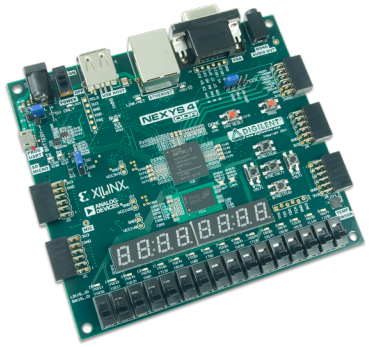
Lab Hardware Handout 2021

ECE/CS 153A, ECE 253

This document provides a brief overview of Vivado and Xilinx SDK use for the labs. It is intended as a reference for labs and the undergraduate course project. The hardware handout includes information for creating Microblaze projects in Vivado, connecting peripherals and debugging tips.



Choosing your Hardware

The hardware selected for prototyping and development can vary depending on the exact needs of the embedded engineer. For the Fall 2021 course the selected hardware is the Artix A7-100T (on the Nexys4 DDR or Nexys A7 100T board) [1]

[1] IMPORTANT: Make sure the development board is the **DDR** version.

Introduction

This document shows how to build a complete Microblaze system containing

- MicroBlaze CPU.
- CPU support (such as debugging)
- Interface to the 128MB DDR2 memory on the board
- 128KIns + 128KData SRAM local to the CPU
- 64k Instruction and 64k Data caches local to the CPU.
- Peripherals such as GPIO (General Purpose I/O), timers, interrupt controllers, etc.

These peripherals will be added slightly out of order. In particular, the memory interface generator will be added first. This is because it is electrically convenient to have the memory interface accept the off-chip crystal clock and produce a clock for the rest of the system.

Creating a Vivado 2019.1 Project

1. Open Vivado, Click “Create new project" and Next. Name your project and choose the project location.
2. Select RTL project, click through “Add Sources”

3. Add the constraints file "Nexys4_Handout_2021.xdc" from labfiles. Do this by locating the "Sources" block and clicking the "+" symbol. Select "Add or create constraints", click Next, Select "Add Files" and find the "Nexys_Handout_2021.xdc" file, click Finish.[2] Make sure you check "Copy constraints files into project". [3]

4. To specify the FPGA, in the SEARCH bar type: **xc7a100tcsg324-1** Click Next and Finish. You should now have an open project in Vivado.

5. In the left menu called "Flow Navigator", click on "Create Block Design" and name your block design **system**[4]

Creating the DDR2 Memory Interface

6. In the Block "Diagram" Click on "+" to add new IP.

7. In the pop-up list, select "Memory Interface Generator (MIG 7 Series") to add this module to the design. You can start typing the first few letters into the search box and the correct choice should quickly come to the top of the list. Double click to select.

8. Double click on the Memory Interface Generator to bring up the settings for the module. For most modules this is a properties window, but for the MIG it is a wizard.

9. Click next through the first page, then choose "Verify Pin Changes and Update Design".[5]

10. For "Load Prj File", choose "mig_200Mhz_64bit.prj" from the lab files. For "Load UCF or XDC File" choose "mig_200Mhz_64bit.xdc".

11. On the next screen, you will be presented with pinout of the RAM (loaded from the XDC file). Press "Validate", you should get a dialong box saying "Current Pinout is Valid" and nothing else. If you get an error, try reading the mig_200MHz_64bit.xdc file again – then validate. Then press next.

12. On the "Sytem Signals Selection page", ensure that "sys_clk_i" is set to Bank 35, pin E3, and that all other signals are set to "No Connect" under Pin.

13. Continue clicking "Next"(and "Accept", etc.) to finish the wizard.

The provided configuration for the MIG tells it that it is connected to a DDR2 memory, and the details of the memory chip on the board. The .xdc file calls out the appropriate pins for the memory. The controller is configured to accept a 100MHz clock (from the crystal oscillator on the PCB), as a reference for its internal signals. In particular

[2] **/labfiles** contains start files:

- Nexys4_Handout_2021.xdc
- mig_200MHz_64bit.prj
- mig_200MHz_64bit.xdc
- Driver for Custom Peripherals

The constraints file (XDC) is used to specify what the FPGA pins are wired to which logical names. Throughout your design, when new peripherals (LEDs, buttons, microphones etc...) are added, the corresponding pins must be uncommented in the constraints file, and the names made to match between the constraints file and the block diagram.

