525.612.81/82 – Computer Architecture
Setting up for the FPGA tools
Windows 10 Version

Dr. Nicholas Beser

Welcome to the Spring Computer Architecture class. During this semester we will be studying RISC architectures by simulating them using Verilog code, running them on the Xilinx Artix-7 FPGA using the Nexys board. Since many of our students have recently taken the VHDL class at JHU, you have a Windows 10 system running Xilinx Vivado.

This getting started guide has been written to ease the addition of tools and hardware. During this semester we are using the latest version of Vivado (at this time 2020.2). If you have an older version you can either update to the latest, or be forced to save the files we offer the class in the older format. While I recommend using the latest version, all of the code should run on 2018 and 2019 Vivado.

In this document we are going to:

- Load the snapshot of the repository from the mips-cpu.zip file
- Install the MTI bare metal toolchain
- Set the environment variables for the MIPS assembler, and the support files (make, xxd)
- Install Python (if you don't have it)
- Run Vivado (I am assuming that it is already installed)
- Run the assembler, generating the ihex and imem files needed by Vivado and the python script
- Assemble the FPGA board, and the pmodUSBUart module, connect to the PC
- In VIvado, load the bitstream file to run a simple MIPS example
- Run putty (you may have to load it if you don't have it) to identify which communications port you are using for the FPGA and Uart module.
- Run the python script to load the ihex file to the FPGA board, starting a new program.

At the end of this document, you should be able to assemble a test MIPS program, and then run it using the Vivado simulator to watch it's execution. Depending on the program, you may also be able to see it execute on the FPGA board.

I am assuming at this point you already have a version of Vivado on your computer. If not, you need to go to https://www.xilinx.com/support/download.html to download and install the latest version of Vivado. We will be using the Web installer, and using the free Webpack version.

MiniRepo:

Go to Canvas and select Module 1 Course Module. Go to the readings and download the mips-cpu.zip file. The structure of the file is the same that we will use with the bitbucket.org repository.

Unzip the file in a directory like (c:\Distro). You will have the following structure:

- mips-cpu
  - RTL
    - 3rd Party IP
      - Examples
      - Uart-debug
    - Common
      - io
    - CPUs
      - mccompbasicFPGA
      - Single Cycle with Interrupts
  - Software
    - Assembly
      - makesystem
      - mc7segment
      - mcSwitchLED7segment
      - scintcode7segment
      - scintcodeSwitchLED7segment
    - UART

