University of Colorado Boulder

PSOC PROJECT 3

Report

Date: 5/3/2017

Team:

Jeet Baru
jeet.baru@colorado.edu
107189037

Sudeep Kulkarniirlekar
suku8854@colorado.edu
106996755

Executive Summary

The DE1-SoC, FPGAs integrate an ARM-based hard processor system (HPS) consisting of processor, peripherals and memory interfaces, using a high-bandwidth interconnect backbone. With this in mind, 1st and 3rd modules were completed that explored the various functionalities of Altera DE1-SoC.

PYNQ is an open-source project from Xilinx to design embedded systems with Zynq All Programmable Systems on Chips. Using Python language and libraries, designers can exploit the programmable logic and microprocessors in Zynq. With this taken into consideration, the module 2 was completed that explored various functionalities of PYNQ.

In 1st module we are required to program the FPGA fabric for various designs. In Lab 1 we program various circuits involving MUXs and learn to connect various IO peripherals to the FPGA including switches, pushbuttons, LEDs and 7 segment displays. In Lab 5 we initially start building from a counter and finally build a Real Time clock and Morse Code encoder.

In module 2 we appreciated the Ethernet capabilities of the PYNQ board. This module required us to use the different functionalities of Jupyter notebook, like using terminal, integrate the hardware with the Python 3.

In module 3 we learn to integrate a Hard Processor System with our FPGA fabric. Write C code which shall execute on linux in HPS to control LEDs connected to the FPGA fabric.

By the independent module (module 4) we aim at applying all skill garnered through the course in Programmable Systems on a Chip on a small project designed completely by us. We decided to implement a software application on HPS that utilizes FPGA resources for enhanced functionality. The application software we implemented is a menu driven program for multiple games. We implemented snakes, minesweeper and tic-tac-toe. Every special event in the game, like score, player turns, winner, loser, game over etc. shall be reflected on the displays on-board.

University of Colorado Boulder

Module 1

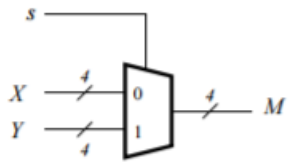
Objective:

- To become familiar with a large SoC device like De1 SoC and appreciate its capabilities in terms of hardware peripherals and huge configurable fabric.
- To develop several hardware and software designs of circuits and timing systems and implement them on the De1-SoC board.
- To learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices like LEDs, 7-segment displays, buttons and switches.
- To study the use of clocks in timed circuits. Also to design and implement such timed circuits like Morse code encoder and Real time clock.

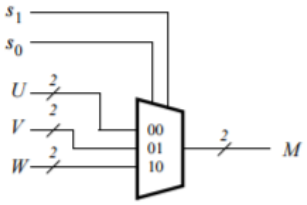
Procedure:

Lab 1:

- In part 1, we first connect each switch to its respective LED. Design is such that on switching respective LED shall turn on or off. We write the code in Verilog and use pin planner for pin assignments.
- In part 2, we implement a mux that shall be used to select the output. If 1, the LEDs shall glow corresponding to SW[7:4] and if 0, LEDs shall glow corresponding to SW[3:0]. The circuit implemented is as shown below:



- In part 3, we implement a 3 to 1 MUX. Now the select line is 2 bits wide. The output is 2 bits and corresponds to 1 of the 3, 2-bit inputs. The circuit implemented is as shown below:



- In part 4, we write a Verilog code to display on the first 7-segment display depending on the input in SW[1:0]. We display D, E or 1 depending on the input.
- In part 5 we use our code from the previous part and now display on 3 7-segment displays. Also we write code to swap places of D, E and 1 displayed on the 3 displays depending on input from SW[9:8].
- In part 6, we extend our design from the previous part so that we use all 7-segment displays we write code such that on changing the select line input we change the position of the “DE1” word on the display.

