Wishbone Interconnection Architecture compatible Framework V1.1 (Monday 2nd June, 2014)
This document describes a setup for a SoC implementation that will be the basis for the assignments for the ET4351 ‘VLSI systems on chip’ course. We composed a coherent subset of SystemC descriptions, C-program code and scripts, etc. that can be used for the assignments. These can be downloaded from http://ens.ewi.tudelft.nl/Education/courses/et4351/et4351_sc.zip. Due to the diversity of the assignments, the information in this document may be a bit too elaborate for a particular assignment. However, it will be very helpful for understanding the complete system and will show to be of importance for debugging purposes.

This User Guide is organized as follows. First, an overview is given of the SoC which is built around a processor core derived from the Xilinx MicroBlaze, a Wishbone interconnection architecture and one or more Wishbone compatible slaves. The whole system -and the SystemC software that describes the system- is described in detail.

Use this document as guidance and particularly look closely at the parts that are specific for your assignment.
Contents

Preface i

1 Introduction 1
  1.1 RISC Processors .................................. 1
  1.2 From MicroBlaze to MBL1c .......................... 1
  1.3 Memory-mapped I/O ................................. 3
  1.4 Wishbone Interconnection Architecture .......... 3
    1.4.1 MBL-Wishbone bridge ......................... 3

2 Design Possibilities 5

3 MBL1C ISS Setup for ET4351 7
  3.1 MBL1C ISS for ET4351 ............................. 7
    3.1.1 Memory architecture ......................... 7
    3.1.2 Data Alignment ................................ 7
  3.2 The Wishbone Interconnection Architecture ..... 9
    3.2.1 Handshaking protocol ........................ 10
    3.2.2 The Wishbone specification .................. 10
    3.2.3 The Wishbone signals at the slaves side .... 11

4 Programming the MBL1c 13

5 The ET4351_SC Package 15

A Software 17
  A.1 Installation and software requirements .......... 17
    A.1.1 Creating contents for the instruction memory . 17
  A.2 Files .......................................... 17
    A.2.1 Installation .................................. 18
  A.3 Building and running the example WBSlave .... 19
  A.4 Course Lab facilities ............................. 19
    A.4.1 Remote Access (Rdesktop) ..................... 19
    A.4.2 File transfer .................................. 20
    A.4.3 Basic steps project assignment ............... 20
Bibliography
Chapter 1

Introduction

One of the very popular 32-bit microprocessors nowadays is the MicroBlaze: a 32-bit RISC processor, for use in FPGA designs.

1.1 RISC Processors

RISC, or Reduced Instruction Set Computer, is a term that is conventionally used to describe a type of microprocessor architecture that employs a small but highly-optimized set of instructions, rather than the large set of more specialized instructions often found in other types of architectures. This other type of processor is traditionally referred to as CISC, or Complex Instruction Set Computer. Early RISC processors emerged in the late 1970s and early 1980s, and the basic design architecture of all RISC processors has generally followed the characteristics that came from those early research projects and which can be summarized as follows: One instruction per clock cycle execution time: RISC processors have a CPI (clock per instruction) of one cycle, due to the optimization of each instruction on the CPU. To allow for high clock frequencies, pipelining is used. This technique allows each instruction to be processed in a set number of stages that are processed in parallel. This in turn allows for the simultaneous execution of a number of different instructions, each instruction being at a different stage in the pipeline. Load/Store machine with a large number of internal registers: the RISC design philosophy typically uses a relatively large number (commonly 32) of internal registers. Most instructions operate on these registers, with access to memory made using a very limited set of Load and Store instructions. This limits the need for continuous access to usually slower memory for loading and storing intermediate data. Separate Data Memory and Instruction Memory access paths: different stages of the pipeline perform simultaneous accesses to memory.

