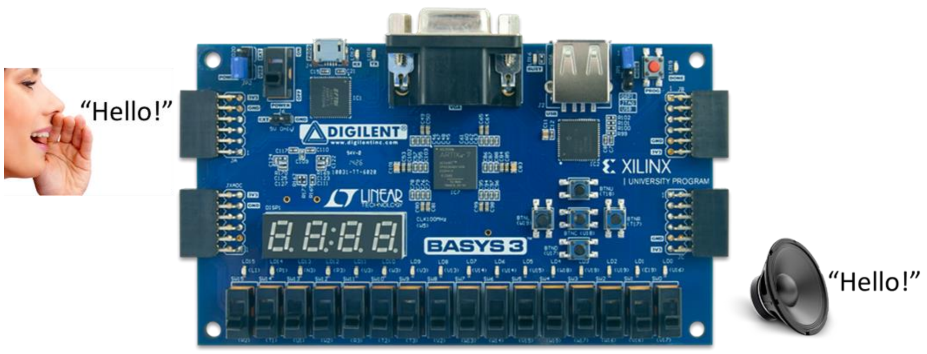
Department of Electrical & Computer Engineering

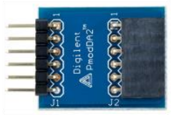
EE2020 Digital Fundamentals

FPGA Design Project: Real-Time Audio Effects

Abstract

As your EE2020 FPGA Design Project, you will create a real time audio effects machine! You will be provided with a MEMs microphone to capture human voice and an audio amplifier to output your signal through earphones. This manual introduces you to the various concepts involved, and guides (not walks!) you through getting an audio FX machine up and running.





Semester 1, AY 2017/2018	
EE2020 Lab Instructors	DR. CHUA DINGJUAN CHRISTOPHER LEE LAN CHONG GAO JIEYI

1. Introduction – System Overview

The block diagram of the overall system is as shown in Figure 1. It can be divided into four sub-system blocks, the Clock Generator (green box), the Audio Input Capturer (blue box), the Features Select (orange box) and the Audio Output Player (purple box). The entire system consists of three peripheral PMOD hardware components and the *AUDIO_FX_TOP.v* module on your BASYS3.

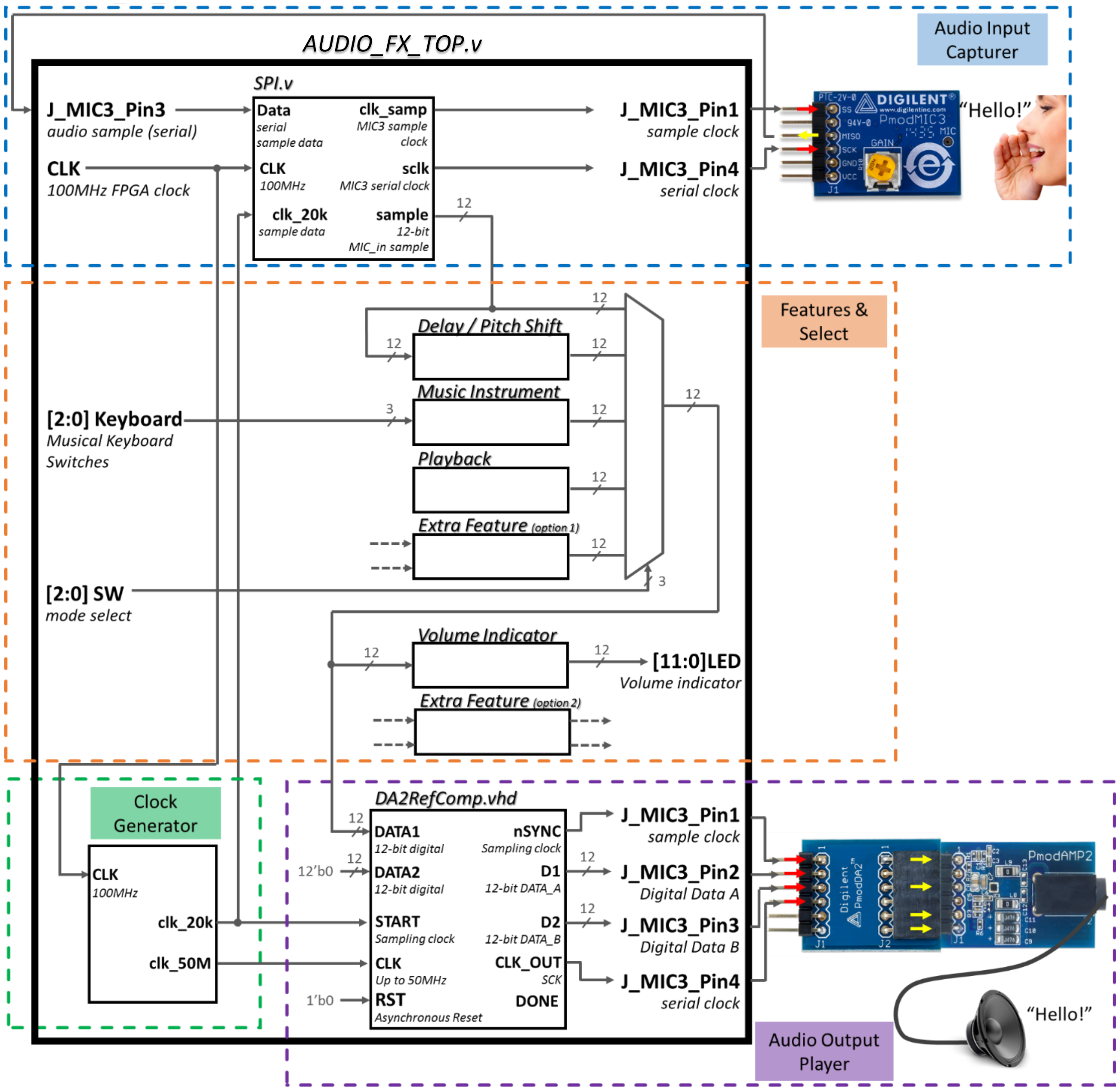


Figure 1. System block diagram of the project (EE2020)

2. Schedule & Weightage

This is a project that includes individual and pair-work components. The schedule and weightages for the various components in the different project weeks are listed below. The requirements of the tasks will be explained in detail in Section 4.

All of the features will be assessed during the project assessment week.

Week	EE2020 Tasks	Percentage
8	Teamwork: Real-time microphone-speaker system (capture voice from them PmodMIC3 and output at PmodAMP2).	5%
9	Student A: Volume indicator using LEDs. Student B: Electronic instrument using switches.	10% individually
10	Student A: Real-time delay in microphone-speaker system. Student B: Real-time playback of digitized male “hello” voice sample.	10% individually
11	Teamwork: System integration. One extra feature (open-ended).	10%
12	Project Assessment	

3. Design Files and Resources

The following design files accompany this project manual, and are available on IVLE. Open the archived Vivado project to get started.

Design Sources

AUDIO_FX_TOP.v: the top-module of the design. This will be your main module, or typically called the Top Level module, where you instantiate the sub-modules and make the necessary links between these modules. Within this module, the **SPI.v** and **DA2RefComp.vhd** modules have already been instantiated.
You are required to add code to provide the necessary signals for the SPI.v and DA2RefComp.vhd modules to work!

SPI.v: an interface module between the microphone and your design. This module converts the serial data input into a 12-bit parallel *MIC_in* data when provided with a sampling clock and a serial clock. Once added to your project, do not modify this code!

DA2RefComp.vhd: an interface module between your design and the output boards. This is a VHDL file that helps to translate your instructions into appropriate signals to the Pmod DA2 module board. DAC conversion transforms your digital bits into an analog signal which are then amplified and heard on your earphones. Once added to your project, do not modify this code!

Constraint Sources

Basys3_Master.xdc: a master constraints file that defines the I/O constraints for BASYS 3.

Other Resources

memory_demo.mp4: an instructional video to assist you in completing the real-time playback task. This video teaches you how to make use of Vivado's IP catalog to store and access data using a ROM (Read-Only-Memory) block on the FPGA.

hello20.wav: an audio male voice sample .wav file. This audio file contains a male voice saying "hello". The .wav file is digitized at a sample rate of 20kHz.

hello_data.coe: a data file containing the data set of the hello20.wav voice sample. This file contains the data set of a male "hello" voice sample digitized at 20kHz and scaled to 12-bits. This file is prepared in .coe format to be used in the Vivado 2016.2 IP Catalog Memory Generator Wizard.

