FPGA LAB REPORT

ILLINOIS INSTITUTE OF TECHNOLOGY

ECE-597

FPGA SPECIAL PROJECTS

Title – Audio Processing Using ZED board

Date: 05/14/2021

Vishnu Nambiar & Ishan Loganathan Reddy

INTRODUCTION

What do we implement in the special project ECE597 course?

We use a software code names The Zynq book, that has predefined blocks which is implemented in the Xilinx Zynq-7000 chip. This chip is the programmable System on Chip from Xilinx. The Zynq book covers specific milestones in the field of programmable arrays and SoC's. The major examples would be:

- High Level Synthesis
- Embedded system designs

We implement the project in two steps. We first create the software design blocks needed to build the circuitry as shown in the turorial 1 and 2 in the Zynq book. The next step is to use the ZEDboard for the hardware implementation for chapter 5 in this project.

The ZEDboard:

The ZEDboard is a low-cost developement board designed by Xilinx which is used in the Zynq-7000 SoC. This board consists of all the necessary parameters to create any OS or RTOS based design. Hence, it is compatible with Linux, Android and Windows. Also, ZEDboard enables easy user access which helps to expose the processing system and programmable logic input/outputs. These properties make the ZEDboard ideal for rapid prototyping and concept developement.

The ZEDboard features are:

- 512 MB DDR3
- 256 Mb Quad-SPI Flash
- USB OTG 2.0 and USB UART
- Audio Codec preinstalled

In this course and project, we cover the basic tutorials on the implementation of the Zynq book. We use Vivado to implement the block designs. Next we will discuss the Tutorial 1 and Tutorial 2:

- Tutorial 1: Creating a First IP Integrator design
- Tutorial 2: Expanding the basic Integrator design using Vivado.

In tutorial 1 we create a simple Zynq embedded system which uses a General purpose I/O controller in the Zynq device – in our case the ZEDboard Developement kit. This GPIO controller connects to the LED outputs and is also connected to the Zynq processor via the AXI bus connection.

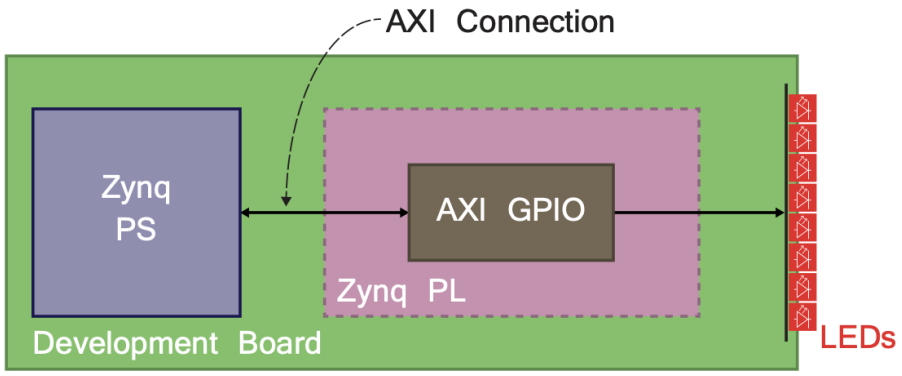


Figure – Zynq Embedded Design

We add the Zynq7 Processing System from the IP Catalog that we obtain from the Zynq book repository we downloaded for this project. We then connect the DDR and the I/O interface ports to the Zynq device by running block automation to the created basic block design. The IP Integrator automatically maps the memory for all the existing IP that is available in the design. After adding the AXI GPIO interface using the IP Integrator Designer Assistance Tool we connect the LED outputs to this port. The design we obtained is shown in the Figure Below:

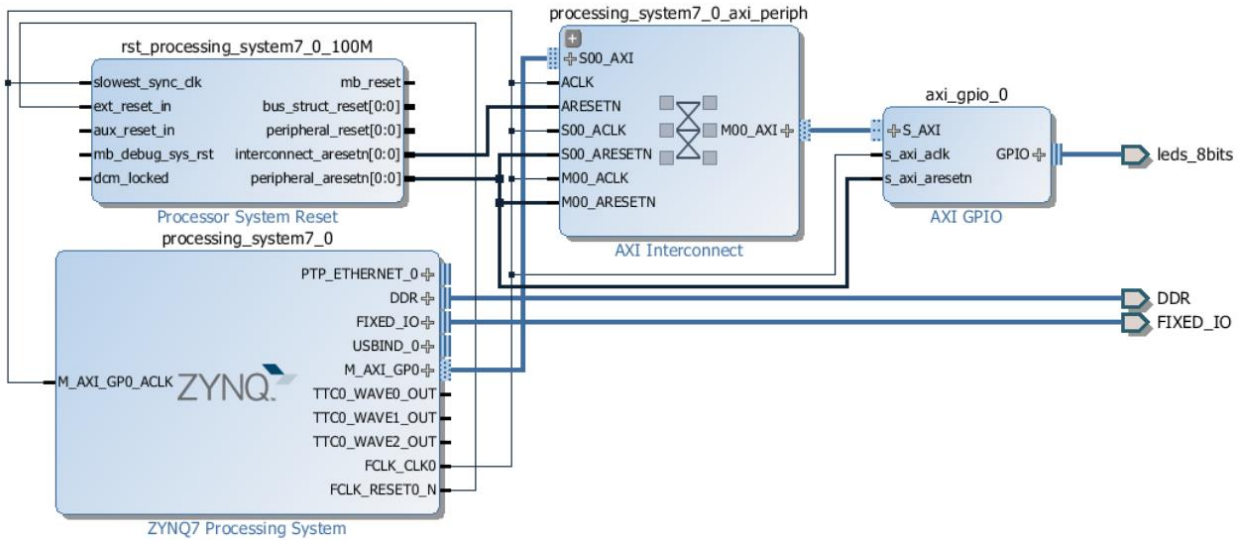


Figure – ZEDboard Zynq Processing System Design Block

SDK Implementation:

The last step in the tutorial 1 is the software application in the SDK. We imported the LED_test_tut_1C.c to build the console. The function xGPIO is provided by the GPIO driver as we are specifying the LED. There are 8 LED outputs in total. Bits set to 0 are output and bits set to 1 are input. Therefore we set the all the LED to outputs by specifying 0x0000000 in binary. After successful Hardware Implementation GDB, we see the lights flashing between the LED outputs.

In this tutorial 2 we create a Zynq design utilising interrupts. We use the IDE and IP integrator environment to create a simple Zynq processor design. We then use the SDK patch to run on the Zynq processor's ARM system to control the hardware that we have implemented in the Programmable Logic.

First we created the simple Zynq embedded system which consists of two general I/O purpose controllers in the Programmable Logic.

One of the controller uses Zynq board's push button functionality to generate interrupts while the other will connect to the LED. They both are finally connected to the Zynq processor using an AXI bus connection.