[3] Otherwise when you edit constraints it'll edit the original where you downloaded it, and if 2 projects both do this it leads to errors.

[4] Actually, you can name your block design anything here. I use the name "system" throughout this document.

[5] We are loading a completed design with a prj file, not specifying a new one.

the DDR2 memory is clocked at 200MHz, and ui_clk, the clock the MIG expects to talk to the processor on, is half that at 100MHz. In fact, we will use this ui_clk to clock the processor, since this avoids synchronization overhead between the MIG and the processor. The MIG also needs a clock of exactly 200MHz to calibrate its delay lines. Since it already has a PLL inside it, it lets us use the extra PLL taps it doesn't need for other purposes. Thus, we also create ui_addn_clk_0 at 200MHz using the MIGs internal PLL.

14. Back on the block diagram right click on sys_rst pin of the MIG and select "Make External". Selecting the pin and pressing Ctrl+T has the same effect. This will create a pin going out of your block diagram, wired to the MIG sys_rst pin.
15. Rename the external pin to "btnCpuReset', by selecting the arrow and then editing the name field in the "External Port Properties" pane to the left.
16. Make the "sys_clk_i" pin of the MIG external as well, and rename the external pin to "clock_rtl".
17. If the DDR2 port of MIG hasn't automatically been made external, make it external and if needed rename it from "DDR2_0" to just "DDR2"
18. Connect, by click and dragging, the "ui_addn_clk_0" output of the MIG to "clk_ref_i"

Adding the MicroBlaze processor

19. Click on "Add IP", and add a "Microblaze" module to add to the design.
20. Click on "Run Block Automation". [6]
21. Choose:
    - **Preset**: None
    - **Local Memory**: 128KB
    - **Local Memory ECC**: None
    - **Cache Configuration**: 64KB
    - **Debug Module**: Debug & UART
    - **Peripheral AXI Port**: Enabled
    - **Interrupt Controller** (selected)
    - **Clock Connection**: /mig_7series_0/ui_clk (100MHz)

[6] Vivado displays 'Block' or 'Connection' at different times, depending on the order of your design creation.

After running the block automaton, Vivado will add in addition to the processor several support modules:

- A local memory block. You can set your programs to run in the local memory (faster) if there is space or the much larger DDR memory. Some local memory is needed for the tools to work correctly.
- A "Processor System Reset" block to coordinate the sequencing of reset signals to different parts of the system.
- A "MicroBlaze Debug Module" debugger.
- An "Interrupt controller" to coordinate peripherals interrupting the processor.
- A "Concat" (concatenate) block that takes individual interrupt wires from different sources and passes them as a bus into the interrupt controller.
- An "AXI Interconnect" connecting the processors peripheral bus (M_AXI_DP) to interrupt controller and debug module. As we add more peripherals to the project, they will connect to this interconnect. [7].

[7] AXI (also called AMBA) is ARM's bus standard for their processors. Since the higher end Xilinx FPGAs include hardwired ARM processors, MicroBlaze was also designed to use AXI so the same peripherals could be used with either

It is probably a good idea to click "scale to fit" which is the four arrow symbol third from the left on the Block Diagram header. You can rearrange the hardware placement to make the diagram easier to read by click-dragging the blocks to where you want them. Wires are re-routed when you move the blocks.

Next we connect the processor to the memory:

22. Click on "Run Connection Automation", and check **All Automation**. This will add a second interconnect block, this time called "AXI SmartConnect" that connects the processors two cache fill ports (M_AXI_IC and M_AXI_DC) to the memory interface we generated earlier.,

Next we change some configuration settings on the processor: [8]

[8] Microblaze is a configurable Microprocessor. The options we just selected are sensible defaults for this course, but the processor could be configured many different ways.

23. Open the settings for "Microblaze_0" block by double clicking on it.
24. Leave the first page as is, click next.
25. On page 2 of 5, "General":
    - Check "Enable Barrel Shifter".
    - Change "Enable Integer Multiplier" to "MUL32".

26. On page 3 of 5, "Caches":
    - Change “Data Width” on the left side (Instruction cache) to “Full Cacheline”
    - On the right side (Data cache) check “Enable Write-back Storage Policy”
    - Change "Data Width" on the right side (Data cache) to “Full Cacheline”. This is only possible after changing the storage policy.
27. Click “OK” to close the MicroBlaze settings window.[9]
28. Connect the “Interrupt” pin of the “Microblaze Debug Module” to the “Ino” port of the interrupt “Concat”. This allows the debug module to interupt the processor at need.

