ECE532 Group Report
FPGA Implementation of the NES Audio Processing Unit

Cedomir Segulja, Bill Dai
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
seguljac@eecg.toronto.edu, bill.dai@utoronto.ca

March 31^{st}, 2008

1 Introduction

In this report, we describe the work done for our ECE532 class project. We designed and implemented the Nintendo Entertainment System (NES) Audio Processing Unit (APU) using an FPGA platform. Furthermore, for purposes of testing the developed core and for project demonstration, we have design *NSF player*, a system capable of playing NSF files using our developed core.

The remainder of this report is organized as follows. Section 2 presents our projects goals, covers background material and gives high-level overview of the architecture of our system. Section 3 discuss the outcome of our project and presents the software architecture of the NSF Player. Section 4 provides a detail description of used components, concentrating the most on the developed IP, the OPB APU. Section 5 explaines how we used the NES emulator to extract needed information from NSF files. Section 6 provides the details about the software structure of our project. Finally, we include an appendix that provides additional details about the NES APU register organization.

2 Overview

2.1 Goals of the Project

The main goal of the project is to implement the NES Audio Processing Unit on the FPGA of Xilinx XUP VirtexTM-II Pro Development Board. By the ending phase of the project, our APU should be able to demonstrate standalone music playing ability by using MicroBlaze as the NES CPU.

2.2 Project Background

The original CPU of Nintendo Entertainment System was 8-bit Ricoh 2A03 microprocessor based on a MOS Technology 6502 core [2]. This low-cost chip differed from similar product back at year 1983 in that it had the ability to handle sound, serving as pseudo Audio Processing Unit (APU). In other words, the CPU and APU were combined on one chip. The processing speed of this chip is around 1.79MHZ. The APU contains in total twenty 8 bit memory mapped registers.

The NES APU is composed of five channels. These included two pulse wave channels of variable duty cycle (12.5%, 25%, 50%, and 75%), with a volume control of sixteen levels, and hardware pitch bending supporting frequencies ranging from 54 Hz to 28 kHz. Additional channels included one fixed-volume triangle wave channel supporting frequencies from 27 Hz to 56 kHz, one sixteen-volume level white noise channel supporting two modes (by adjusting inputs on a linear feedback shift register) at sixteen preprogrammed frequencies, and one delta pulse-width modulation channel (DMC) with six bits of range, using 1-bit delta encoding at sixteen preprogrammed sample rates from 4.2 kHz to 33.5 kHz. The DMC channel used DMA to fetch previously stored samples. This final channel was also capable of playing standard pulse-code modulation (PCM) sound by writing individual 7-bit values at timed intervals. The complete functional description of the NES APU can be found in [6, 8, 9, 7].

2.3 System Description

The system architecture is shown in Figure 1. The OPB Audio Processing Unit (APU) represents the implementation of the the NES APU. To convert digital output to analog signals, the on-board AC97 codec is used. MicroBlaze, runing the APU Driver simulates the NES CPU and drives the OPB APU. Memory samples for the DMC channel are stored in the BRAM which is accessed by the OPB APU through the OPB BRAM controller. OPB Microprocessor Debug Module (MDM) is used for debugging and download. UART Lite was used through the developement process, it is obsolete in the final design.

List of used IPs is shown in Table 1. We have designed and implemented the OPB APU core, which is in detail described in section 4.3. Also, created software (APU Driver) is described in section 4.8.