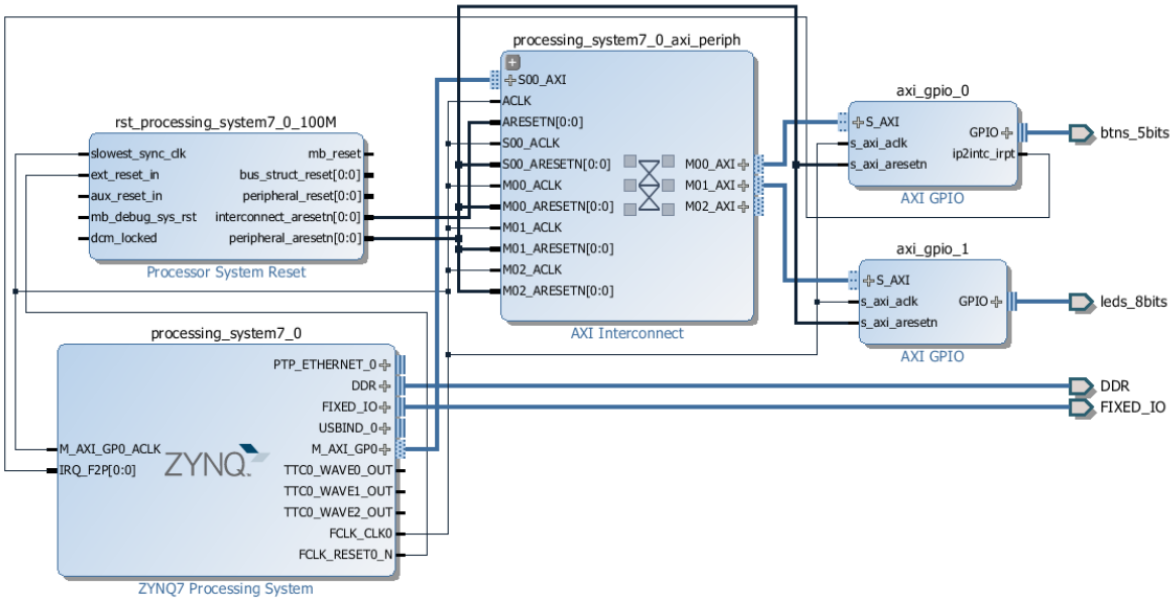


Figure – Zedboard Zynq Processor System Design with Interrupts.

An important note to make here is: the function BTN_Intr_Handler(void*InstancePtr); is the interrupt handler function for the push buttons and this function is invoked everytime a request from the programmable logic is received by the PS. This function performs the increment and decrement of the counter on each invoked iteration and also displays the value of the counter of the LEDs in binary.

After this we run the Hardware implementation – GDB and we can obtain the outputs by pushing the buttons on the Zedboard. The values we found were as follows:

BTNU – 00110000 - 48

BTNR – 00001000 - 08

BTND – 01111110 - 126

BTNL – 00001110 - 14

BTNC – 11110100 – 244

CREATING DIGITAL SIGNAL PROCESSING SYSTEM:

From the completed tutorials 1 and 2 we now use this knowledge and built designs to implement the next stage towards a digital signal processing. We focused on implementing all the IP modules into the Vivado IP catalog by forcing an inclusion in the IP Integrator design.

We also get introduced to the the predefined audio codec available in the Zedboard. The audio codec used is SSM2603. And these IP use I^2S serial communication for transmitting and receiving the audio samples from the audio codec. The audio codec is transmitted in between the Programmable logic and PS by the AXI bus connection.

For using the audio codec, we enable the I^2C interface for communicating via control signals between PS and codec. In order to map the external interfaces in the design for the physical pin mapping in the Zedboard, we created a constraints file (XDC) file. This XDC file allows the Zynq design to give data to the synthesis and implementation processes in Vivado about where they need to route the external/output interface signals.

