0001            0000001         639
5      01     1111            0000            0000001         671
6      10     1101            0010            0000001         1013
7      10     1001            0100            0000001         1081
8      01     1011            0000            0000001         1155
9      10     0011            1000            0000001         1161
10     01     0111            0000            0000001         1506
11     01     1111            0000            0000001         1523
12     10     1110            0001            0000001         2028
13     10     1100            0010            0000001         2096
14     01     1101            0000            0000010         2145
15     01     1111            0000            0000010         2162
16     10     1011            0100            0000011         2164
17     01     1111            0000            0000010         2543
18     10     0111            1000            0000011         2879
19     01     1111            0000            0000010         2970
20     10     1110            0001            0000011         3414
21     01     1111            0000            0000011         3465
22     10     1101            0010            0000011         3482
23     01     1111            0000            0000011         3814
24     10     1011            0100            0000011         4156
25     10     0011            1000            0000011         4236
26     01     0111            0000            0000011         4290
27     01     1111            0000            0000011         4307
28     10     1110            0001            0000011         4800
29     10     1100            0010            0000011         4868
30     10     1000            0100            0000011         4936
```

COM13

Port read timeout (ms) 4000

Tester Configuration

Test Duration 10 ms

Hold Time 1 ns

Time Scaler 3

□ Burst Mode

☑ INT_A Period 10 us

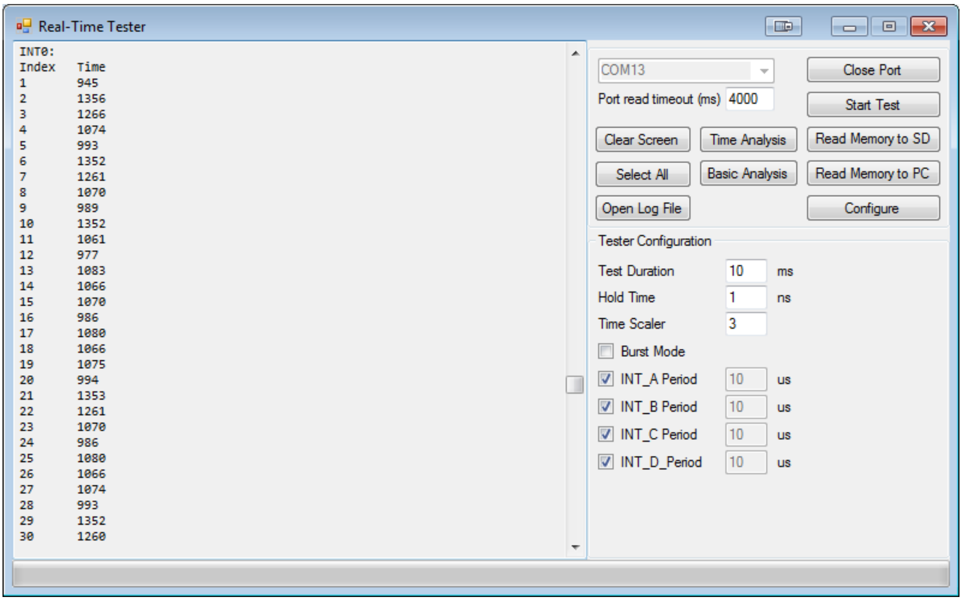
☑ INT_B Period 10 us

☑ INT_C Period 10 us

☑ INT_D_Period 10 us

Figure 7.6: Log data analysis result created by the tester software

presented earlier.



Real-Time Tester

```
INT0:
Index   Time
1       945
2       1356
3       1266
4       1074
5       993
6       1352
7       1261
8       1070
9       989
10      1352
11      1061
12      977
13      1083
14      1066
15      1070
16      986
17      1080
18      1066
19      1075
20      994
21      1353
22      1261
23      1070
24      986
25      1080
26      1066
27      1074
28      993
29      1352
30      1260
```

COM13

Port read timeout (ms) 4000

Tester Configuration

Test Duration 10 ms

Hold Time 1 ns

Time Scaler 3

□ Burst Mode

☑ INT_A Period 10 us

☑ INT_B Period 10 us

☑ INT_C Period 10 us

☑ INT_D_Period 10 us

Figure 7.7: Interrupt-to-acknowledgment data extracted by the tester software

Chapter 8

Discussion

8.1 Board manufacturing and assembly

Several steps has been taken to reduce PCB manufacturing cost. Assembly cost, however, has not been taken into account when calculating the final system cost. This is mainly because we are lucky at NTNU to have our own PCB assembly lab, the Electronics and Prototyping Lab (ElproLab). This lab has most of the necessary equipment such as a pick-and-place machine, a reflow oven and a device for soldering/desoldering surface mount components.

The pick-and-place machine could not be used for this project because it requires the components to be fed to it in reels. In other words, a large quantity of components is needed which was not feasible for our project. Therefore, assembly needed to bed done manually by the lab staff using the stencil the author ordered with the PCB.

The lab took two weeks to assemble the board and then the author spent another week debugging the PCB. Several Problems were found including reversed diodes and a few shorted pads under one of the SMD components, the micro-SD card socket to be specific. This required taking the boards back to the lab to do some further rework.

The main advantages of local PCB production are cost saving and being able to personally speak to the lab staff and clearly deliver feedback. However, if more boards are to be ordered, it is recommended that the PCB manufacturer is asked to do the assembly as well. The author checked with the used PCB manufacturing company (elecrow.com) and they do offer assembly services. Components can be shipped directly to their location as well. The author could not check the exact assembly cost though because it is job dependent.

To be fair, the assembly problems at ElproLab could have happened because the pick-and-place machine could not be used resulting in error caused by hand placement. Trying to clarify the concerns mentioned above to the lab staff before any future work is probably a good idea. Figure 8.1 shows the project PCB after assembly.

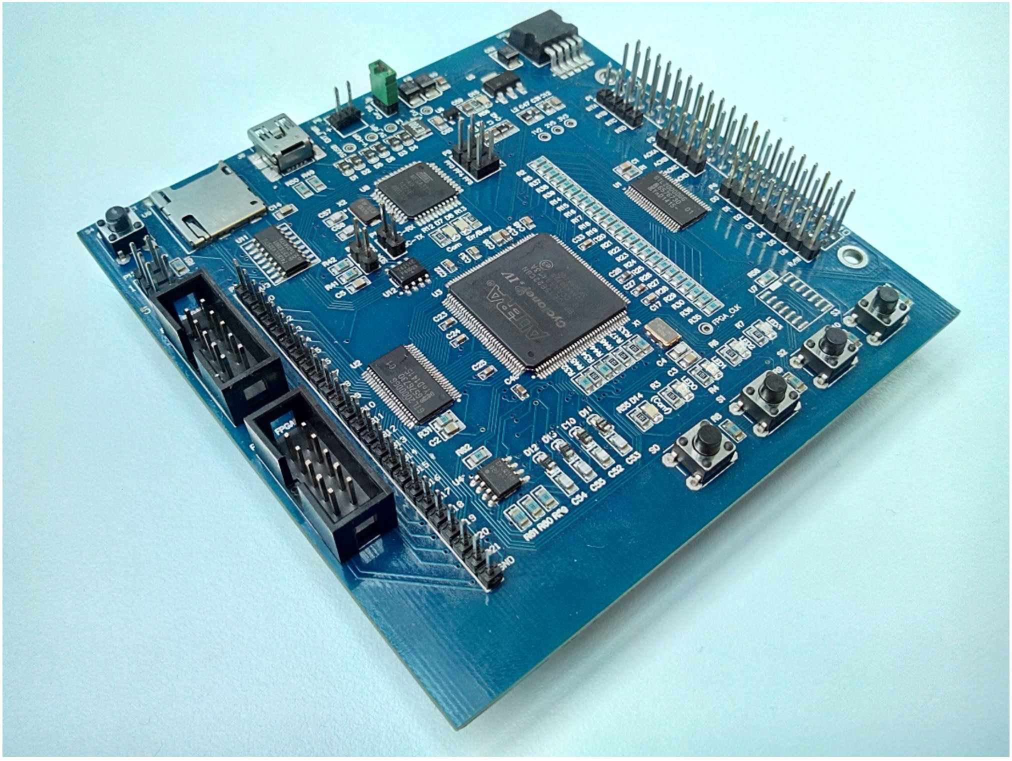


Figure 8.1: Assembled Tester PCB

8.2 Micro SD card connection

As mentioned in the previous section, some rework was needed to correct the micro SD card socket connection to the PCB. Even after this rework and removing the shorted connections, the microcontroller could not communicate with the card. Apparently, this time the socket was not properly soldered to the board. To work around this, the SPI bus test points were used to connect an external SD card adapter in order to get access to the micro SD card. Figure 8.2 shows the connection made. Although it does not look very appealing, it was the only way to connect the memory card to the system and prove its operation without wasting more time on reworks at the lab.

