Arty MicroBlaze Soft Processing System Implementation Tutorial

Daniel Wimberly, Sean Coss

***Abstract*—A Microblaze soft processing system was set up and then uploaded to a Arty Artix-7 FPGA Evaluation board using the Xilinx Vivado software. Once this soft processor was created using this software, C code was used to program the Microblaze processor. The program that was created was set up such that it would read from built in switches and buttons and use these inputs to output to the built in LEDs and to output to a terminal via serial communication. The following document outlines the process that was taken to set up the soft processor, write it to the Artix-7 board and then program it using the provided SDK that comes with the Vivado software.**

I. Arty Board and MicroBlaze

The board used for the project was the Arty board which is a development board built around the Artix-7 FPGA. This was developed by Xilinx for use with the MicroBlaze soft processor which is an HDL defined processor which can be written to the Artix-7 FPGA. The board is shown in Figure 1. The evaluation board provides connectivity such as switches, buttons, LEDs, RGB LEDs, Pmod connectors, shield connectors, USB, and Ethernet to work with HDL components and those defined through the MicroBlaze.

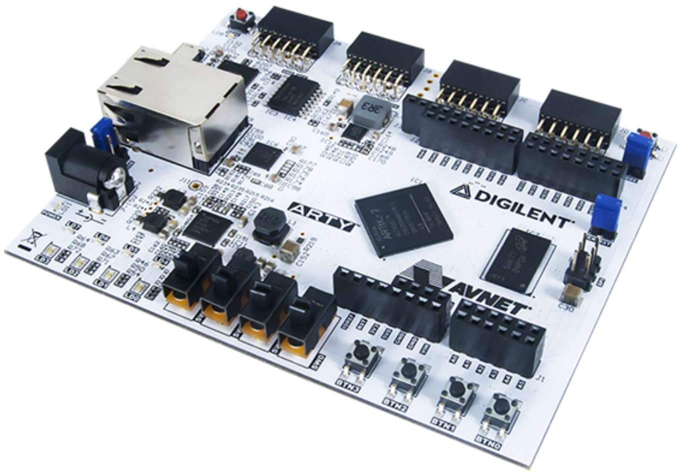
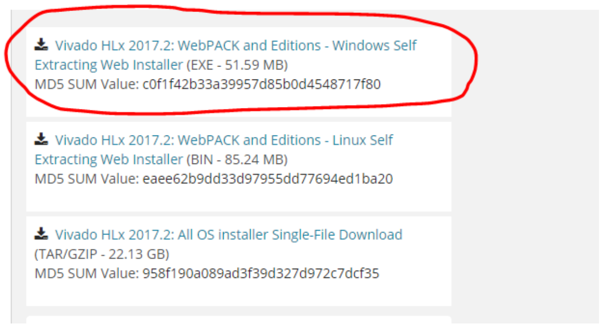


Fig. 1. Arty Board

II. Installing Vivado

The first step in the implementation of this project was to install the Vivado software by downloading from [1]. In order to install this software an Xilinx account had to first be created. Once the account was created, the download version was selected. The version that was used for this project and is suggested to install is the 2017.2 WebPack Windows self extracting web installer shown in Figure 2.



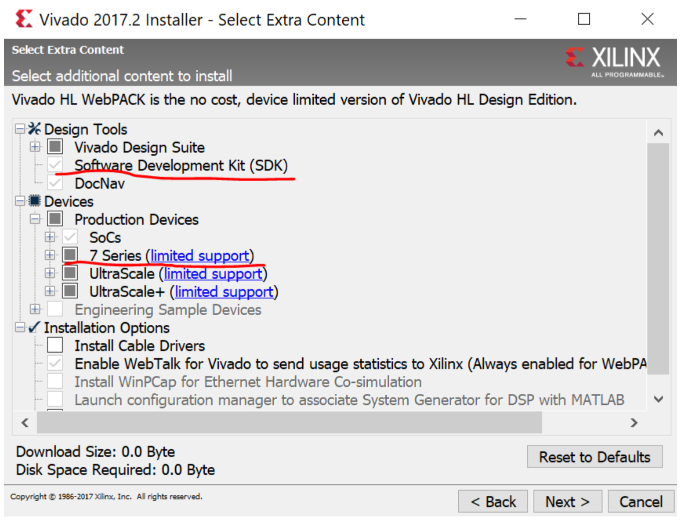
Vivado HLx 2017.2: WebPACK and Editions - Windows Self Extracting Web Installer (EXE - 51.59 MB)
MD5 SUM Value: c0f1f42b33a39957d85b0d4548717f80

Vivado HLx 2017.2: WebPACK and Editions - Linux Self Extracting Web Installer (BIN - 85.24 MB)
MD5 SUM Value: eaee62b9dd33d97955dd77694ed1ba20

Vivado HLx 2017.2: All OS installer Single-File Download (TAR/GZIP - 22.13 GB)
MD5 SUM Value: 958f190a089ad3f39d327d972c7dcf35

Fig. 2. Install Version

Once this was downloaded, the installer was run, following all of the standard install instructions. Note that the user name and password for the Xilinx account that was created were needed for the install. When the step where the user chose which packages to install was reached, the SDK and 7 Seriecs device options were checked, as these were required for the project. These are shown in Figure 3.