4. Tasks & Requirements

4.1 Real-time Microphone-Speaker System

In this session, you will setup the microphone-speaker system using the PmodMIC3, PmodDA2, and PmodAMP2 with Basys3. Your system will capture and digitize the audio signal input from the microphone on the PmodMIC3, recover and play the audio signal at the output via the PmodDA2 and PmodAMP2 into a speaker or earphones.

PROJECT TASK 1

Audio Input Capturer – Convert Analog Audio Signal into 12-bit Digital Sample

We will be making use of PmodMIC3 to capture and sample the analog audio signal provided to the microphone. The analog-to-digital conversion (ADC) is done internally on the PmodMIC3 to produce a digital serial (bit-by-bit) output.

Please refer to Section 5.2 for background knowledge of (analog-to-digital conversion) ADC, and 5.4 and 5.5 for the details of Pmod Ports on Basys 3 and PmodMIC3.

By completing steps 1) to 4), you will obtain a 12-bit audio sample `[11:0] MIC_in` captured by the microphone and updated at every cycle of `clk_20k`.

1) **Create `clk_20k`.**
   Design a 20kHz clock signal and provide it to **SPI.v** instantiated in the top module.

2) **Attach the PmodMIC3 to one of the J-Ports on Basys 3 (e.g. JA).**
   IMPT *To avoid damaging the boards*, make sure the GND and VCC pins on the Pmod and Basys3 are connected correspondingly.

3) **Edit the constraints file.**
   Map signals `J_MIC3_Pin1`, `J_MIC3_Pin3`, `J_MIC3_Pin4` to the corresponding J-Port selected above.

4) **Generate your bitstream and observe MIC_in on LEDs.**
   Observe `[11:0] MIC_in` on the LEDs as you speak into the microphone. In addition, observe the effects of adjusting the volume control dial on the PmodMIC3.

Audio Output Player – Convert Digital Sample back into an Analog Signal

Make use of PmodDA2 to convert the digital data captured by the microphone into an analog form. The analog audio signal will then be amplified by PmodAMP2.

Please refer to Section 5.3 for background knowledge of digital-to-analog conversion (DAC), and 5.4, 5.6 and 5.7 for the details of Pmod Ports on Basys 3, PmodDA2 and PmodAMP2.

By completing steps 5) to 8), you will be able to convert the digital data `[11:0] MIC_in` captured by the microphone into a continuous analog signal at the PmodDA2 output Pin 1. This signal is amplified by the PmodAMP3 into the earphones or speaker.

5) **Create `clk_50M` (50 MHz).**
   DA2RefComp.vhd has been instantiated into the top module. This module requires `clk_50M`, `clk_20k`, and `speaker_out` as inputs. Design a 50MHz and 20kHz clock signal and provide it to DA2RefComp accordingly.
   For initial verification, you may use a clock signal of approximately 16Hz for `speaker_out`. Explore what happens when the frequency of this changes.

6) **Attach the PmodDA2 with one of the J-Ports on Basys 3. (e.g. JA).**
   Attach PmodAMP2 to the output ports of PmodDA2 (Figure 2).
   IMPT *To avoid damaging the boards*, make sure the GND and VCC pins on all boards are connected correspondingly.

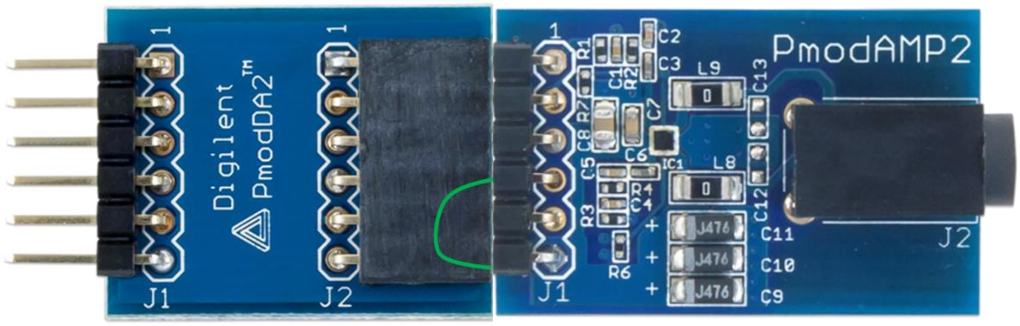


Figure 2. Connection between PmodDA2 and PmodAMP2

7) **Edit the constraint file.**
   Map signals `J_DA2_Pin1`, `J_DA2_Pin2`, `J_DA2_Pin3` and `J_DA2_Pin4` to the corresponding J-Port selected above..

8) **Generate bitstream.**
   DO NOT PLUG THE EARPHONES INTO YOUR EARS! The volume of the output signal may be VERY loud, observe the output cautiously.

Integration of Real-time Microphone-Speaker System on Basys 3 FPGA.

9) **Assign `MIC_in` to `speaker_out`.**
   The signal `[11:0] MIC_in` represents the data captured by the microphone.
   The signal `speaker_out` represents the digital signal to be converted to analog form for playing through the earphones / speaker. Connect the two signals together to create a real-time Microphone-Speaker System.

10) **Generate bitstream and upload it onto the FPGA.**

4.2(A) Electronic Musical Instrument

An electronic musical instrument is a musical instrument that produces sound using electronic circuitry and/or digital devices. In this task, we are going to build an electronic musical instrument on the Basys 3 FPGA.

To create an instrument, we would need to generate different pitches (Do, Re, Mi etc). A musical note is essentially a periodic signal at a specific frequency. Therefore, we can generate a periodic signal and by adjusting the frequency we can obtain different notes. The switches on the Basys can be used as the keys on the musical instrument and by turning on the different switches, different notes can be played at the output speaker/earphone.

PROJECT TASK 2 (A)

Name:____________________________

Create a basic musical instrument on the Basys 3.

1) **Create a basic instrument that is able to play musical notes of C (do), D (re), E (mi), F (fa), G (so) on your instrument.**

2) **In the context of a musical instrument, create an improvement for this feature.**
   For example, you can create different types of sounds or instruments.

Hint:

1) Research on the frequencies of the notes provide above.

2) Generate square waves of these frequencies, and map them to the switches.

3) Different types of periodic signals will produce different sounds.

3.2 (B) Audio Volume Indicator

An audio volume indicator displays a representation of the intensity of the audio signal. It helps people to observe the changes in the audio signal visually. An example is as shown in Figure 3.

In this task, we can use the LED array on Basys 3 as an audio volume indicator (if viewed vertically), displaying a digital value which reflects the peak volume of the signal at regular intervals.

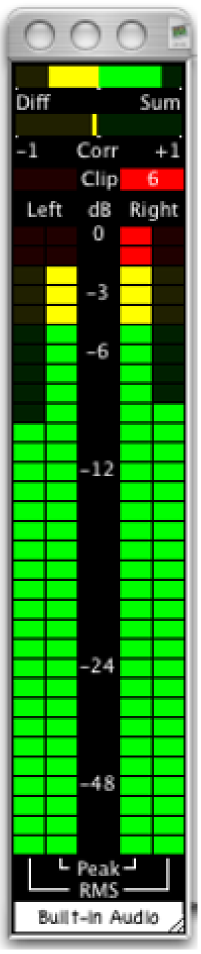


Figure 3. An example of an audio volume indicator

PROJECT TASK 2 (B)

Name:__________________________

Create a peak volume indicator for the microphone's input signal.

1) **Display the peak intensity of the volume by on the LED array on Basys 3 and refresh it at an observable rate.**

2) **In the context of a volume indicator, create an improvement for this feature.**
For example, you can display the volume in dB.

Hint:

1) The LED array should display information at a rate observable by human eyes. The voltage of an audio signal varies drastically as it is a periodic signal and you may need a register to record the peak value.

2) To use the LEDs as a linear display where the number of LEDs lighting up indicate the value directly, the binary value would need to be decoded.

4.3 Microphone-Speaker System with Delay and Pitch Shifting Functions

A. Delay

This task aims to create a time delay to the Microphone-Speaker system. That is, when we speak (for example, “Hello”) into the PmodMIC3, the “Hello” will be recovered and played out after a fixed time delay.

PROJECT TASK 3 (A)

Name:____________________________

Create a module that allows us to obtain a delayed input signal at the output.

1) Output the input audio signal by a 0.25 second delay.

2) In the context of a timed delay, create an improvement for this feature.
For example, output the input audio signal by variable delay time.
Alternatively, you can create a pitch shifter by introducing some changes in the codes created in this task.

Hints:
You may practise the tasks below before implementing the delay module.