The good news is that this connection is not permanent. Instead of soldering, it was connected to the board through a female-to-male header so it is easily removable. In fact, after proving that the system is working correctly with the SD card, this connection was

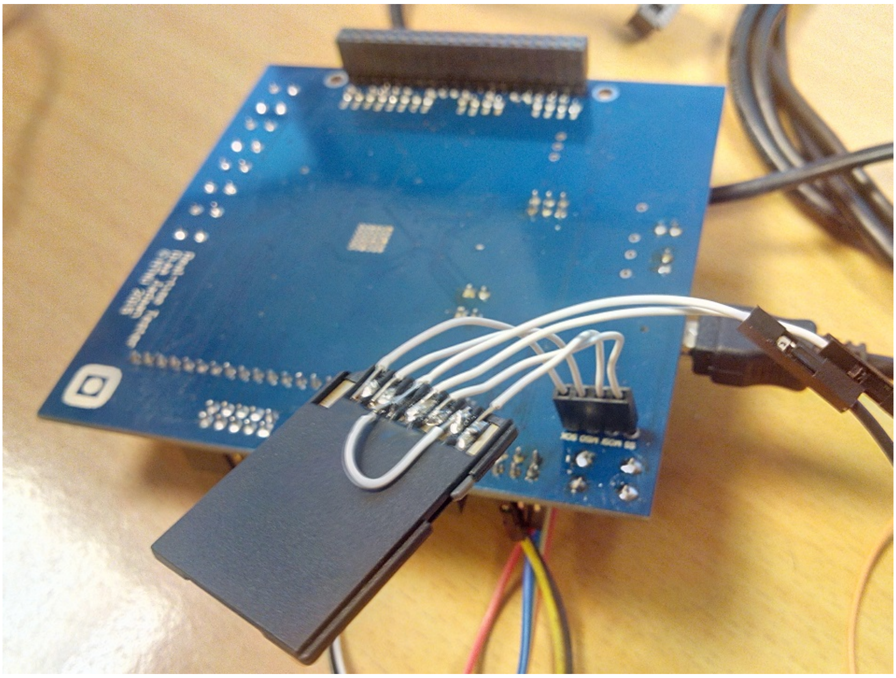


Figure 8.2: Connecting an SD card adapter to SPI test points

removed and data was transferred directly to the PC in the later phases of development. So the prototype board will look perfectly normal except when the user wants to write the log data directly to the micro-SD card instead of the PC.

8.3 Issues with PCB design

After soldering the PCB, examination revealed a problem with the design that prevented FPGA configuration over JTAG. The TDI and TDO pins from the JTAG connector were flipped relative to the ones on FPGA side causing the FPGA not to be detected by the programming cable. This problem can be easily fixed, but it needs some soldering effort.

As shown in the simplified connection figure below (figure 8.3) TDI and TDO are connected to the FPGA through 100Ω resistors.

These resistors were added for extra protection, but neither Altera nor the JTAG protocol requires them. Therefore, they can be harmlessly removed to correct the wire connections as shown in figure 8.4.

This patch was applied to the actual hardware as shown in figure 8.5, successfully resulting in operational JTAG connection.

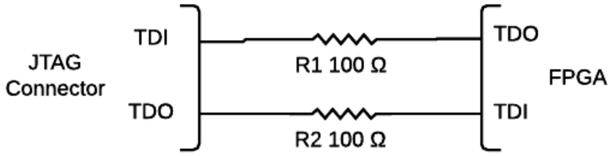


Figure 8.3: Connection problem causing JTAG configuration failure

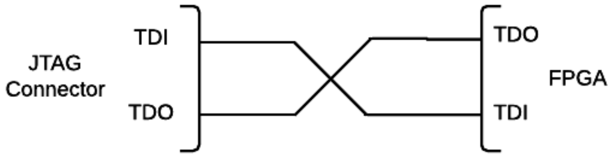


Figure 8.4: Correcting the connection problem

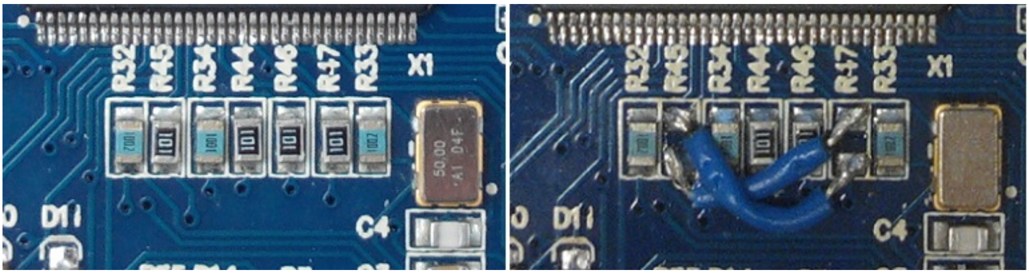


Figure 8.5: Tester PCB before (left) and after (right) the correction

It is worth mentioning that this problem has been corrected in the PCB design files, so the user does not need to worry about this if a new batch of PCBs is to be manufactured.

8.4 Recommendations for PCB design improvements

Although the final PCB is fully functional, there are some notes that might make dealing with the hardware more convenient if a future version is made.

- **Adding more power pins**
  Six GND pins and one 3V3 pin are available on the PCB. Anyway, having extra 5VUSB and 3V3 pins can be very useful.

- **Making test points have Dual Inline Package (DIP) pin pitch**
  Currently, the distance between neighboring test points is slightly larger than DIP

pin pitch, which is 2.54mm. This is useful if a DIP connector is intended to be used.

- **Test points labeling** Having the test points labeled on the silkscreen of both sides of the PCB rather than one side.
- **Labeling Raspberry Pi connections on both PCB sides**
- **Changing the Schmitt trigger chip**
  The 74HC7014 hex Schmitt trigger buffer is used in the design. This chip is very rare and probably discontinued. The 74HC7541 octal Schmitt trigger buffer chip is recommended as a replacement.
- **Removing extra components**
  As mentioned in section 3.5 there are several components used for redundancy and prototyping and they can be removed from future designs.
- **Changing board shape**
  If conforming to Raspberry Pi HAT specifications is intended, board dimensions should be changed to fit the specified shape.

8.5 RTS-tester vs. a logic/bus analyser [15]

Dedicated Systems Experts uses a PCI bus analyzer in their work [23]. This requires direct access to the system bus, something very few microcontrollers have. The tester developed in this project require GPIO pins, something most microcontrollers have. Another similar option is a logic analyzer.

Neither bus analyzer nor logic analyzer can produce stimuli and they require that the user uses a form of signal generator. The signal generator may be a microcontroller that is programmed to send interrupt and produce data packets. Beyond that, it will be necessary to verify the correct behavior of the microcontroller. Pin number may be another problem. The logic analyzer must use one or more pins to sample the outputs of the signal generator in addition to sample data from the system under test.

It is important to record accurately the timing of stimuli applied the system to carry out a good analysis. If all data lines from the microcontroller are sampled, one must also get around the problem of synchronization between the assumed sent data and the actually sent data. In addition to that, high-quality logic analysers are very expensive so the user must choose between an expensive analyzer and limited access to information from the system. The system implemented in this project provides test information and verified behavior along with low cost.

8.6 System cost

One of the project goals was to keep the system cost below EUR 50. Table 8.1 lists the prices of the used components in addition to the PCB cost. The following points should be taken into consideration when reading the table.

- The cost of development, research, debugging, etc. is not considered.
- Board assembly was done at NTNU with no cost.
- PCB is manufactured at elecrow.com
- PCB shipping cost is not considered. If the normal China mail is used, shipping cost per PCB can be negligible. This option would have been used if the project did not have a tight deadline.
- "Lowest Price" column shows the prices of some components when purchased from other retailers besides Mouser. Some parts, like the micro-SD card socket and USB connector, can be found for extremely cheap prices when 10 or 20 pieces are ordered.
- The Configuration Device listed is the Altera chip equivalent from Spansion.

Table 8.1: An overview of components cost

Component	**Component Name**	**Mouser Price (€)**	**Lowest Price (€)**
Passive components		5	4
50MHz Oscillator	VCC1-B3B-50M	0.51	0.51
8MHz Crystal	4SMX-8MHz	1.27	1.27
Micro-SD Card Socket		2.548	0.1
USB Mini-B Connector		0.45	0.05
3.3V Regulator	LD29300P2M33R	1.41	1.41
2.5V Regulator	LDK130M25R	0.48	0.48
1.2V Regulator	LD1117AS12TR	0.64	0.64
Voltage Level Shifter	GTL2000	2.73	2.73
Bus Multiplexer	74CBTLV3257D	0.57	0.57
Configuration Device	S25FL116K	0.43	0.43
EEPROM	CAT24C256WI-G	0.73	0.73
Microcontroller	Atmega32U4	6.19	3.44
FPGA	EP4CE15E22C8N	22.65	22.65
PCB		1.45	1.45
Sum		**47.06**	**40.46**

The table shows that the project target price has been successfully reached. Furthermore, the cost can be easily driven lower to 40€with just 3 components ordered from other retailers, that's even without considering ordering the other components in quantities over 10, which will drive the cost even lower.

8.7 Conforming to HAT specifications

As mentioned in section 3.5 the tester has been designed to conform to HAT specifications of Raspberry Pi. However, due to the need of adding extra components to the PCB for debugging purposes or for extra functionality, the RTS-tester does not fully adhere to these specifications. Extra components like the AS configuration interface, user GPIO pins and their protection circuit and non-essential user peripherals can be eliminated from the future designs. By doing so, it will be easier to modify the PCB layout and publish it as standard Raspberry Pi HAT.