Vivado 2017.2 Installer - Select Extra Content

Select Extra Content
Select additional content to install

Vivado HL WebPACK is the no cost, device limited version of Vivado HL Design Edition.

- Design Tools
  - Vivado Design Suite
  - ✓ Software Development Kit (SDK)
  - ✓ DocNav
- Devices
  - Production Devices
    - ✓ SoCs
    - 7 Series (limited support)
    - UltraScale (limited support)
    - UltraScale+ (limited support)
  - Engineering Sample Devices
- ✓ Installation Options
  - Install Cable Drivers
  - ✓ Enable WebTalk for Vivado to send usage statistics to Xilinx (Always enabled for WebPA
  - Install WinPCap for Ethernet Hardware Co-simulation
  - Launch configuration manager to associate System Generator for DSP with MATLAB

Download Size: 0.0 Byte
Disk Space Required: 0.0 Byte

Copyright © 1986-2017 Xilinx, Inc. All rights reserved.

Fig. 3. Install Packages

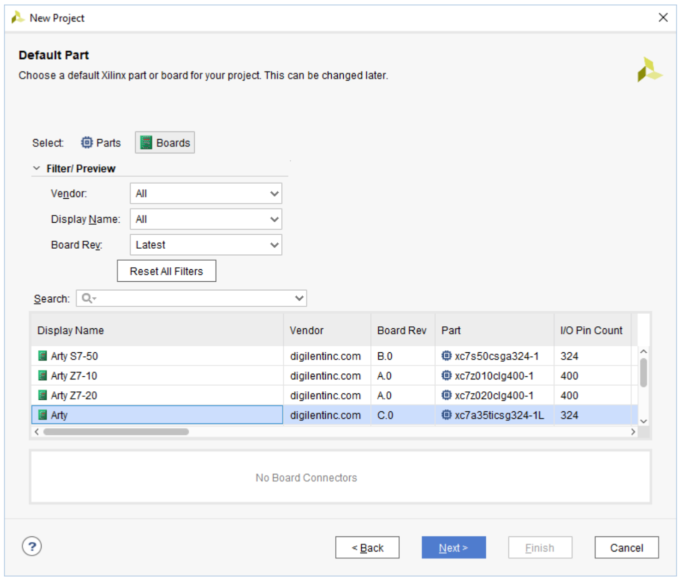
After this step the standard install options were chosen and the install wizard was followed until the program was successfully installed.

III. MICROBLAZE IMPLEMENTATION

The process for project implementation is highlighted below.

- Open Vivado SDK
- Create new project
- Create block diagram
- Populate block diagram with necessary ports, blocks, and IPs
- Wire block diagram as needed
- Ensure Settings and names for all blocks are as desired
- Run design validation
- Generate bitstream
- Export bitstream
- Open SDK
- Create SDK project
- Write Microblaze code
- Compile code
- Write bitstream to board
- Write code to board and run program

The first step of the project implementation was to open Vivado 2017.2 and to create a new project. A directory and filename without spaces was used, and it was set up as an RTL project with the "Do not specify sources at this time" box checked. After clicking "Next", the Arty board should be selected by selecting the "Boards" icon, and selecting the "Arty" board, as shown in Figure 4. Once the project was created, a new block design was created, by clicking the "Create Block Design" under the "IP INTEGRATOR" pane Figure 30. The block diagram was opened, and the process of populating the diagram was begun.



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Vendor: All

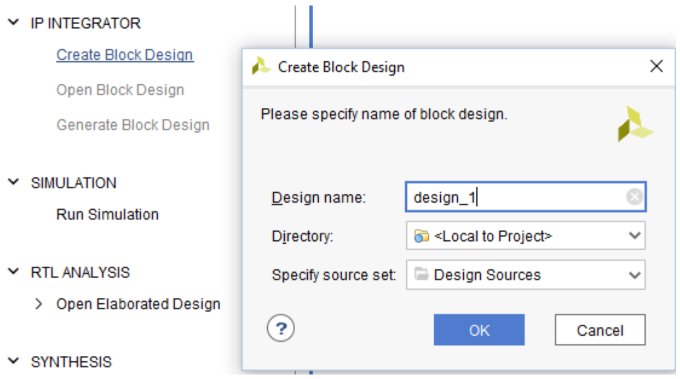
Display Name: All

Board Rev: Latest

Display Name	Vendor	Board Rev	Part	I/O Pin Count
Arty S7-50	digilentinc.com	B.0	xc7s50csga324-1	324
Arty Z7-10	digilentinc.com	A.0	xc7z010clg400-1	400
Arty Z7-20	digilentinc.com	A.0	xc7z020clg400-1	400
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324

No Board Connectors

Fig. 4. Select Arty Board



Please specify name of block design.

Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

Fig. 5. Create Block Design In New Project

A system clock block was added by clicking on the "Board" pane, and dragging the "System Clock" block into the block diagram Figure 6. The block is shown by Figure 7. The clock block settings were set as shown in Figure 8 and Figure 9 to have 3 clock outputs with an active low reset.