[9] These settings create a static memory heavy processor with adequate performance and good real-time properties. If you need the space or want to add more processors or peripherals in your design, you can substantially reduce the local memory and cache sizes.

This is the simplest ‘complete’ version of Microblaze. In theory, you could Generate a Bitstream and program the FPGA right now. If you try to do this though, you will receive Critical Warnings and and Error messages. This is because the Nexys4_master_DDR.xdc file we are using to setup the project expects additional peripherals, and because we have not yet created a top level wrapper. We will be able to **Create HDL Resources** immediately after connecting the interrupt ports.

Adding the “AXI Timer” peripheral in Vivado

29. Open the block design
30. Click on the “Add Ip” menu button
31. Begin to type “Timer” and select the AXI Timer Module
32. Click “Run Connection Automation”, and check **All Automation**. Select “Auto” for **Clock connection (for unconnected nets)**. click OK. This will connect the AXI bus on the timer to the processor by adding an extra port to the interconnect module.
33. Connect the “interrupt” pin of AXI Timer to the “In1” of the interrupt concat.

Adding Peripherals

Adding a second timer

34. Add a second timer IP block to the block diagram and conn

35. Connect the AXI bus on the new timer with the connection automation as above.
36. Double click the “Concat” module and increase the number of ports to 3. You can increase this further as you add even more interrupt sources to the system.
37. Connect the interrupt pin of the new timer to the concat module.

Adding the AXI GPIO peripheral for LEDs (Output)

38. Open the block design
39. Click on the “‘Add Ip” menu button
40. Begin to type “GPIO” and select the AXI GPIO Module
41. Click on the module and rename it to **axi_gpio_led**
42. Click “Run Connection Automation”, and check **“S_AXI”only**. Click OK
43. Double click on the block, make the port GPIO width 16, and check "All Output". Click OK
44. Expand the GPIO port by clicking on the + next to it.
45. Click on the port gpio_io_o[15:0] and select “Make External”. [10]

[10] Name your output pin port and make sure the .xdc file has the pins corresponding to the 16 LED’s uncommented and directed to the name used for your output pin port (“led” in this case). Use the notation {myPinName[0]}

Adding the AXI GPIO peripheral for Btns (Input)

46. Open the block design
47. Click on the “‘Add Ip” menu button
48. Begin to type “GPIO” and select the AXI GPIO Module
49. Click on the module and rename it to **axi_gpio_btn**
50. Click “Run Connection Automation”, and check **“S_AXI”only**. Click OK
51. Double click on the block, make the port GPIO width 5, and check "All Input". Check "Enable Interrupts". Click OK
52. Expand the GPIO port by clicking on the + next to it.
53. Click on the port gpio_io_i[4:0] and select “Make External”. Name your input pin port "btn". [11]
54. Make a new port on the interrupt “Concat” and connect the “ip2intc_irpt” pin to the new “Concat” pin.

[11] Make sure that the name of the input pins match the pin names for buttons in the .xdc file and uncomment corresponding pins in the .xdc file.

Adding a custom peripheral (Example, Lab 1 seven segment display)

55. Use the IP provided in the Lab Files, in the folder "vivado"
56. Click Project Settings in the Flow Navigator.
57. Select "IP" on the left pane.
58. Select the "Repository" tab, click on "+" and browse to the custom IP folder **../labFiles/vivado/seven/ip_repo** Click select
59. Click OK
60. Open the block design, and click on the "Add IP" menu button
61. Begin to type "seven" and select the sevenseg_v_1_0 module
62. Click "Run Connection Automation", and check **"All Automation"**. Click OK
63. Click on the port seg[6:0] and select "Make External"[12]
64. Click on the port an[7:0] and select "Make External"

[12] Make sure to check the .xdc file and uncomment corresponding pins. Noramlly, Vivado adds a part number after each external name – make sure that the names exactly match the .xdc file.

Compiling the Hardware

Create HDL resources

65. Above the block, diagram, switch to the "Address Editor" tab. Right click on "microblaze_0" and choose "Auto Assign Addresses". Note that this option may be greyed out if vivado has already assigned addresses to everything fully automaticaly.
66. Right click on "system" in the "Sources" window (top of the middle-left column). Click "Create HDL Wrapper". In the dialog box, choose "Let Vivado manage wrapper and auto-update."
67. In the "Flow Navigator" pane on the left, choose "Generate Block Design".

