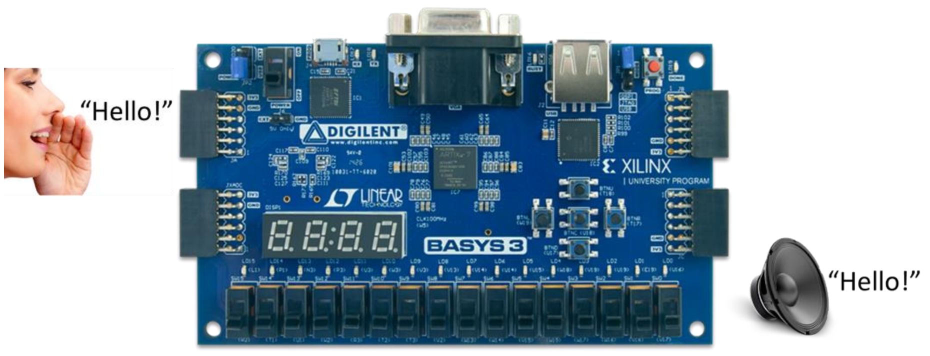
Department of Electrical & Computer Engineering

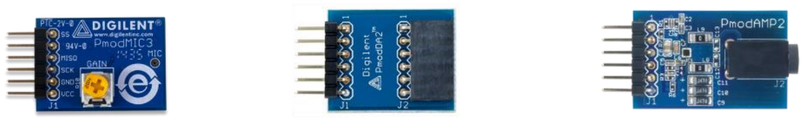
EE2026 Digital Fundamentals

FPGA Design Project: Real-Time Audio Effects

Abstract

As your EE2026 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 2, AY 2017/2018	
EE2026/EE2020/TEE2020 Lab Instructors	DR. CHUA DINGJUAN DR. GU JING CHRISTOPHER 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.