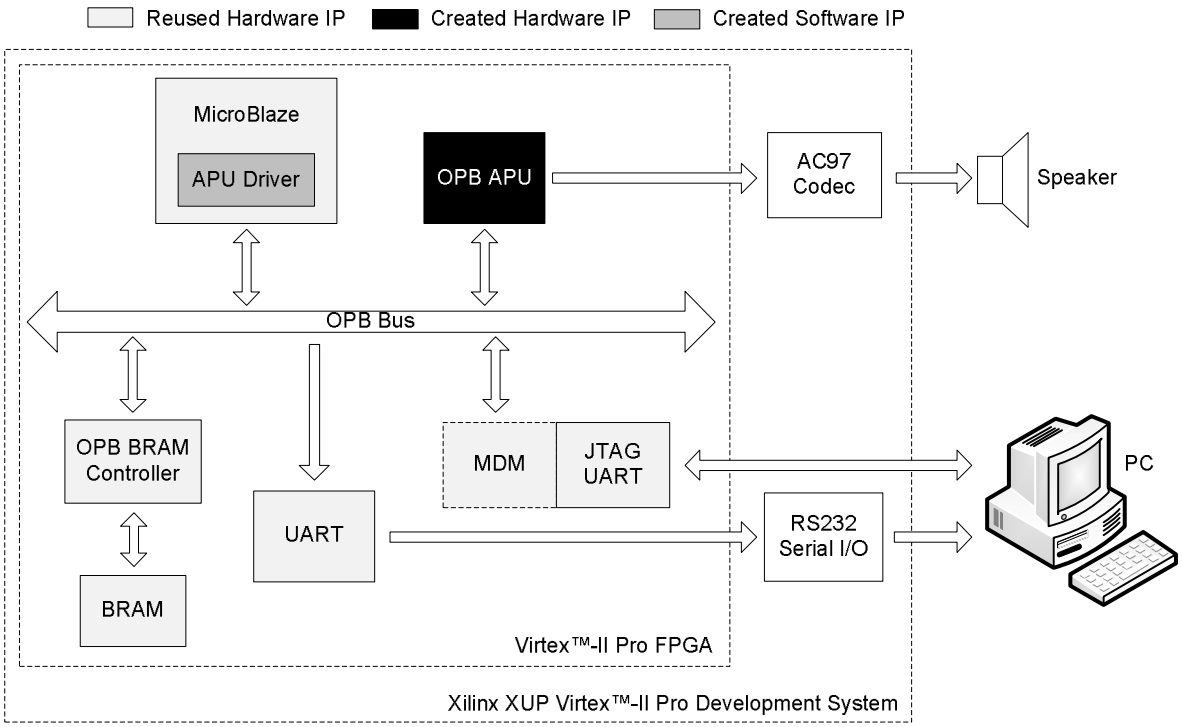

Figure 1: System Block Diagram

IP Name	Hardware/ Software IP	Used/Modified/ Created	Function
MicroBlaze Processor	Hardware IP	Used (Xilinx IP)	Drives the OPB APU. See section 4.1
On-Chip Peripheral BUS (OPB)	Hardware IP	Used (Xilinx IP)	System backbone. See section 4.2
OPB Audio Processing Unit (APU)	Hardware IP	Created/Modified	NES APU implementation. See section 4.3
OPB Block RAM (BRAM) Interface Controller	Hardware IP	Used (Xilinx IP)	Memory controller for the DMC dedicated memory. See section 4.4
Block RAM (BRAM) Block	Hardware IP	Used (Xilinx IP)	DMC dedicated memory. See section 4.4
Digital Clock Manager (DCM) Module	Hardware IP	Used (Xilinx IP)	Used to derive system and NES CPU clock. See section 4.5
OPB Microprocessor Debug Module (MDM)	Hardware IP	Used (Xilinx IP)	Used for debugging and executable download. See section 4.6
OPB UART Lite	Hardware IP	Used (Xilinx IP)	Used for debugging. See section 4.7
APU Driver	Software IP	Created	Simulates the NES CPU. See section 4.8

Table 1: Brief description of IPs

3 Outcome

We have successfuly implemented the NES Audio Processing Unit on the FPGA. The developed IP was tested using series of microbenchmarks. Furthermore, for in-depth testing and for the purposes of the project demo we have designed and implemented *NSF player*, a system which plays NSF files using our developed IP. NES Sound Format (NSF) file is a sound data file containing instructions for the Nintendo Entertainment System (NES) sound hardware. The overwiev of the NSF Player system is given next.

3.1 NSF Player

NSF Player is the system which utilizes the implemented system for playing NSF files. Software flow chart of the NSF Player is displayed on Figure 2. The input is an NSF file. The file is translated in the form plausible for interpreting by the MicroBlaze. Information about used NSF emulator and modification of it are described in section 5. The generated file (`music.h`) is compiled with the `apu_driver.c`, which implements the APU Driver. Finally, using the Xilinx Microprocessor Debugger (XMD) and the JTAG cable, the executable is downloaded on the FPGA. MicroBlaze is running XMD Stub which allows as to download and run the executable on the fly. The whole process is automatized by the script which as an input takes the NSF file.

As shown in Figure 2 the NES emulator also generates the `out.wav` file. This is a waveform audio file, and it is used for comparison with the music played on the board. Our final results is that, for many examples, the music played on the board is very similar (indentical) to the waveform file played back on the computer.

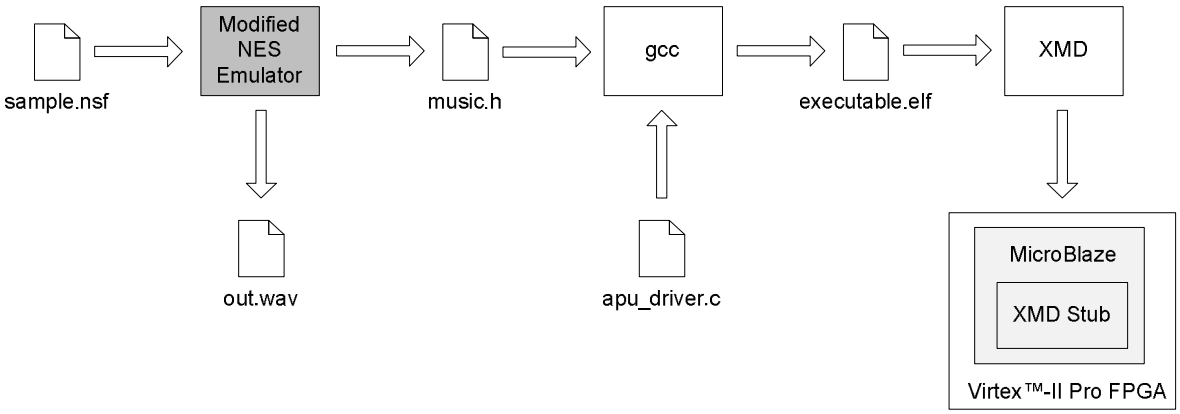

Figure 2: NSF Player Software Flow

4 Description of the Blocks

This section provides detail information about used IP blocks. The created IPs, OPB Audio Processing Unit (APU) and APU Driver are explained in detail, while for the other IPs we denote used version, system use and provide reference to corresponding data sheets.

4.1 MicroBlaze Processor

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx FPGAs. We used version 5.00.c. More information on this IP can be found in [16]. In our design, MicroBlaze is used to simulate NES CPU and drive OPB Audio Processing Unit (APU).

4.2 On-Chip Peripheral Bus (OPB) V2.0

The On-Chip Peripheral Bus (OPB) V2.0 is designed for easy connection of on-chip peipheral devices. We used version 1.10.c. More information on this IP can be found in [18][3]. In our design, OPB serves as a system backbone connecting MicroBlaze and OPB APU with memory and peripherals. We use 32-bit address and data bus implementations.

4.3 OPB Audio Processing Unit (APU)

In this section we describe the design specification for the OPB Audio Processing Unit (APU). The OPB APU is our custom developed IP, written using Verilog and VHDL. The used resources are reported in Table 2.

	OPB APU	APU Core
Slices	822	663
FFs	573	316
LUTs	1498	1188
Block RAMs	0	0

Table 2: Used resources. Column denoted with OPB APU reports resources used for implementing the OPB Audio Processing Unit (APU). Column denoted with APU Core reports resources needed to implement the core functionality of the NES APU (see section 4.3.6). Each Virtex-II Pro Slice has two LUTs.

4.3.1 Functional Description

The OPB Audio Processing Unit (APU) implements the NES APU. The block diagram of the module is shown in Figure 3.