1.2 From MicroBlaze to MBL1c

The MicroBlaze is a 32-bit RISC machine that follows the classic RISC architecture described above. It is a load/store machine with 32 general purpose registers. All instructions are 32-bits wide and most of them execute in a single clock cycle. However, the processor is designed specifically for Xilinx FPGAs and therefore highly optimized for their FPGA circuits. The MicroBlaze
is distributed with the Xilinx Embedded Development Kit (EDK) as a parameterizable netlist, and although the VHDL source code can be obtained from Xilinx at additional costs, it is not to be distributed freely. Several Microblaze inspired processors are available as open source projects, like e.g. the aeMB and the Openfire, but neither of them did exactly what we were looking for. We developed a version with only the features that we really need.

Like the original MicroBlaze, this MB-Lite [MB-Lite] uses a pipelined architecture. Most of the instructions take only 1 clock cycle, except for the branch- and return-from-subroutine instructions. These have to flush the pipeline to start fresh from a new instruction address. Also, trying to process data that isn’t available, since not having been read yet by a previous instruction, causes the processor to stall for one or more cycles.

For our purpose here, i.e. the development and investigation of (slave) IP hardware that has to communicate with the processor via a Wishbone interface, this behavior can be misleading and distracting. The fact that primarily all Wishbone I/O signals should be generated and handled adequately, while being able to run unchanged MB-Lite software, did us decide to develop the single-instruction-cycle MBL1c simulation model presented here and which should be used for our assignments. This MBL1c is available both as a VHDL entity and as a SystemC module. All instructions are executed in a single cycle. To remain compatible with the MB-Lite and the MicroBlaze where they are intended to (partly) fill the gaps after taken branches, the so called “delayed instruction” also need to be executed.

Figure 1.1: Block scheme of the MBL1c ISS environment
1.3 Memory-mapped I/O

For connecting to the outside world, memory mapped I/O can be used. Since the MicroBlaze is a 32-bit processor, reserving ranges of memory address space for I/O is generally no real problem, as the memory address space is usually much larger than the required space for all memory and I/O devices together. There are two major advantages of using memory-mapped I/O instead of dedicated ports for I/O. One of them is that the CPU requires less internal logic and thus will be cheaper, faster, easier to build, less power hungry and physically smaller, which is according to the basic RISC philosophy. The other advantage is that, because regular memory instructions are used to address devices, all of the CPU’s addressing modes are available for the I/O as well as the memory, and instructions that perform an operation directly on a memory operand-loading an operand from a memory location, storing the result to a memory location, or both- can be used with I/O device registers as well.

In fact, all that is needed is an interface to facilitate communication and data transport between the processor’s memory bus and the peripheral device. Clearly, there are several I/O kinds of connection possible. Here, we have chosen for easy connecting to devices that are designed for interfacing using the popular Wishbone architecture. Note that, if the intention is to interface devices that can take more than one processor clock cycle for reading and/or writing data, the need to be able to stall the processor for one or more clock cycles becomes obvious.

1.4 Wishbone Interconnection Architecture

The Wishbone bus is a simple scalable bus specification to connect IP blocks [WBSpec]. The main objective is to use a flexible, robust, easy to understand and technology-independent communication interface. This bus was initially specified by the Silicore company and is now being further developed by OpenCores, so the specification is public domain. As a consequence, many IP blocks have been developed using this type of interface and many are available. All Wishbone bus data transfers can execute in one clock cycle. It can be configured as an 8, 16 or 32 bit wide bus. All bus cycles use a handshaking protocol between the master and the slave IP block(s). The architecture of the bus is not defined; it is up to the user/designer to choose one.

1.4.1 MBL-Wishbone bridge

From the processor side, no distinction has to exist between ‘real’ memory and a Wishbone I/O device. Seen from the other side of the bus, everything has to behave like a fully compliant Wishbone master. This can be accomplished with an appropriate bridge circuit that is responsible for the correct transfer of data, address values and control signals between the MBL1c and the Wishbone compliant peripherals (slaves) using the specified Wishbone control signals. If more than one slave is involved, everyone of them needs its own bridge and a private section in memory space.
Chapter 2

Design Possibilities

The ET4351SC_Pack software package can be downloaded from our web-site http://ens.ewi.tudelft.nl/Education/courses/et4351/. All SystemC types and module descriptions, utilities, etc. that are needed to simulate a SoC as described in the previous section, are at your disposal. Only the code for specific slaves has to be written (both a SystemC-description, as well as the software to be executed by the MBL1C) and merged with the existing software.