8.8 Notes about using the tester

- The R/!W line should be kept at logic '1' whenever the tester is required to perform an action, e.g. issue interrupts or change data lines. Having this line on logic '0' will put the tester in "reading" mode, because read/!write here is from the SUT perspective.

- Problems with memory access have been noticed when the memory writing mode in the tester is set to Normal. It is highly recommended that Burst mode is activated at all times.

- Due to the bug mentioned in Section 2.3, all ACK lines must be held at logic '0' in order for the tester to issue any interrupts, even if only some of the interrupt lines are activated in configuration.

- The 74HC7014 Schmitt trigger chip is used in the design to provide basic debouncing for the push buttons and buffering for the Configuration Done LED. However, as seen in Figure 8.1 the place of the chip is empty because it was not available at Mouser when the components were order. If the chip is not found and the push buttons are needed, the user can solder the input and output pins of the chip connected to them and deal with the debounce in software. The Done LED connection should not be soldered. To know the exact pin locations please refer to Appendix A for the design schematics.

- The default tester configurations are:

Active Interrupts:	All
Interrupt Period:	10us
Burst Mode:	On
Time Scaler:	3
Hold Time	1ns
Test Duration:	10ms

- Another bug has been discovered in the tester IP. The actual test results are not pushed from the buffer to the memory until the next test. Therefore, two consecutive tests need to be performed in order to get the results for the first one. This issue has been taken care of in the GUI and the user does not need to worry about it.

The test parameters will be reset to these values after each power on. Otherwise, the tester will keep the programmed parameters even if the GUI was closed and opened again.

8.9 Future work

The produced system can be considered a "complete product". However, more improvements can be made to enhance its features and increase its usability.

Enhanced log data analysis

As explained in section 7.3.3, the PC software already has the capability of analysing the raw log data to produce the test results. Anyway, this operation can still be improved to offer more useful details to the user. For example, expanding it to cover other test modes besides interrupt-to-acknowledgment tests. An extra nice feature to have is the ability to recreate the signal timing diagram from the log data in a manner similar to figure 7.4. Adding additional features to the software is mostly straightforward because of its the modular structure and the features of C#.

Making the PCB compliant to Raspberry Pi HAT specification

The reader can refer to section 3.5 where the required modifications are explained in details.

Controlling the tester from a smartphone

This feature has been planned and the necessary connection was added to the PCB. A future developer can use the TX/RX pins on the PCB to connect a Bluetooth module similar to the one in Figure 8.6 and interface it to any smartphone. Since the implemented interface between with Nios II is simple ASCII based commands, this task should be easy to implement.

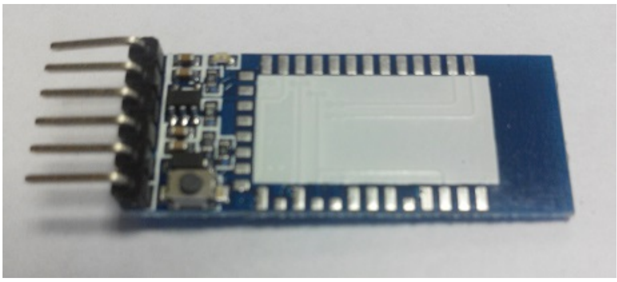


Figure 8.6: A proposed Bluetooth module with UART interface

It is important that the UART unit in the microcontroller is disabled before connecting the Bluetooth module.

Open-source materials

The most of the materials of this project have been published as open-source on the author's Github web page at the address: `https://github.com/ditek`

Chapter 9

Conclusion

This project was set out to create a platform independent, low-cost solution for testing real-time systems. The basis of the project is a real-time testing IP created within a previous master thesis at NTNU. The report shows how the functionality of this IP was debugged, corrected and verified before a dedicated hardware of the tester was created.

The resulting platform has dimensions of 10*10 cm^2 that partially follows the standard form factor of Raspberry Pi 2 model B as a proposed SUT. Despite that, it is possible to connect any external device to the tester. Additional IOs and peripherals were included to allow the user to add custom functionality. In addition, extra care has been taken of protection against power supply problems and user mistakes.

The test system has a resolution of 20ns and can perform a new logging every 80ns, which corresponds to a rate of 12.5 MHz. It can maintain an interruption rate of 2.5 MHz with four interrupt lines and 8.33 MHz with one line. The tester may log data with unique timestamps over a period of $n * 2^{43} * 20ns$ with a resolution of $n * 20ns$ where n is clock scaling. That is, a test time of 48.9 hours with a resolution of 20ns. Clock scaling will be sufficient for most test requirements for real-time systems.

An easy to use graphical user interface has also been created to allow the user to customize test parameters, perform data analysis and save test results to the PC or the the on-board micro SD card.

This work can be considered as a complete system, which the user can connect to an external SUT and then use the accompanying PC software to analyze the resulting test data log. A good way to improve the user experience is to enhance the analysis operation by adding more features to it. Other potential improvements include modifying the design of the device to become a standard add-on for a popular embedded system (e.g. a HAT for the Raspberry Pi) and enhancing device usability by developing a Bluetooth-based smartphone interface for example. The basis of all these improvements has been implemented with various degrees of completeness as explained in the Discussion chapter.

Finally, this project has successfully met its goals and produced a fully functional real-time testing platform that costs less than EUR 50. The various features of this device ensure that it can provide labs and universities with a reliable solution that is both high-quality and low-cost.

Bibliography

[1] Altera. *Avalon Interface Specifications*.

[2] Altera. Configuration and remote system upgrades in cyclone iv devices, 5 2013.

[3] Altera. Altera device package information, 8 2014.

[4] Altera. Cyclone iv device datasheet, 10 2014.

[5] Altera. Cyclone iv device overview, 10 2014.

[6] Altera. Cyclone v device overview, 10 2014.

[7] Altera. Quartus ii help pages. http://quartushelp.altera.com/14.0/master.htm#mergedProjects/quartus/gl_quartus_welcome.htm, 2014.

[8] Altera. Pdn tool. http://wl.altera.com/technology/signal/power-distribution-network/sgl-pdn.html, 2015.

[9] Atmel. Atmega32u4 datasheet, 9 2014.

[10] Dean Camera. Lufa usb framework. www.fourwalledcubicle.com/LUFA.php, 26.6.2015.

[11] Analog Devices. Decoupling techniques, 3 2009.

[12] Raspberry Pi Foundation. Add-on boards and hats. https://github.com/raspberrypi/hats, 26.6.2015.

[13] Raspberry Pi Foundation. Hat id eeprom format specification. https://github.com/raspberrypi/hats/blob/master/eeprom-format.md, 26.6.2015.

[14] Raspberry Pi Foundation. Introducing raspberry pi hats. https://www.raspberrypi.org/introducing-raspberry-pi-hats, 26.6.2015.

[15] Kyrre E. A. Gonsholt. Sanntidsystemtester. Master's thesis, NTNU, 2015.

[16] Mouser. Altera cyclone v fpgas. http://no.mouser.com/new/altera/altera-cyclonev/, 18.06.2015.

[17] Philips. Bi-directional level shifter for i2c-bus. http://www.adafruit.com/datasheets/an97055.pdf, 2004.

[18] Rocketboards. Arrow sockit evaluation board. http://www.rocketboards.org/foswiki/Documentation/ArrowSoCKitEvaluationBoard, 18.06.2015.

[19] Rocketboards. Gsrd v14.0 - user manual - arrow sockit edition. http://www.rocketboards.org/foswiki/Documentation/GSRDGhrd, 18.06.2015.

[20] Cypress Semiconductor. Protecting your low voltage electronic devices from electrical overstress. http://www.embedded.com/design/prototyping-and-development/4423709/Protecting-your-low-voltage-electronic-devices-from+electrical-overstress, 2013.

[21] Siemens. Ptc thermistors guide. http://www.hems.de/uploads/media/PTC_siemens.pdf, 2015.

[22] William Stallings. *Operating Systems: Internals and Designs Principles*. Pearson Education, 7 edition, 2012.

[23] Dedicated Systems. Real-time testing documents. http://download.dedicated-systems.info/, 26.6.2015.

[24] terasic.com.tw. Sockit producer webpage. http://www.terasic.com.tw/cgi-bin/page/archive.pl?CategoryNo=167&No=816, 18.06.2015.

