ECE532 Draft Project Proposal

NES Audio Processing Unit

January 21, 2008.

Project Team

Cedomir Segulja, *seguljac@eecg.toronto.edu*
Bill Dai, *bill.dai@utoronto.ca*

Project Description

NES Audio Processing Unit

The goal of our project is to implement Nintendo Audio Processing Unit in the FPGA. The three[1] major processing units of the original NES hardware are Central Processing Unit (CPU), Picture Processing Unit (PPU) and the Audio Processing Unit (APU). The original CPU was modified 8-bit MOS 6502 microprocessor which used memory mapped I/O to communicate with the other components.

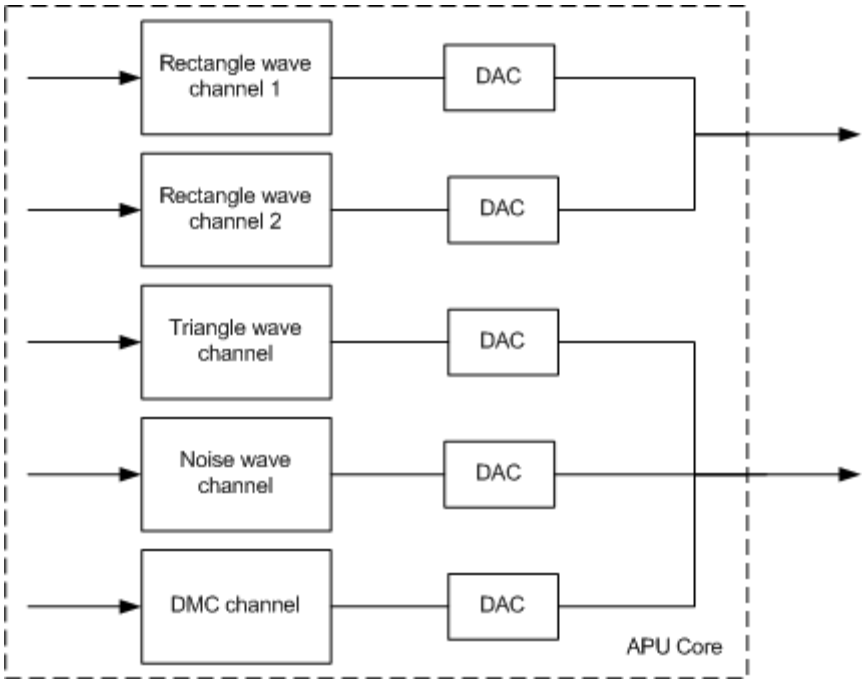

Figure 1: APU Core

The APU has 5 internal channels (components). Each channel is configured with memory-mapped registers (typically three or four 8-bit registers per channel), and connected to a Digital-Analog Converter (DAC) which generates the final, analog signal. The four of them generate simple waveforms: rectangle, triangle and the noise wave forms. The fifth channel is the Delta-Modulation Channel (DMC) which is able to play samples. The DCM has two modes of operation. With the first, DMA mode, samples are fetched directly from the memory (without CPU's intervention). In this case the

[1] The original NES CPU, the 2A03, incorporated the APU. However, we will be looking at it like two separate components.

modulation used is delta-modulation: each bit in 8-bit sample represents the increase or decrease of the current output. With the second mode, samples are fed directly to the DCM in 7-bit PCM form.

The APU has two sound outputs. One carries the two square wave channels and the other one carries the triangle wave channel, the noise channel and the DCM channel. However, in the original NES the final output is mono sound as both intermediate outputs are mixed together.

System Design

Proposed system design is shown in Figure 1. We will be using the Xilinx XUP Virtex™-II Pro Development System. MicroBlaze soft-processor will play the role of NES CPU, driving the APU. The APU is designed as peripheral unit that communicates with the CPU through the OPB bus. CPU writes and reads the APU internal registers using memory-mapped interface. Using OPB bus offers modularity of the system; in that way other components (primary the PPU) will be easily added. Although using the bus could potentially cause bandwidth issues, this is not the case in this system as the original NES CPU operates on 1.7 MHz's.