The design flow for obtaining a simulation wave file is expected to be known. It can be found in many tutorials and beginners guides on the internet.

On the PC’s in the MSc-lab, full functional installations of the SystemC library files, the GTKWave viewer, and the ModelSim simulator are available.

Of course, it is still possible to edit all files in your favorite editor and use the Linux utility Make together with an appropriate Makefile from the command line to create the simulator-executable, followed by also calling the GTKWave viewer from the command line.

If you are up and running, ET4351_SC offers the following possibilities:

If you start by designing e.g. the core of a slave, this core can be tested directly at a low level by using binary files for supplying input data and writing back the results to be analyzed. See the File I/O example in our BMP-directory. A next step can be to connect the core to a Wishbone interface, again using binary file I/O for feeding the testbench. At a higher level, you can do a simulation with the MBL1c, the Master-Wishbone bridge(s) and the Wishbone slave(s). All files (except for your slave, of course, although there are a few example slaves) are available. Some can be used completely unchanged; some have to be tailored to your application. The software to be executed by the soft-processor should be written (c-file) and compiled, which results in initialized binary instruction- and data-file (the .elf file) that will be read by the simulator. Stimuli and/or input data is either generated by your testbench or again (partly) read from a binary file. The resulting output data can be observed in a simulation waveform viewer or written to a file.
Chapter 3

MBL1C ISS Setup for ET4351

In this chapter an overview will be given of the MBL1c ISS (Instruction Set Simulator) and, after a short summary of the Wishbone interface, how to connect it to peripheral circuitry.

3.1 MBL1C ISS for ET4351

Being a lite version of the regularly improved and expanded MicroBlaze, only a subset of the MicroBlaze’s instruction set can be executed. Table I list the available mnemonic opcodes. In practice, i.e. for the projects for this course, this subset is more than adequate. See the Reference Guide [MicroBlaze] for a detailed explanation of each instruction.

3.1.1 Memory architecture

The MB-Lite is based on a Harvard architecture and thus features separate address- and data-buses for instruction memory (imem) and data memory (dmem). Both instruction memory and data memory start at address 0x00000000, while each address refers to a byte-wide memory location. Given the 32-bit address widths, both memories have a maximum size of 4 GBytes. Thus, although the MB-Lite is a 32-bit machine which addresses and processes 32-bit data units, memory sizes and addresses are specified in bytes, i.e. the 32-bit instructions are found on addresses on a 4-byte boundary only, e.g. the program counter’s lsb-value will always be 0, 4, 8, or c (hex). The same holds true for data memory accesses when addressing 32-bit data.

3.1.2 Data Alignment

As mentioned before, the MBL1c is a 32-bit cpu working with 32-bit data (WORDS), but also capable of handling 16-bit (HALFWORDS) and 8-bit data (BYTES) units. Nevertheless, in all memory accesses to c-data types, pointers, and also the program counter, are addressing bytes. Regarding data handling, it is important to know that the MicroBlaze uses the Big Endian data format, which means that the most significant byte of an operand or data unit is stored at the lowest address in memory. To enable the access of HALFWORDS and BYTES, a 4-bit ‘sel’ signal can be used to select the appropriate part in a 32-bit unit. The relationship between addresses, data types and the sel-signal is illustrated below.
CHAPTER 3. MBLIC ISS SETUP FOR ET4351

**WORD alignment on 4-byte boundaries only**

least significant address nibble 0, 4, 8 or c (sel = "0001")

```
  d31--d0
```

**HALF WORD alignment in 32-bit data field (2-byte boundaries only)**

least significant address nibble 0, 4, 8 or c (sel = "1110")

```
  d31--d16  d15--d0
```

least significant address nibble 2, 6, a or e (sel = "0011")