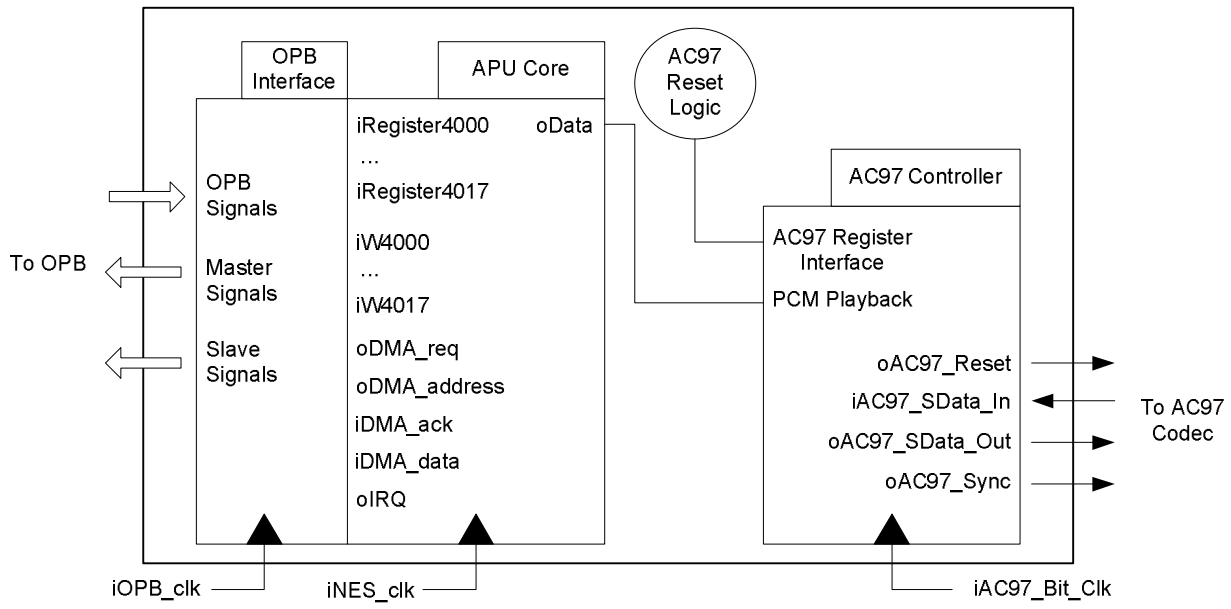

Figure 3: OPB Audio Processing Unit (APU) Block Diagram

For connecting to the OPB, APU provides OPB interface with both Master and Slave signals. Masters signals are needed to implement DMA functionality of the Delta Pulse-Width Modulation Channel (DMC). The APU Core implements the functionality of the NES APU, and outputs 16 bits data to the AC97 Controller. The AC97 Controller implements the AC Link protocol and provides register interface for AC97 Codec. AC97 Reset Logic makes sure that upon system reset AC97 codec is initialized for playback. The OPB APU consists of three clock domains. OPB interface is clocked with the OPB system clock (25 MHz). The APU Core module is clocked with the NES clock (1.79 MHz). The AC97 Controller is clocked with the AC97 Bit Clock (12.288 MHz). For more information about clock generation see section 4.5.

4.3.2 OPB APU I/O Signals

The I/O signals for the OPB Audio Processing Unit (APU) are listed and described in Table 3. Detailed information about OPB protocol and signals can be found in [3].

Signal Name	Interface	I/O	Description
iOPB_clk	OPB	I	OPB clock for OPB APU interface
iOPB_rst	OPB	I	OPB reset for OPB APU interface
iOPB_RNW	OPB	I	OPB read not write
iOPB_select	OPB	I	OPB Reset for OPB APU interface
iOPB_ABus	OPB	I	OPB address bus
iOPB_DBus	OPB	I	OPB data bus
iOPB_MnGrant	OPB	I	OPB master bus grant
iOPB_xferAck	OPB	I	OPB transfer acknowledge
oMn_request	OPB	O	Master bus request
oMn_ABus	OPB	O	Master address bus
oMn_DBus	OPB	O	Master data bus. Not used (constantly set to zero).
oMn_Select	OPB	O	Master select
oMn_DBusEn	OPB	O	Master data bus enable. Not used (constantly set to zero).
oMn_RNW	OPB	O	Master read not write
oSln_xferAck	OPB	O	Slave transfer acknowledge
oSln_DBus	OPB	O	Slave data bus
iAC97_Bit_Clk	AC97	I	Input clock for AC97 controller. Provides a 12.288 MHz clock for the AC Link.
oAC97_Sync	AC97	O	Output to the LM4550 codec. Defines the boundaries of AC Link frames. Sync is a 48 kHz positive pulse with a duty cycle of 6.25% (16/256).
oAC97_SData_Out	AC97	O	This is the output for AC Link Frames from an AC97 controller to the LM4550 codec
iAC97_SData_In	AC97	I	Not used. (This is the input for AC Link Frames from the LM4550 codec to an AC97 controller)
oAC97_Reset	AC97	O	This active low signal causes a hardware reset of the LM4550 codec which returns the control registers and all internal circuits to their default conditions. Triggered on the OPB reset signal.
iNES_clk		I	NES CPU clock (1.79 MHz)

Table 3: OPB APU I/O Signals

4.3.3 Design Parameters

Table 4 lists and describes the features that can be parameterized in the OPB Audio Processing Unit (APU).

Parameter Name	Feature/Description	Allowable values	Default value
C_BASEADDR	Base address for peripheral on OPB bus	0x00000000 to 0xffffffff, with the constraint that the range should be greater or equal than 0x5c (23 8-bits register * 4), where the range is C_HIGHADDR- C_BASEADDR + 1	0xffffff00
C_HIGHADDR	Highest OPB address within peripherals address range	0x00000000 to 0xffffffff	0xffffffff
DMC_MEMORY	Base address of the memory designated for DMC samples storage	0x00000000 to 0xffffffff	0x80400000

Table 4: OPB APU Design Parameters

4.3.4 OPB APU Register Description

The OPB APU registers are listed and described in Table 5. All OPB APU registers are organized in the same manner as the register of the NES APU, as specified in [6]. For detailed information about NES APU registers organization please refer to Appendix A.

4.3.5 Additional Constraints

The OPB Audio Processing Unit (APU) communicates with the AC97 codec. To configurate physical connections between the AC97 codec and the FPGA pins some entries need to be added to the UCF file. For more information consult the submitted `system.ucf` file and/or the board schematic [12].

4.3.6 APU Core

The APU Core implements the functionality of the NES APU, and outputs 16 bits data to the AC97 Controller. The APU Core interface is described in Table 6.