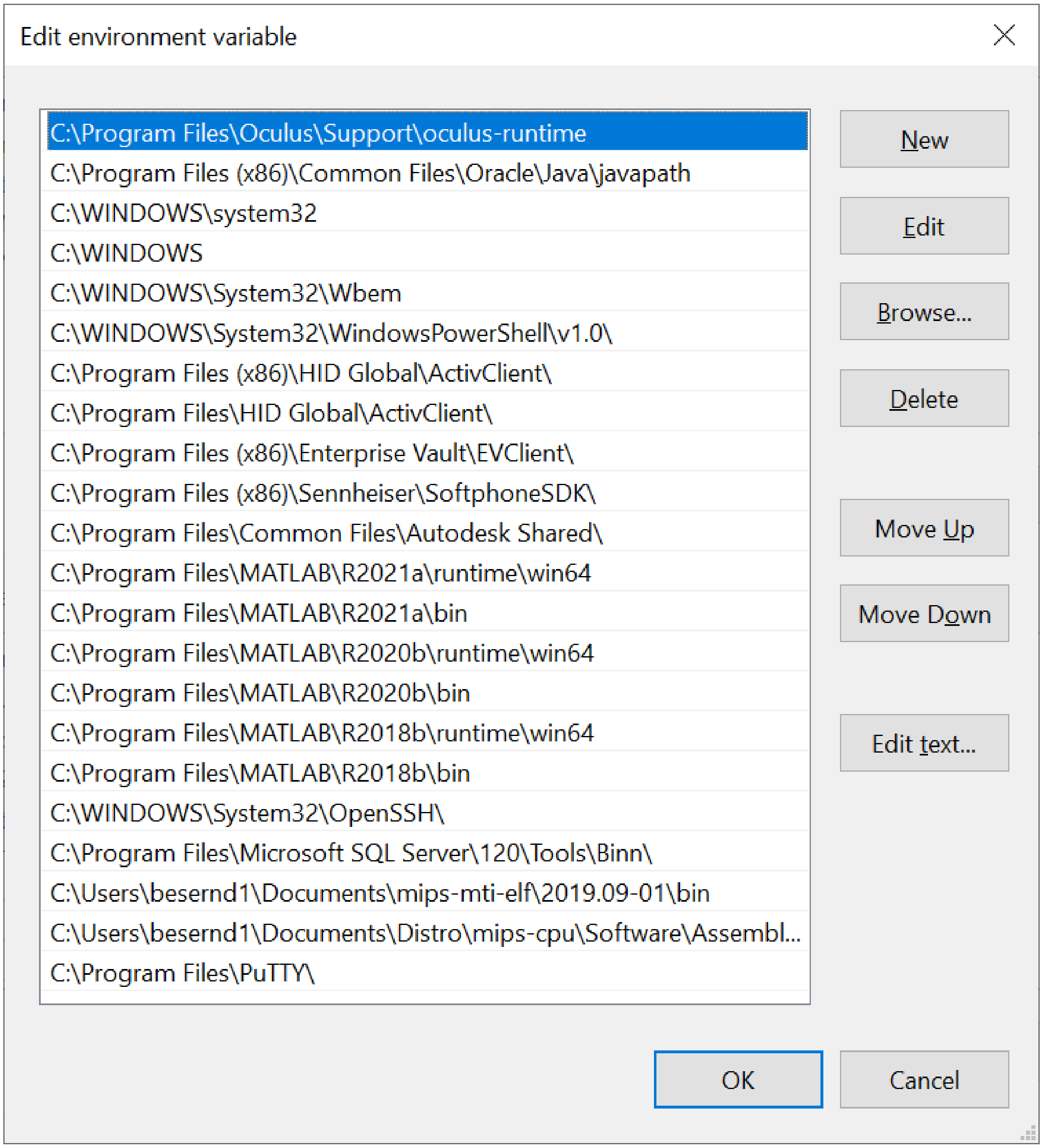
MIPS-MTI-ELF Assembler and Compiler

The RISC processors that we will be studying are based on the MIPS architecture as described by Patterson and Hennessy. The initial processor has a very small instruction set, and as we study the architectures we will add additional instructions. The MTI-ELF toolchain contains an Assembler and C compiler, plus additional utilities that will convert the various output files, link them, and create memory images. The code is available from: https://codescape.mips.com/components/toolchain/2019.09-01/downloads.html

Download the MTI Bare Metal Toolchain for windows x64 (450Mbytes). Unarchive it in the Downloads directory. Once it is available:

1. Go to the start memu and type in env and chose Edit the system environment variables.
2. Click the Environment Variables button
3. Set the environment Variable PATH to include the directory when the bin files from the MTI Bare Metal Toolchain are located

4. You will also add the directory where the makesystem folder is located so you can run the make and xxd utility.



Edit environment variable

```
C:\Program Files\Oculus\Support\oculus-runtime
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
C:\WINDOWS\system32
C:\WINDOWS
C:\WINDOWS\System32\Wbem
C:\WINDOWS\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\HID Global\ActivClient\
C:\Program Files\HID Global\ActivClient\
C:\Program Files (x86)\Enterprise Vault\EVClient\
C:\Program Files (x86)\Sennheiser\SoftphoneSDK\
C:\Program Files\Common Files\Autodesk Shared\
C:\Program Files\MATLAB\R2021a\runtime\win64
C:\Program Files\MATLAB\R2021a\bin
C:\Program Files\MATLAB\R2020b\runtime\win64
C:\Program Files\MATLAB\R2020b\bin
C:\Program Files\MATLAB\R2018b\runtime\win64
C:\Program Files\MATLAB\R2018b\bin
C:\WINDOWS\System32\OpenSSH\
C:\Program Files\Microsoft SQL Server\120\Tools\Binn\
C:\Users\besernd1\Documents\mips-mti-elf\2019.09-01\bin
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\Assembl...
C:\Program Files\PuTTY\
```

Figure 1 – Set Environment window from control panel. Notice that the two lines near the bottom define the MTI-ELF tools and the makesystem folder with the make and xxd tools.

Install Python on your windows computer

We use python3 to load the intel hex formatted file to the MIPS processor on the FPGA board. I went to the Microsoft store and downloaded a free version of python. Once python is installed, you can then use the requirements file to load additional packages required to run the script.

From a command prompt run:

Pip3 –version

This will indicate everything is installed and ready to begin (I am running Python 3.9

Go to the UART directory and type:

Pip3 install –r requirements.txt

This will install intelhex and pyserial.

Install putty on your windows computer

If you don’t have putty installed, go to https://www.putty.org/ and download the version for windows 10. We will use putty to determine which serial port is talking to the FPGA and which port is talking to the pmodUSBUart.

Assembling the MIPS code

Four MIPS programs are provided, 2 for each architecture. The first program sets the 7 segment display with numbers: 76543210. The other program reads the slider switches and lights the LED with switch settings. Each directory contains a Makefile that will access support files in the makesystem folder. It should be noted that the contents of the makesystem folder assume a linux environment. The parent.mk will need to be modified for windows.