**BYTE alignment in 32-bit data field**

least significant address nibble 0, 4, 8 or c (sel = "1010")

```
  d31--d24  d23--d16  d15--d8  d7--d0
```

least significant address nibble 2, 6, a or e (sel = "0010")

least significant address nibble 1, 5, 9 or d (sel = "1100")

Figure 3.1
### Table 3.1: Subset of available MicroBlaze instructions

<table>
<thead>
<tr>
<th>Category</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add, subtract and compare functions:</strong></td>
<td>ADD, ADDC, ADDK, ADDKC, ADDI, ADDIC, ADDIK, ADDIKC, RSUB, RSUBC, RSUBK, RSUBKC, RSUBI, RSUBIC, RSUBIK, RSUBIKC, CMP, CMPU</td>
</tr>
<tr>
<td><strong>Logical functions:</strong></td>
<td>AND, ANDI, OR, ORI, XOR, XORI, ANDN, ANDNI</td>
</tr>
<tr>
<td><strong>Extend instructions:</strong></td>
<td>IMM SEXT8, SEXT16</td>
</tr>
<tr>
<td><strong>Shift right:</strong></td>
<td>SRA, SRC, SRL</td>
</tr>
<tr>
<td><strong>Unconditional branch instructions:</strong></td>
<td>BR, BRD, BRLD, BRA, BRAD, BRAID</td>
</tr>
<tr>
<td><strong>Conditional branch instructions:</strong></td>
<td>BEQ, BNE, BLT, BLE, BGT, BGE, BEQD, BNED, BLTD, BLED, BGTD, BGED, BEQI, BNEI, BLTI, BLEI, BGTI, BGEI, BEQID, BNEID, BLTID, BLEID, BGTD, BGEID</td>
</tr>
<tr>
<td><strong>Load and store instructions:</strong></td>
<td>LBU, LHU, LW, LBUI, LHUI, LWI, SB, SH, SW, SBL, SHI, SWI</td>
</tr>
<tr>
<td><strong>Return from interrupt, subroutine</strong></td>
<td>RTID, RTSD</td>
</tr>
</tbody>
</table>

### 3.2 The Wishbone Interconnection Architecture

In the following sections, we will take a closer look at the specific modules that constitute an ET4351 SoC built around a Wishbone Interconnection Architecture [WBSpec].

The Wishbone interface works with a so-called master-slave architecture. This means that one component can claim to be the bus master: that component will then decide what slave to communicate to and what slave may communicate back. Multiple masters and multiple slaves are allowed on the bus. In the designs for this course, there will usually be only one master.

The Wishbone ‘bus’ can support multiple modes of data transfer. The simplest form is the single read- or write-operation. However, so-called block-transfers are also possible: in this case the master keeps the connection to a certain slave open for multiple sequential read- or write-operations. This block-transfer mode can also be used for a combination of read- and write-operations, called a Read-Modify-Write cycle. In such a case the master reads the input data, modifies it, and sends it back to the slave in one block-transfer.

All signal names in the Wishbone specification have ‘_I’ or ‘_O’ characters as postfix. These indicate if the signals are respectively an input (to the module where the name is defined) or an
output (from the module). Names that are followed by a set of parenthesis indicate that we deal with a signal array, e.g. DAT_O().

### 3.2.1 Handshaking protocol

All bus cycles use a handshaking protocol between Master and Slave. As shown in Figure 4, the Master asserts a signal called STB_O when it is ready to transfer data. STB_O remains asserted until the Slave asserts in our case the cycle terminating signal ACK_I. This means either that the Slave is ready to read data from the master, or that the Slave has stable data available for the master to read. At every rising edge of CLK_I i.e. the events that either master or slave can read the data the terminating signal is sampled. If it is asserted, reading takes place and STB_O is negated, so in turn the Slave can negate its ACK_I. This gives both Master and Slave interfaces the possibility to control the rate at which data is transferred.