The architecture of the APU Core is displayed in Figure 4. It consists of five channels: Rectangle Channel 1 and 2, Triangle Channel, Noise Channel and the DMC Channel. The Frame Sequencer module generates clock needed by the channels. The Channel Mixer determines the final output of the APU Core module. In the rest of this section we provide block diagrams and short description of each module of the

Register Name	Size	Address Offset	Initial State	Description
Register4000	8	0	0x10	Rectangle channel 1 register 1
Register4001	8	4	0x00	Rectangle channel 1 register 2
Register4002	8	8	0x00	Rectangle channel 1 register 3
Register4003	8	12	0x00	Rectangle channel 1 register 4
Register4004	8	16	0x10	Rectangle channel 2 register 1
Register4005	8	20	0x00	Rectangle channel 2 register 2
Register4006	8	24	0x00	Rectangle channel 2 register 3
Register4007	8	28	0x00	Rectangle channel 2 register 4
Register4008	8	32	0x10	Triangle channel register 1
Register400A	8	40	0x00	Triangle channel register 2
Register400B	8	44	0x00	Tirangle channel register 3
Register400C	8	48	0x10	Noise channel register 1
Register400E	8	56	0x00	Noise channel register 2
Register400F	8	60	0x00	Noise channel register 3
Register4010	8	64	0x10	DMC channel register 1
Register4011	8	68	0x00	DMC channel register 2
Register4012	8	72	0x00	DMC channel register 3
Register4013	8	76	0x00	DMC channel register 4
Register4015	8	84	0x0f	Status register
Register4017	8	92	0x00	Frame register

Table 5: OPB APU Registers

Signal Name	I/O	Description
iRegister40xx	I	8 bit register. Asynchronous signal. See Table 5 for register description.
iWx	I	Asynchronous signal raised upon write to the corresponding register.
oDMA_req	O	DMA request signal.
oDMA_address	O	DMA address (16 bits)
iDMA_ack	I	DMA acknowledgement signal
iDMA_data	I	DMA input data (8 bits)
oIRQ	O	Interrupt signal. Currently not implemented.
oData	O	APU Core output derived by mixing the outputs of all channels. 16 bits.

Table 6: APU Core Interface

APU Core. Full specification of the APU channels can be found in [6, 8, 9, 7]. Register organization can be find in the Appendix A.

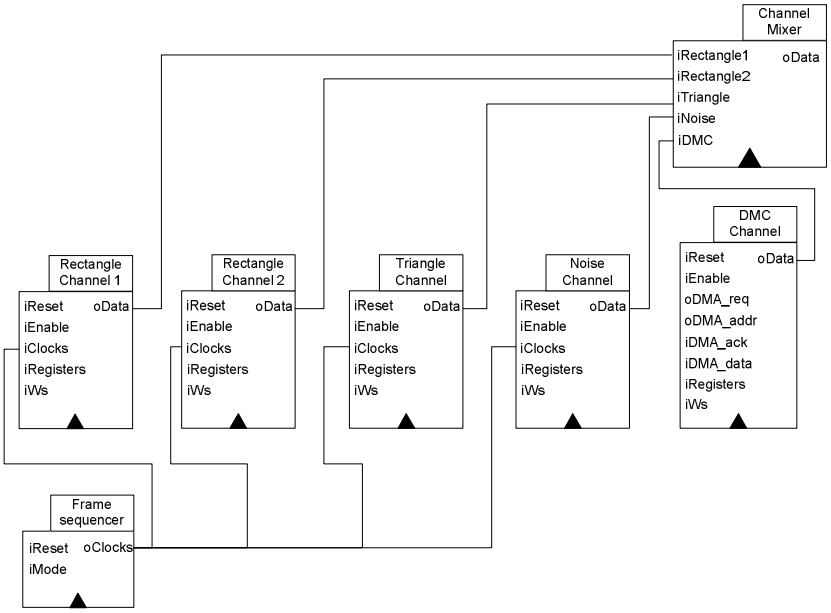

Figure 4: Audio Processing Unit (APU) Core Block Diagram

Frame Sequencer Frame Sequencer generates needed frequencies for Volume Generator, Length Counter, Frequency Generator and Linear Counter. These are the building blocks of the APU Core channels. Frequencies are derived using counters and the NES clock as an input clock. Generated frequencies are denoted in Table 7.

Output signal name	Frequency
oLength_clk	120 Hz (mode == 0) or 96 Hz (mode == 1)
oEnvelope_clk	240 Hz (mode == 0) or 192 Hz (mode == 1)
oLinear_clk	240 Hz (mode == 0) or 192 Hz (mode == 1)
oSweep_clk	120 Hz (mode == 0) or 96 Hz (mode == 1)
IRQ_clk	60 Hz (mode == 0). For mode == 1 this output is constantly 0.

Table 7: Frequencies generated by the Frame Sequencer. Frequencies depend on the input signal $mode$

Rectangle Channel The Rectangle Channel generates rectangle waveforms. The block diagram of the Rectangle Channel is shown in the Figure 5.

The Frequency Generator generates the frequency of the output wave and implements the frequency sweep. If the $iSweep\_enable$ is set to 0, the frequency is determined by $\frac{NES\_clock\_freq.}{iPeriod+1}$. Otherwise the

frequency is constantly updated with the frequency $\frac{Sweep\_clock\_freq.}{iSweep\_refresh\_rate+1}$. Period is updated as shown in Table 8.

The Length counter is used to set the total duration of the waveform. Duration (8 bit) is defined by encoding the value $iDuration$. The enconding table is implemented using LUTs. When the Length Counter reaches zero, the channel is muted.

The Volume Generator determines the volume (4 bits) of the waveform. If $iEnable$ is set to 1 the volume is determined by the $iVolume$. Otherwise, the volume is constantly decreased by 1 with the frequency $\frac{Envelope\_clock\_freq.}{iDecay\_rate+1}$. When it reaches zero it will (1) stop (channel muted) or (2) wrap-around, depending on the value of $iEnable\_loop$ signal.

iSweep_mode	New period
0	$current\_period + (current\_period >> iSweep\_shift)$
1	$current\_period - (current\_period >> iSweep\_shift)$

Table 8: Calculation of the new period for the frequency sweep. Documentation [6] reports that term -1 should be added for Rectangle Channel 1 when iSweep_mode is 1. This is not currently implemented.