Note that the change is the > put in front of the mem.mem files.

```
singlesim: assemble

	$(OBJCOPY) -O binary -j .text -j .data $(PROGNAME).elf mem.bin

	-$(OBJDUMP) -s -j .data $(PROGNAME).elf

	-$(OBJDUMP) -d -S -M reg-names=numeric -j .text $(PROGNAME).elf


	xxd -c 4 -ps mem.bin > mem.mem


idsim: assemble

	$(OBJCOPY) -O binary -j .text $(PROGNAME).elf imem.bin

	$(OBJCOPY) -O binary -j .data $(PROGNAME).elf dmem.bin

	$(OBJCOPY) -O ihex -j .text $(PROGNAME).elf imem.ihex

	$(OBJCOPY) -O ihex -j .data $(PROGNAME).elf dmem.ihex
	-$(OBJDUMP) -s -j .data $(PROGNAME).elf
	-$(OBJDUMP) -d -S -j .text $(PROGNAME).elf

	xxd -c 4 -ps imem.bin > imem.mem
	xxd -c 4 -ps dmem.bin > dmem.mem

fullsim: assemble
	$(OBJCOPY) -O binary -j .init_text $(PROGNAME).elf mem0.bin
	$(OBJCOPY) -O binary -j .tlb_data $(PROGNAME).elf mem1.bin
	-$(OBJDUMP) -s -j .tlb_data $(PROGNAME).elf
	-$(OBJDUMP) -d -M reg-names=numeric -j .init_text $(PROGNAME).elf
	$(OBJCOPY) -O binary -j .text $(PROGNAME).elf mem2.bin
	$(OBJCOPY) -O binary --gap-fill 0x00 -j .data -j .rodata -j .bss -j
.early_stack $(PROGNAME).elf mem3.bin
	-$(OBJDUMP) -s -j .data $(PROGNAME).elf
	-$(OBJDUMP) -d -S -M reg-names=numeric -j .text $(PROGNAME).elf

	xxd -c 4 -ps mem0.bin > mem0.mem
	xxd -c 4 -ps mem1.bin > mem1.mem
	xxd -c 4 -ps mem2.bin > mem2.mem
	xxd -c 4 -ps mem3.bin >mem3.mem
```

Go to the following directory in the mips-cpu respository:
mips-cpu/Software/Assembly/scintcode7segment

Examine the Makefile:
Contents of Makefile:

```
SOFTWARE_ROOT = ..
A_SOURCES = scintcode7segment.S
C_SOURCES =
PROGNAME = scintcode7segment
include $(SOFTWARE_ROOT)/makesystem/parent.mk
```

In a command window (cmd) in the scintcode7segment folder type:

```
make split_i_d_mem
```

Since the single cycle with interrupts processor has a separate memory for program and data, we needed to split the instruction and data memory. This will create two new files: imem.mem (instruction memory text file) and dmem.mem (data memory text file). The two files must be read into the Verilog source file in Vivado. Notice that the make file also generated a listing of the assembly code. This can be used when running Vivado simulator to debug the program. The Makefile will also make a ihex version of the two files. Those files are used by the python script for loading with the download-ihex.py to the pmodUSBUart.

In order to test the USB device, go to the other directories (scintcodeSwitchLED7segment, and make the MIPS code in the same way as the first program.

If you build the code for the multicycle MIPS processor, that system has a single memory (both instruction and data). The make procedure for the multicycle processor is:

```
make single_mem
```

One caveat: At this time I am having problems getting the make command working correctly. One step seems to be failing:

xxd -c 4 -ps mem.bin > mem.mem (for single memory command).

Or

xxd -c 4 -ps imem.bin > imem.mem

xxd -c 4 -ps dmem.bin > dmem.mem (for split I_D memory command).

Since the xxd command, failed, the next command also did not run:

mips-mti-elf-objcopy -O ihex programname.elf > programname.ihex

The command creates a ihex file which is used in the python script to download. The imem and dmem files are used in Vivado to load the program and memory into the design. If the make file does not work, you may have to type the commands by hand.

Running VIvado

Once Vivado starts, go to the following directory (assuming that you put the repository in Documents):

```
mips-cpu/RTL/CPUs/Single Cycle with interrupts
```

Click on sccompintFPGA.xpr to open the single cycle with interrupts design. Since the project was created with an earlier version of Vivado, you will get a message that it has to be converted to the current version. You will also get a Project Upgraded message. I usually select Report IP Status. The conversion should be automatic; however we will need to update the instruction and data memory files.

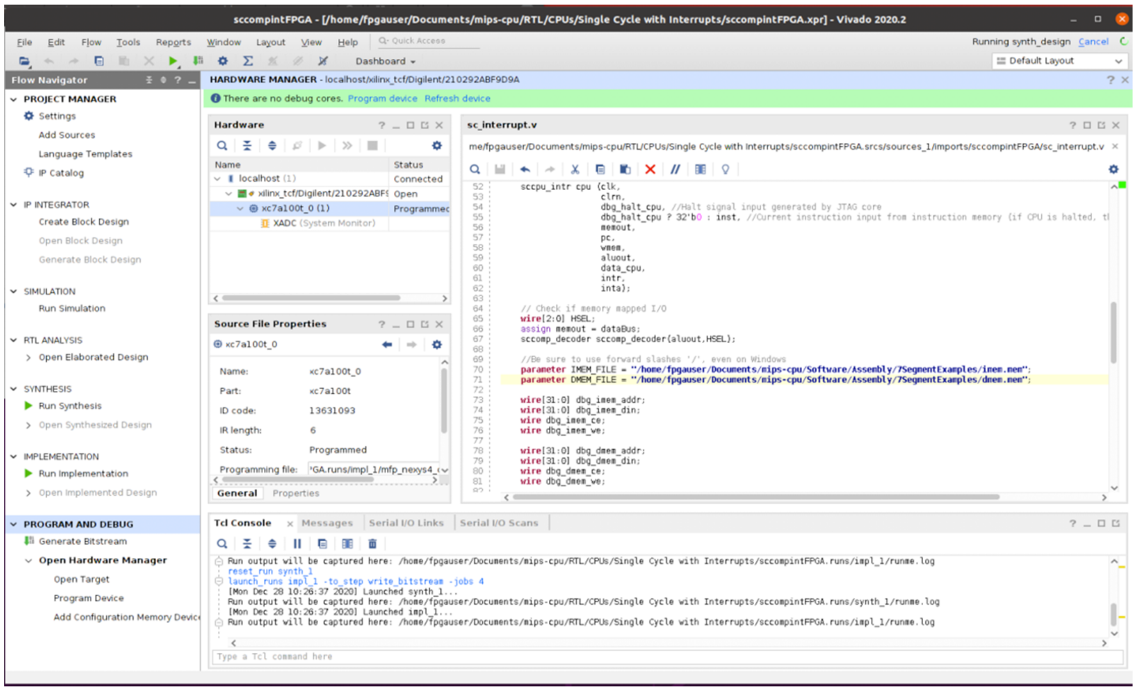
Open the Project Manager, and click on mfp_nexys4_ddr file. We need to open the sc_interrupt.v file to update the links. Scroll to lines 70 and 71 and you will see the parameter IMEM_FILE and parameter D_MEM_FILE definitions. They need to be updated to reflect the newly assembled code for the scintcode7segment routine. Note, this is the most common mistake most people make when updating programs. The exact directory location must be given.