![Local bus handshaking protocol](image)

**Figure 3.2:** Local bus handshaking protocol

### 3.2.2 The Wishbone specification

The Wishbone bus specification has a number of rules that apply directly to the Wishbone slave(s) that will be created. The most important basic ones are:

**RULE 2.30** states that all Wishbone interface signals MUST use active high logic.

**RULE 3.00** specifies that the Wishbone reset signal is synchronous.

**RULE 3.10** specifies that the Wishbone reset signal should override all other signals.

**RULE 3.15** specifies that all self-starting state machines and counters in Wishbone interfaces should initialize themselves as long as a Wishbone reset signal is present.

**RULE 3.30** specifies that the slave may only respond to signals when CYC_I is active.

**RULE 3.35** specifies that the cycle termination signals ACK_O must be generated in response to the logical AND of CYC_I and STB_I.
RULE 3.40 specifies that the slave should at least have the signals ACK_O, CLK_I, CYC_I, STB_I, and RST_I. The bus master however specifies that also the signals WE_I, ADR_I, DAT_I, DAT_O, and an interrupt signal INT_O are used.

RULE 3.50 specifies that the ACK_O signal must react to every change of STB_I. PERMISSION 3.30 however specifies that this may be done asynchronous to the CLK_I. The slave can also control transfer speed this way.

RULE 3.90 specifies the method of data ordering. Both Big Endian and Little Endian ordering are supported. For the ET4-351 course Little Endian ordering will be used.

RULE 3.100 specifies the organization to which an 8-bit port must comply. However, nothing extraordinary is mentioned here.

RULE 5.00 specifies that all Wishbone signals are clocked on the positive clock edge.

RULE 5.05 specifies that all Wishbone interfaces must be synchronous.

Next to that, there are also 2 so-called ‘permissions’ that should be mentioned here:

PERMISSION 3.10 If the SLAVE guarantees it can keep pace with all MASTER interfaces and if the [ERR_I] and [RTY_I] signals are not used, then the SLAVE’s [ACK_O] signal MAY be tied to the logical AND of the SLAVE’s [STB_I] and [CYC_I] inputs. The interface will function normally under these circumstances.

PERMISSION 3.35 Under certain circumstances SLAVE interfaces MAY be designed to hold [ACK_O] in the asserted state.

Support for extensions to the basic Wishbone operations, such as the Burst-cycle, will not be added. Figure 5 shows the timing relationships for the most common single_read and single_write operations. The signal names shown here are from the master’s perspective, with of course the master’s ‘_I’ signals connected to the corresponding slave’s ‘_O’ signals, and vice versa.

In the above, only a very concise description of the most important rules and signals were given. For the complete specification, the reader is referred to [WBSpec].

3.2.3 The Wishbone signals at the slaves side

The Wishbone signals connecting the bus to a slave, with a short description of their functions, are:

CLK_I: This is the "CLocK Input" signal.

RST_I: The "ReSeT Input" signal initiates a synchronous reset of the wishbone bus.

ADR_I: With the "ADdRes Input array" the Wishbone master selects a specific slave address to write to or to read from.

DAT_I: The "DATa Input array" signal is used to receive data from the master.
DAT_O: The "DATa Output array" signal is used to transfer data from the slave to the master. Slaves that are inactive must disable this output using tri-state buffers, as this is a bus-signal.

WE_I: With this "Write Enable Input" signal the Wishbone master signals the slave to receive or to send data (WE_I high means that the master writes valid data that the slave should read).

STB_I: This is the "STRobe Input" signal the Wishbone master sends to start a data transfer. This signal can be used to select a particular slave by only activating the STB_I for that slave.

CYC_I: This is the "CYCle Input" signal the Wishbone master sends to initiate a transfer cycle. Multiple data transfers can take place within one cycle.