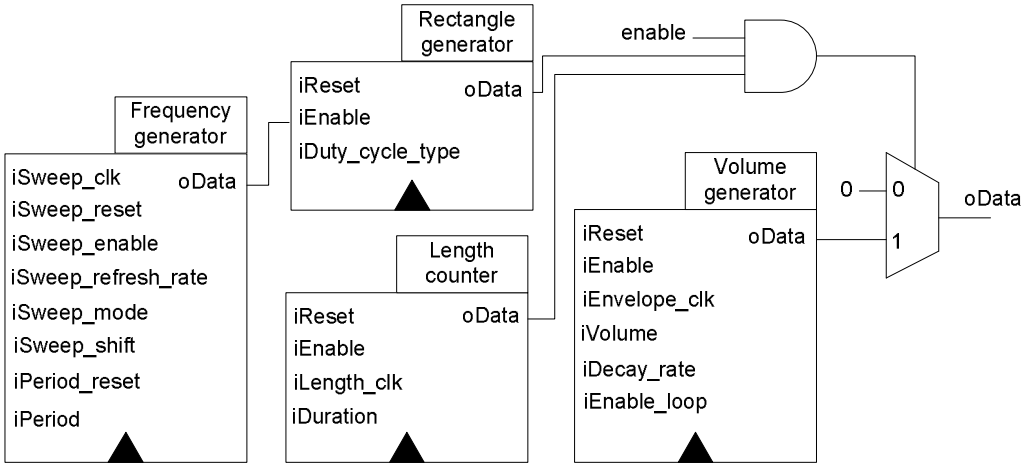

Figure 5: Rectangle Channel Block Diagram

Triangle Channel The Triangle Channel generates triangle wave forms with a 4-bit resolution. The block diagram of the triangle channel is shown in the Figure 6. Frequency of a generated waveform is determined by the Clock Divider which divided the NES clock frequency by $iDivider$. The Length Counter and the Linear Counter determine the duration of the output wave.

Noise Channel The main component of the noise channel is Random Generator, which as output produces pseudo-random binary sequences. The block diagram of the noise channel is shown in the Figure 7. It is capable of producing 93-bit (short) and 32,767-bit (long) sequences. The type of sequence is defined by the $iMode$ signal.

Delta Pulse-Width Modulation Channel The Delta Pulse-Width Modulation Channel (DMC) as an output provides PCM sample. For generating PCM samples channel uses Delta Pulse-Width modulation and Direct Memory Access (DMA). The block diagram of the DMC channel is shown in the Figure 8.

Using direct memory access, the DMC channel fetches one byte at the time from dedicated DMC memory. The DMA Sample Counter register defines the number of bytes that needs to be fetched. The DMA

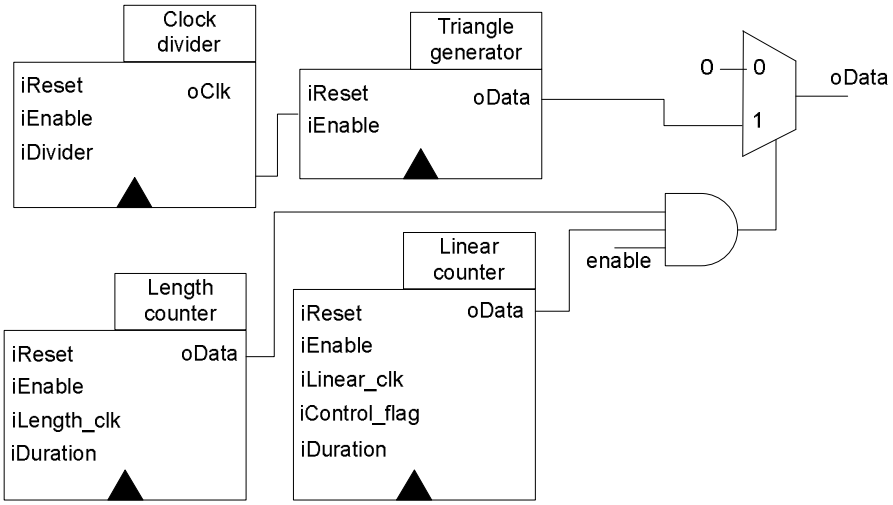

Figure 6: Triangle Channel Block Diagram

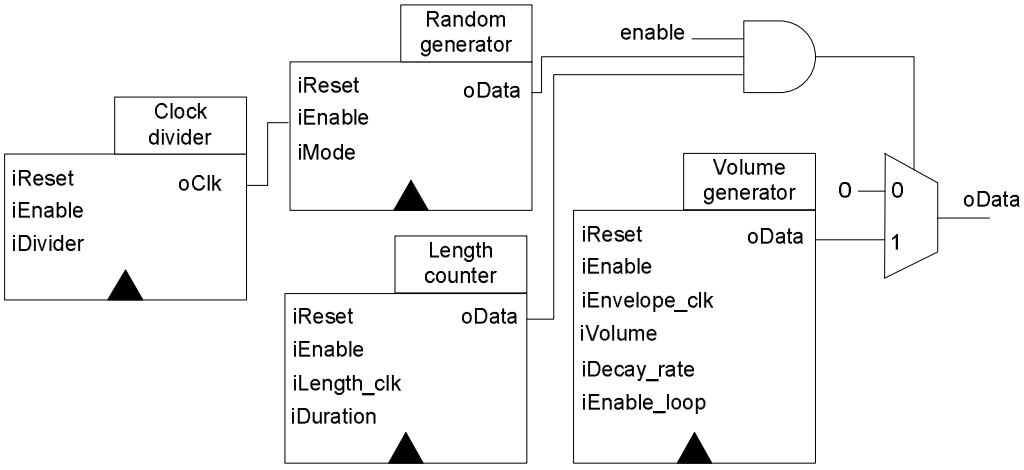

Figure 7: Noise Channel Block Diagram

Sample Address register contains the address from which the next byte should be fetched. When the DMA Buffer is empty, $oDMA\_req$ signal is raised, and the $oDMA\_addr$ is set. When $iDMA\_ack$ signal is raised, the value $iDMA\_data$ is valid, and is stored into DMA Buffer.

Clock Dividers produce two clocks, the $Cycle\_clock$ and the $Shift\_clock$, where $Shift\_clock$ is clocked at 8x of the $Cycle\_clock$. If, on the positive edge of the $Cycle\_clock$ DMA Buffer is empty, the cycle is declared as silence cycle, and during that period the channel is muted. Otherwise, the output is determined by the DMC Shift Register and the DMC Delta Counter. These registers are clocked at the $Shift\_clock$. The bits shifted from the DMC Delta Counter are used as an input of the DMC Sample Counter which is up-down counter with clipping behaviour.