Note that although windows uses \ as the file delimiter, Vivado uses / instead. (Just like Linux)

The new lines should be:

```
parameter IMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/imem.mem";

parameter DMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/dmem.mem";
```



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210292ABF	Open
xc7a100t_0 (1)	Programmed
XADC (System Monitor)	

Name:	xc7a100t_0
Part:	xc7a100t
ID code:	13631093
IR length:	6
Status:	Programmed
Programming file:	'GA.runs/impl_1/mfp_nexys4_

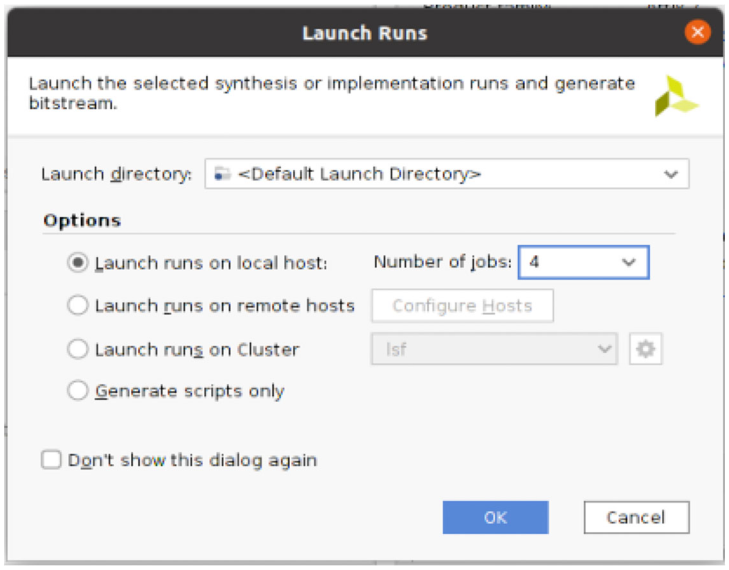
me/fpgauser/Documents/mips-cpu/RTL/CPUs/Single Cycle with Interrupts/sccompintFPGA.srcs/sources_1/imports/sccompintFPGA/sc_interrupt.v

```
52    sccpu_intr cpu (clk,
53                    clrn,
54                    dbg_halt_cpu, //Halt signal input generated by JTAG core
55                    dbg_halt_cpu ? 32'b0 : inst, //Current instruction input from instruction memory (if CPU is halted, t
56                    memout,
57                    pc,
58                    wmem,
59                    aluout,
60                    data_cpu,
61                    intr,
62                    inta);
63
64    // Check if memory mapped I/O
65    wire[2:0] HSEL;
66    assign memout = dataBus;
67    sccomp_decoder sccomp_decoder(aluout,HSEL);
68
69    //Be sure to use forward slashes '/', even on Windows
70    parameter IMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/imem.mem";
71    parameter DMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/dmem.mem";
72
73    wire[31:0] dbg_imem_addr;
74    wire[31:0] dbg_imem_din;
75    wire dbg_imem_ce;
76    wire dbg_imem_we;
77
78    wire[31:0] dbg_dmem_addr;
79    wire[31:0] dbg_dmem_din;
80    wire dbg_dmem_ce;
81    wire dbg_dmem_we;
```

```
Run output will be captured here: /home/fpgauser/Documents/mips-cpu/RTL/CPUs/Single Cycle with Interrupts/sccompintFPGA.runs/impl_1/runme.log
reset_run synth_1
launch_runs impl_1 -to_step write_bitstream -jobs 4
[Mon Dec 28 10:26:37 2020] Launched synth_1...
Run output will be captured here: /home/fpgauser/Documents/mips-cpu/RTL/CPUs/Single Cycle with Interrupts/sccompintFPGA.runs/synth_1/runme.log
[Mon Dec 28 10:26:37 2020] Launched impl_1...
Run output will be captured here: /home/fpgauser/Documents/mips-cpu/RTL/CPUs/Single Cycle with Interrupts/sccompintFPGA.runs/impl_1/runme.log
Type a Tcl command here
```

Figure 2 – Update the sc_interrupt.v file with the correct location of the imem.mem and dmem.mem files. Save the file.