ACK_O: This is the "ACKnowledge Output" signal that acknowledges correct transfer over the Wishbone bus. The active slave replies to the STB_I signal when data is transferred.

INT_O: This is a signal for the "INTerrupt Output". Interrupt signals are not explicitly mentioned in the Wishbone specifications. It is nevertheless regarded to be a Wishbone signal, because the soft-processor can access it via the Wishbone interface.
Chapter 4

Programming the MBL1c

Code for the instruction memory can be developed by writing a `.c`-file, with an accompanying `.h` header file. The open source mb-gcc compiler, when started from the supplied Makefile will result in one binary file: `.elf`, that will be used to automatically ‘program’ the mbl1c processor when a reset is detected during the simulation, and `dmem.bin` that will be used to initialize data memory. Note: Although an interrupt mechanism is implemented in the hardware, we don’t supply the software for implementing interrupt service routines, since the necessary Xilinx code has not been released in the public domain.
Chapter 5

The ET4351_SC Package

The aforementioned SoC architecture is described in a number of SystemC files. Some of these files can be used without any alterations (e.g. the ISS) as they are independent of the rest of the design, while others need to be tailored to the exact wishes of the designer. A slave can be an IP block that has been designed to interface to the Wishbone bus directly or also as a combination of a non-Wishbone part and an bridge to become Wishbone compatible. Sometimes, a slave’s IP block can be totally implemented on the same FPGA or ASIC as the SoC, but it most cases it will consist of peripheral ICs. In such cases, assuming that a slave interface is not Wishbone compatible, an bridge will have to be created on the FPGA/ASIC and brought to I/O pads..

Figure 5.1: General SystemC design flow
Figure 5.2: SystemC software structure
Appendix A

Software

A.1 Installation and software requirements

The SystemC files are Operating System independent, and should work on any system, provided that you have access to a fully functional C++ compiler, the SystemC library and a waveform viewer. These can be all separate programs (e.g. GNU’s gcc or g++, and GTKWave), or can be combined in a single one, e.g. as in Vivado_HLS.

For creating the initialized .elf file, the mb-gcc compiler will be needed. This compiler, together with a bunch of other mb targeting tools is provided in the package.

Also a number of shell scripts and Makefile’s is available, that are intended to be run on one of the numerous Linux OS distributions.

The ET4351SC_Pack can be downloaded from http://ens.ewi.tudelft.nl/Education/courses/et4351/

Unzip it to a directory of your choice, while preserving full path names.

A.1.1 Creating contents for the instruction memory

The program code is made by compiling and linking the c-code with mb-gcc, resulting in a binary .elf-file. Ensure yourself that the #defines and assignments in the .h- and .c-files are in accordance with the ‘hardware architecture’ defined in main.h.

We provide a small Makefile in the sw-directory, such that a simple

$ make

should be enough to create the .elf file.

A.2 Files

./settings.csh
./setup.csh
./script.tcl.tmpl
./script.tcl
./x\_hls.tcl

./vivadohls.csh
./vsim.csh
./synplify\_pro.csh
./synth/WBStructs.h
APPENDIX A. SOFTWARE

A.2.1 Installation

These are the basic steps (we will work with a 'csh' shell):

```bash
source settings.csh
./setup
``` (only once!. This will create the top level cpp file, 'tb' Makefile and vivado_HLS TCL script file)

```bash
cd sw; make ; cd ..
``` (This will create the MBL executable ELF file)

```bash
./vivadohls.csh -f script.tcl
```
(You have to run this command at least one time; It will create a project direcory)

```bash
vim ./x_hls.tcl
```
(edit/change the number; It controls the flowsteps of Vivado HLS)

```bash
./vivadohls.csh
```
(This will start vivado_HLS user interface mode)

```bash
cd tb; make ; cd ..
```
(This will create a SystemC executable in the 'tb' directory)
A.3 Building and running the example WBSlave

To repeat: the steps to build and run the wbs32_test slave are:

cd sw; make ; cd .. (This will create the wbs32_test executable ELF file)

cd tb; make ; cd .. (This will create the SystemC executable ‘top’ in the ‘tb’ directory)

cd tb; ./top ../sw/wbs32_test (This will run the SystemC executable ‘top’, and load the wbs32_test executable ELF file. It will produce runtime messages on the console terminal.

gtkwave tr.vcd (This will display the signal traces)

Important: Currently we get an unexplained error; Simulation results are however correct. Just ignore for now.