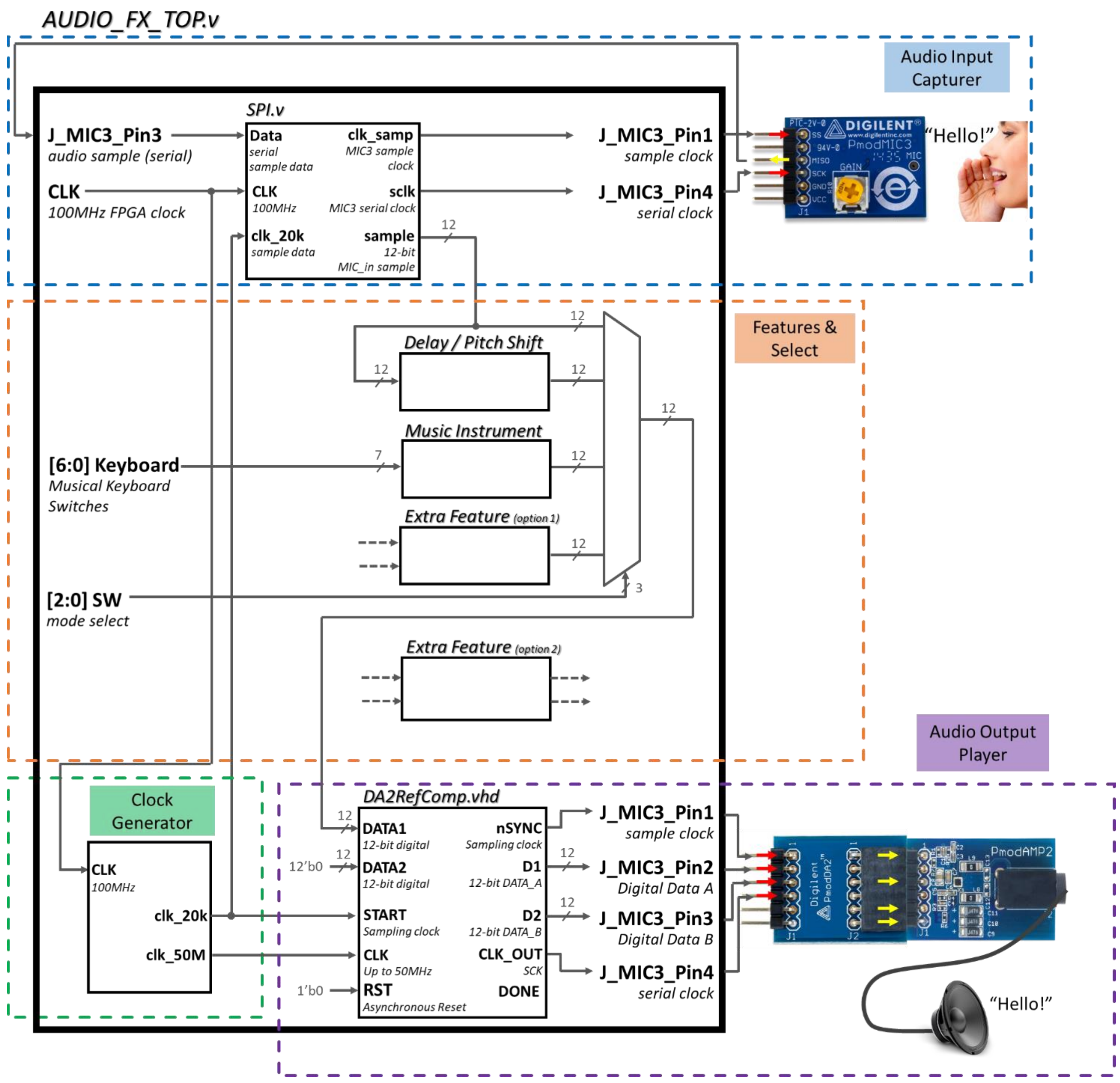


Figure 1. System block diagram of the project

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 detailed in Section 4.

All of the features will be assessed during the project assessment in week 12.

Week	EE2026 Tasks	Percentage
8	Teamwork: Real-time microphone-speaker system (capture voice from them PmodMIC3 and output at PmodAMP2).	5%
9	Student A: Real-time delay in microphone-speaker system. Student B: Electronic Music Instrument.	20% individually
11	Teamwork: System integration. One extra feature (open-ended).	10%
12	Project Assessment	

Table 1. Project schedule and mark percentage breakdown

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.

Module Wiki

tiny.cc/ee2026wiki

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.

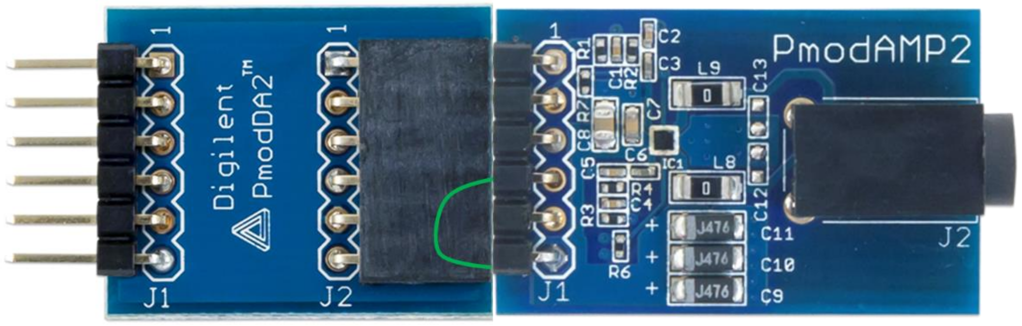


Figure 2. Connection between PmodDA2 and PmodAMP2

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 the bitstream and upload it onto the FPGA.**

4.2 Individual Components

A. Microphone-Speaker System with 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 2 (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 1 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 new module.
Create a 2-dimensional array which can be used to store some number of audio samples.
For example, a `reg [7:0] memory [0:15]` represents an array that consists of four registers, that are indexed from 0 to 15. The size of each register is 8 bits (Figure 4).