1) Create a 2-dimensional array which can be used to store some number of audio samples.
For example, a `reg [7:0] memory [0:3]` represents an array that consists of four registers, that are indexed from 0 to 3. The size of each register is 8 bits (Figure 4).

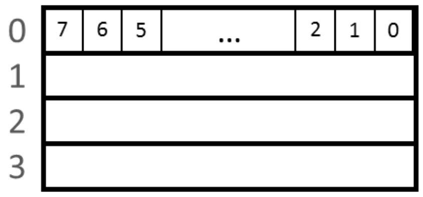


Figure 4. An example of a 2-dimensional register [7:0] memory [0:3]

2) Store the audio input sample into the memory by making use a “for” loop.

For example,
```
for (i=0; i<3; i=i+1) begin
    memory[i+1] <= memory[i];
end
```

Please take note that `i` here represents the index number of the registers in the array.
In the example, every clock cycle, contents in all registers are shifted by one location in the memory array (Figure 5).

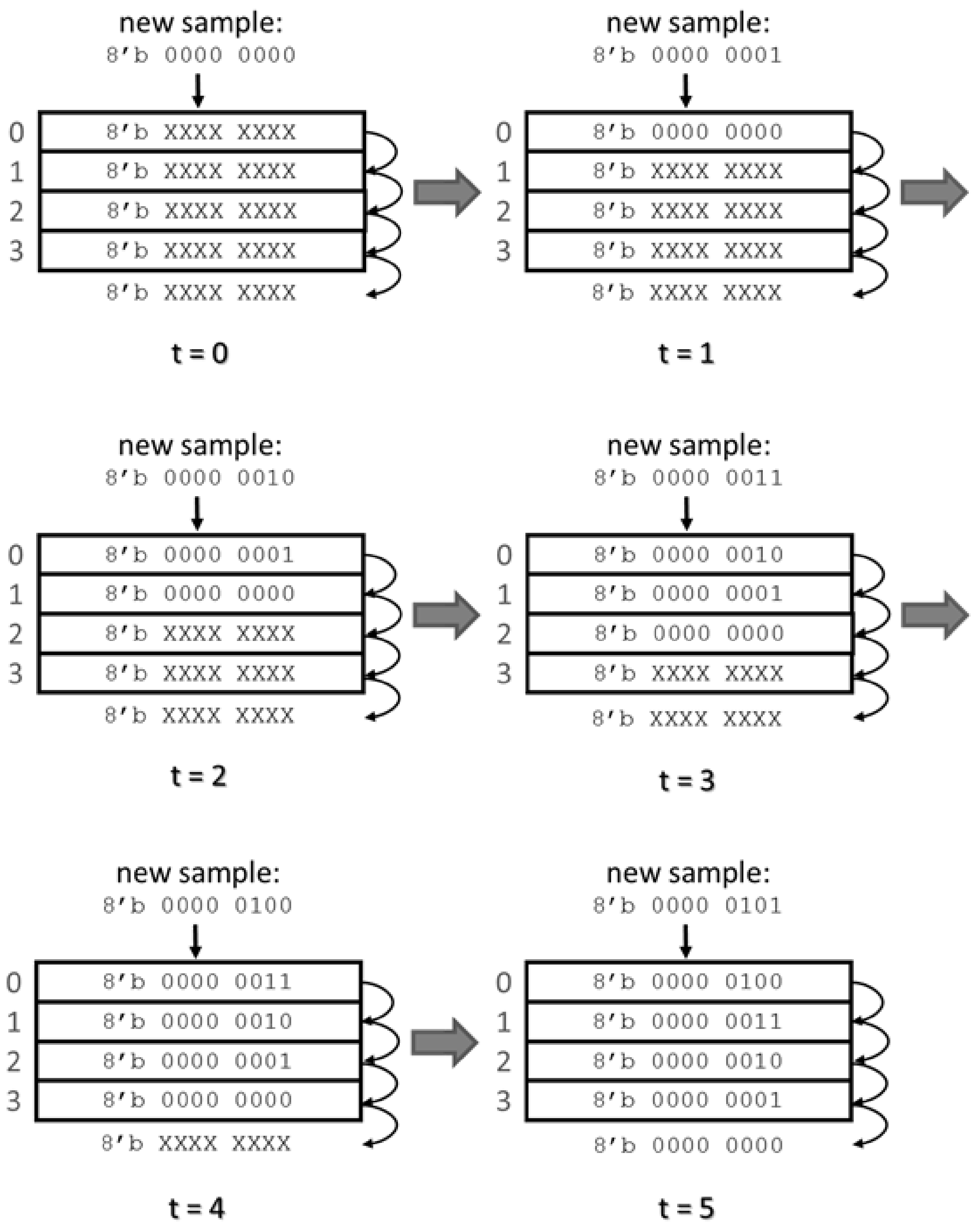


Figure 5. An example of shift registers – change of [7:0] memory [0:3] over time

3) Create a testbench for your code and simulate your code to ensure that the data samples are being shifted correctly. By using the for loop structure, you would be able to obtain a delayed sample from the memory.

B. Playback "Hello"

Apart from playing an audio signal that is received by the microphone in real-time, an audio signal can also be stored on the FPGA and played through the audio amplifier.

Go through **memory_demo.mp4** which guides you on how to use the Vivado IP Catalog to generate a Read-Only-Memory (ROM) circuit block for storing data.

In this task, you are storing "hello" audio data provided in **hello_data.coe** in a ROM memory block using the IP Catalog. By making use of this memory module, you can playback the "Hello" sound continuously.

PROJECT TASK 3 (B)

Name:__________________________

Store "Hello" data into FPGA ROM. Play back the "Hello" by accessing the memory.

1) **Make use of hello_data.coe to create a ROM memory module through IP Catalog.**

2) **Access the memory to obtain the audio data. Play the data continuously (on a loop).**
   The sampling rate of **hello_data.coe is** 20 kHz.

3) **Make additional improvements to this feature.**
   For example, introduce a pause/resume of the playing by a pushbutton.

Hints:
You may practise the tasks below before implementing the delay module.
1) Watch **memory_demo.mp4** and follow the steps to create a 16-by-8 (depth-by-length) memory (Figure 6). Store some arbitrary data into the memory. Generate a memory circuit.

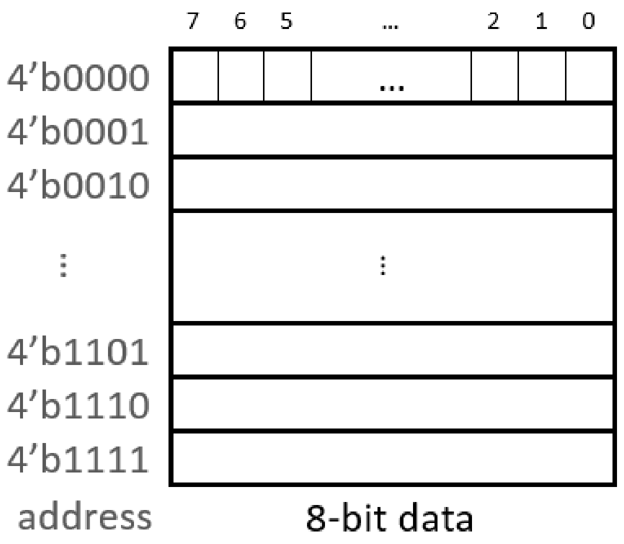


Figure 6. A 16-by-8 memory (with 16 registers and each register has 8 bits)

2) Instantiate the memory circuit into a new module with one input and one output. Obtain the memory data from output by providing in its address as an input. Simulation is very helpful in examining your memory data.

4.4 System Integration

Working together with your partner, integrate the multiple features into the system using switches / pushbuttons for basic feature selection. Ensure that your audio FX machine is easy to use and user-friendly.

4.5. One Extra Feature

As part of the project requirements, you are required to implement ONE additional feature to add value to your audio FX machine and distinguish your product from the rest. You are allowed to make use of the other peripherals available on the Basys 3 to make the project – your own product – more interesting/functional/user-friendly.

This additional feature is open-ended and you will be evaluated on the metrics of quality, complexity, functionality, creativity, functionality and uniqueness.

4.6. Project Report

You will be required to submit a product summary for your project and more details will be released at a later date.

5. Appendix

5.1. Human Voice Frequency Band & Hearing Range

The human ear is able to hear a range of frequencies from 20Hz to 20kHz.

However, the range of frequencies in the human voice band is actually of a much smaller frequency range. It ranges from 85Hz to 180Hz for a typical adult male and from 165Hz to 255Hz for a typical adult female.