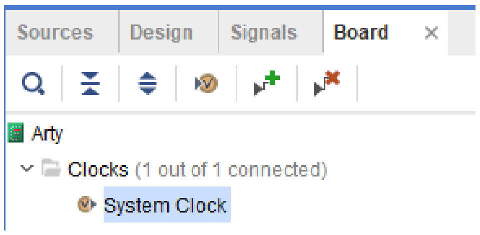


Fig. 6. Add Clock Block

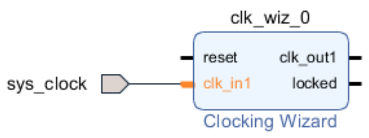
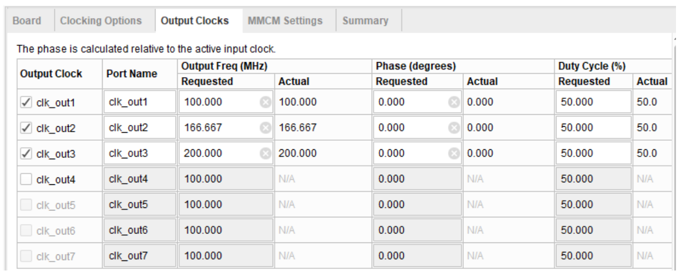


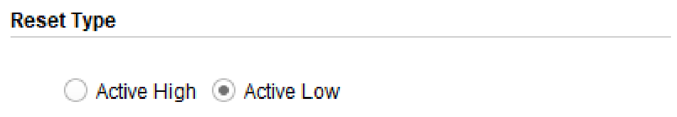
Fig. 7. Clock Block



The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
		Requested	Actual	Requested	Actual	Requested	Actual
✓ clk_out1	clk_out1	100.000	100.000	0.000	0.000	50.000	50.0
✓ clk_out2	clk_out2	166.667	166.667	0.000	0.000	50.000	50.0
✓ clk_out3	clk_out3	200.000	200.000	0.000	0.000	50.000	50.0
clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A
clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A
clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A
clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A

Fig. 8. Clock Settings

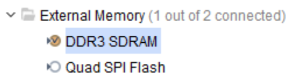


Reset Type

Active High [✓ Active Low]

Fig. 9. Clock Settings Ctd.

Next, the DDR3 SDRAM external memory interface was added. This is shown in Figure 10. When added, the software automatically added the memory interface generator block with the memory interface, as shown in Figure 11.



External Memory (1 out of 2 connected)
- DDR3 SDRAM
- Quad SPI Flash

Fig. 10. DDR3 Menu Item

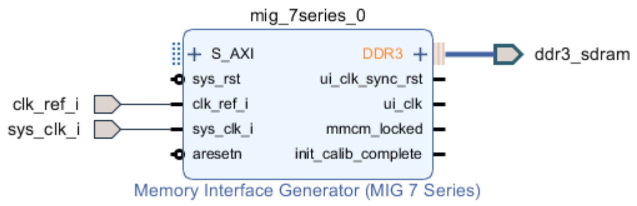


Fig. 11. DDR3 Interface With MIG Block

The system reset was added, and the blocks were connected as shown by Figure 12.

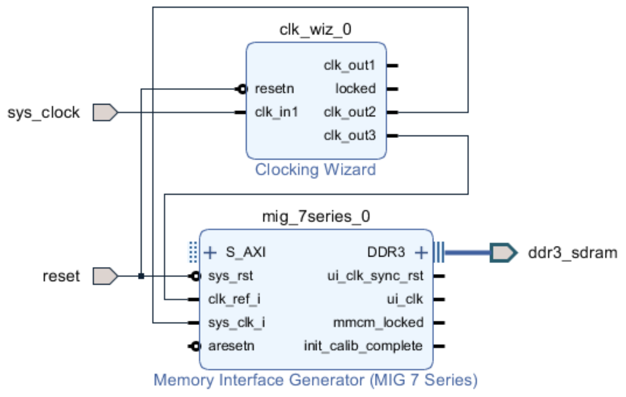
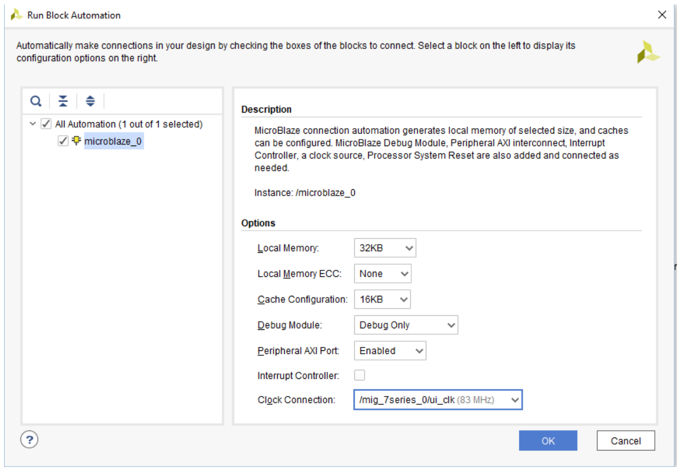


Fig. 12. MIG Block With Clock Block

The Microblaze IP was added next. To add this item, the ”+” icon shown in Figure 14 was clicked and the Microblaze IP was chosen. Next, the ”Run Block Automation” designer assistance, shown shaded in green at the top of the block diagram, was used. The settings for local memory, cache configuration, and clock connection were changed to reflect Figure 13. After this step was complete, the block diagram looked like Figure 15.