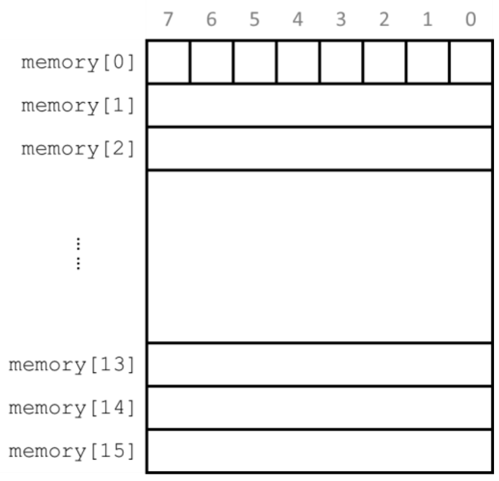


Figure 4. An example of a 2-dimensional register `[7:0] memory [0:15]`

2) Store/write the input data into the `memory[i]` by making using of an index parameter `i`.
For example,

```
reg [3:0]i = 0;

always @ (posedge clk_write) begin
    memory[i] <= data_in;
end
```

Write a simulation testbench to assign a series of values “8’d0, 8’d1, 8’d2, …, 8’d16, 8’d17, 8’d18…” to data_in every clk_write cycle.
How do you expect [7:0] memory [0:15] to be updated across the time?
Run simulation, observe the waveforms and verify your answer to the above question.

3) From the observation, every clock cycle, the content in a register memory[i] is updated and will maintain until the index i goes one round and comes back to the same location.

   To easily understand, the [7:0] memory [0:15] is comprehended as a circular by placing memory[0] and memory[15] next to each other (Figure 5).

   The change of circular buffer over time is as shown in Figure 6 (data_in is written into location memory[i] according to the blue arrow at the outer circle).

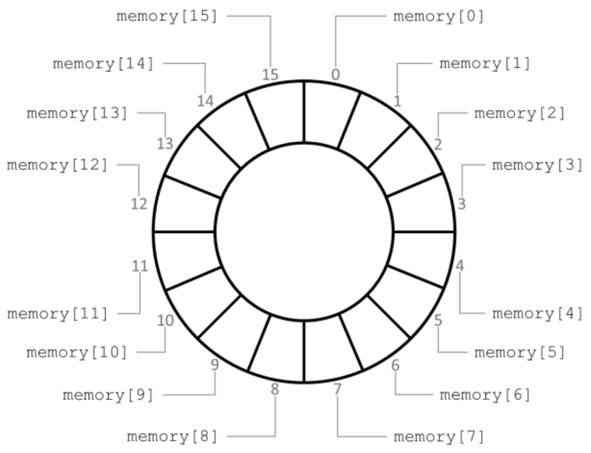


Figure 5. Circular buffer representation of [7:0] memory [0:15]

4) Write codes that are similar to 2). Create an index parameter register j and assign an initial value to it. Update the value of j and assign memory[j] to data_out according to a clock clk_read.

   In the example of Figure 6, the initial value of j is 14 and the frequency of clk_read is the same as clk_write. Predict and fill in the data_out values in Figure 6. Is there any delay at data_out with respect to data_in? How long is the delay?

   Run simulation and verify your answers by observing the waveforms.
   Please take note that if there are two clocks appearing in the module, in the testbench, two ‘always begin … end’ blocks need to be written for creating the clocks respectively.

5) Modify the size of memory, the initial values of i and j, the frequencies of clk_write and clk_read etc. separately. How will they introduce the distortion (eg. delay) in data_out with respect to data_in.
   Run simulation and verify your answers.

6) Make use of this module to implement the delay feature in this project.