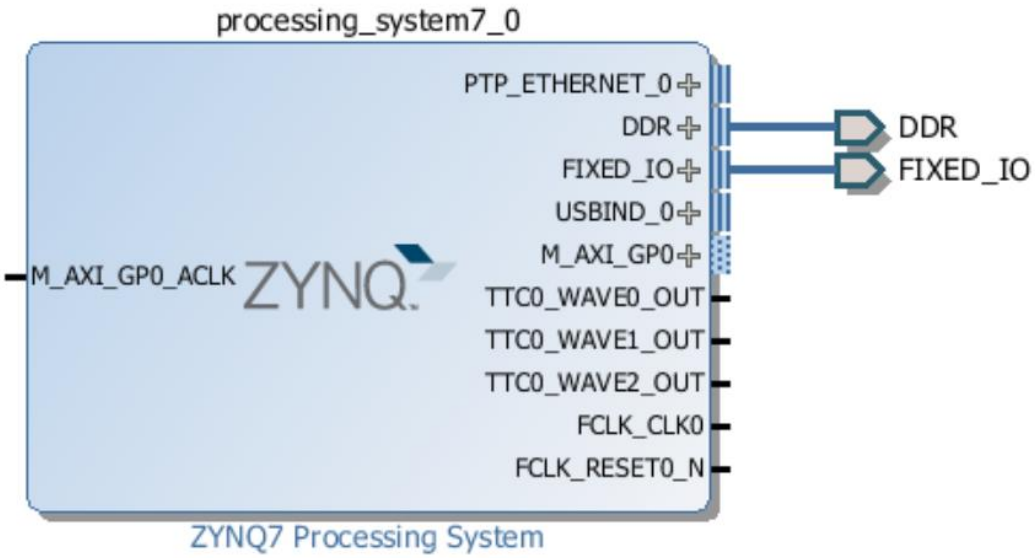


Figure – Zedboard ZYNQ Processing System External Connections.

We run the connection automation for the three IP blocks and connect them via the AXI connection bus. We then made the LED_ports external.

The final block diagram is as seen below:

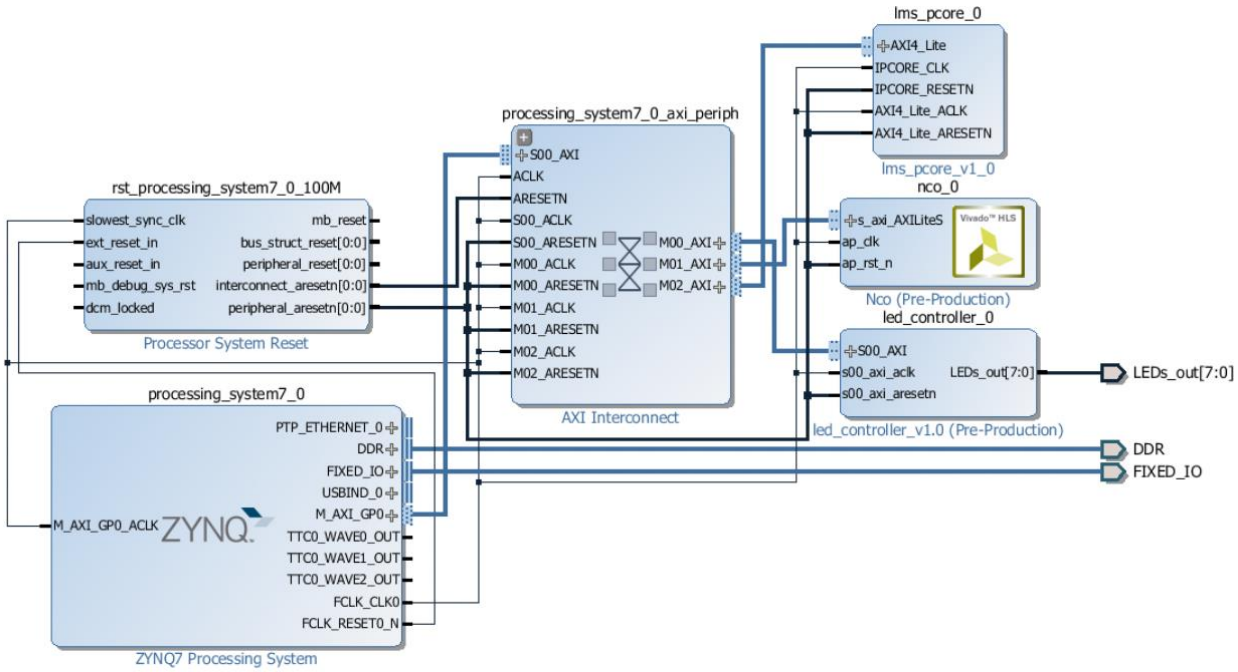


Figure – Final block design created as a digital signal processing system.

We make the necessary modifications to the Zynq7 processing system we have created to make changes to the configuration of the Zynq PS. We added the PL fabric clock as a separate clock input that is required for the CLK pin on the Audio Codec. We then enable the board's I^2C communication interfacing to allow for the PS to communicate with the audio codec.