Click on Generate Bitstream. Since this is the first time this has been run, Vivado will go through the entire process, synthesis, implementation and layout, generating the bit stream. The launch menu will provide an option to set the number of jobs. Set it to the maximum number of processors to speed up the compilation.


Launch Runs

Launch the selected synthesis or implementation runs and generate bitstream.

Launch directory: <Default Launch Directory>

Options

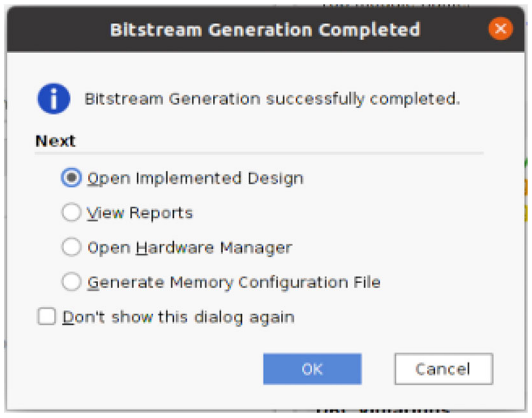
- [✓ Launch runs on local host:] Number of jobs: 4
- Launch runs on remote hosts — Configure Hosts
- Launch runs on Cluster — lsf
- Generate scripts only

Don't show this dialog again

OK Cancel

Figure 3 – This setting will allow 4 CPU’s to work on the jobs in parallel.

After Vivado completes the run, it will produce a window with the option to Open Hardware manager:


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

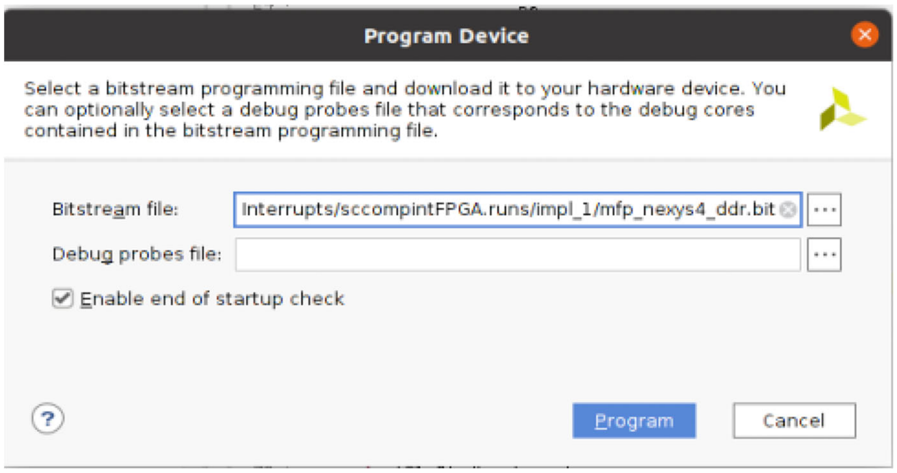
- [✓ Open Implemented Design]
- View Reports
- Open Hardware Manager
- Generate Memory Configuration File

Don't show this dialog again

OK Cancel

Figure 4 – Select Open Hardware Manger to load the Nexys board.

Make sure the Nexys board is on and connected to the computer. Select open target, and then auto connect. The system should return with the Digilent board visible, and Program Device as an option to be selected:


Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: Interrupts/sccompintFPGA.runs/impl_1/mfp_nexys4_ddr.bit

Debug probes file:

☑ Enable end of startup check

Figure 5 – Loading the bitstream to the Nexys Board

After loading the bitstream the board should produce:

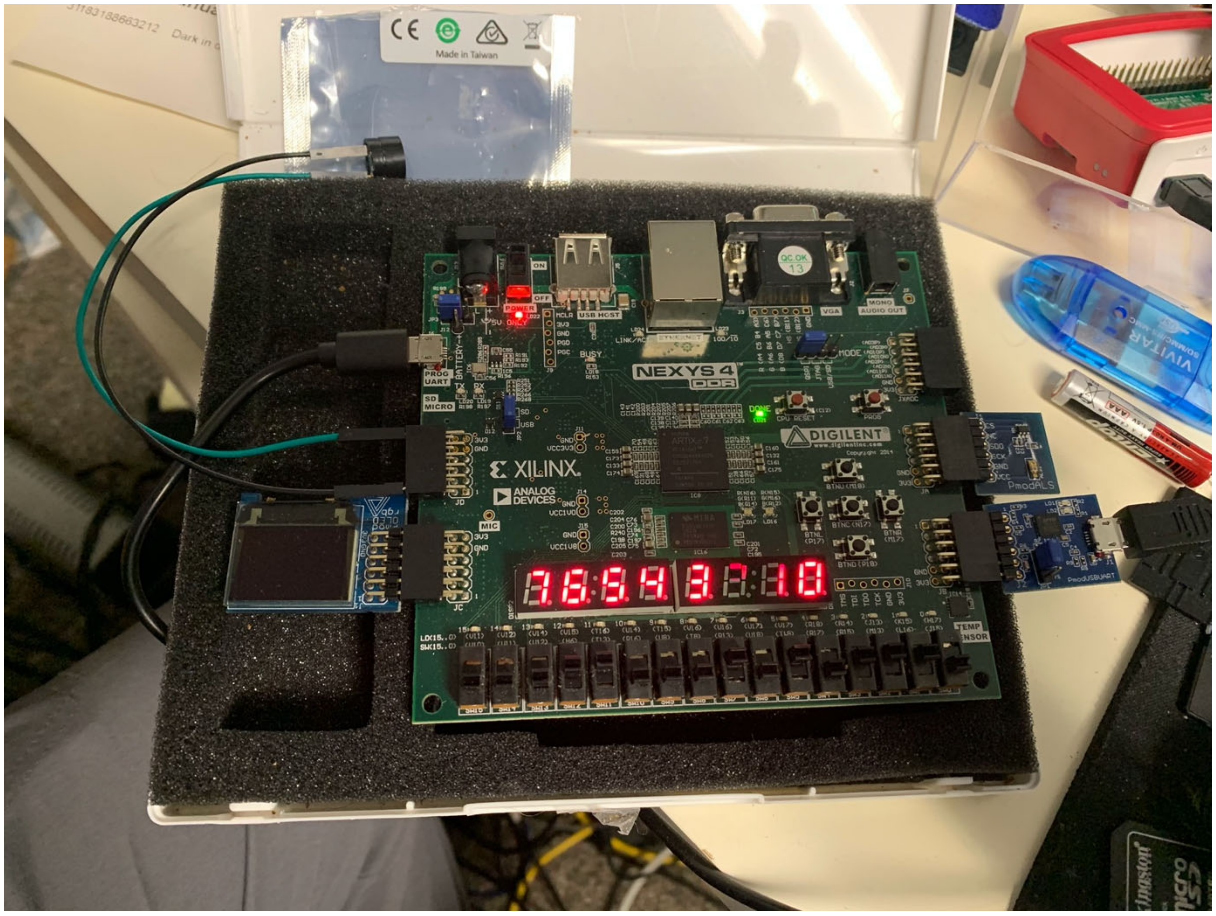
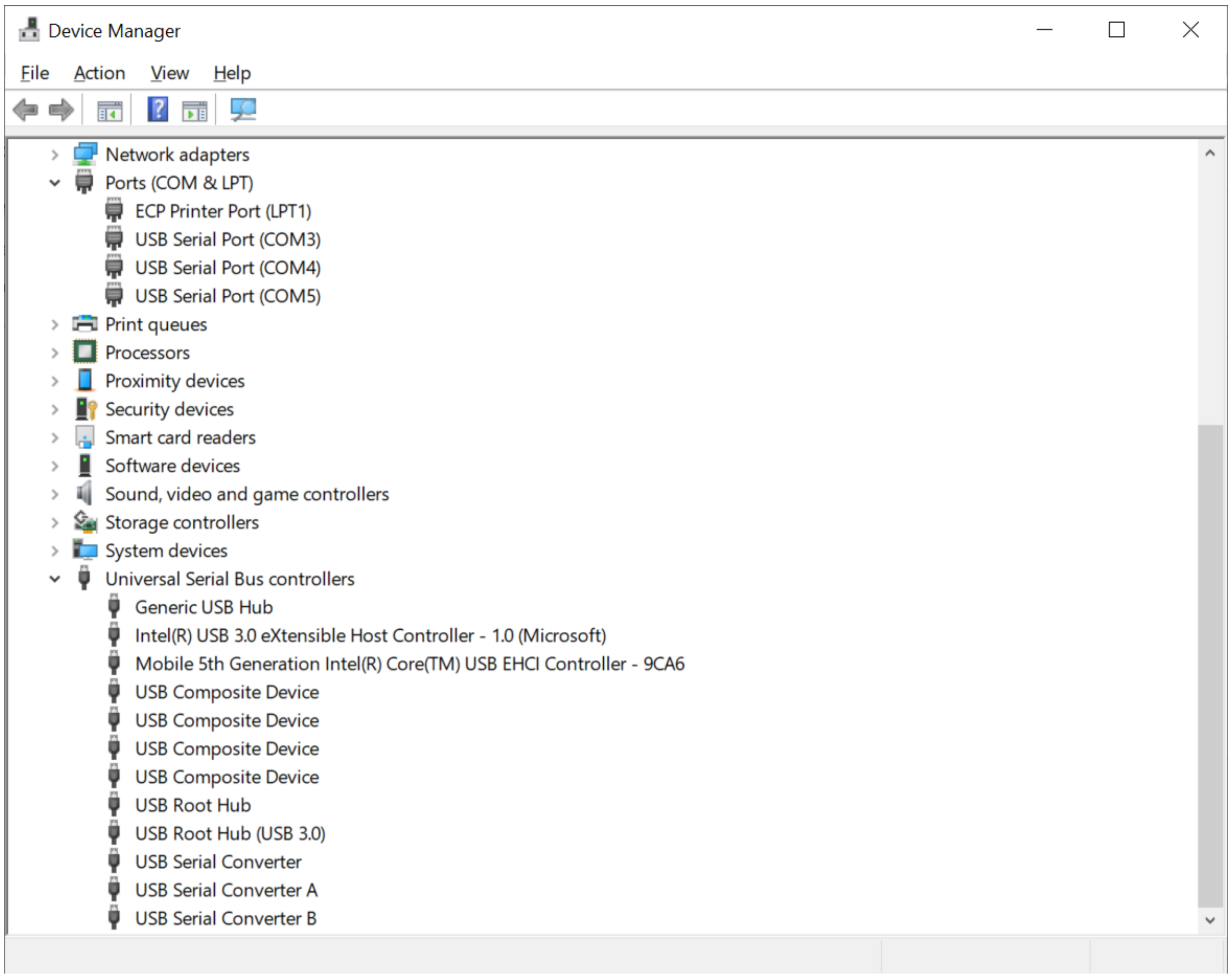