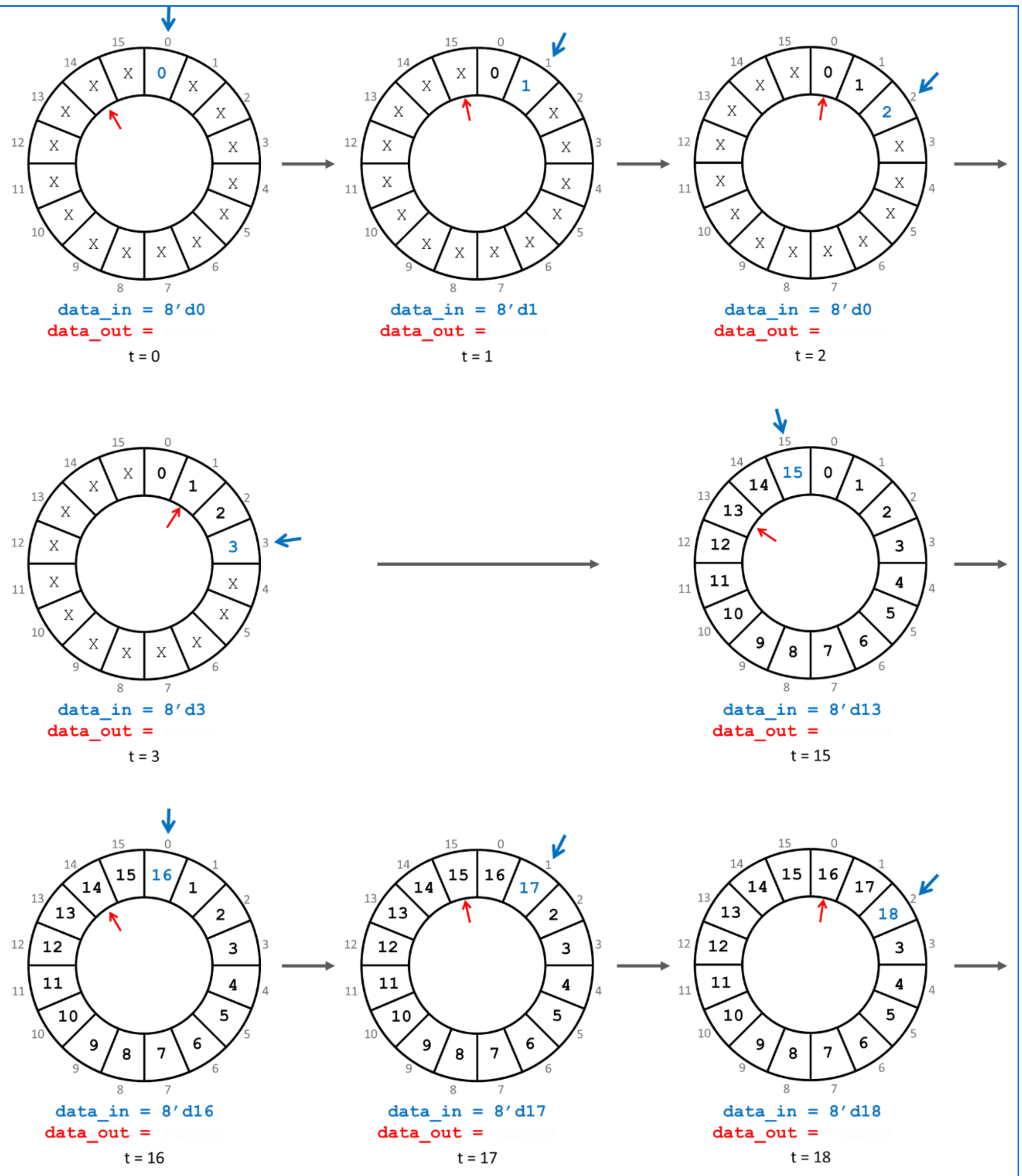


Figure 6. An example of the change of a circular buffer `[7:0] memory [0:15]` *over time.*
The blue arrow indicates the writing location, to which `data_in` *is stored.*
The red arrow indicates the reading location, from which `data_out` *gets the value.*

B. 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.

Figure 7. An example of an electronic instrument with a keyboard

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 (B)

Name:__________________________

Create a basic musical instrument on the Basys 3.

1) **Create a basic instrument that is able to play a musical scale of C (do), D (re), E (mi), F (fa), G (sol), A (la), B (ti) on your instrument.**
   Each note is controlled by one FPGA switch. When the switch is on, the corresponding note is output at the speaker.