The sampling rate of any signal needs to be a minimum of twice the frequency of the signal, in order to digitize and recover the signal sufficiently (Nyquist Theorem). Thus, the typical sampling frequency used in audio CDs is 44.1kHz which accommodates a maximum input frequency of 20kHz.

However, as we are only dealing with real-time voice signals, we are using a lower sampling rate of 20kHz which is able to pass through voice and other signals sufficiently.

5.2. Analog to Digital Conversion (ADC)

Physical quantities (or signals) in the real world are analog – sound, light, temperature, voltage, current etc. However, analog signals cannot be stored or processed by the electronic devices, like Basys 3 FPGA. All computers and modern electronic systems are digital in nature. The digital signals, representing by a series of “1”s and “0”s, have discrete amplitude values and are defined at discrete points in time. They are easy to store and manipulate (compress, filter, enhance, etc). Analog to digital conversion (ADC) is the process of converting a continuous-voltage continuous-time representation of the signal into a digital one.

The process of ADC is as illustrated below, including **sampling** and quantization.

Figure 7 shows an analog signal (e.g. an audio signal) in a shape of a sine waveform. The frequency of the periodic wave is f_a= 1Hz. The amplitude is 1V peak-to-peak, and the signal has a dc offset of 0.5V in order to keep the signal unipolar and simplify things. Observe that an analog signal is continuous in both time and amplitude. In order to perform analog-to-digital conversion, the first step is to **sample** the signal at discrete points in time. In Figure 7, a **sampling frequency**[1] of f_s = 10Hz is employed. This means 10 samples (green dots in Figure 7) of data from the sine wave are taken every second. This results in a discrete-time signal as shown in Figure 8.

[1] Since our analog signal is 1Hz, observe this results in obtaining 10 samples / cycle.

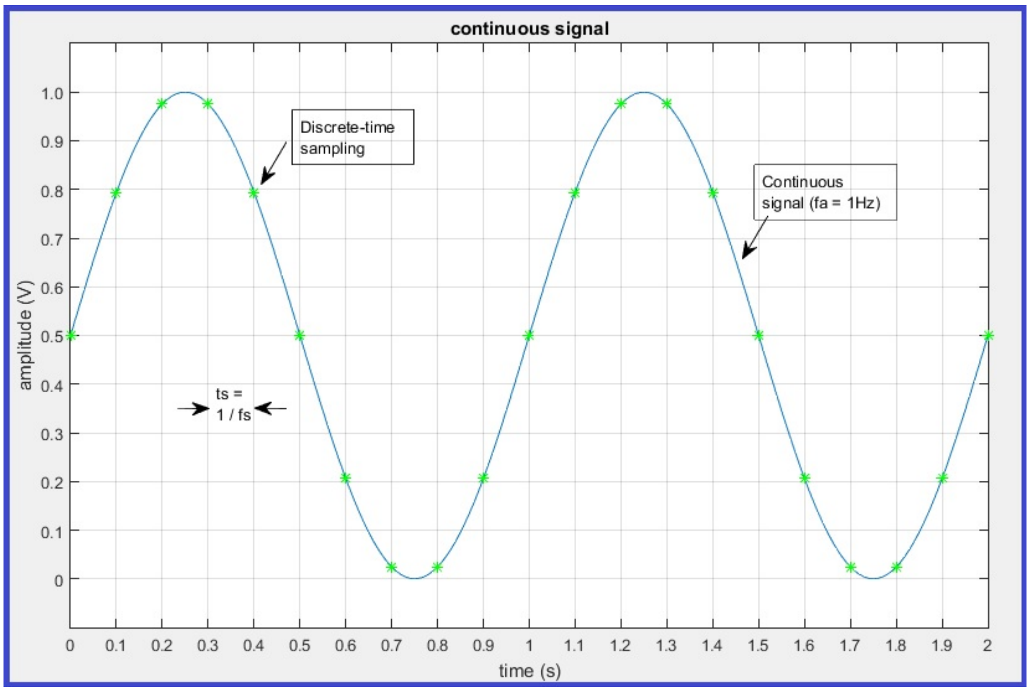


Figure 7. An analog waveform of 1Hz. It is to be sampled at f_s = 10Hz, i.e. 10 samples/second.

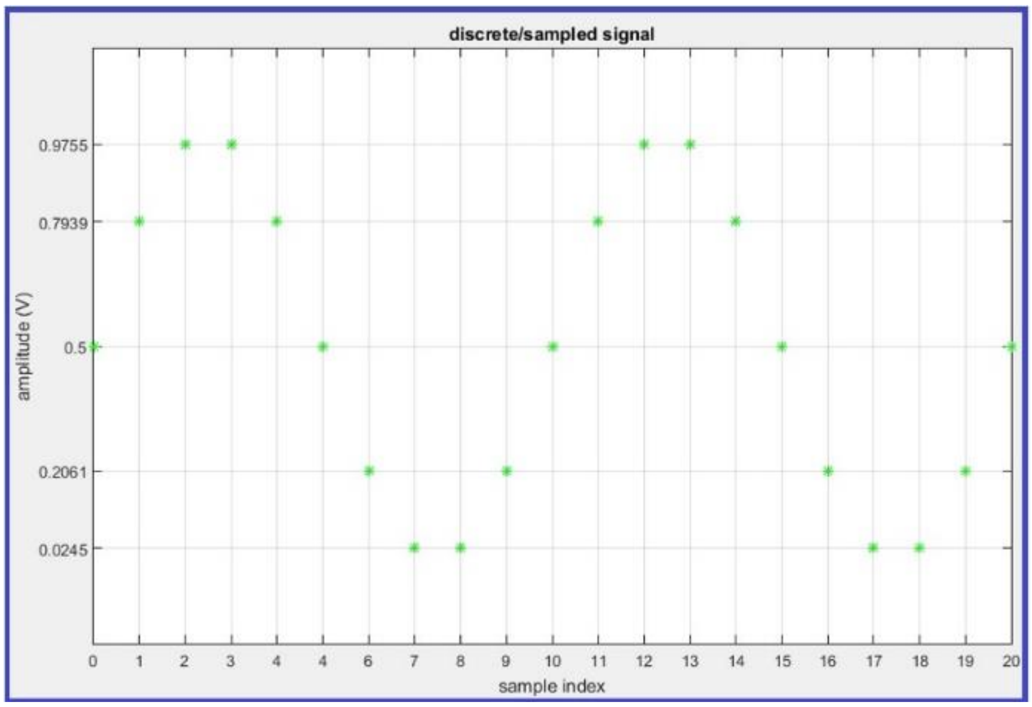


Figure 8. Discrete-time signal at sampling rate of f_s = 10Hz

Now that the horizontal axis has been discretized, the next step in the A2D conversion process is to discretize the vertical axis. This is also known as **quantization**. Quantization is to represent the continuous amplitudes in the discrete signal using the nearest digital value available. Given an 8-bit ADC, we have $2^8 = 256$ possible digital values. The continuous values {0V, 1V} are mapped to {0, 255} and intermediate values are mapped to the nearest integer value between 0 and 255. The resulting **digital signal** is shown in Figure 9.

Quantization incurs an error due to the difference between original value and its discrete representation. The more bits (higher resolution), the more discrete levels of representation it provides, and the less error is incurred.