Lab 5:

- We write a modulo-k counter module in Verilog. We implement k=20 modulo counter with pushbutton 0 as asynchronous reset and pushbutton 1 as clock input. We display the output on the LEDs and configure one of the LEDs to display the rollover signal.
- In part 2, we use our module in part 1 and design a 3 digit BCD counter. For the BCD counter we set the value of k at 10. We also trigger the nth place digit only when the n-1th place digit generates a rollover signal. We display each digit on 7-segment displays. To make the counter increment at 1 sec interval we use a counter that counts 50000000 clock edges of the 50 MHz clock. We use the push button 0 to reset the counter to 0.
- In part 3, we use out counter module to create a real time clock. Here we have 2 digits that count at 100^{th} and 10^{th} of a second, further we have 2 digit seconds counter that counts till 60 and lastly we have a minutes counter that counts till 60 minutes. To achieve this, we implemented 6 counters for each digit. Each counter counts till 10 except tens place digit of seconds and minutes. We instantiated all the counters reusing the available module. Also, we configured KEY2 to load the minutes place by value on switches SW[7:0]. Lastly the clock stops when KEY1 is pressed and continues when released. Also we have a reset configured for KEY0.
- In part 4, we reutilize out counter in order to create a Morse code generator. The design displays the Morse code for one of 8 letters specified by switches SW[2:0] on LEDR[0]. The Morse code of each letter is hardcoded in an array. Which is left shifted after every 0.5 secs. Hence the duration of dots is 0.5 secs and that of dash is 1.5 sec. A dot is specified by a single 1 and a dash is represented by a 111 in the array. The last bit of the array is connected to the LED and hence on left shifting we obtain the required output. Also our design has KEY0 configured for reset, KEY1 is confired to load the character and begin displaying the Morse code. We have also added a functionality to display the SOS signal on pressing KEY3.

Observations:

- In Lab 1 and Lab 5 we observe the designs to function as expected. We observe that the push buttons on the board are active low i.e it generates a low signal when pressed and high when released. Also the push buttons are debounced and hence can be used for clock and reset. The 7-segment displays too are active low. And the respective LED glows when a corresponding low signal is applied to it.
- Hence initially the display gave a complemented output, but on reading the datasheet and learning about it being inverted we could rectify it.
- The timer on the board is very accurate. This was verified by measuring timing on stopwatch and on our implementation of the timer. The screenshots of the same are shown below. The 200 ms difference could be because of our inability to start both timers exactly on the same time.

Lab 1 Fmax:

Part	% utilization (ALMs)
1	1% (1)
2	1% (3)
3	1% (2)
4	1% (1)
5	1% (6)
6	1% (18)

Lab 2 Fmax and % Utilization

Part	Fmax (MHz)	% utilization (ALMs)
1	447.43	1% (7)
2	243.61	1% (48)
3	286.37	1% (74)
4	266.88	1% (63)

Lesson Learnt:

- LEDs, 7-segment displays and switches can be used very effectively to debug HDL codes in order to identify bugs and understand the functioning of our code.
- De1-SoC has a huge FPGA fabric, even our most complex design utilized less than 1 percent of total (32000) logic elements.
- Pin planning for larger designs can be more efficiently done by writing Tcl script as compared to assigning each pin manually. Editing and Importing csv files is also a viable option.
- Time dependent modules can be effectively implemented and variable clock speeds can be achieved by simple use of counters. The same 50Mhz clock can be used to generate any lower frequency. The accuracy of the clock on board is reliable.

University of Colorado Boulder

Module 2

Objectives:

- Learn to boot the PYNQ board with the help of Micro SD card.
- To appreciate the Ethernet capabilities of the PYNQ FPGA and python based software and hardware interaction.

Procedure:

- Downloaded the PYNQ-Z1image and wrote the image to the blank Micro SD card by using win32DiskImager.
- Attached micro USB, Ethernet cable, and changed the jumper's position (JP5) to USB from REG.
- Inserted the micro SD card with the PYNQ-Z1 image already wrote. This card will have the booting drivers to set up the PYNQ board.
- Ethernet connection to the board was enabled by adding the wired device in the allowed devices list.
- Then checked for the successful connection by giving the command 'ping pynq'. The screenshot of the result from the terminal is attached below.
- With pynq:9090 we accessed Jupyter. And updated the pynq package.

Task :

1. Followed the instructions from the link on github.com/Xilinx/PYNQ/ and checked whether the LEDs can be configured with the python3.
2. The buttons were configured to shown MORSE code in the output for different cases
   Added an IN[] block where the button 1 is configured for (.) and button 2 is configured for (-) and button 3 is configured for the end result. The end results for different cases is shown below.

Lessons Learnt:

In module 2, We learned the functionalities of Jupyter. We learned to interact to hardware of the PYNQ board with the python code. We learned the different functionalities of Jupyter like using terminal and Python3 for the tasks.

University of Colorado Boulder

Module 3

In module 3 we learn to integrate a Hard Processor System with our FPGA fabric. Write C code which shall execute on linux in HPS to control LEDs connected to the FPGA fabric.