Figure 6. The Nexys 4 DDR board running the single cycle with interrupts MIPS assembly code. Notice the pmodUSBUart module plugged into the top pins of JB. Connector JA has the pmodALS board (light detector), and Connector JC has the pmodOLEDrgb module. Connector JD has the speaker wired into pins 1 and 6 (signal and power).

Testing the COM's Ports

If both USB cables are attached to the computer, you need to find out what port to use for the python script. First go to the Device Manager and examine both the USB ports and the serial ports:

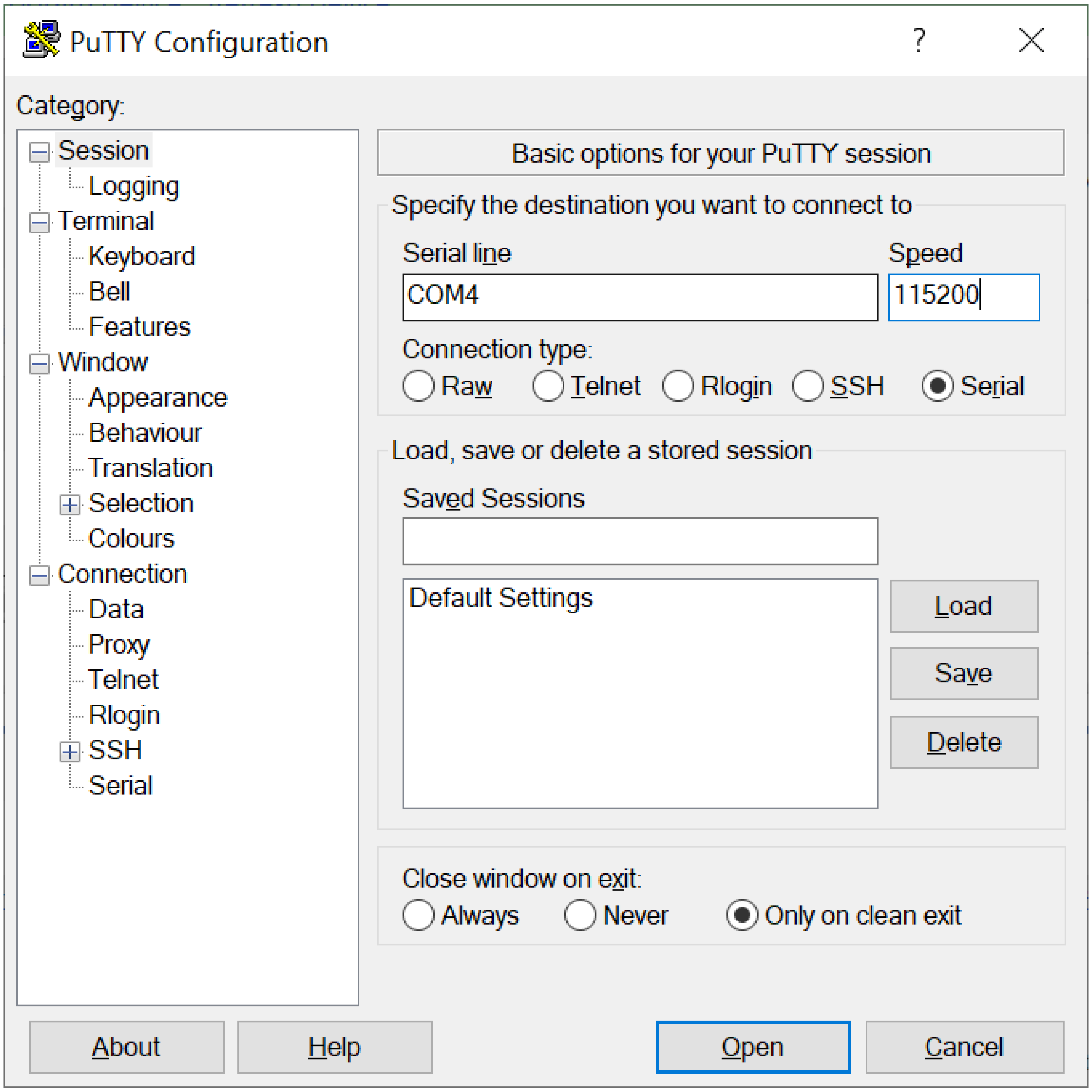


```
> Network adapters
v Ports (COM & LPT)
    ECP Printer Port (LPT1)
    USB Serial Port (COM3)
    USB Serial Port (COM4)
    USB Serial Port (COM5)
> Print queues
> Processors
> Proximity devices
> Security devices
> Smart card readers
> Software devices
> Sound, video and game controllers
> Storage controllers
> System devices
v Universal Serial Bus controllers
    Generic USB Hub
    Intel(R) USB 3.0 eXtensible Host Controller - 1.0 (Microsoft)
    Mobile 5th Generation Intel(R) Core(TM) USB EHCI Controller - 9CA6
    USB Composite Device
    USB Composite Device
    USB Composite Device
    USB Composite Device
    USB Root Hub
    USB Root Hub (USB 3.0)
    USB Serial Converter
    USB Serial Converter A
    USB Serial Converter B
```