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

- ✓ All Automation (1 out of 1 selected)
  - ✓ microblaze_0

Description

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

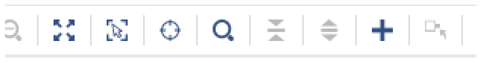
Instance: /microblaze_0

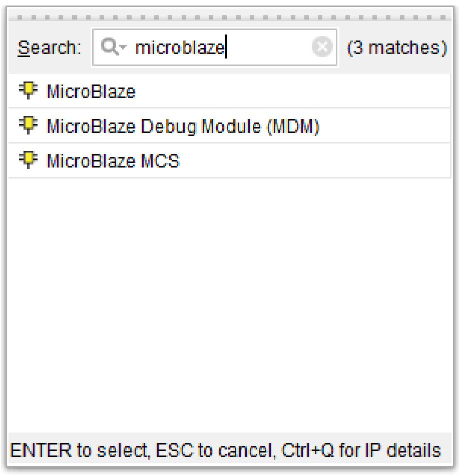
Options

Local Memory:	32KB
Local Memory ECC:	None
Cache Configuration:	16KB
Debug Module:	Debug Only
Peripheral AXI Port:	Enabled
Interrupt Controller:	☐
Clock Connection:	/mig_7series_0/ui_clk (83 MHz)

OK Cancel

Fig. 13. Microblaze Block Automation Settings





Search: microblaze (3 matches)

- MicroBlaze
- MicroBlaze Debug Module (MDM)
- MicroBlaze MCS

ENTER to select, ESC to cancel, Ctrl+Q for IP details

Fig. 14. Add Microblaze IP

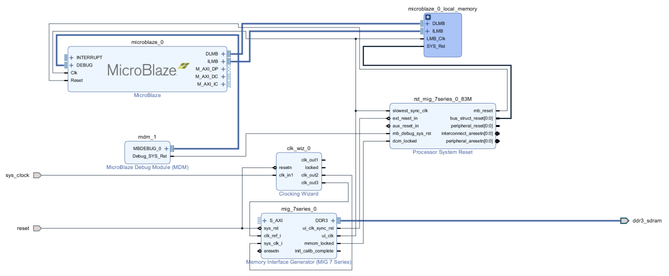


Fig. 15. Microblaze Block Automation Result

Up to this point, the specific wire connections in the block diagram would be largely changed, so little attention was paid to ensuring the wiring was configured in this manner. The GPIO ports shown in Figure 16 and the UART port were added next. The Arty board automatically configures the LED GPIO ports to share one AXI GPIO block, while the buttons and switches share another. After addition of these pins, the result is shown by Figure 17. The blocks were then renamed to have more intuitive names, as shown by Figure 18. The UART port also added a GPIO block, but this block was not renamed.

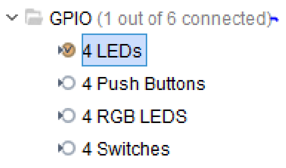


Fig. 16. Adding GPIO Ports

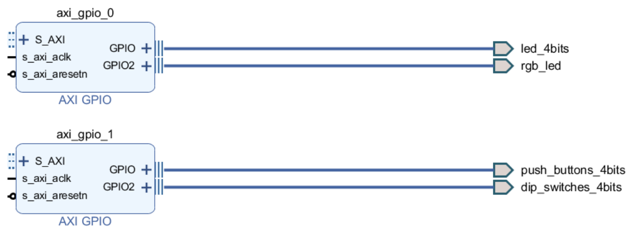


Fig. 17. Added LED and Button Ports

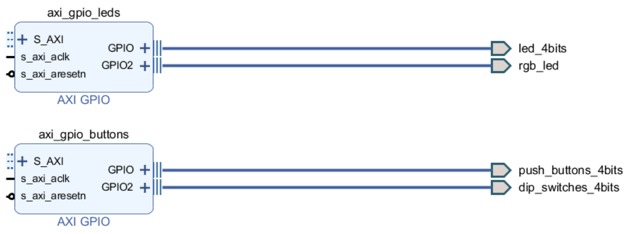


Fig. 18. Renamed GPIO Blocks

At this point, connection automation was run on the GPIO ports to add all of the remaining necessary blocks to the diagram, but the connections made were not (necessarily) used. Blocks were renamed and connected as shown by Figure 19 (see Appendix for larger image). Note that some items were deleted, and that some were duplicated and/or renamed to achieve the result shown.