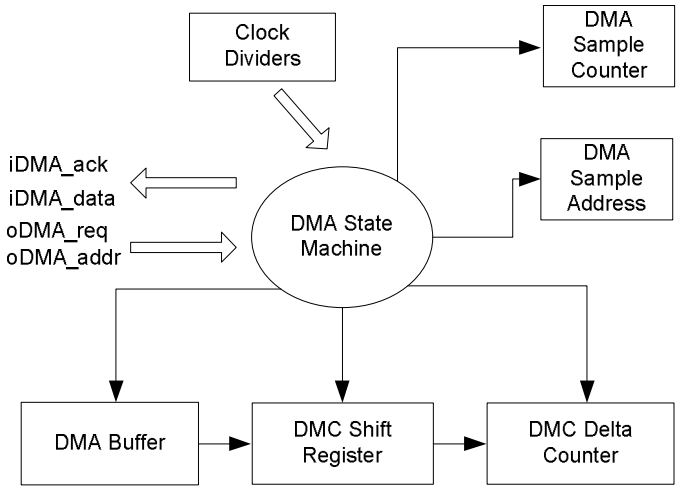

Figure 8: Delta Pulse-Width Modulation Channel Block Diagram

Channel Mixer Channel mixer implements non-linear function which defines the final 16 bit output of the APU Core. As stated in [6], the output is defined with equation:

$$output = \frac{95.88}{\frac{8128}{rectangle_1 + rectangle_2} + 100} + \frac{159.79}{\frac{1}{\frac{triangle}{8227} + \frac{noise}{12241} + \frac{dmc}{22638}} + 100}$$

where $rectangle_1$, $rectangle_2$, $noise$, $triangle$ represent 4 bit outputs from the corresponding channel, and dmc represents 7 bit output from the DMC channel. For efficient implementation in the FPGA we use an approximation proposed in [6]. This is accomplished using 31-entry table for two rectangle channels and a

203-entry table for the triangle, noise and the DMC channel.

$$rectangle\_table[n] = \lfloor M * \frac{95.52}{\frac{8128.0}{n} + 100} \rfloor$$

$$tnd\_table[n] = \lfloor M * \frac{163.67}{\frac{24329.0}{n} + 100} \rfloor$$

M represents scaling factor to get 16 bit result. We use $M = 2^9$. Based on these equation, the initialization code for tables is automatically generated. The final output is then calculated using equation

$$output = rectangle\_table[rectangle_1 + rectangle_2] + tnd\_table[3 * triangle + 2 * noise + dmc]$$

4.3.7 AC97 Controller

The audio system on the XUP Virtex-II Pro Development System consists of a National Semiconductor LM4550 AC97 audio CODEC [5] paired with a stereo power amplifier (TPA6111A) made by Texas Instruments. The LM4550 DACs have 18-bit resolution, and sample rate in the range 4 kHz - 48 kHz.

The AC97 Controller implements the AC Link protocol for communicating with the LM4550 chip. We have modified AC97 Controller, version 2.00.a, developed by Mike Wirthlin which comes with the EDK XUPV2P Pack. More information on this IP can be found in [10]. Originally, this AC97 controller is a standalone OPB peripheral, providing slave OPB interface and asynchronous FIFO for crossing OPB-AC97 clock domains. As we have incorporated this logic into our APU OPB we do not use original AC97 Controller OPB interface. Furthermore, as output of the APU OPB is continous signal which yet needs to be sampled, we do not use asynchronous FIFO. Good description of the AC97 Controller and the AC Link Protocol can be found in [4].

4.4 OPB Block RAM (BRAM) Interface Controller and Block RAM (BRAM) Block

The BRAM Block is a configurable memory module that attaches to a variety of BRAM Interface Controllers. We used verion 1.00.a. More Information on this IP can be found in [15].

The OPB BRAM Interface Controller is the interface between the OPB and the BRAM block peripheral. We used verion 1.00.a. More Information on this IP can be found in [14].

In our design, these components are used to store and retreive samples for Delta Pulse-Width Modulation

Channel. For that reason, the C_BASEADDR parameter of the OPB BRAM Interface Controller should be set to the same value as the DMC_MEMORY parameter of the OPB Audio Processing Unit (APU).

4.5 Digital Clock Manager (DCM) Module

The Digital Clock Manager (DCM) primitive in Xilinx FPGA parts is used to implement delay locked loop, digital frequency synthesizer, digital phase shifter, or a digital spread spectrum. We used version 1.00.a. More information on this IP can be found in [13]. In our design we use two instances of this module. The first one, dcm_0 is used to generate OPB system clock (25 MHz) by dividing input clock by 4. The second one, dcm_nes_cpu is used to generate frequency of 1.79 Mhz, which is the frequency of the original NES CPU. This is derived by dividing the output clock of the first DCM by 14.

4.6 OPB Microprocessor Debug Module (MDM)

The Microprocessor Debug Module (MDM) enables JTAG-based debugging of one or more MicroBlaze processors and/or PowerPC 405 processors. We used version 2.00.a. More information on this IP can be found in [17]. We do not use this IP for hardware debugging solution; instead we use only its JTAG UART module to communicate with the XMDStub running on the MicroBlaze.

4.7 OPB UART Lite

The OPB UART Lite (v1.00b) module connects the OPB bus to the serial port of the board. This allows the user to interact with the MicroBlaze by sending and receiving characters over the RS232 port via the OPB. More information on this IP can be found in [11]. It was used during development process for debugging purpose. It is obsolete in the final design.

4.8 APU Driver

The APU Driver is a simple C program executing on the MicroBlaze processor and driving the OPB Audio Processing Unit (APU). Its main is shown in Figure 9. The task of the APU Driver is two -fold: to initialized DMC dedicated memory and to simulate NES CPU instructions. Initializtion of the DMC memory is done using arrays `dcm_address` and `dcm_data`. For simulating the NES CPU, APU Driver uses arrays `delay`, `address`, `data` and parameter `SLOW_FACTOR`. Delays between instructions are simulated using a simple

loop with variable number of iteration. Details about construction of used parameters and arrays can be found in Section 4.

```
...
//Memory initialization for DMC
for (i = 0; i < DCM_MEM_SIZE; i++)
{
  XIo_Out8(XPAR_OPB_BRAM_IF_CNTLR_0_BASEADDR + dcm_address[i], dcm_data[i]);
}

for (i = 0; i < NUM_OF_INSTRUCTIONS; i++)
{
  //Simulate delay between NES CPU instructions
  for (j = 0; j < SLOW_FACTOR * delay[i]; j++);

  //Write to APU registers
  XIo_Out32(XPAR_APU_OPB_0_BASEADDR + 4*address[i], data[i]);
}

//Silence all channels
XIo_Out32(XPAR_APU_OPB_0_BASEADDR + 84, 0);
...
```

Figure 9: APU Driver Source Code

5 NSF Translation

During the NSF Translation procedure, input NSF file is translated into C header file which will be added to the APU Driver and executed on the MicroBlaze. For implementing the NSF Translation, open source C++ NES emulator is used. More information about used emulator can be found in [1].