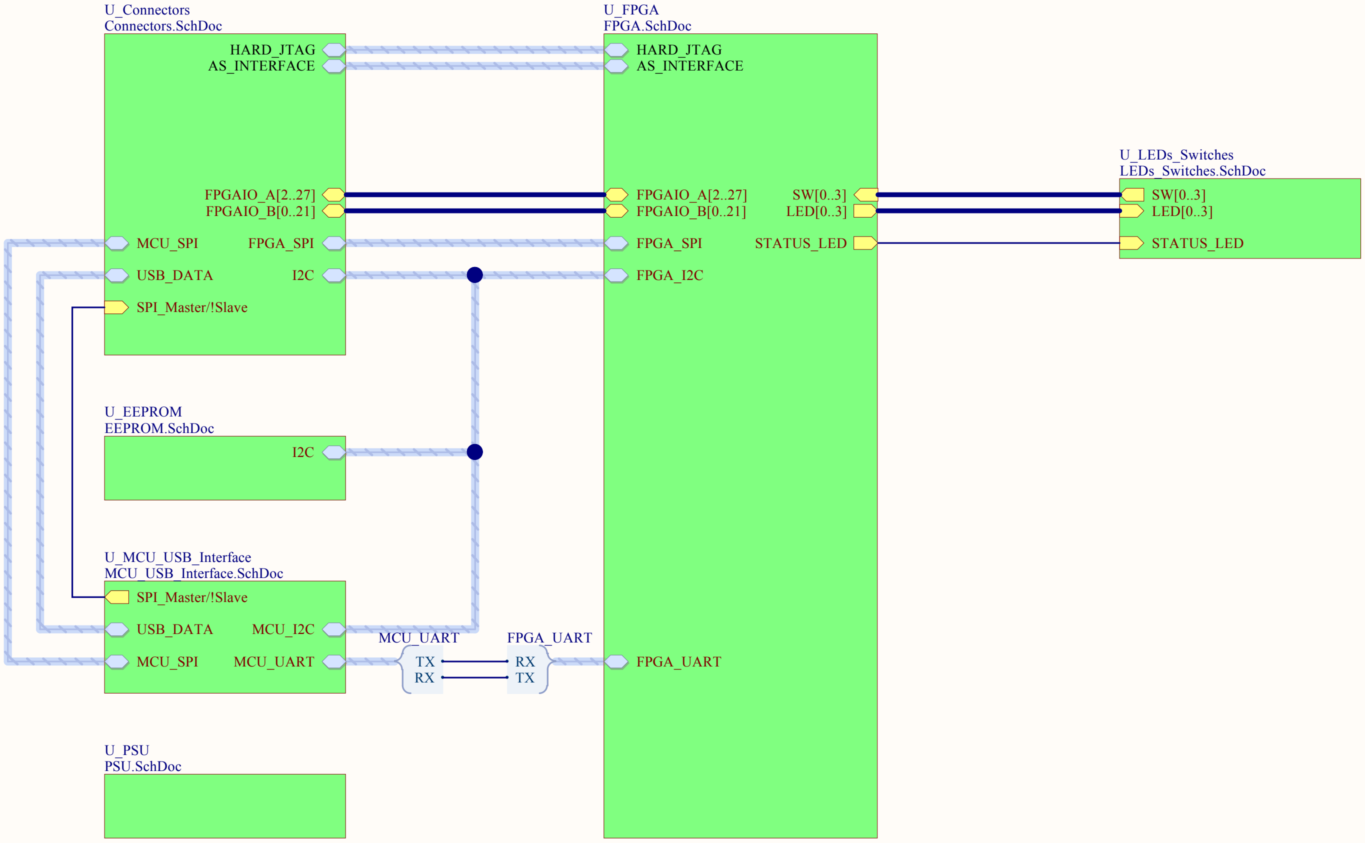
[25] Kolja Waschk. Usb jtag project. http://ixo-jtag.sourceforge.net/, 26.6.2015.

[26] Mosaic Documentation Web. Raspberri pi gpio electrical specifications. http://www.mosaic-industries.com/embedded-systems/microcontroller, 26.6.2015.

[27] Wikipedia. Raspberry pi. https://en.wikipedia.org/wiki/Raspberry_Pi, 26.6.2015.

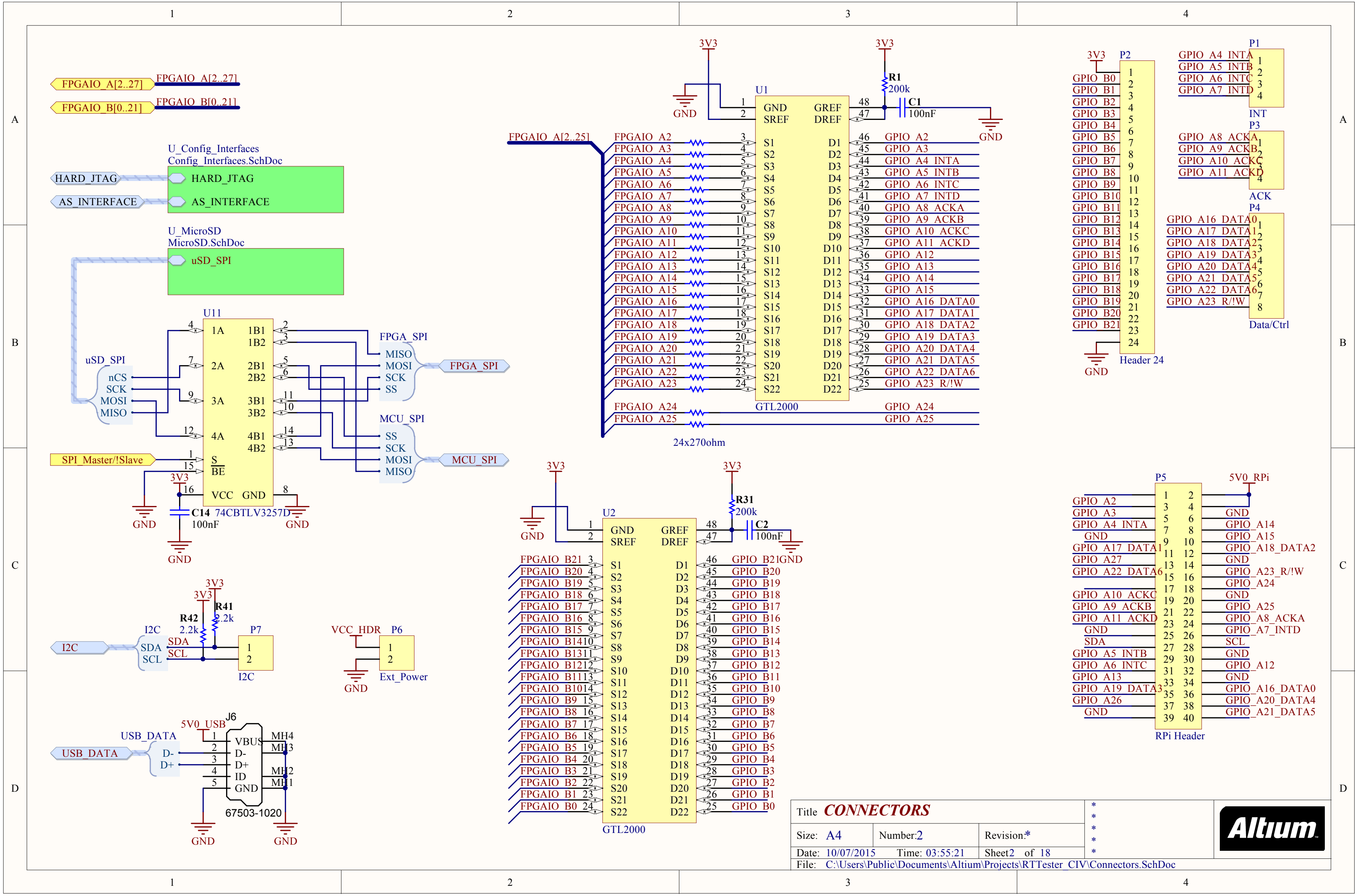
Appendix A

PCB Design Schematics



Title *TOP*		
Size: A4	Number:1	Revision:*
Date: 10/07/2015	Time: 03:55:21	Sheet1 of 18
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\Top.SchDoc		

*
*
*
*
*



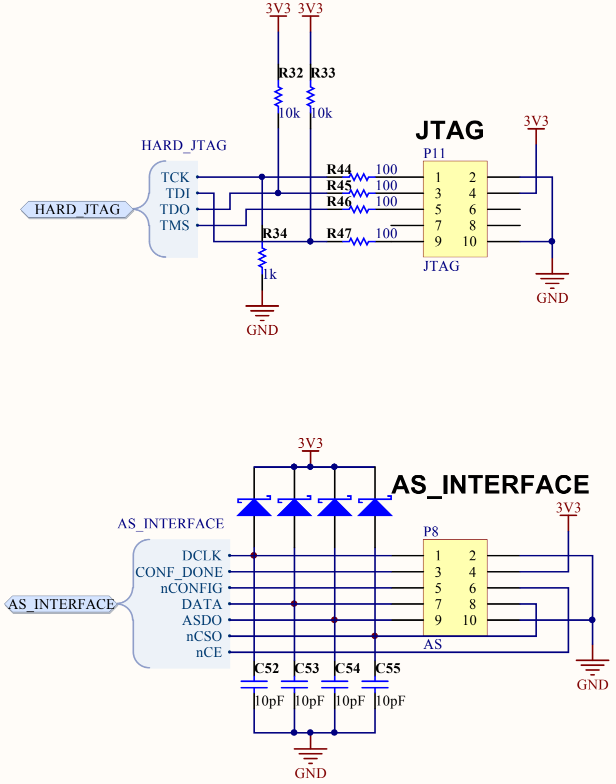
Title CONNECTORS

Size: A4 Number:2 Revision:*

Date: 10/07/2015 Time: 03:55:21 Sheet2 of 18

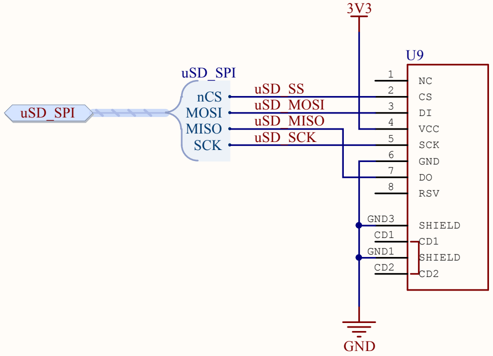
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\Connectors.SchDoc

Altium

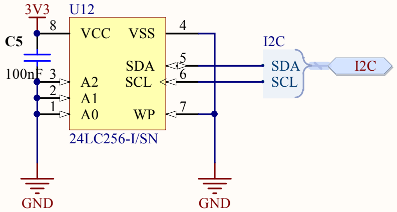


Title *CONFIG. INTERFACES*		
Size: A4	Number:3	Revision:*
Date: 10/07/2015	Time: 03:55:21	Sheet3 of 18
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\Config_Interfaces.SchDoc		

*
*
*
*
*



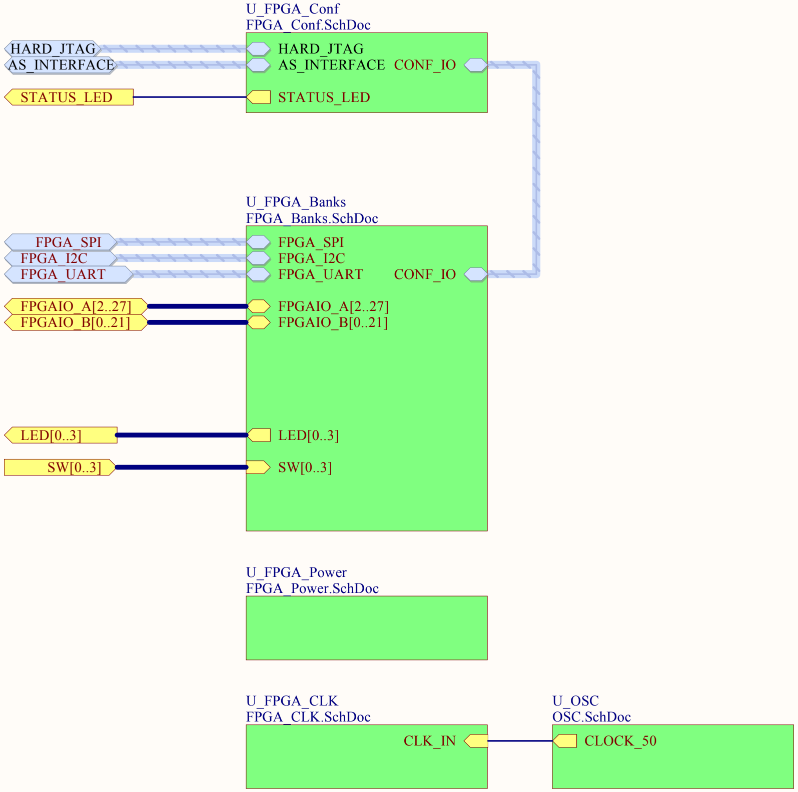
Title *MICRO-SD*			*
Size: A4	Number:4	Revision:*	*
Date: 10/07/2015	Time: 03:55:21	Sheet4 of 18	*
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\MicroSD.SchDoc			*
			*



I2C Address:
0xA0 = 1010000

Title *EEPROM*		
Size: A4	Number:5	Revision:*
Date: 10/07/2015 Time: 03:55:21		Sheet5 of 18