Generate Bitstream

68. Open the block design, right-click and select "Validate Design". This checks for possible errors or critical warnings
69. In the Flow Navigator, click on "Generate Bitstream". Vivado may ask if it can resynthesize the design and run implementation. Click yes, and wait. It can take up to 30 minutes to generate a bitstream.[13]

[13] Errors might occur during the generation of the bitstream. The most common reason for errors and critical warning's comes from the constraints file, nexys4_master_DDR.xdc. To fix errors, open the constraints file in Vivado (under Sources box) and check the line number given in the error message. Make sure the name of the pin matches the block diagram pin name.

Exporting your hardware design to the software SDK

70. Open the implemented design and the block design[14]
71. Click on File : Export : Export Hardware
72. Leave the default destination and check **Include Bitstream**.
73. Click on File : Launch SDK, OK.

[14] Failing to have the implemented design and block design open in Vivado may cause the bitstream to not be exported.

Starting a New project in SDK

74. Click on "File", "New", "Application Project"
75. Name your project.
76. Ensure that hardware platform is set to "system_wrapper_hw_platform_0" (where system is the name of your block diagram) and processor is set to "microblaze_0". In case these options are not listed in the drop-down menu, click on the "New.." button next to Hardware Platform, for the Target Hardware Specification, browse to select the .hdf file for your project.
77. For your first project, choose 'Create New' for the Platform. For subsequent projects, you can create separate Platforms for each or re-use the first one you created. Press Next.
78. When you get to the Template choice for your first project, select "Hello World" from the choices and press "Finish".

A project in Xilinx SDK has 3 parts. Hardware exported from Vivado shows up as a "Hardware Platform Specification". A "Board Support Package" is a set of drivers and specifications (like compiler flags and software library selections) for a particular hardware platform. The application project (the software you write) is linked to the board support package. [15]

79. Open the BSP (inside the system wrapper) and then the "system.mss" file. You should get a summary of what drivers and libraries are in the BSP. Click on "Modify this BSP's settings" and click on "drivers" in the left column.
80. In the rwo for sevenSeg select 'generic'. Click OK.
81. In the Project Explorer, your project folder (named same as the project) contains a **"src"** folder which has all the .c and .h files for the project.

[15] If, in Vivado, you are re-exporting a previously exported hardware, close SDK first. You want the current hardware platform to be updated with the new bitstream and related data, so that the existing BSP will in turn be updated. If SDK is open when exporting, a new hardware platform is created instead. You will then have to create a new board support package and update your applications references to run on the new hardware. The bitstream file is loadable hardware configuration that added to the compiled elf binary executable that is downloaded to the FPGA.

82. Select your project, and from the top menu choose Xilinx and then "Generate Linker Script". For each of the "Place [something] In" options ensure the "mig_7series" option is selected instead of "microblaze_0_local_memory" option. This will ensure your program is compiled into DDR instead of the local memory, since the DDR is much larger. Set the size of the heap to 4096 (4k) and that of the stack to 4096 (4k) as well. [16]

[16] What the different sections of the program are will be discussed later in the class

83. If the project was created using many of the Xilinx templates, it will have an"init_platform();" function call at the top of "main()". If so, delete this function call.

84. To enable caches, insert the following line in the includes at the top of the program (if not already there):

```
#include <xil_cache.h>
```

and insert the following 4 lines of code at the top of the"main()" function:

```
Xil_ICacheInvalidate();
Xil_ICacheEnable();
Xil_DCacheInvalidate();
Xil_DCacheEnable();
```

Running your program in the DDR2 with caches enabled gives you a very large storage space and good average time performance, though at the cost of performance. You can run your program without caches by omitting these lines of code however, DDR access is surprisingly slower than internal memory. You can also generate a linker script selecting local memory to load your program into the local memeory inside the FPGA, which is the fastest possible but limited in size (though if you are careful with the programs in this class, they should all still fit). If you do this you can still choose whether or not to enable caches for any parts of the program you might put in DDR.

Running your Project

Connect the Artix7 board to the USB port of the Computer. Make sure the Power jumper is connected to USB and make sure the JTAG is enabled by connecting the two center pins in the mode selector.