We must implement HDL wrapping of the block design so that we can implement the same code with any of the other tools in vivado. A top level HDL wrapper is defined as follows:

The actual HDL wrapper is the file that connects the output/input port of your design to the physical pin described in the created constraint file(XDC). The constraints file is imported from the adventure_in_ip repository. We then follow the same procedure as seen in the first two tutorials of the zynq book and finally generate the bitstream from the program and debug section.

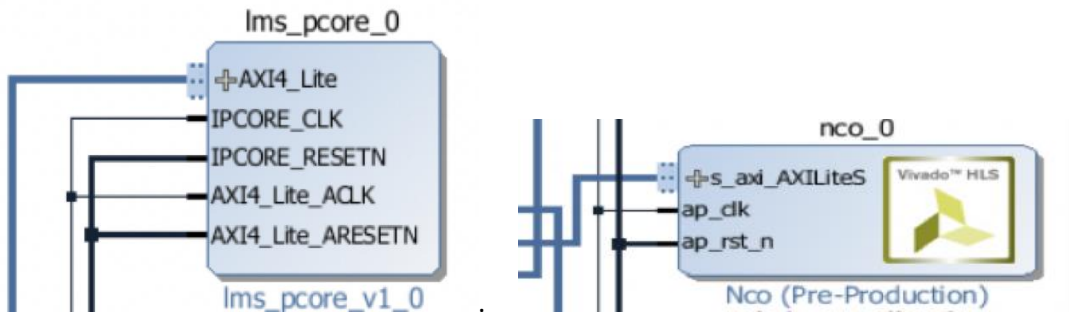
We finally have to export the hardware to the software development kit, which enables us to create a software application to control the system in the final part of this project.

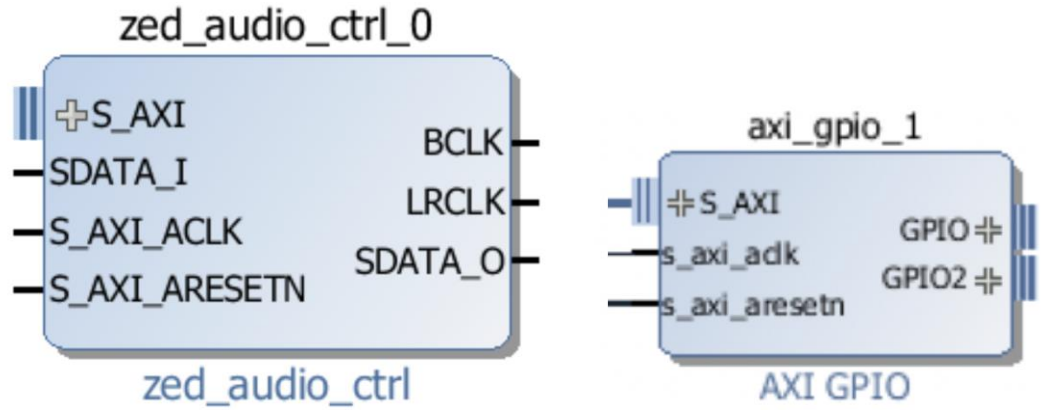
CREATING AN AUDIO SOFTWARE APPLICATION IN SDK:

We created a software application which bundles up and connects all the custom IP modules we have created. This results in creation of a DSP-oriented system.

Stepwise, we add the led_controller_1.0 into the IP repository which gives us the interfacing for the LED I/O ports between the user interface(Computer) and the FPGA board(Zedboard), this is followed by the xilinx_com_hls_nco_1_0 file which is the tonal noise file. The nco file is basically a file that generates the

tonal noise by creating sinusoidal noises. And these tonal noises' frequencies can be varied by external switches on the Zedboard. This is possible due to the I2C interfacing. Lastly we add the lms_pcore_v1_00_a file. This is the file responsible for the filtering of the tonal noise and the original audio signal.