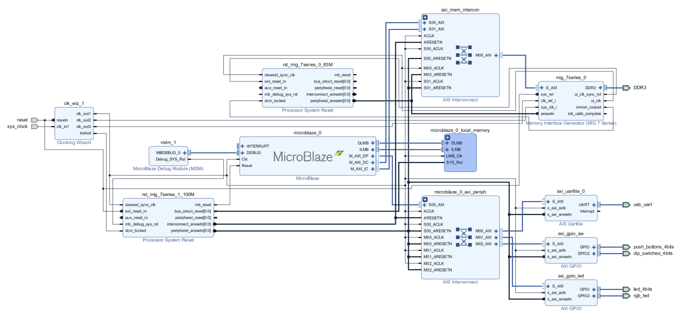
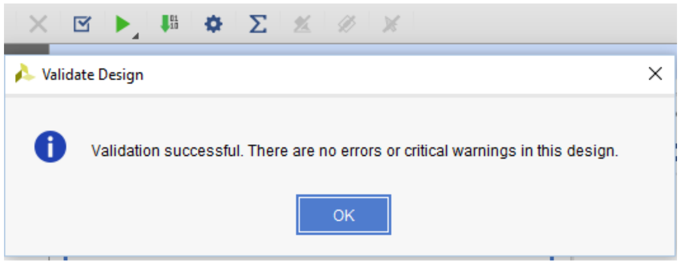


Fig. 19. Final Block Diagram Layout

The next step was to validate the design by clicking the checkbox icon shown in Figure 20. After completion, a message was given confirming that no connection errors existed.



Validation successful. There are no errors or critical warnings in this design.

Fig. 20. Block Diagram Validation

After this, an HDL wrapper was created by navigating to ”Sources”, right-clicking on the block diagram design, and clicking ”Create HDL Wrapper” Figure 21. Note that whenever the block diagram is changed, the current HDL wrapper should be deleted and a new one created.

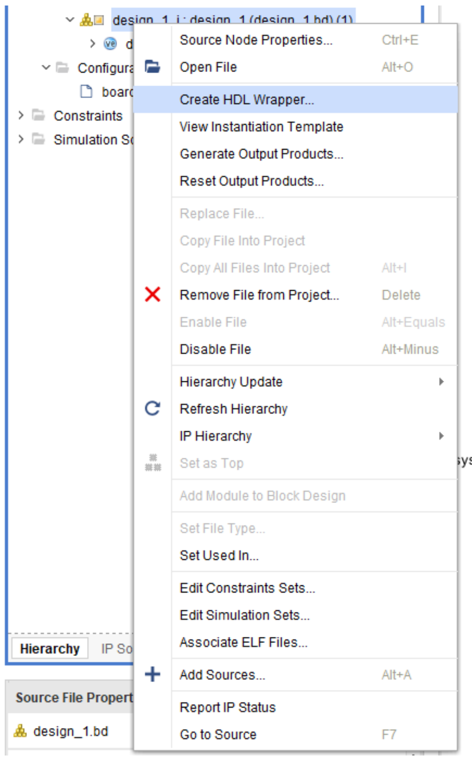
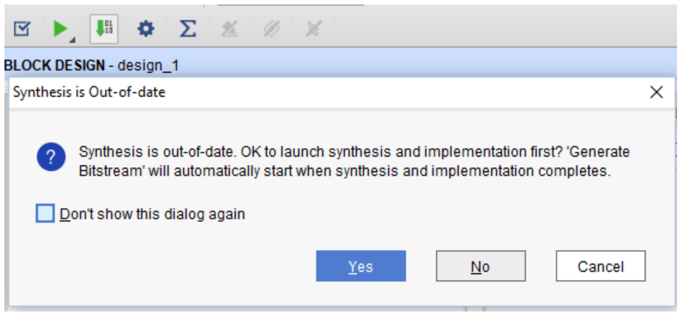


Fig. 21. HDL Wrapper Creation

The design must then be synthesized and a bitstream generated. To do this, the ”Generate Bitstream” icon shown by Figure 22 was clicked, and the ”Yes” box selected. Default settings in the following window were selected Figure 23. For a short time, a status bar showed, then the synthesis and implementation process followed. Its status could be seen by looking at the status indicator at the top right of the Vivado software, next to the exit button Figure 24.

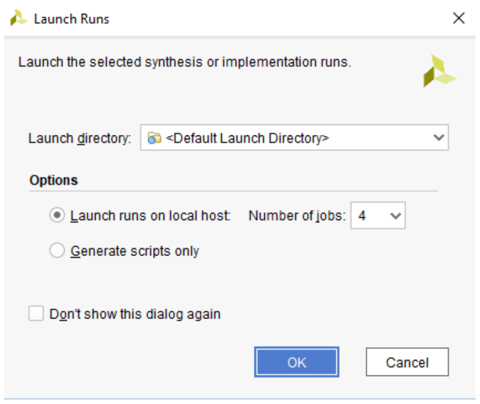

BLOCK DESIGN - design_1

Synthesis is Out-of-date

Synthesis is out-of-date. OK to launch synthesis and implementation first? 'Generate Bitstream' will automatically start when synthesis and implementation completes.