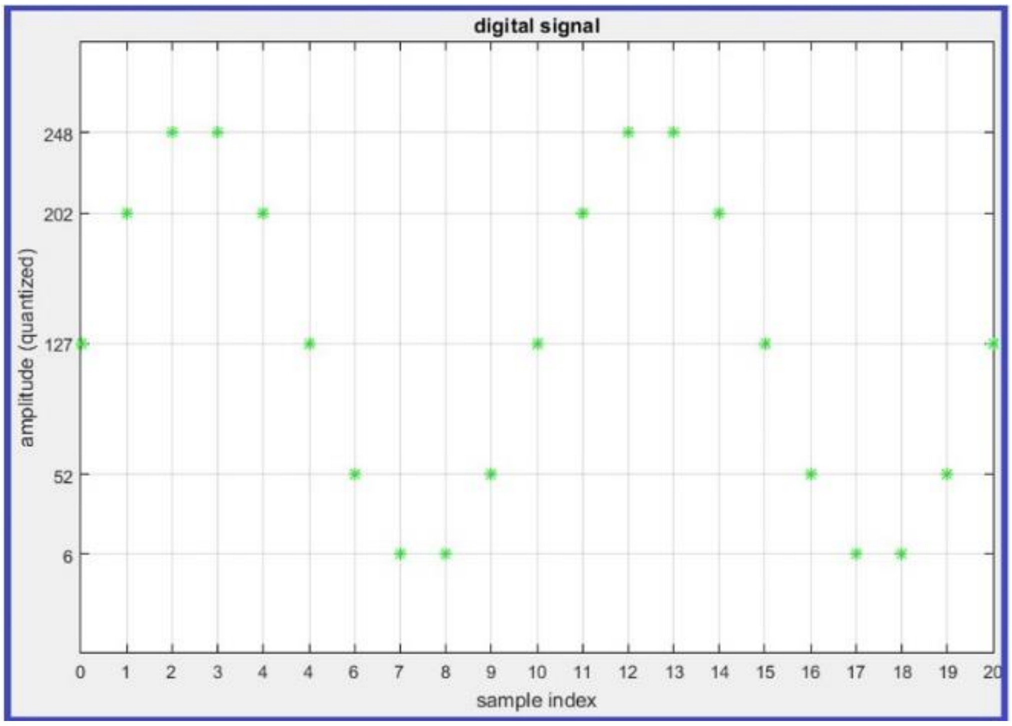


Figure 9. Digital signal quantized by 8 bits at sampling rate of f_s = 10Hz

Figures 10 and 11 show discrete-time signals when the sampling rate is increased to 20 and 100 samples per second respectively.

In the **discrete signals**, please take note that the horizontal axis is indexed by the numbering of the samples, rather than time.

The increase in sampling frequency improves the similarity between the digitized signal and the original analog signal.

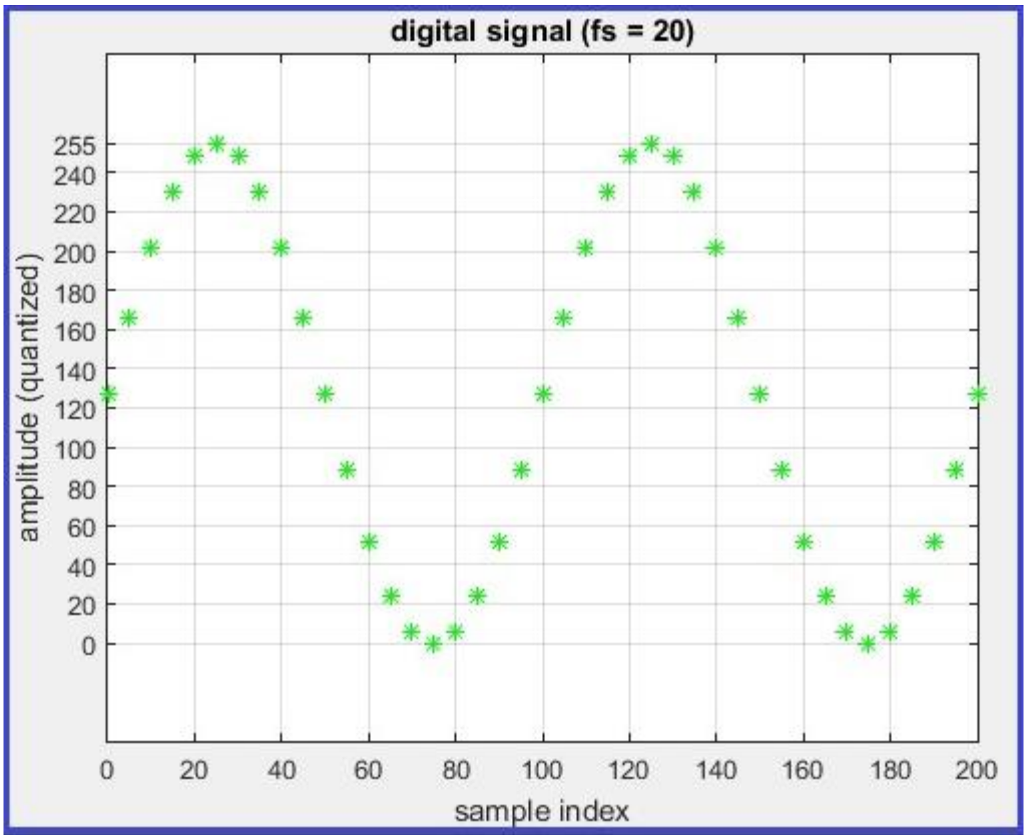


Figure 10. Digital signal quantized by 8 bits at sampling rate of f_s = 20Hz

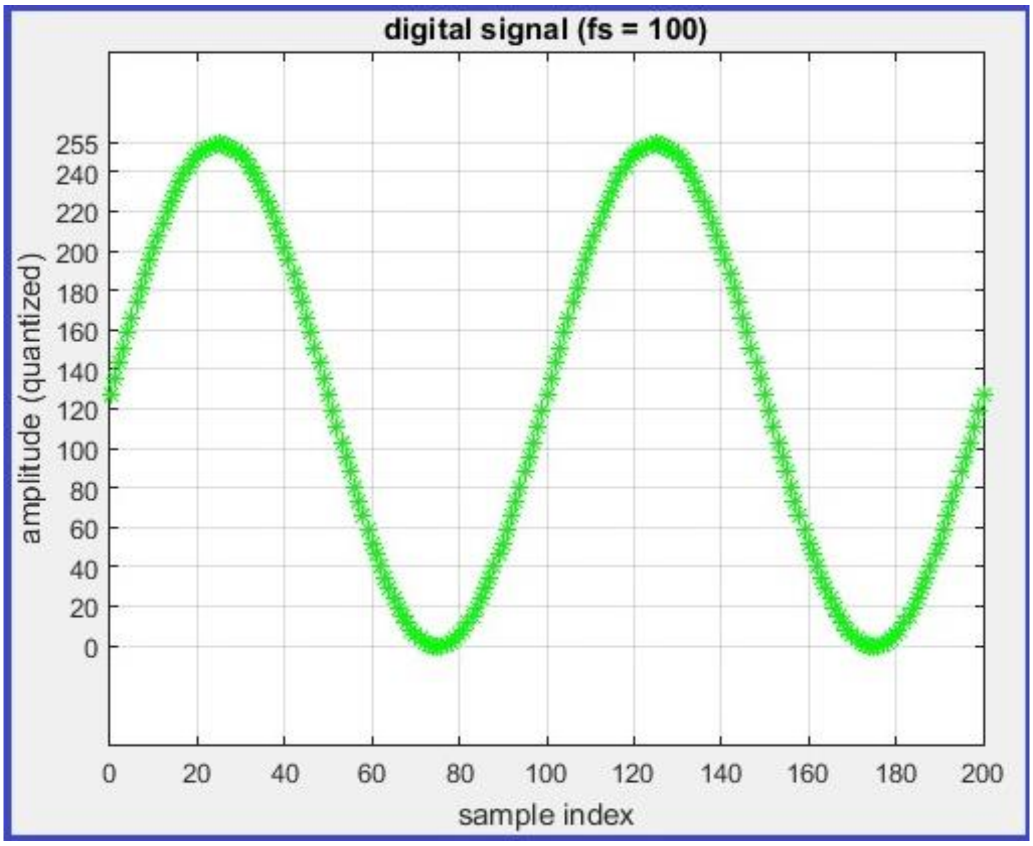


Figure 11. Digital signal quantized by 8 bits at sampling rate of f_s = 100Hz

In this project, the ADC is done via an ADCS7476 12-bit Analog-to-Digital Converter on PmodMIC3. PmodMIC3 working as a voice capturer (Figure 1) obtains analog voice input from its on-chip microphone and takes up to 1 million samples per second (MSPS) based on the sampling clock. Each sample is quantized and output as a 12-bit digital data.

5.3. Digital to Analog Conversion (ADC)

The reverse process of DAC is Digital to Analog Conversion (ADC).

To recover a human voice, the digital signal needs to be converted to analog before outputting at a speaker. The human voice is captured by PmodMIC3 and is output in a digital form.

Similar to ADC, the process of ADC consists of two steps. To convert discrete amplitude representing by bits to an analog number, and to convert the discrete time signal into a continuous form.

One possible design of a DAC is shown in Figure 12. It consists of a digital decoder and an analog voltage divider which converts a 3-bit binary number into an analog value.