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\EEPROM.SchDoc		

*
*
*
*
*



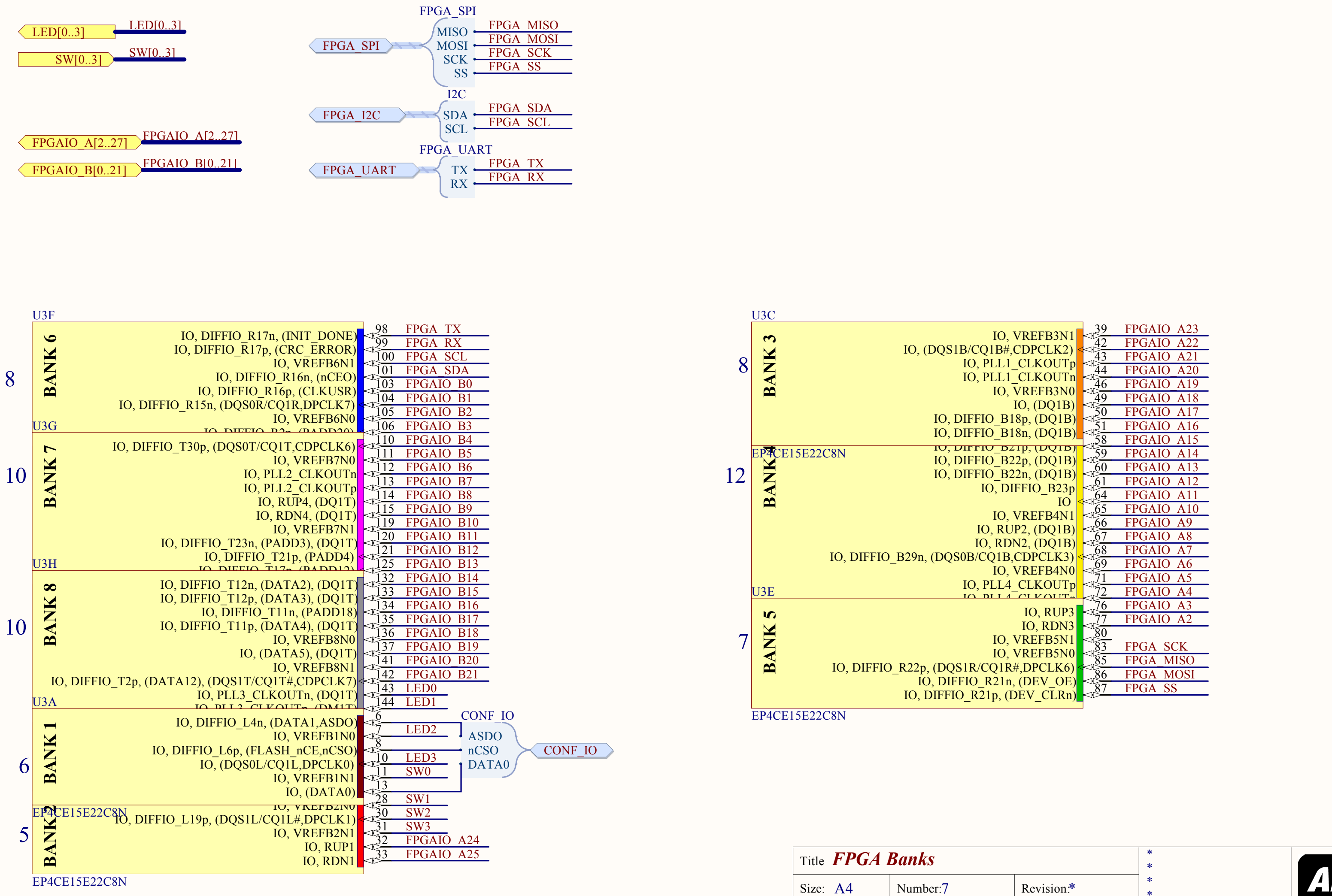
Title **FPGA TOP**

Size: A4 | Number:6 | Revision:*

Date: 10/07/2015 | Time: 03:55:21 | Sheet6 of 18

File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\FPGA.SchDoc

*
*
*
*
*



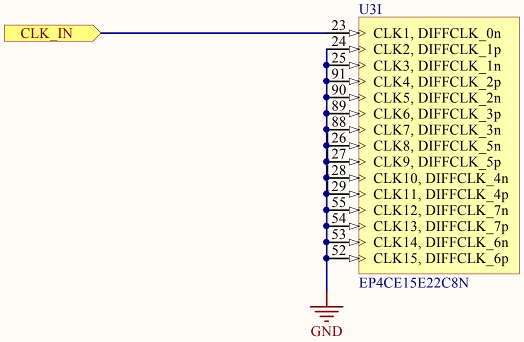
LED[0..3]
LED[0..3]
SW[0..3]
SW[0..3]
FPGAIO_A[2..27]
FPGAIO_A[2..27]
FPGAIO_B[0..21]
FPGAIO_B[0..21]
FPGA_SPI
FPGA_SPI
MISO
MOSI
SCK
SS
FPGA_MISO
FPGA_MOSI
FPGA_SCK
FPGA_SS
I2C
FPGA_I2C
SDA
SCL
FPGA_SDA
FPGA_SCL
FPGA_UART
FPGA_UART
TX
RX
FPGA_TX
FPGA_RX
U3F
BANK 6
8
IO, DIFFIO_R17n, (INIT_DONE) 98 FPGA_TX
IO, DIFFIO_R17p, (CRC_ERROR) 99 FPGA_RX
IO, VREFB6N1 100 FPGA_SCL
IO, DIFFIO_R16n, (nCEO) 101 FPGA_SDA
IO, DIFFIO_R16p, (CLKUSR) 103 FPGAIO_B0
IO, DIFFIO_R15n, (DQS0R/CQ1R,DPCLK7) 104 FPGAIO_B1
IO, VREFB6N0 105 FPGAIO_B2
U3G
106 FPGAIO_B3
BANK 7
10
IO, DIFFIO_T30p, (DQS0T/CQ1T,CDPCLK6) 110 FPGAIO_B4
IO, VREFB7N0 111 FPGAIO_B5
IO, PLL2_CLKOUTn 112 FPGAIO_B6
IO, PLL2_CLKOUTp 113 FPGAIO_B7
IO, RUP4, (DQ1T) 114 FPGAIO_B8
IO, RDN4, (DQ1T) 115 FPGAIO_B9
IO, VREFB7N1 119 FPGAIO_B10
IO, DIFFIO_T23n, (PADD3), (DQ1T) 120 FPGAIO_B11
IO, DIFFIO_T21p, (PADD4) 121 FPGAIO_B12
U3H
125 FPGAIO_B13
BANK 8
10
IO, DIFFIO_T12n, (DATA2), (DQ1T) 132 FPGAIO_B14
IO, DIFFIO_T12p, (DATA3), (DQ1T) 133 FPGAIO_B15
IO, DIFFIO_T11n, (PADD18) 134 FPGAIO_B16
IO, DIFFIO_T11p, (DATA4), (DQ1T) 135 FPGAIO_B17
IO, VREFB8N0 136 FPGAIO_B18