Objective:

- To learn to use ARM HPS to communicate with FPGA.
- To understand the working of the AXB bus interface between HPS master and PIO slave.
- To get familiar with Qsys and appreciate the ease with which a SoC hardware design can be implemented on it.
- To write software for functionality which we previously implemented in hardware like blinking LEDs.
- To design an ARM C program to control the components connected to the FPGA fabric. To understand how Linux system calls map the physical base address of the FPGA peripheral to a directly accessible virtual address.

Procedure:

- We load the my_first_hps-fpga project to build it from scratch. This project has the interconnects between the FPGA and the HPS defined in Qsys.
- We now add a Parallel input output block to the design of width 10 (number of LEDs) and set the reset value at 0x3ff (which is “1111111111” 1 for all LEDs)
- On adding the PIO component we connect the AXI master port of the HPS to the slave of PIO. We also connect its clock and reset to system clock and reset. We export the external connection of the PIO to connect it to LEDRs in our Verilog code. The base address of the PIO port is by default 0x00.
- To make the information about PIO to the C program we generate a header file for the program by running a Linux shell batch batch file provided. We read the header file to understand the mapping of address.
- We then write a makefile and main program to control the LEDs and compile the program to generate a C-executable file.
- We the boot the HPS with Linux Console provided by Altera from out SD card.
- To copy executable file into our Linux system we used USB stick. We mount the USB stick by using mount command. We first list all the connected USB devices using the ‘fdisk -l’ command. We then mount it to any directory using the mount command. The screenshot of the same is shown below:
- On mounting the USB we copy our executable to our root folder and run the file. We see our on board LEDs blinks as required.

Observations:

Yes, the board behaves as expected. We observe the LEDs to turn off 1 by 1 and this repeats for 60 cycles. The percentage utilization of the board is as shown below:

Flow Summary	
Flow Status	Successful - Sun Apr 23 22:31:20 2017
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	ghrd_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,223 / 32,070 (7 %)
Total registers	3304
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	526,336 / 4,065,280 (13 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Fmax for the design is as shown below:

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	63.92 MHz	63.92 MHz	altera_reserved_tck	
2	81.65 MHz	81.65 MHz	clock_50_1	
3	1184.83 MHz	717.36 MHz	soc_system…_write_clk	lim…in)

Lessons Learnt:

The A9 chip on the De1-SoC can be used to load Oss like Linux on which software apllications can be written to drive the hardware. The use of AXI bus to communicate between HPS and FPGA fabric was learnt. The memory address on fabric connected PIOs is mapped ssuch that it is accessible to the HPS. Cross compiling applications on EDS to generate an executable for the processor was understood. The power and flexibility of an SoC device is hence appreciated

University of Colorado Boulder

Module 4

Objective:

- To apply all skills gained in the course and apply it to self-designed project.
- To understand the functionality of the AXI bus for HPS to communicate with FPGA logic.
- To design a system in Qsys where master of HPS is connected to slave of the memory mapped IO components.
- To create a software build system including the sources, headers and a makefile to cross compile the application for arm A9 board.
- To learn to boot the A9 arm chip with Linux and hence setup the environment for execution for our application.
- To understand how the physical address of parallel IO connected to FPGA gets mapped to a virtual address which is accessible by the application software.
- Finally, to understand how a complete system on a chip application functions by making use of Hard Processor, FPGA, IO and memory.

Procedure:

- To create the desired System on a chip application we use Quartus, a software tool from Altera for system design on their chips. We create a new project and start designing in Qsys.
- We add the HPS IP to our design. We the add JTAG to Avalon module and JTAG to UART module IPs to ensure communication between HPS and FPGA and HPS and display respectively. We also add On chip memory to provide memory for the design. We also add an interrupt capturer for the HPS to capture any IO interrupts via FPGA. Finally we add the 7 segment display module and PIO module to connect the 7 segment display and the LEDs.
- We then export HPS IO conduit and PIO-LED and 7 segment conduits to make further connection via Verilog. We then generate the design and import the qip file into out project. After making all the connections in the top level design file we compile the design. And generate a .sof file.
- To use the IO in our software application we shall need a header file that contains the memory mapping information of the IO components in our case it is the LEDs and 7 segment display. To generate the header file we use the bash script provided by altera. We run this script in Altera's EDS tool.
- We then design functions that map this address of the IO component to the virtual address for the HPS. We make use of the header files provided by Altera in the EDS folder The location of these files is: /ip/altera/hps/altera_hps/hwlib/include. Apart from these initialization functions we also design a layer of abstraction between the application software and the underlying hardware memory mapping by writing functions to write specific content onto the display and to drive specific LEDs. In these source files it is essential to include the header files in our project.
- Now we design our application. We have created an application with 3 games. We created c project that creates a menu driven application for games and beneath this layer lies the logic for the game. We referred to an online resource for the logic of the snake game in our application.