The following modifications are done with the used NES emulator. First, new class $APU\_Analyze$ is created. This class implements methods for capturing writes to APU registers, initialization of the DMC dedicated memory and generating output C header file. Then, corresponding calls to the methods of this class have been inserted. All the calls are coupled with comment `NSF TRANSLATION` so they can be easily tracked.

Delay between NES CPU instruction is calculated as a function of the NES CPU clock cycles. We used a small training set of several NSF files, and determined that delay can be represented with equations:

$$SLOW\_FACTOR = 7 * 10^6 * (\frac{10 * total\_number\_of\_cycles}{NUM\_OF\_SECONDS})^{-0.96345}$$

$$delay(i) = SLOW\_FACTOR * (cycle(i) - cycle(i-1))$$

, where $cycle(i)$ denotes the current cycle number of the NES CPU when executing instruction i. $NUM\_OF\_SECONDS$ represents the duration of the song, i.e. only the first $NUM\_OF\_SECONDS$ seconds of the song will be translated (and consequently played). This constraint is addedd due to the finite amount of the available memory for storage (<64KB).

Sample file generated during the NSF Translation is shown in Figure 10. At the begining of the file, information about used channels is displayed. Addresses of the DMC samples are relative to the base address of the DMC dedicated memory. Addresses of the APU register are relative to the base address of the OPB APU core and divided by 4. Generated information is used by the APU Driver as explained in the previous section.

```
//Auto-generated file

//STATISTICS

//Used Channels:
//Rectangle 1
//Noise

#define NUM_OF_INSTRUCTIONS 2403

#define SLOW_FACTOR 23

#define DCM_MEM_SIZE 589

static const unsigned int dcm_address[DCM_MEM_SIZE] = {..};

static const unsigned char dcm_data[DCM_MEM_SIZE] = {...};

static const unsigned char address[NUM_OF_INSTRUCTIONS] = {...};

static const unsigned char data[NUM_OF_INSTRUCTIONS] = {...};

static const unsigned short int delay[NUM_OF_INSTRUCTIONS] = {...};
```

Figure 10: Sample file generated during the NSF Translation procedure

6 Description of the Design Tree

The submitted directory has the following structure:

- `readme.txt`

  Describes the structure of the design directory

- `\demo`

  The lab demo. Includes

  - `\nsf`- Sample NSF files
  - `\sample_files` - scripts called from the ppt presentation

- `\doc`

  Documentation files. These include:

  - Draft Project Proposal
  - Project Proposal
  - Presentation
  - Group Report

- `\emulator`

  Used emulator, Blargg's NES APU. Includes:

  - `\apu` - standalone software NES APU
  - `\player` - modified NSF emulator
  - `\Debug` - contains the player.exe, emulator executable which as an input takes NSF file
  - `\vanilla_versions` - original sources codes
  - `nes.sln` - Visual C++ 2008 Express Edition solution

- `\scripts`

  - `download.bat` - automatizes the proces of the bitstream download

  - `nsf_player.bat` - automatizes the process of translating the NSF file, downloading and running executable
  - `synthesis.bat` - automatizes the synthesis process (warning make clean will be called first)

- `\system`

  Contains the XPS project. APU OPB HDLs, and data files can be found in `\pcores` directory.

NOTE: All links are hardcoded. Original path of this directory was `d/ece532/project`. So, in order to use scripts you should adjust the links to point to your current directory.

References

[1] *Blargg's Audio Libraries.* (`http://www.slack.net/~ant/libs/audio.html`).

[2] Patrick Diskin. *Nintendo Entertainment System Documentation.*

[3] IBM. *On-Chip Peripheral Bus, Architecture Specification, Version 2.1*, April 2001.

[4] Wendy Cheung Leonard Sio. *Voice Over IP*, 2004. ECE532 Group Report.

[5] National Semiconductor. *LM4550: AC 97 Rev 2.1 Multi-Channel Audio Codec with Stereo Headphone Amplifier, Sample Rate Conversion and National 3D Sound*, 2004.

[6] Brad Taylor. *2A03 Technical Reference*.

[7] Brad Taylor. *Delta Modulation Channel Tutorial 1.0.*

[8] Brad Taylor. *NES APU Sound Hardware Reference*.

[9] Brad Taylor. *The NES Sound Channel Guide 1.8.*

[10] Mike Wirthlin. *OPB AC97 Controller*. Brigham Young University.

[11] Xilinx. *OPB UART Lite (v1.00b)*, July 2004. Datasheet.

[12] Xilinx. *XUP Virtex-II Pro Development System Board Schematic*, November 2004.

[13] Xilinx. *Digital Clock Manager (DCM) Module (v1.00a)*, December 2005. Datasheet.

[14] Xilinx. *OPB Block RAM (BRAM) Interface Controller*, December 2005. Datasheet.

[15] Xilinx. *Block RAM (BRAM) Block (v1.00a)*, August 2006. Datasheet.

[16] Xilinx. *MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 8.2i*, August 2006.

[17] Xilinx. *Microprocessor Debug Module (MDM) (v2.00a)*, February 2006. Datasheet.

[18] Xilinx. *On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.10c)*, August 2006. Datasheet.

7 Appendix A: NES APU Register Map

This appendix describes the NES APU registers organization. The register of the implemented OBP APU IP are organized in the same manner.