IO, (DATA5), (DQ1T) 137 FPGAIO_B19
IO, VREFB8N1 141 FPGAIO_B20
IO, DIFFIO_T2p, (DATA12), (DQS1T/CQ1T#,CDPCLK7) 142 FPGAIO_B21
IO, PLL3_CLKOUTn, (DQ1T) 143 LED0
U3A
144 LED1
BANK 1
6
IO, DIFFIO_L4n, (DATA1,ASDO) 6
IO, VREFB1N0 7 LED2
IO, DIFFIO_L6p, (FLASH_nCE,nCSO) 8
IO, (DQS0L/CQ1L,DPCLK0) 10 LED3
IO, VREFB1N1 11 SW0
IO, (DATA0) 13
CONF_IO
ASDO
nCSO
DATA0
CONF_IO
EP4CE15E22C8N
BANK 2
5
IO, VREFB2N0 28 SW1
IO, DIFFIO_L19p, (DQS1L/CQ1L#,DPCLK1) 30 SW2
IO, VREFB2N1 31 SW3
IO, RUP1 32 FPGAIO_A24
IO, RDN1 33 FPGAIO_A25
EP4CE15E22C8N
U3C
BANK 3
8
IO, VREFB3N1 39 FPGAIO_A23
IO, (DQS1B/CQ1B#,CDPCLK2) 42 FPGAIO_A22
IO, PLL1_CLKOUTp 43 FPGAIO_A21
IO, PLL1_CLKOUTn 44 FPGAIO_A20
IO, VREFB3N0 46 FPGAIO_A19
IO, (DQ1B) 49 FPGAIO_A18
IO, DIFFIO_B18p, (DQ1B) 50 FPGAIO_A17
IO, DIFFIO_B18n, (DQ1B) 51 FPGAIO_A16
EP4CE15E22C8N
BANK 4
12
IO, DIFFIO_B21p, (DQ1B) 58 FPGAIO_A15
IO, DIFFIO_B22p, (DQ1B) 59 FPGAIO_A14
IO, DIFFIO_B22n, (DQ1B) 60 FPGAIO_A13
IO, DIFFIO_B23p 61 FPGAIO_A12
IO 64 FPGAIO_A11
IO, VREFB4N1 65 FPGAIO_A10
IO, RUP2, (DQ1B) 66 FPGAIO_A9
IO, RDN2, (DQ1B) 67 FPGAIO_A8
IO, DIFFIO_B29n, (DQS0B/CQ1B,CDPCLK3) 68 FPGAIO_A7
IO, VREFB4N0 69 FPGAIO_A6
IO, PLL4_CLKOUTp 71 FPGAIO_A5
U3E
72 FPGAIO_A4
BANK 5
7
IO, RUP3 76 FPGAIO_A3
IO, RDN3 77 FPGAIO_A2
IO, VREFB5N1 80
IO, VREFB5N0 83 FPGA_SCK
IO, DIFFIO_R22p, (DQS1R/CQ1R#,DPCLK6) 85 FPGA_MISO
IO, DIFFIO_R21n, (DEV_OE) 86 FPGA_MOSI
IO, DIFFIO_R21p, (DEV_CLRn) 87 FPGA_SS
EP4CE15E22C8N

Title *FPGA Banks*

Size: A4	Number:7	Revision:*