Don't show this dialog again

Fig. 22. Generate Bitstream Icon


Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

Launch runs on local host Number of jobs: 4

Generate scripts only

Don't show this dialog again

Fig. 23. Generate Bitstream Icon

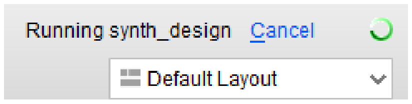
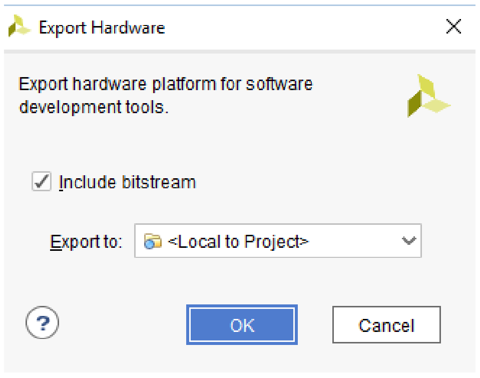

Running synth_design Cancel

Fig. 24. Bitstream Generation Status

When the bitstream generation is complete, a message box popped up asking whether the user would like to see the implemented design, view reports, open hardware manager, or generate memory configuration file. Since none of these options were relevant, ”Cancel” was selected.

The next step was to export hardware. This was done by selecting ”File - Export - Export Hardware,” checking the ”Include Bitstream” box, and clicking ”OK” Figure 25. If an overwrite dialogue box came up, the choice to overwrite the previous bitstream file was selected.


Export Hardware

Export hardware platform for software development tools.

Include bitstream

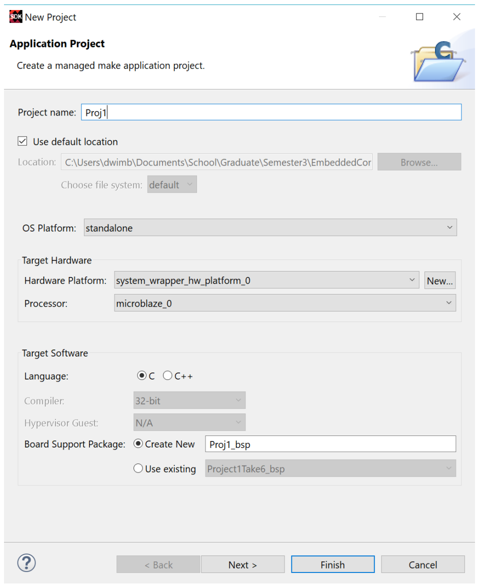
Export to: <Local to Project>

Fig. 25. Export Hardware Step

Lastly, the SDK was opened by selecting ”File - Launch SDK”. The SDK allows the user to write the bitstream to the Arty and to write programs for the Microblaze soft processor.

IV. Programming with SDK

When the SDK is launched it should open up a new window. Once this new window is up a new project was created by going to ”File - New Application Project”. A window will pop up in which the project can be named. The rest of information should be auto-populated as shown Figure 26. After the project was named, ”Next” was selected and then the ”Hello World” Template was chosen Figure 27.


New Project

Application Project

Create a managed make application project.

Project name: Proj1

☑ Use default location

Location: C:\Users\dwimb\Documents\School\Graduate\Semester3\EmbeddedCor

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: system_wrapper_hw_platform_0

Processor: microblaze_0

Target Software

Language: [✓ C] C++

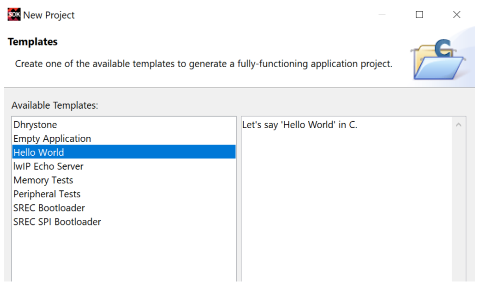
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] Proj1_bsp

Use existing Project1Take6_bsp

Fig. 26. Name SDK Project


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
Peripheral Tests
SREC Bootloader
SREC SPI Bootloader

Let's say 'Hello World' in C.