# ** Note: simulation done! #
# ** Failure: NORMAL EXIT (note: failure is to force the simulator to stop) #
# Time: 5445 ns Iteration: 0 Process: /autotb_top/proc_tv_out File: WBSlave.autotb.vhd
# Break in Process proc_tv_out at WBSlave.autotb.vhd line 362
@F [SIM−301] Simulation FAILED with 498 mismatches detected.
@E [SIM−4] *** C/RTL co-simulation finished: FAIL ***
command 'ap_source' returned error code
while executing
"source $argv 1"
("uplevel" body line 1)
invoked from within
"uplevel $argv 1 source $argv 1"

A.4 Course Lab facilities

The Lab facility is located on the ground floor, the ‘Veemhallen’, LH 00.530. Twelve computers are available; All have installed the required (Modelsim, Xilinx, Vivado_HLS, Synopsys and Cadence) software you need for the design and implementation of your project.

In order to get access to these PC’s, you need to ‘register’ your NetID.

Send email to Antoon Frehe, a.p.frehe@tudelft.nl, Subject: NetID ET4351, and your NetID.

A.4.1 Remote Access (Rdesktop)

It is possible to use the ‘Rdesktop’ software tool to get access to a central Rdesktop server. Rdesktop is available for any machine including Windows, Apple, and Linux.

Logon with your NetId. In order to get access to the Rdesktop server, you need to ‘register’ your NetID.

Send email to Antoon Frehe, a.p.frehe@tudelft.nl, Subject: NetID ET4351, and your NetID.

rdesktop -g 1280x900 -u YOURNETID xrdp.ewi.tudelft.nl

Then open a terminal window and connect to the Lab virtual machine:

ssh -Y et4351.et.tudelft.nl
You have now the same software facilities available as when using the Course Lab PC’s. Note: You have a disk quota on this machine. Default quota is 4GB.

The default shell is: ‘csh’.
Exit the remote ssh shell with: ‘exit’.
Remote Desktop Log Out
Logoff NOT by clicking x but use the green man button aside of the Applications Menu button.
It is ESSENTIAL that you uncheck/clear the ’Save session for future logins’ checkbox.
You will have to do this only once.

A.4.2 File transfer

If you need to transfer files from/to the Lab virtual machine and your local machine you need to use a VPN network connection.
(see also: http://vpn.tudelft.nl/.

This is an example of a default configuration for Linux when using VPNC (default.conf located in /etc/vpnc):

IKE DH Group dh2
Perfect Forward Secrecy nops
IPSec gateway luchtbuag.tudelft.nl
IPSec ID public
IPSec secret wireless
Xauth username YOURNETID
Xauth password YOURNETIDPASSWD
Debug 0

A.4.3 Basic steps project assignment

![Diagram of hardware components](image-url)

**Figure A.1:** The default example: wbs32_test
install software
run demo project (wbs32_test) and get familiar with the tools and software organization.
Then,
- analyze your selected algorithm
- determine fixed point datatype size
- adjust code to interface with WB bus
- investigate optimization options, e.g. pipelining, loop unrolling, code specification
- implement the SC code; Test and debug using the `tbMakefile`
- When simulation is successful, start using the Vivado_HLS tool: simulate and synthesize, analyze and optimize until you are satisfied.
write report
Bibliography


[7] EasyBMP is a simple, cross-platform, open source C++ library designed for easily reading, writing, and modifying bitmap (BMP) image files. http://easybmp.sourceforge.net/