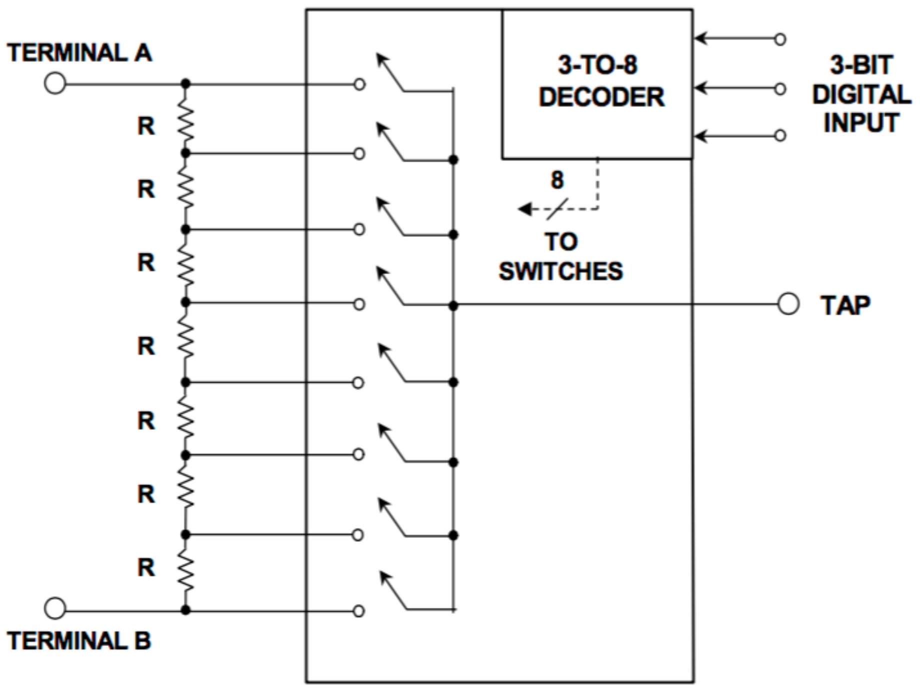


Figure 12. Configuration of a 3-bit DAC, consisting of a 3-to-8 decoder and a voltage divider

The 3-bit input *n* (eg. “010” or 2 in decimal) goes into a 3-to-8 decoder. The corresponding *nth* output pin (2nd pin) will become high. This will turn on the switch connecting between the decoder output pin and the analog voltage divider.

Since 8 decoder outputs are used, the same amount of identical resistors are connected in series between Vcc and GND. Ideally, a voltage of V_R (= Vcc/8) is evenly distributed to each resistor. As the *nth* switch (2nd) controlled by the decoder is on, the voltage connected to port *n (port 2)* will become the output of the DAC.

The resulting output can be expressed by Vout = Vcc * (*n*/8) = *n* * V_R. Table 1 shows the input and the correspondent output of a 3-bit DAC.

3-bit Digital Input	n (in decimal)	DAC Output
000	0	0 * V_R = Vcc * 0 / 8
001	1	1 * V_R = Vcc * 1 / 8
010	2	2 * V_R = Vcc * 2 / 8
011	3	3 * V_R = Vcc * 3 / 8
100	4	4 * V_R = Vcc * 4 / 8
101	5	5 * V_R = Vcc * 5 / 8
110	6	6 * V_R = Vcc * 6 / 8
111	7	7 * V_R = Vcc * 7 / 8

Table 1. (Digital) input and (analog) output of a 3-bit DAC

Figure 9 is a discrete analog signal, generated at a sampling frequency of 10Hz (i.e. 10 sample points/second). This signal is an input of the DAC, a sample comes in every 0.1 second. If the same sampling clock (10Hz) is applied to the DAC, the DAC reads an input sample at every 0.1 second. In one second, all the 10 points are captured and converted to analog by the DAC, as shown in Figure 13.

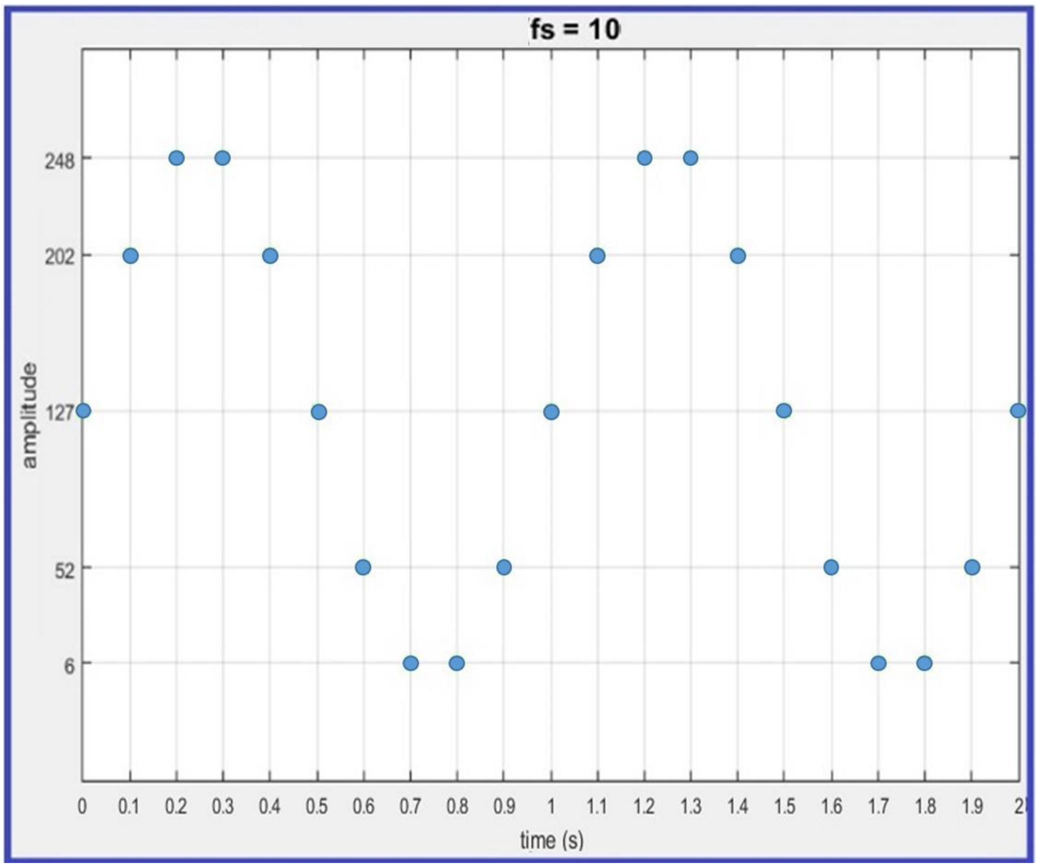


Figure 13. DAC capture inputs (10Hz) at sampling frequency fs=10Hz

By adjusting the DAC clock frequency, the DAC sampling rate can be adjusted accordingly. For example, the DAC clock is running at a frequency of 2Hz, which is one-fifth of the input sampling rate (10Hz) generated in Figure 9. The DAC reads and captures sample every 0.5 second. Only two out of ten input samples are captured every second (blue dots in Figure 14), while the other eight data (green dots in Figure 14) are missing due to the difference between the input and DAC sampling frequencies.