Fig. 27. Select Hello World Template

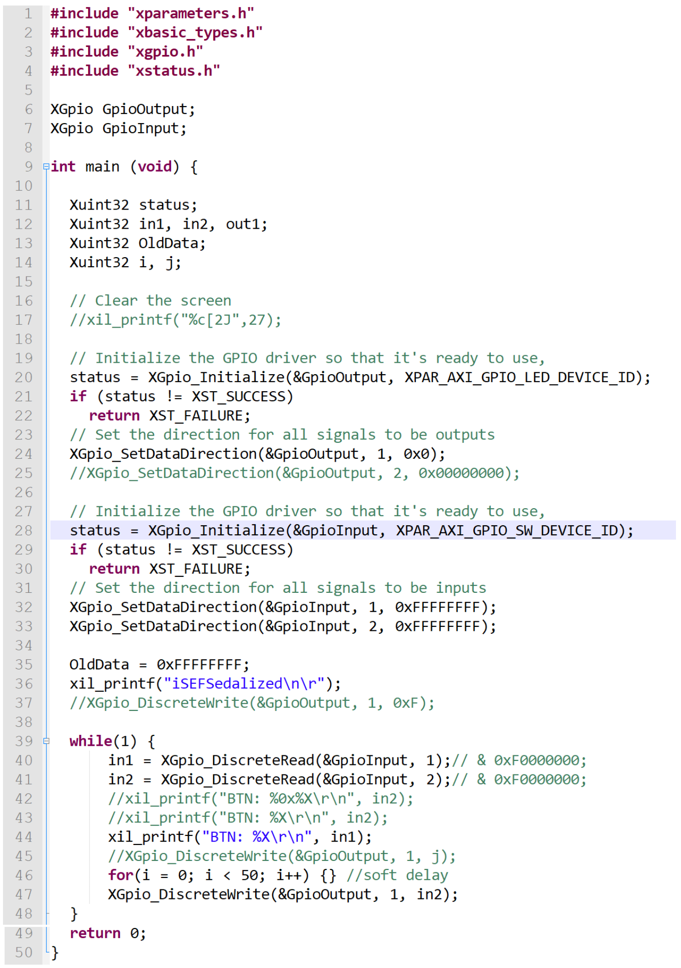
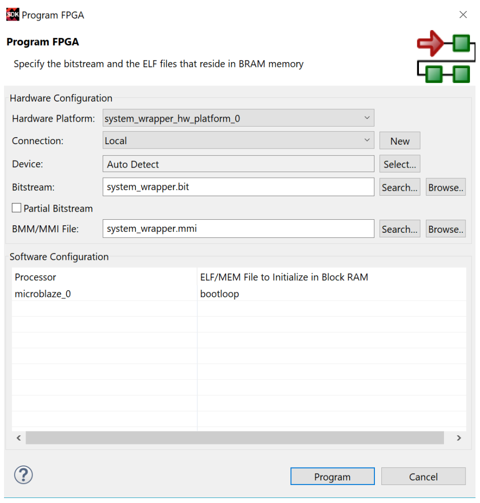

```c
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xstatus.h"

XGpio GpioOutput;
XGpio GpioInput;

int main (void) {

  Xuint32 status;
  Xuint32 in1, in2, out1;
  Xuint32 OldData;
  Xuint32 i, j;

  // Clear the screen
  //xil_printf("%c[2J",27);

  // Initialize the GPIO driver so that it's ready to use,
  status = XGpio_Initialize(&GpioOutput, XPAR_AXI_GPIO_LED_DEVICE_ID);
  if (status != XST_SUCCESS)
    return XST_FAILURE;
  // Set the direction for all signals to be outputs
  XGpio_SetDataDirection(&GpioOutput, 1, 0x0);
  //XGpio_SetDataDirection(&GpioOutput, 2, 0x00000000);

  // Initialize the GPIO driver so that it's ready to use,
  status = XGpio_Initialize(&GpioInput, XPAR_AXI_GPIO_SW_DEVICE_ID);
  if (status != XST_SUCCESS)
    return XST_FAILURE;
  // Set the direction for all signals to be inputs
  XGpio_SetDataDirection(&GpioInput, 1, 0xFFFFFFFF);
  XGpio_SetDataDirection(&GpioInput, 2, 0xFFFFFFFF);

  OldData = 0xFFFFFFFF;
  xil_printf("iSEFSedalized\n\r");
  //XGpio_DiscreteWrite(&GpioOutput, 1, 0xF);

  while(1) {
      in1 = XGpio_DiscreteRead(&GpioInput, 1);// & 0xF0000000;
      in2 = XGpio_DiscreteRead(&GpioInput, 2);// & 0xF0000000;
      //xil_printf("BTN: %0x%X\r\n", in2);
      //xil_printf("BTN: %X\r\n", in2);
      xil_printf("BTN: %X\r\n", in1);
      //XGpio_DiscreteWrite(&GpioOutput, 1, j);
      for(i = 0; i < 50; i++) {} //soft delay
      XGpio_DiscreteWrite(&GpioOutput, 1, in2);
  }
  return 0;
}
```

Fig. 28. C-Code To Implement With MicroBlaze

Once this is built, the ”src” file under the project explorer was opened and ”helloworld.c” was opened since this was the main file for the SDK C code to be written to the device. The code originally contained in this file was then replaced with the code shown in Figure 28. This code is set up to read from the development board’s buttons and switches and then write to the LEDs. To do this the Gpio peripherals are initialized and then their directions were set. Then inside of a loop, the values of the switches and buttons were read. The value of the switches were written to the LED outputs. The values of the buttons were displayed by writing to the serial port.

Once the code had been written, it was saved which will automatically compile and error check the code. After the code has compiled it can then be written to the FPGA. Before this step was done, the Arty board was plugged into the computer. In the quick selection tool bar the Program FPGA button was clicked which opens the window for programming the FPGA shown in Figure 29. The settings were ensured to match that shown in Figure 29. After this ”Program” was selected which programmed the FPGA.


Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform: system_wrapper_hw_platform_0

Connection: Local

Device: Auto Detect

Bitstream: system_wrapper.bit

Partial Bitstream

BMM/MMI File: system_wrapper.mmi

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM
microblaze_0	bootloop

Fig. 29. Program The FPGA

Once the FPGA had been programmed the program needed to be ran. This was done by selecting the drop down menu next to the green run button and then selecting ”Run as - Launch on Hardware (System Debugger)”. This will run the program on the FPGA which can be tested by flipping the switches on the board and seeing that the corresponding LEDs light up. The serial output can be tested by opening a serial terminal such as Putty, and properly connecting to the COM port that the board is connected through.

V. Conclusion

The implementation in this report configured the Arty FPGA evaluation board to use several of the built-in LEDs and switches to perform basic actions using a microcontroller-like C programming environment. While the ability to program a microcontroller using C or similar languages is useful in its own right, the Arty board provides much more versatility because it can be programmed to have custom logic gate-based functionality that does not necessarily rely on the built-in interrupt or loop-based structure of the Microblaze microcontroller, meaning that some actions can be completed much more quickly and/or reliably than if one were to rely on the base microcontroller alone. In addition, both the FPGA and microcontroller are effectively integrated into a single chip, meaning the complexity of wiring both devices together is completely eliminated. This advantage over conventional solutions can save large amounts of time when designing and testing early prototypes.

References

[1] Vivado Installer
https://www.xilinx.com/support/download.html

APPENDIX A
FINAL BLOCK DIAGRAM

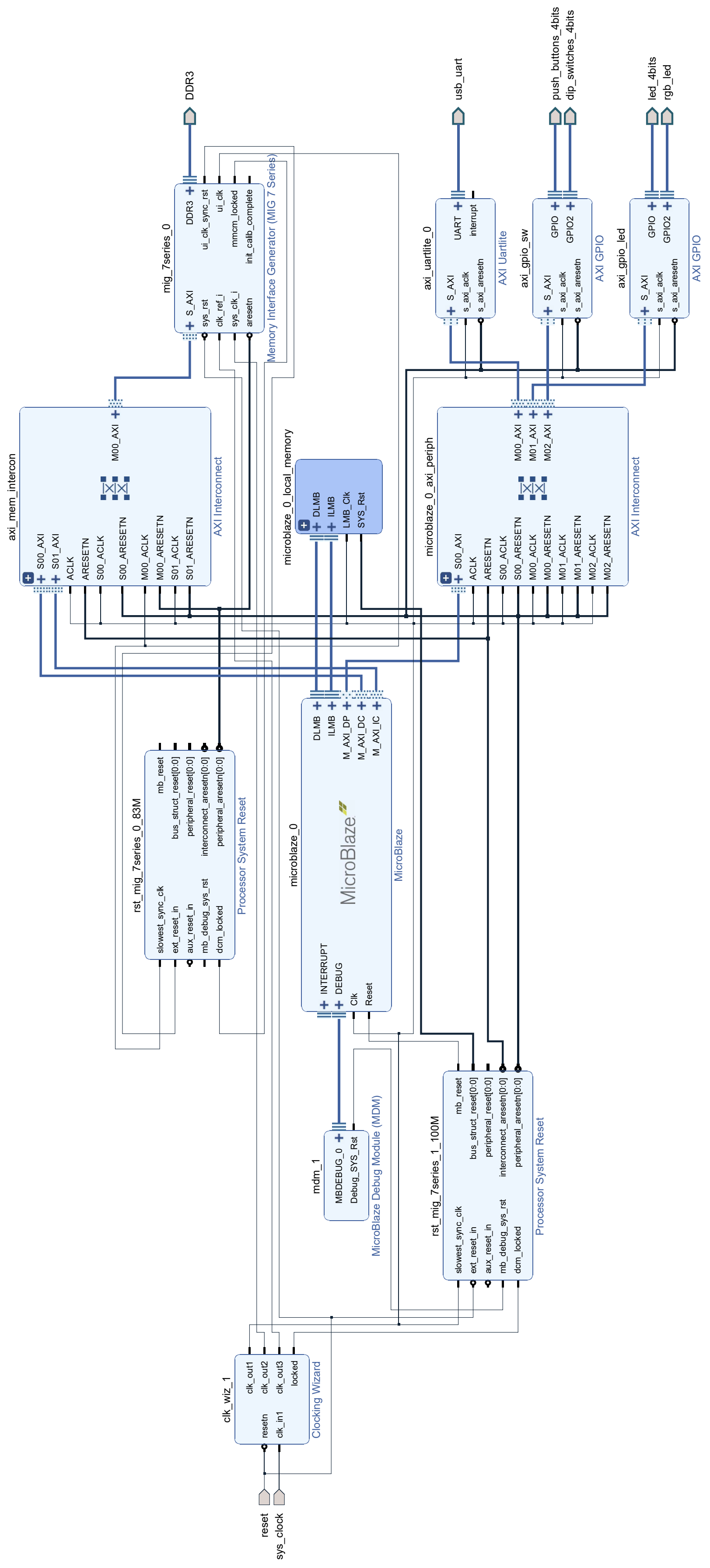


Fig. 30. Enlarged Block Diagram