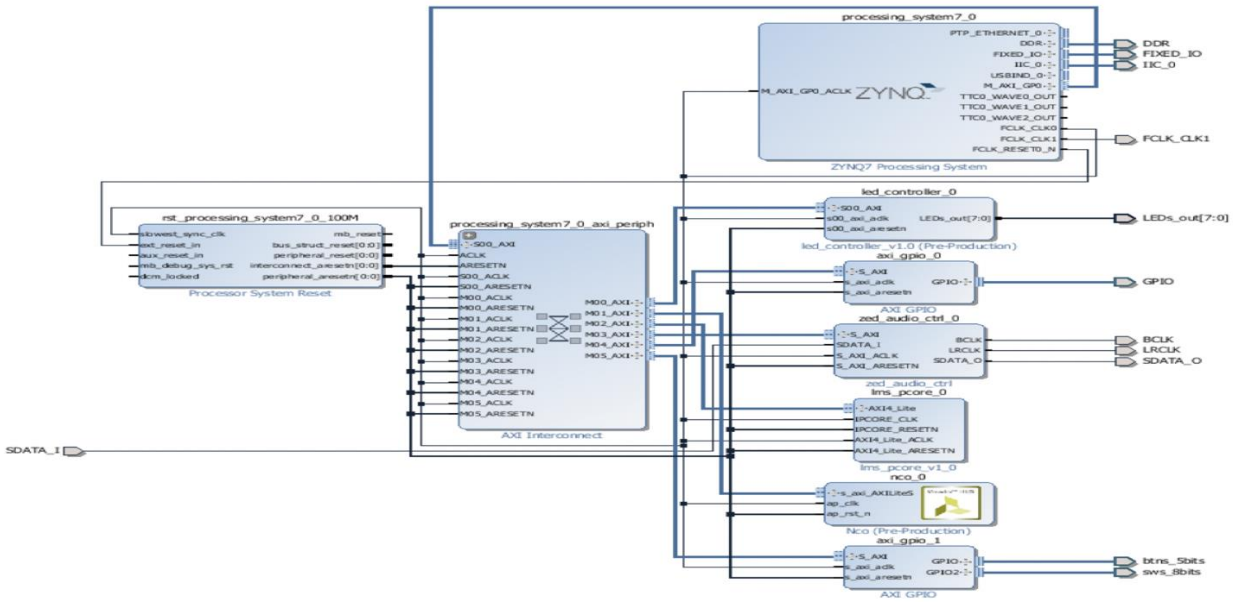




Here we used Putty.exe which acts as a medium for I2C interface between the computer and zedboard. Once we connect the UART port from the zedboard we could see the port in the device manager. In our case it was COM10. And we could use putty for I2C communication which will read the input given through the computer and get the knowledge about the output audio.

The filtering is accessed by implementing a low pass filter. In the output video file we can see that during the execution, in the filtering mode, when we use the push button, we can hear just the tonal noise. This is basically filtering of the tonal noise, separated from the original audio signal.

Now we will see the final implementation block design including all the IP block.



CONCLUSION

Once we run the program through the putty.exe we have access to three option s, n, and f. Pressing ‘s’ will make the original audio stream through zedboard without any addition of noise. Pressing ‘n’ will allow us to add tonal noise to the noise signal. The frequency of the tonal noise can be varied using the switches and its combination. Pressing ‘f’ allows us to filter the noisy signal. Initially we need to add tonal noise by varying the combination of switches and the pressing the push button we can view the filtered signal. And ‘q’ is pressed to exit from any of the functions.

With the help of Vivado and FPGA we could successfully develop a system for the audio processing. Where in we added the input audio added tonal to it and filtered the signal in order to get pure output all using the zedboard as FPGA. We can such create bigger design for a bigger signal processing carrying out more functions required to get a refined output. Through this project we were able to gets hands on experience with the Vivado block design, using of interrupt and creating custom IP packages as per the user requirement.

APPENDIX

For this project the output video files and the presentation video is attached along with this report.