Date: 10/07/2015	Time: 03:55:21	Sheet7 of 18

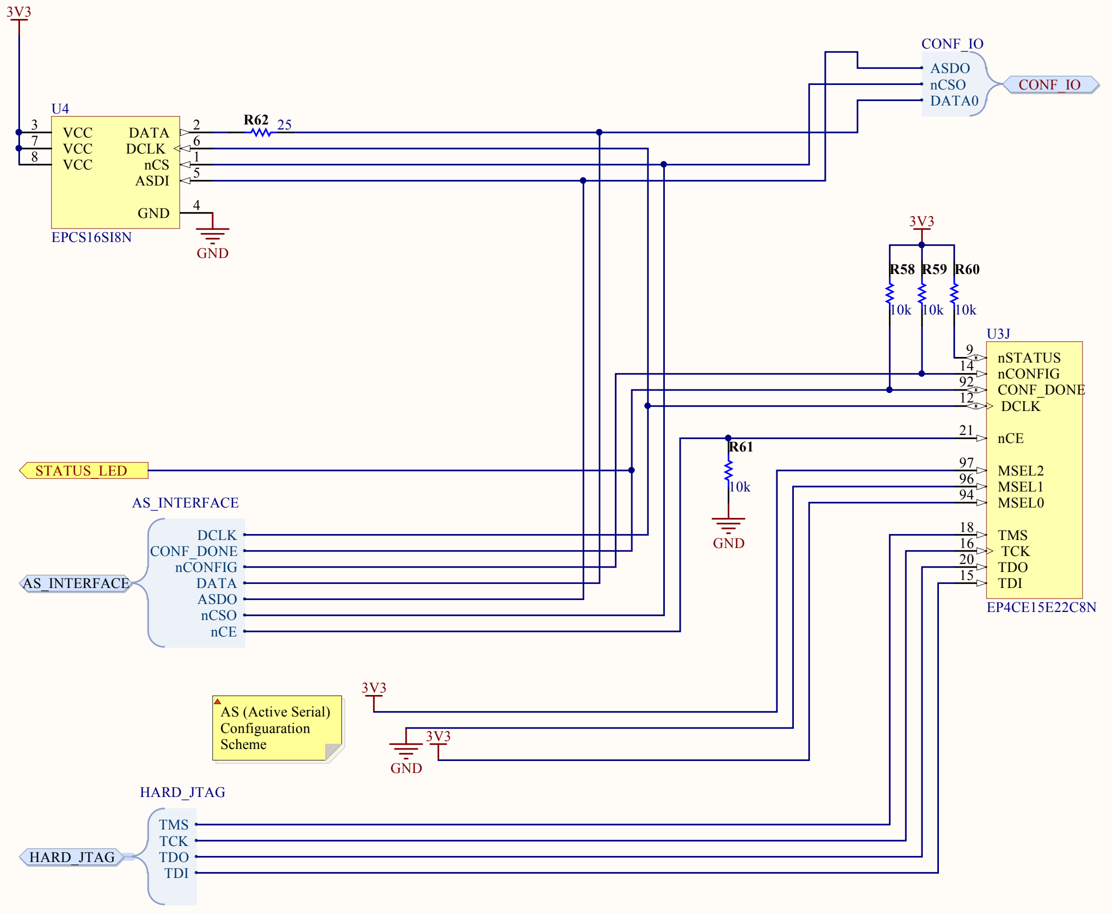
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\FPGA_Banks.SchDoc

Altium



Title *FPGA CLOCK*			* *
Size: A4	Number:8	Revision:*	* *
Date: 10/07/2015	Time: 03:55:21	Sheet8 of 18	*

File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\FPGA_CLK.SchDoc



Title *FPGA CONFIGURATION*

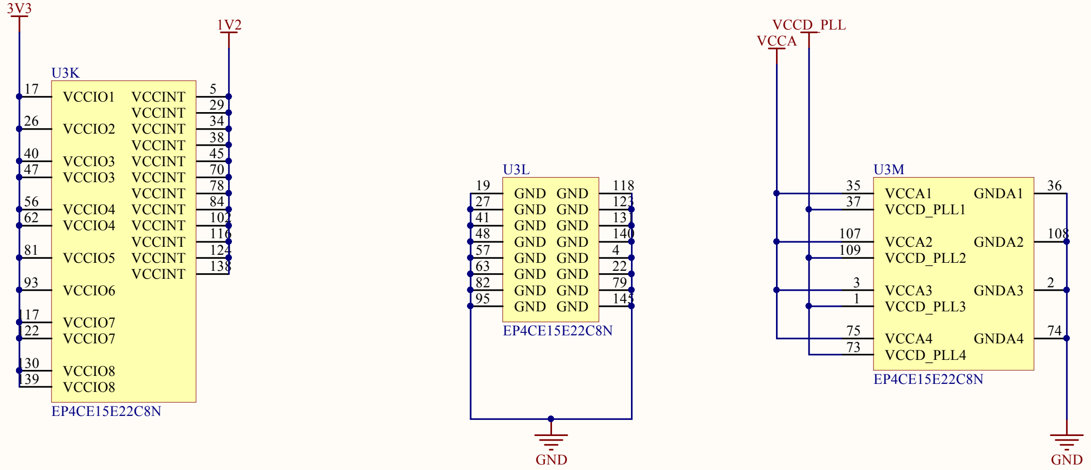
Size: A4 Number:9 Revision:*

Date: 10/07/2015 Time: 03:55:21 Sheet9 of 18

File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\FPGA_Conf.SchDoc

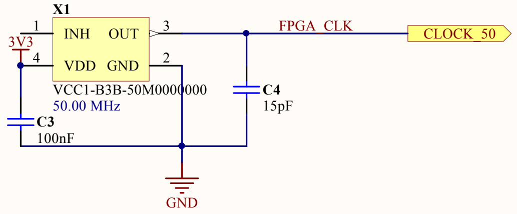
*
*
*
*
*

Altium



Title ***FPGA POWER***		
Size: A4	Number:10	Revision:*
Date: 10/07/2015	Time: 03:55:21	Sheet10 of 18
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\FPGA_Power.SchDoc		

*
*
*
*
*



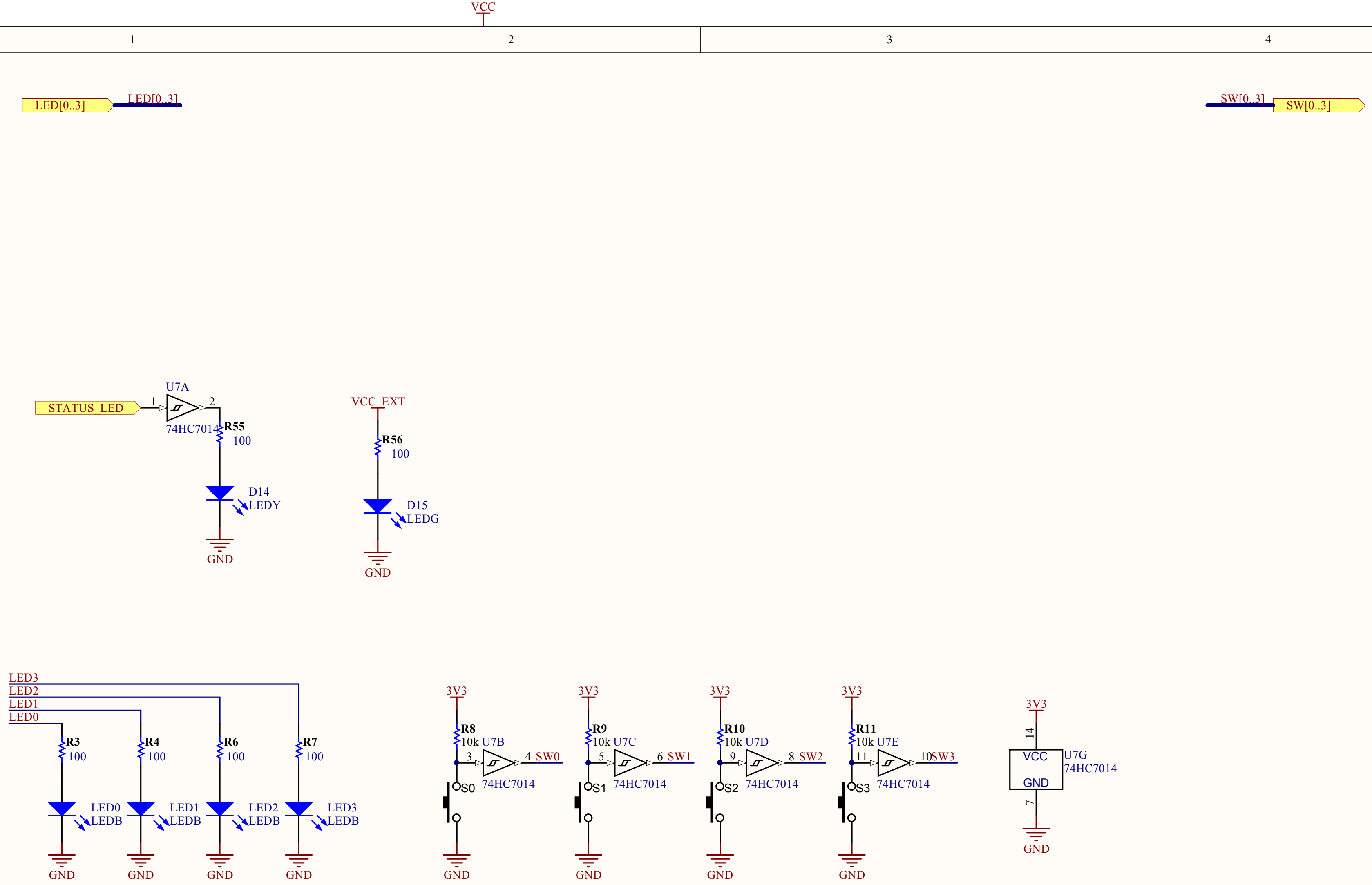
Title **OSCILATOR**

Size: A4 | Number:11 | Revision:*

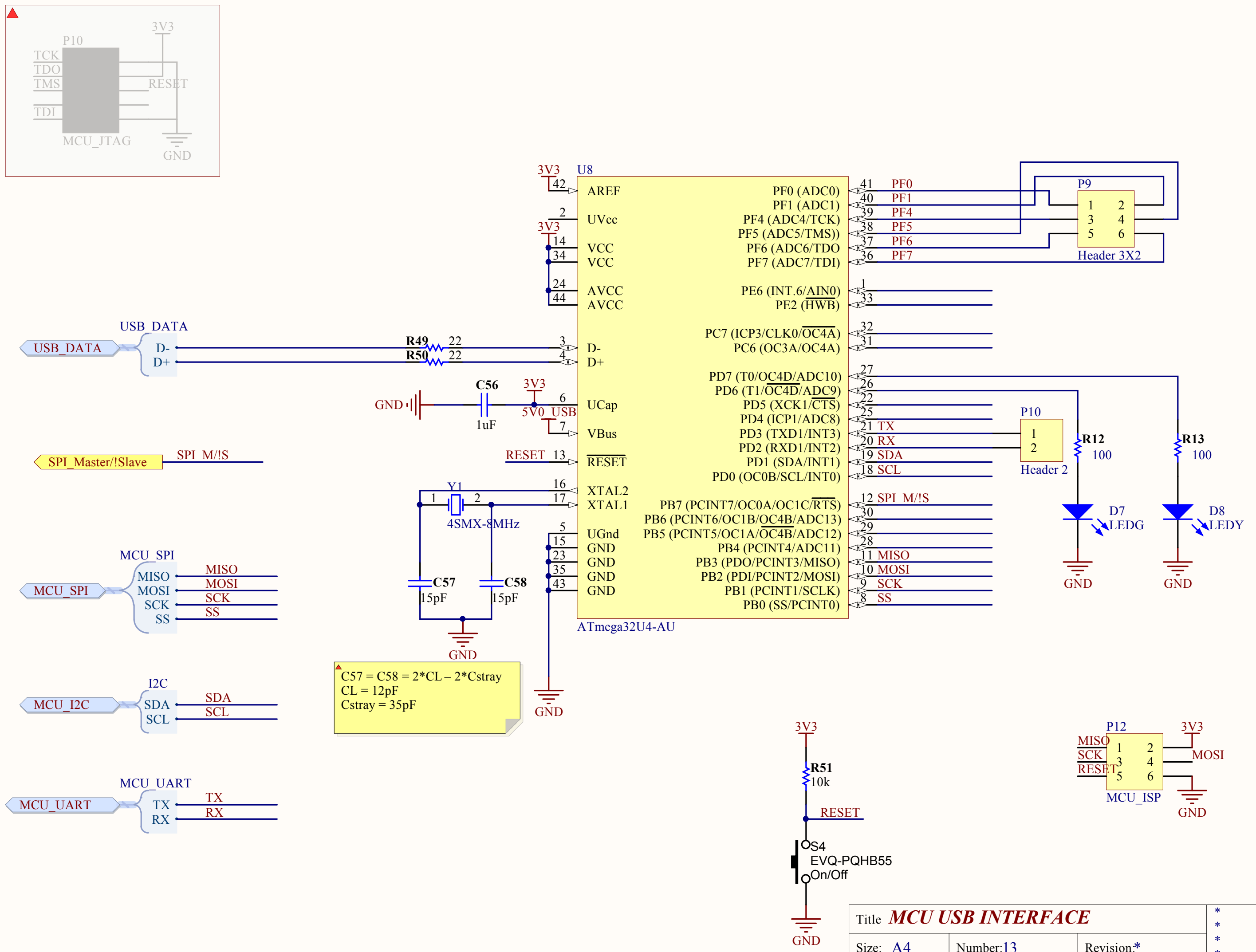
Date: 10/07/2015 Time: 03:55:21 Sheet11 of 18

File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\OSC.SchDoc

*
*
*
*
*



Title *LEDs AND SWITCHES*			*
Size: A4	Number:12	Revision:*	*
Date: 10/07/2015	Time: 03:55:21	Sheet12 of 18	*
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\LEDs_Switches.SchDoc			*

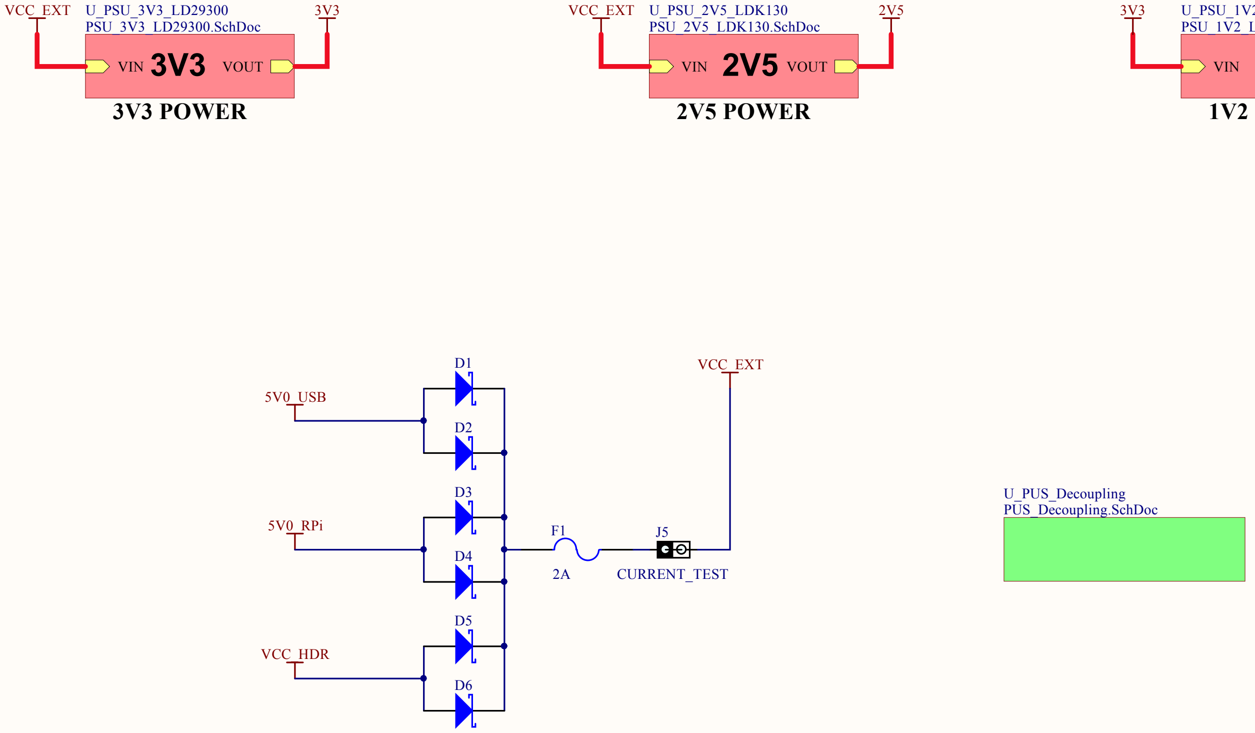


C57 = C58 = 2*CL – 2*Cstray
CL = 12pF
Cstray = 35pF

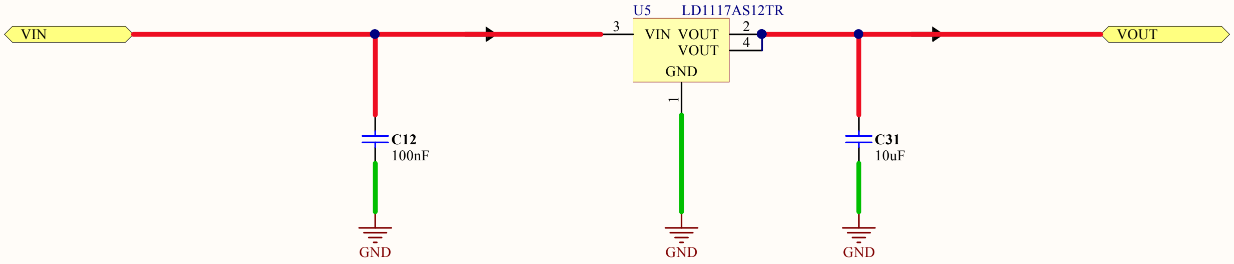
Title *MCU USB INTERFACE*