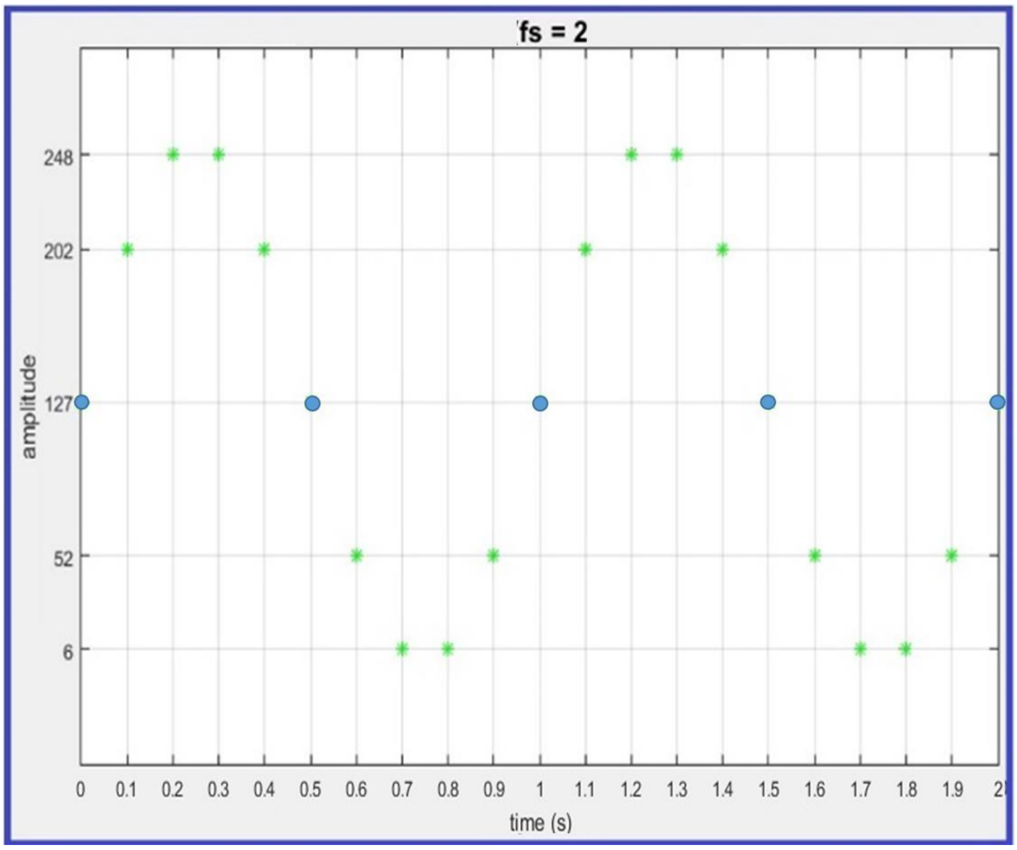


Figure 14. DAC capture inputs (10Hz) at sampling frequency fs=2Hz

Therefore, the sampling frequency of the input data and the DAC's sampling frequency need to be carefully selected. The same sampling clock is suggested to be used for input data generator and the DAC. This is to avoid distortion due to data missing or misalignment between data input to DAC and data capturing by DAC.

Up to this point, the digital signal has been converted into an analog form. However, the signal is still discrete. To convert a discrete signal into a continuous analog signal, a zero-order holder is used to hold the value until the next update (i.e. the next DAC clock cycle).

Figure 15 shows a 10Hz discrete analog signal, converted by DAC at a sampling frequency of 10Hz (i.e. 10 sample points/second). The value of each sample is being held for an entire sampling period of 0.1s (= 1 / 10Hz).

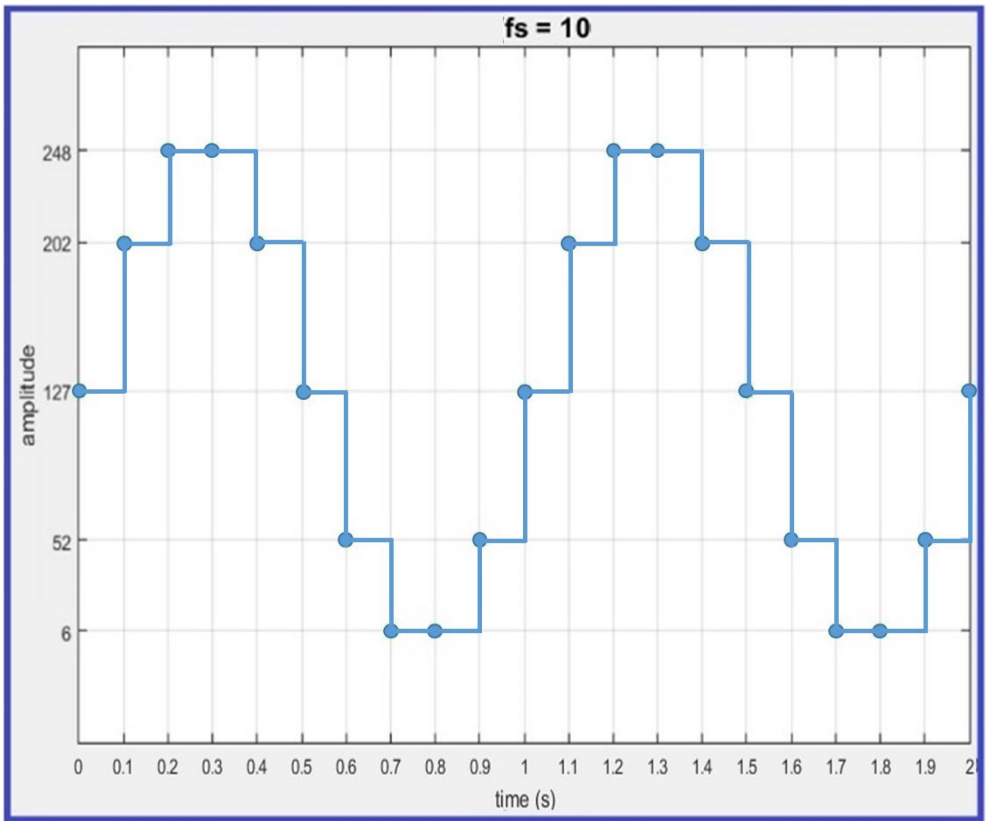


Figure 15. DAC recovered signal (sampling at 10Hz) of a 1Hz sinosoidal analog signal

The recovered signal output from DAC may not be as smooth as the original sinusoidal curve as shown in Figure 7. A relatively smoother curve can be produced if the DAC is able to capture a more accurate sample and covert it at a higher frequency. That is, the DAC output can be improved if the samples are input as shown in Figure 10 and the DAC runs at 20Hz. It can be further improved by increasing the input and conversion rate at 100Hz according to Figure 11.

In addition, by Nyquist Theorem, the frequency of the DAC clock is suggested to be at least twice of the signal frequency expected to be output from the DAC.

5.4. JA, JB and JC Pmod Ports on Basys 3

There are three sets of 12-pin Pmod Ports on the Basys 3 (JA, JB, and JC in red boxes as shown in Figure 16). They are for connecting and communicating with external devices. The Basys 3 Pmod pin assignments are as shown in Figure 16.

There are four pins on each Pmod port reserving for two gounds (eg. JA5 and JA11) and two 3.3V VCC signals (eg. JA6 and JA12) respectively. Another eight pins (eg. JA1 to JA4 and JA7 to JA10) can be configured freely as inputs or outputs to meet our need.

Signals from external devices can be read by Basys 3 by making connection between the external devices and the configured input pins. Basys 3 can output signals to external devices via configured pins. The configuration of input/output pins on Pmod ports can be done via the constrait file.

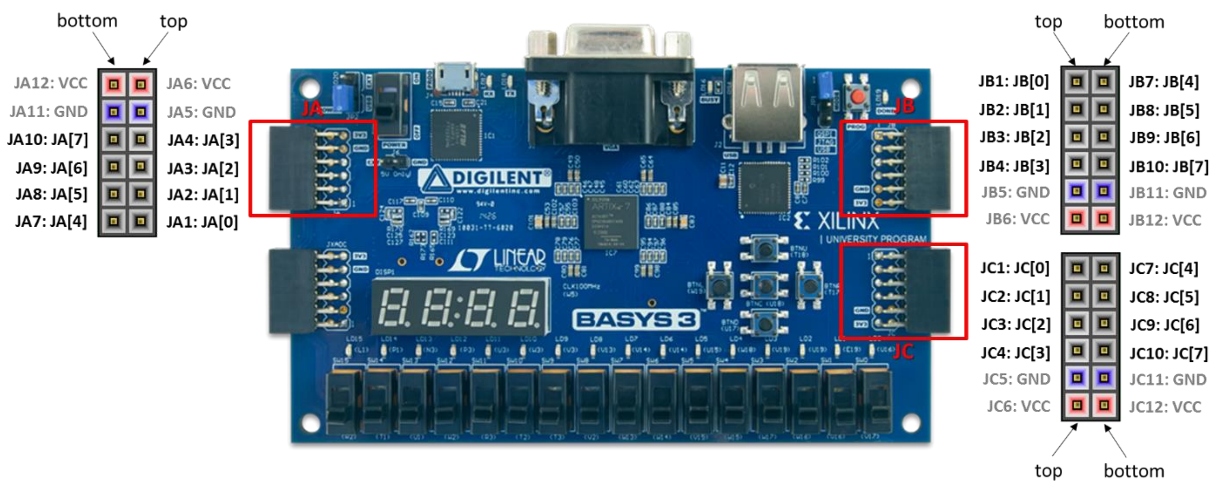


Figure 16. Basys 3 Pmod Pins Assignment

Digilent produces a large collection of Pmod accessory boards that can attach to the Pmod expansion ports to add ready-made functions such as PmodMIC3, PmodDA2 and PmodAMP2 that we are going to use in this project to function as a microphone (audio capturer), a digital-to-anlog converter and an amplifer respectively. They can be simply plugged in with the Basys 3 Pmod ports.