The logic for tic-tac-toe was designed by us. The application make use of the c functions to turn the LEDs on and off and drive the 7 segment display to show desired output on occurrence of special events

- On having written all required C files for the application we write a makefile to generate an executable. In the makefile we must ensure that the location to all EDS headers that have been used is specified. A sample makefile is provided by Altera for the same.
- We now generate an executable by cross compiling our application the arm A9 chip.
- We now boot our DE 1 SoC device with console version of linux. This is done by burning the image file of the OS onto the SD card and booting the device through it. We then copy the executable over to the board and run the application

Observation:

We are able to run the application software and 7 segment display on the device also gives information relevant to the game. The fmax of the design is:

Slow 1100mV 85C Model Fmax Summary

Fmax	Restricted Fmax	Clock Name	Note
60.29 MHz	60.29 MHz	altera_reserved_tck	
79.73 MHz	79.73 MHz	clock_50_0	
1184.83 MHz	717.36 MHz	soc_system:u0|so…fi_clk_write_clk	lim…in)

The % utilization of the fabric is as shown below:

Flow Summary

ow Status	Successful - Wed May 03 17:57:53 2017
uartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
vision Name	HPS_LED_HEX
p-level Entity Name	HPS_LED_HEX
mily	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,637 / 32,070 (8 %)
Total registers	3618
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	526,336 / 4,065,280 (13 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Lessons Learnt:

Implementation of software application on HPS that utilizes FPGA resources for enhanced functionality. And booting the A9 ARM chip, how to setup the environment for execution for our application. Using Hard Processor, FPGA, IO and Memory built using Qsys to complete the system on a chip. Mapping address of parallel IO connected to FPGA to the virtual address was understood.

Conclusion

In this Project we worked with a very powerful SoC device in De1-SoC. In module 1 we concluded that IO assignments can be made to any logic implementation on our device by using pin planner. We understood how it can act as an effective tool for debugging. In module 1, we also can conclude that using the available on board clock of 50MHz, we can generate pulses of any frequency like we did for Real Time Clock and Morse Code Implementation. In module 2 we were introduced to PYNK board from XILINX. We learnt about the latest advancements of using high live languages and their powerful libraries to implement designs on FPGAs. In module 3, we used the HPS to drive peripherals connected to the fabric. We learnt about the AXI bus and how HPS can communicate with FPGA fabric. In the Do It Yourself module we used the knowledge gained from previous module and implemented a software program that runs on Linux on HPS and communicates with fabric via AXI bus. Finally we conclude and agree upon the prowess of an SoC based architecture that uses the power of HPS and the flexibility of the FPGA fabric.

Deliverables

Module 1, 2 and 3 project files and output screenshots submitted in the zip file uploaded on d2l with the name JeetSudeep_Project3.zip

Module 4 project files and screenshots uploaded as JeetSudeep_Module4.zip in the dropbox on d2l.

References

- Lab1 pdf: ftp://ftp.altera.com/up/pub/Altera_Material/Laboratory_Exercises/Digital_Logic/DE1-SoC/vhdl/lab1_VHDL.pdf
- Lab5.pdf: ftp://ftp.altera.com/up/pub/Intel_Material/Laboratory_Exercises/Digital_Logic/Verilog/lab5.pdf
- Getting started with PYNQ: http://pynq.readthedocs.io/en/latest/1_getting_started.html
- Using asyncio buttons: https://github.com/Xilinx/PYNQ/blob/master/Pynq-Z1/notebooks/examples/asyncio_buttons.ipynb
- HPS LEDs: http://terasic.yubacollegecompsci.com/resources/My_First_HPS-Fpga.pdf
- HPS 7-seg: http://blog.embedded.pro/7-segment/
- Minesweeper C code reference: http://raitonshiden.weebly.com/uploads/1/7/7/8/17784759/minesweeper_v_1.0.c
- Snake C code reference: http://matthewvlietstra.com/data/uploads/Snake_Game/snake.c