Size: A4	Number:13	Revision:*
Date: 10/07/2015	Time: 03:55:21	Sheet13 of 18

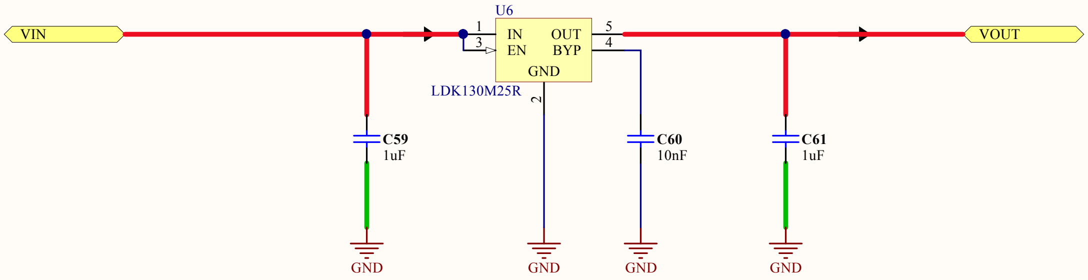
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\MCU_USB_Interface.SchDoc



Title *PSU TOP*			*
Size: A4	Number:14	Revision:*	*
Date: 10/07/2015	Time: 03:55:21	Sheet14 of 18	*
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\PSU.SchDoc			*
			*



Title **PSU 1V2**			*
Size: A4	Number:15	Revision:*	*
Date: 10/07/2015	Time: 03:55:21	Sheet15 of 18	*
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\PSU_1V2_LD1117A.SchDoc			*
			*



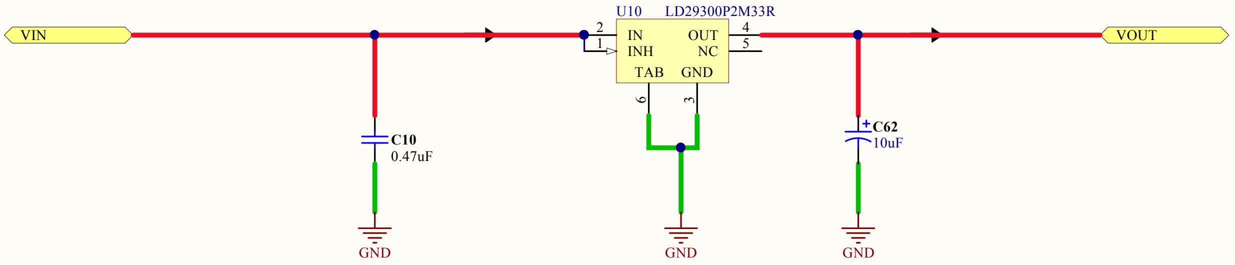
Title *PSU 2V5*

Size: A4 Number:16 Revision:*

Date: 10/07/2015 Time: 03:55:21 Sheet16 of 18

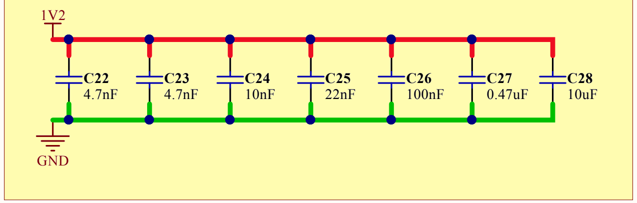
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\PSU_2V5_LDK130.SchDoc

*
*
*
*
*

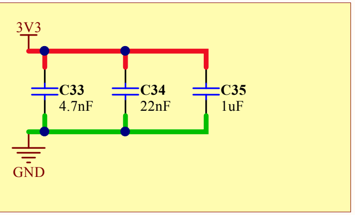


Title *PSU 3V3*			* * * * *
Size: A4	Number:17	Revision:*	
Date: 10/07/2015	Time: 03:55:21	Sheet17 of 18	
File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\PSU_3V3_LD29300.SchDoc			

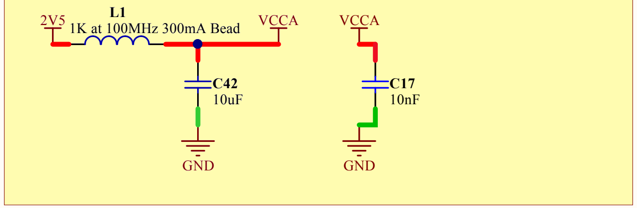
1.2V Decoupling



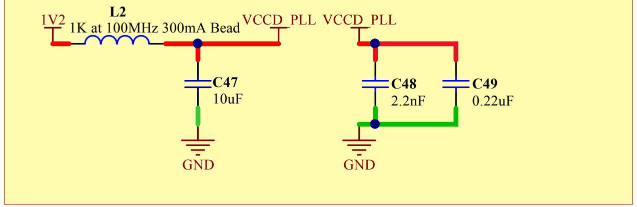
3.3V Decoupling

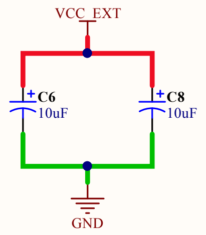


PLL VCCA Decoupling



PLL VCCD Decoupling





Title *PSU_DECOUPLING*

Size: A4 Number:18 Revision:*

Date: 10/07/2015 Time: 03:55:21 Sheet18 of 18

File: C:\Users\Public\Documents\Altium\Projects\RTTester_CIV\PUS_Decoupling.SchDoc

*
*
*
*
*

Altium