85. Turn on the board with the switch

86. Back in the SDK, click on "Xilinx Tools:Program FPGA"

87. Once the board is programmed right click on your project folder and select "Run As: Launch on Hardware (System Debugger)"

88. The console should now display "Hello World"

Starting a Lab: Using .c and .h files in xSDK

When labs are assigned, often some starting code will be provided. To use this, first create a new xSDK project as above. In the project explorer, expand the /src folder and drag the provided source files into it.

After creating a BSP for your Vivado bitstream, create a new project in the xSDK. A Test peripheral project is good because it generates sample code showing how the peripherals are set up.

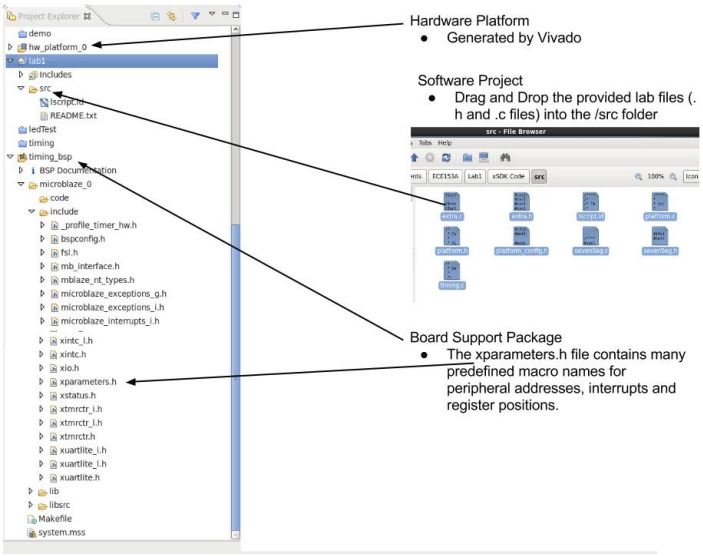


Figure 1: The xSDK has a UI similar to Eclipse. The project explorer is used to organize your source code.

Using the Viavdo Hardware debugger

As well as various software level debug options, Vivado allows you to add a logic analyser into the FPGA to facilitate debugging at the level of hardware signals. This is instructive in understanding how the system works, and is invaluable if you choose to write your own verilog for your project.

89. Open the block design
90. Right click on a wire and choose “Mark Debug”
91. In the Flow Navigator click “Run Synthesis”
92. After synthesis completes, open the synthesised design

93. Select Debug in the main toolbar of the program.
94. Click “Set Up Debug”, and click “Next” until finished.

Generate a bitstream and Export your design to the SDK as usual. While the SDK is running your software, you are able to monitor signals from Vivado.

Observing your Debug signals

95. In the Flow Navigator, under “Program and Debug”, click the triangle next to “Hardware Manager” to expand the dropdown.
96. Click on “Open Target”. There should be a link to localhost... Select that as the target.
97. Add a signal from Debug probes onto Basic Trigger Setup.
98. Select a a trigger value from “Compare Value” dropdown.
99. Click the “Run Trigger for ILA core” play button.
100. When your signal matches the compare it will record a waveform, and switch to displaying the waveform.

Understanding the size of your design

In the Flow Navigator, Click on “Open Implemented Design” to generate reports on your design. This is only available after generating a bitsream.

Resource	Estimation	Available	Utilization %
FF	5494.0	126800.0	4
LUT	1629.0	63400.0	3
Memory LUT	182.0	19000.0	1
I/O	3.0	210.0	1
BRAM	2.0	135.0	1
BUFG	4.0	32.0	13
MMCM	1.0	6.0	17

Figure 2: The Utilization report table in the “Project Summary” for a **default** Microblaze Configuration. To generate a utilization report once synthesis is complete open Project Summary.

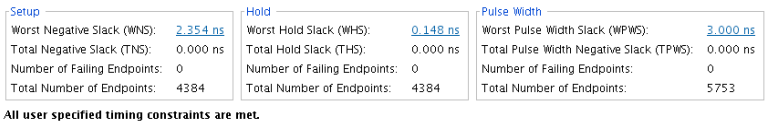


Figure 3: The Timing Report Summary. To generate a timing report, expand the drop down for “Synthesized Design” and click on “Report Timing Summary”