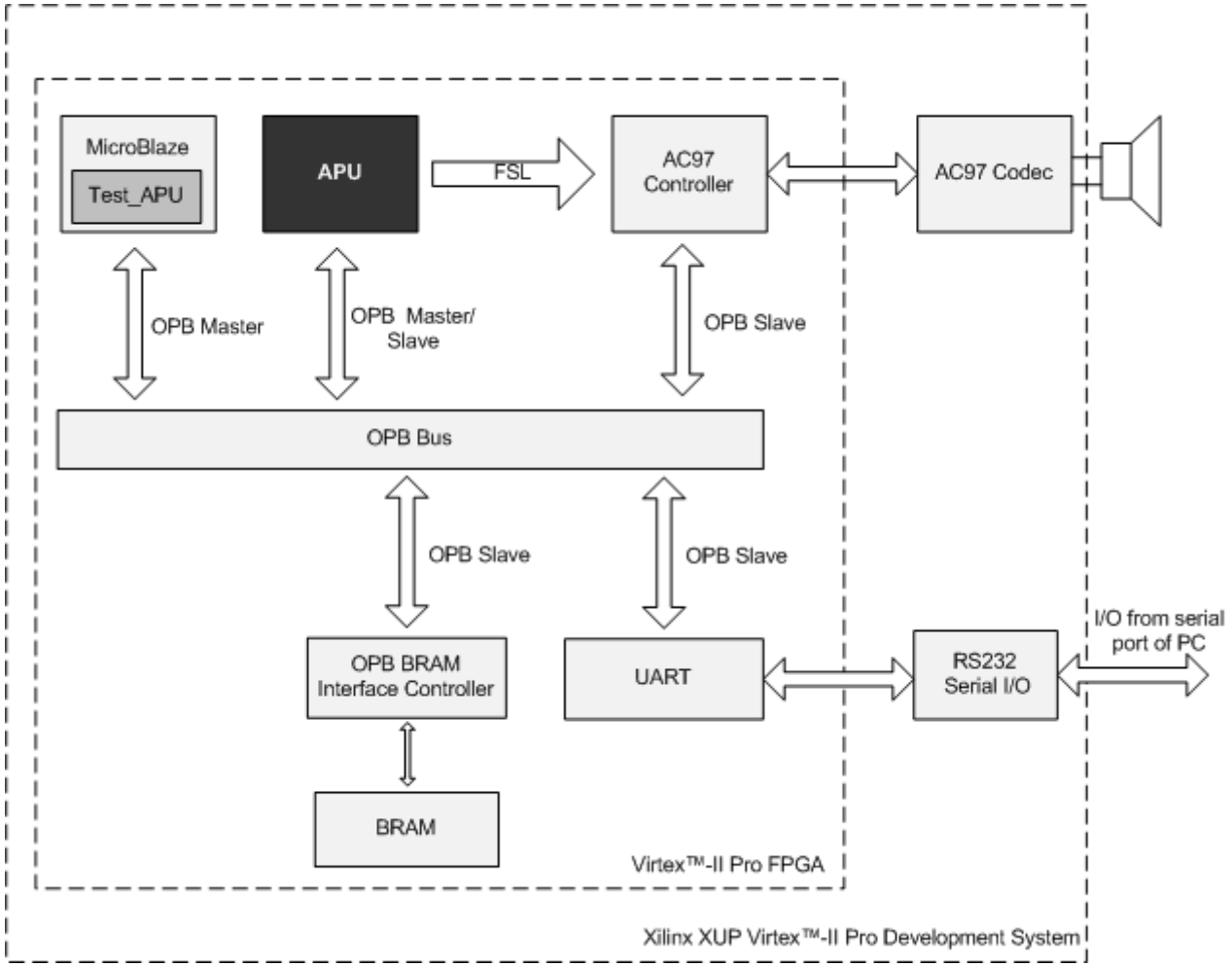

Figure 2: System Block Diagram

The game data and instruction will be stored in the on-chip block RAM (BRAM). The original game sizes are 16KB so this will definitely fit (the XC2VP30 has 2,448Kb BRAM). Although there is a possibility that the final NES system will use the original cartridges that will be connected to the board, for simplicity and testing purposes we have chosen the former design option. The BRAM will be accessed through the OPB BRAM Interface controller. The choice of this design option (instead of using faster LMB to connect the CPU and the BRAM) is described next. Again, there are no bandwidth issues for the reason stated earlier.

Both APU and the PPU should be able to directly access the memory; that is, operate in Direct Memory Access Mode (DMA). Using the OPB bus allows us to (easily) support this operation mode, as OPB bus (as opposite to the LMB bus) allows multiple masters. Having that said, the APU will be connected to the OPB as a master and will be acting as a master only when communicating with the memory.

For digital to analog conversion, we will be using the AC97 codec which is the device placed on the Xilinx XUP Virtex™-II Pro Development System. Codec is controlled with the AC97 controller. The APU communicates with the AC97 controller through the FSL. For configuration and debugging purposes, the AC97 controller is also connected to the OPB bus, which allows communicating with the CPU. Furthermore, for the debugging purposes we will use serial port to display messages on the monitor. The RS232 is controlled by the UART.

The only software component is the Test_APU, which is the simple application for testing the APU.

IP Cores

The following cores will be reused:

- MicroBlaze (Xilinx IP)
- OPB Bus (Xilinx IP)
- AC97 Controller (Xilinx IP, but also using documentation (and maybe code) of past projects)
- OPB BRAM Interface Controller (Xilinx IP)
- BRAM (Xilinx IP)
- UART (Xilinx IP)

We will build the APU as our custom IP core.

Milestones

The main things that we have to do/learn:

- Detail knowledge of behavior of each APU channel
- Implementing each APU channel functionality
- Understanding AC97 Controller
- Developing a user-design peripheral

We propose the following implementation plan:

February 13	Use Xilinx demos (or past projects) to design a MicroBlaze system which uses AC97 controller to play (any) sound. Model and simulate one channel (for example rectangular wave channel) of the APU
February 27	Develop user-design OPB peripheral (APU)
March 5	Implement other channels. The DMC channel is potentially the trickiest as it request communication with memory.
March 12	Testing
March 19	Testing
March 26	Final demo