For more information, please refer to the Digilent Basys3 TM FPGA Board Reference Manual.
https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf

5.5 PmodMIC3

The Digilent PmodMIC3 functioning as a microphone is designed to digitally report to the host board (Basys 3) whenever it detects any external noise. By sending a 12-bit digital value representative of frequency and volume of the noise, this number can be processed by the system board and have the received sound accurately reproduced through a speaker. The on-board potentiometer can be used to modify the gain from the microphone into the Analog-to-Digtal Conversion (ADC). Please refer to 5.1 for more details about ADC.

The pin configuration of PmodMIC3 is as shown in Figure 17.

The audio signal will be sampled according to a sampling clock coming at Pin 1 (`clk_20k` in this project). Each sample will be converted into a 16-bit data, which includes 4 bits of leading zeros followed by 12 bits of sample data. The 16-bit data will be obtained at Pin 3 serially (bit by bit).

By using the same sampling clock `clk_20k` and 100MHz FPGA `CLK`, the Verilog module **SPI.v** generates a serial clock (up to 1MHz) which will be assigned to Pin 4. The each bit of the audio data output from Pin 3 by one cycle of the serial clock. **SPI.v** shifts the serial audio data bit by bit into a 16-bit register `[15:0]temp` by 16 serial clock cycles. The least significant 12 bits of `temp` contains information of the audio sample.

PmodMIC3 and **SPI.v** works together to form a voice capturer. The audio signal captured by microphone will produce a 12-bit audio sample `[11:0] MIC_in` every sampling cycle of `clk_20k`.

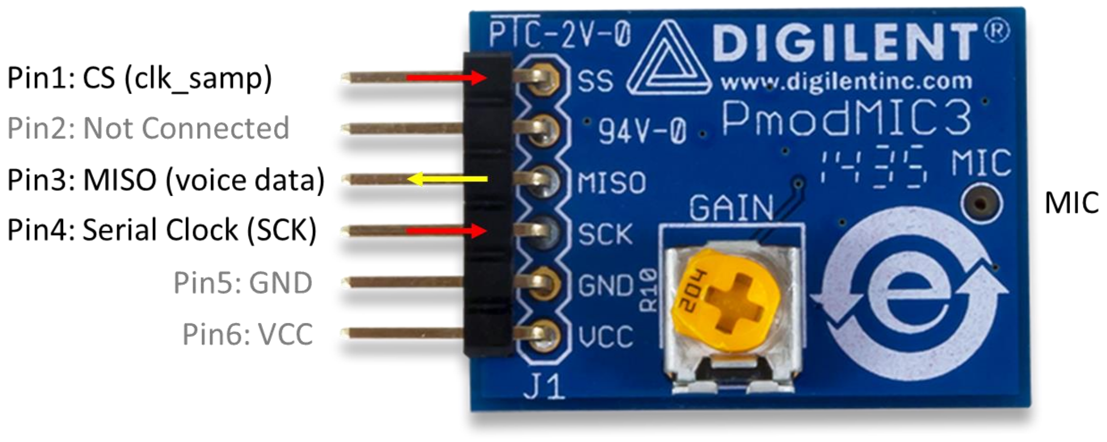


Figure 17. Digilent PmodMIC3 pin configuration

For more information, please refer to the Digilent PmodMIC3 Reference Manual.

https://reference.digilentinc.com/pmod/pmod/mic3/ref_manual

5.6 PmodDA2

The Digilent PmodDA2 board as shown in Figure 18 is a Digital-to-Analog converter (DAC) peripheral board compatible with the Basys 3. The PmodDA2 makes use of two 12-bit Digital-to-Analog Converter chips (Texas Instruments DAC121S101) to convert digital signals to analog signals. It is able to simultaneously convert two separate channels of digital signals (Data_A and Data_B) into two separate channels of analog signals (Channel A and Channel B).

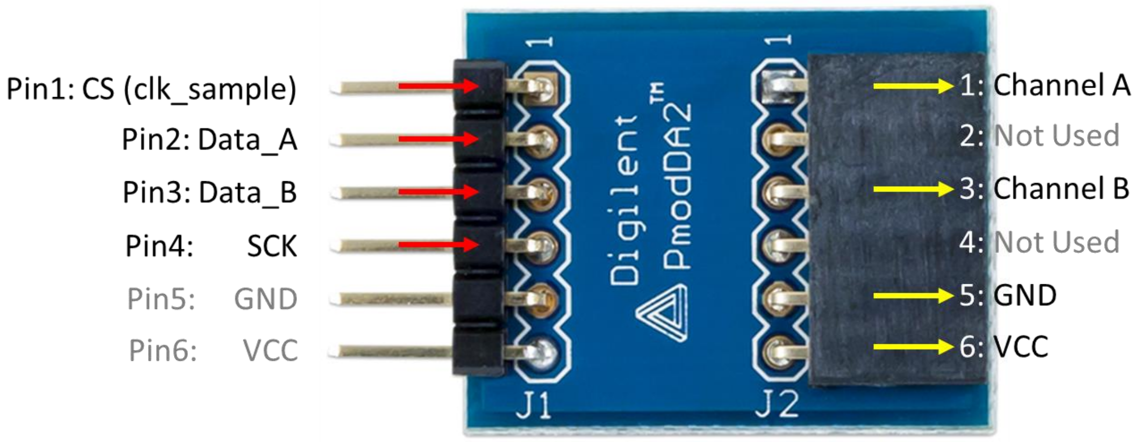


Figure 18. Digilent PmodDA2 pin configuration

By providing necessary inputs to the Verilog module**, DA2RefComp.vhd** produces signals to Pin1 to Pin 4 of PmodDA2. The output signals of PmodDA2 will be ready at the output ports (marked with yellow arrows in Figure 18).

For more information, please refer to the Digilent PmodDA2 Reference Manual.

https://reference.digilentinc.com/pmod/pmod/da2/ref_manual

5.7 PmodAMP2

The Digilent PmodAMP2 amplifies low power audio signals to drive a monophonic output. This module offers a digital gain select to allow output at a 6 or 12 dB gain with pop-and-click suppression. The analog audio output can be played via a speaker or an earphone, by connecting it with the audio jack on PmodAMP2. Figure 19 shows the pins assignment of PmodAMP2.

By connecting Pin 4 and Pin 6, PmodAMP2 is set to ON mode (L.Shutdown = H).

By inputting the audio signal to Pin 1, PmodAMP2 amplifies the signal and produces it through the audio jack.

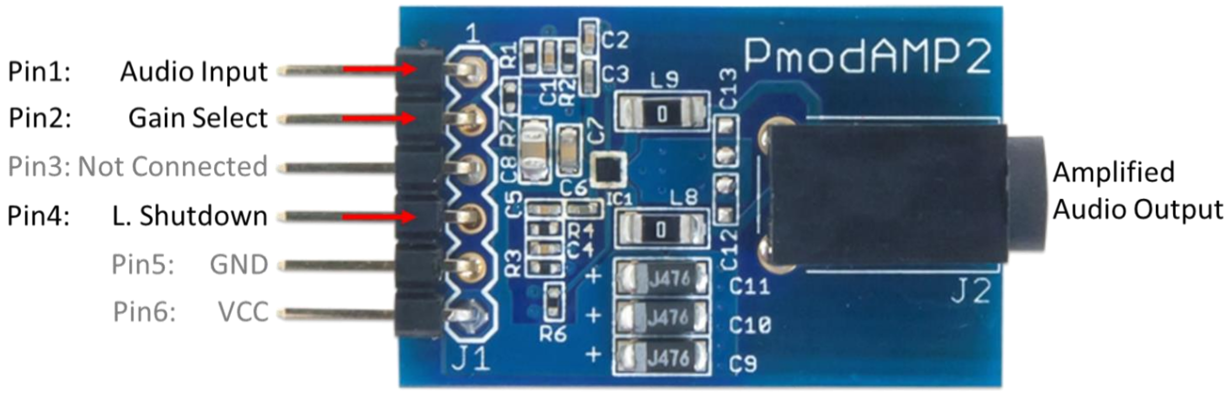


Figure 19. Digilent PmodAMP2 pins configuration

For more information, please refer to the Digilent PmodAMP2 Reference Manual.

https://reference.digilentinc.com/pmod/pmod/amp2/ref_manual

5.8 Distributed Memory Generator

Please refer to the Xillinx Distributed Memory Generator v8.0 for more information.

https://www.xilinx.com/support/documentation/ip_documentation/dist_mem_gen/v8_0/pg063-dist-mem-gen.pdf