Figure 7 – Device Manager application – Identify the COM ports

You will notice that there are two ports labeled USB Serial Converter A and B. Look at the properties of the ports COM4 and COM5. It will be one of those two ports.

Launch putty and select a serial terminal for COM4, setting the baud rate to 115200



Specify the destination you want to connect to

Serial line: COM4

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

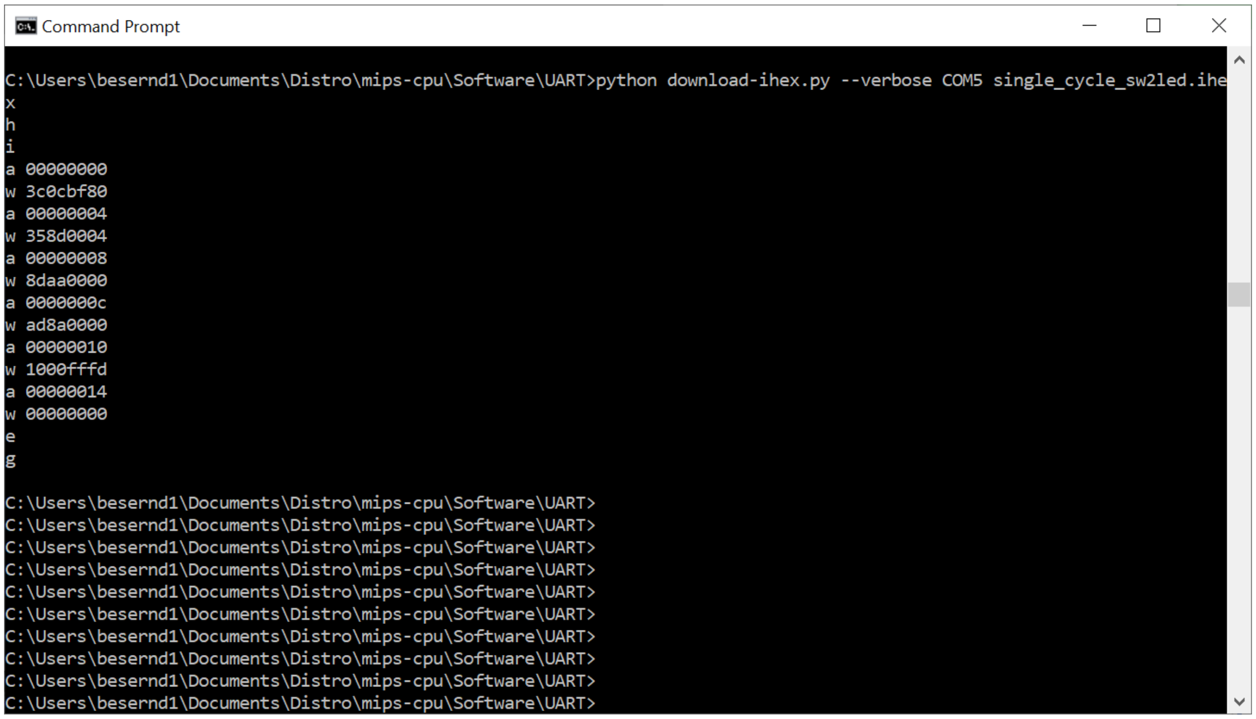
Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Figure 8 – Putty set to COM4 at 115200 baud.

When you open the terminal, type h for halt. If the terminal comes back with OK, you have the correct port. If it does not return with anything, look at the Nexys FPGA board LEDS at Prog UART , TX and RX. If the TX flashes, then the port was for the FPGA board. Switch to COM4 at 115200 baud and rerun the test. If you don't have OK, you might need to check your USB cables.

Once the ports are selected, you will run the python script.



Command Prompt

```
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>python download-ihex.py --verbose COM5 single_cycle_sw2led.ihe
x
h
i
a 00000000
w 3c0cbf80
a 00000004
w 358d0004
a 00000008
w 8daa0000
a 0000000c
w ad8a0000
a 00000010
w 1000fffd
a 00000014
w 00000000
e
g

C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
C:\Users\besernd1\Documents\Distro\mips-cpu\Software\UART>
```

Figure 9 Python script for simplified