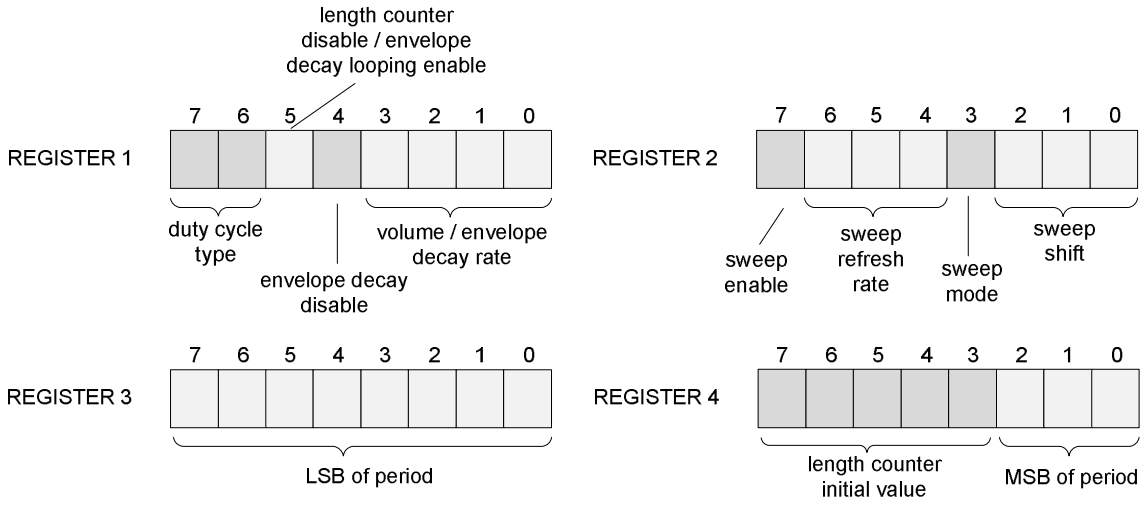

Figure 11: Rectangle Channels registers organization

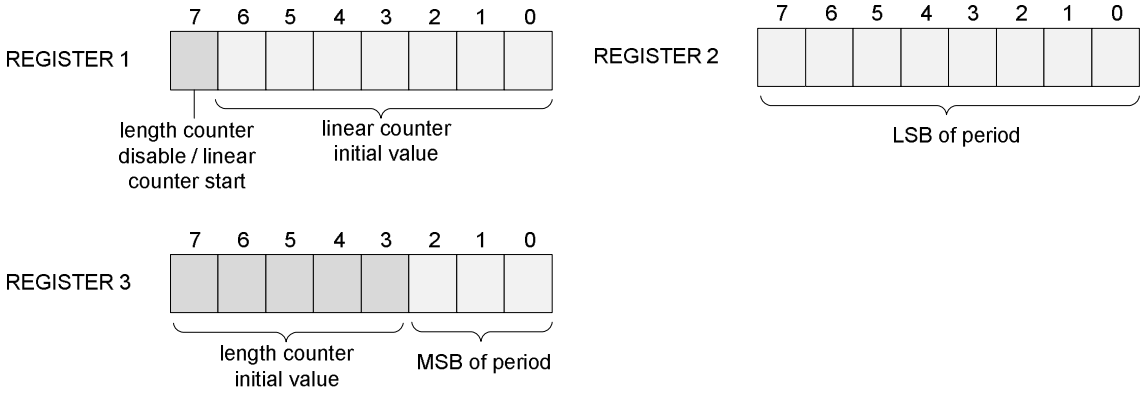

Figure 12: Triangle Channel registers organization

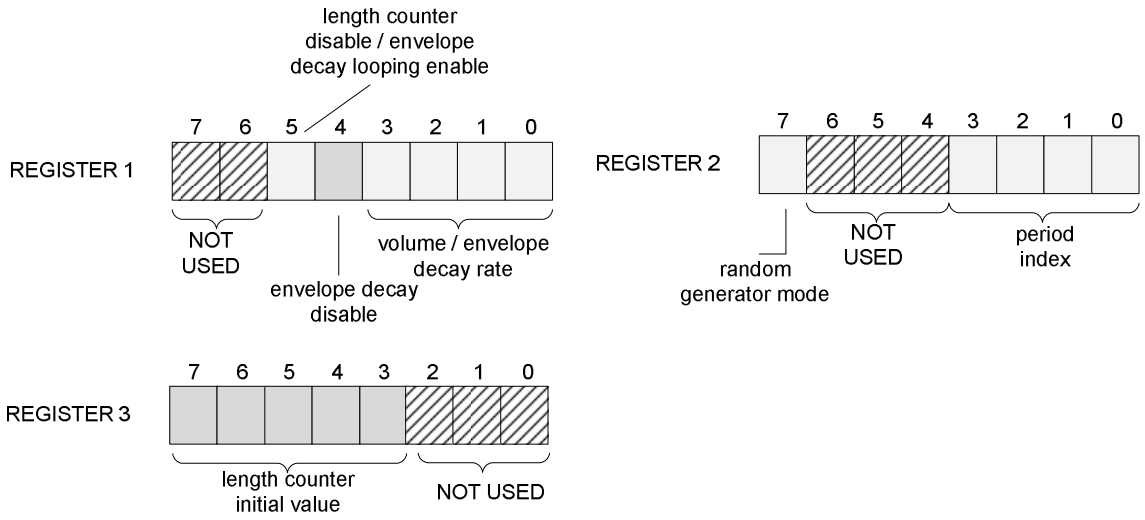

Figure 13: Noise Channel registers organization

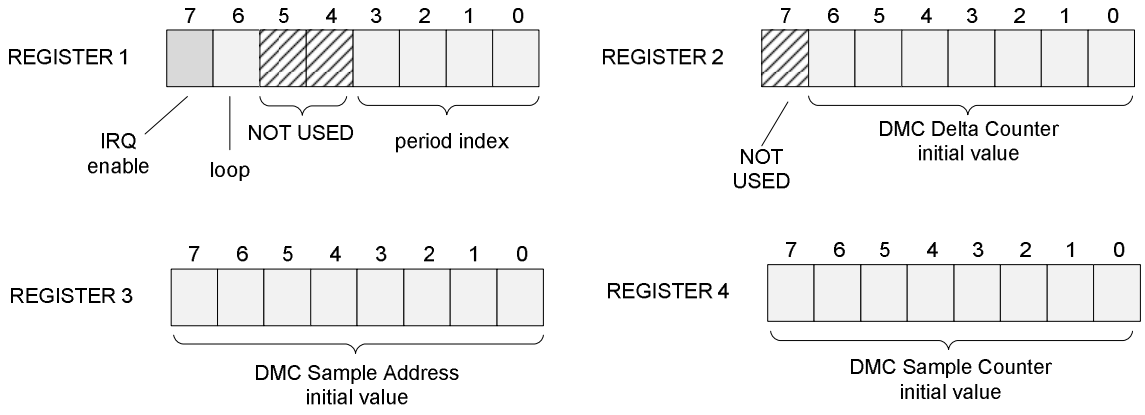

Figure 14: DMC Channel registers organization