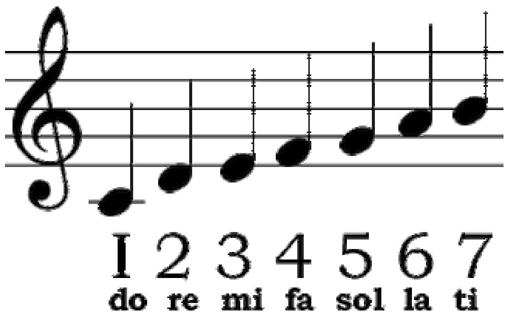


Figure 8. Seven musical notes to be implemented as the basic task

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

Hint:

1) Research on the frequencies of the notes if necessary, or use the table below as the starting point for a scale of notes.

note	note	note	frequency (Hz)
1	C	do	261.626
2	D	re	293.665
3	E	mi	329.628
4	F	fa	349.228
5	G	sol	391.995
6	A	la	440.000
7	B	ti	493.883

Table 2. Frequencies of required musical notes

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

3) Understand how the **DA2RefComp.vhd** takes a 12-bit value as the `DATA1` input and converts it into an equivalent analog signal that can be used to produce sound.

4) Periodic signals of a specific frequency create sounds of specific pitches. Different types of periodic signals will also produce different sounds.

4.3. 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.

4.4. 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 / uniqueness.

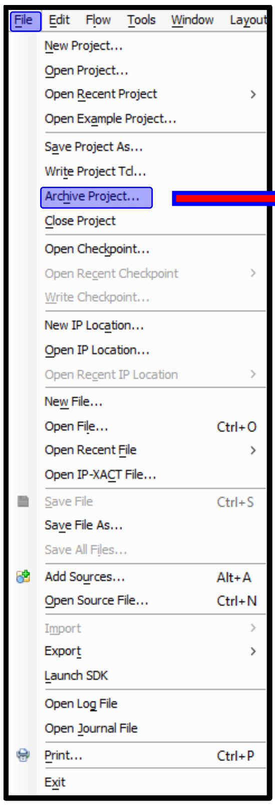
5. Project Submission

Each team is required to submit two items (ITEM A and ITEM B) as indicated below:

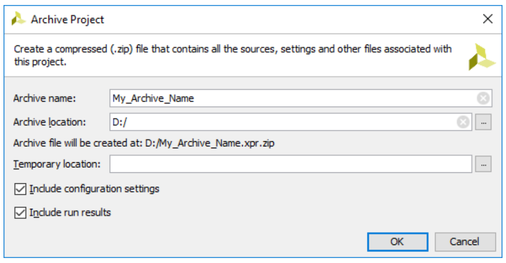
5.1 ITEM A: PROJECT ARCHIVE SUBMISSION

- Only **ONE** Vivado project archive (.zip) for your team. The team marks are not awarded if all the features from the team are not combined into one bitstream
- Ensure that your bitstream has been successfully generated and tested on your Basys 3 development board **BEFORE** archiving your Vivado workspace
- To archive your project, please follow the instructions as shown below (*In Vivado, File -> Archive Project…*):
- Name your project archive in the format indicated below to avoid losing marks:

  Official lab day_Name of any one team member as indicated on the matriculation card_Matriculation number 1_Matriculation number 2_Archive
  Example: Monday_Claude_Shannon_A0300416Z_A0131086Y_**Archive**.xpr.zip



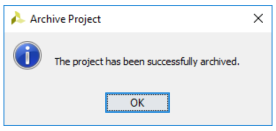
- Enter an appropriate archive name. You can rename it later if required
- Tick the "Include configuration settings" and "Include run results" checkboxes
- Click on OK


Archive Project

Create a compressed (.zip) file that contains all the sources, settings and other files associated with this project.

Archive name: My_Archive_Name
Archive location: D:/
Archive file will be created at: D:/My_Archive_Name.xpr.zip
Temporary location:
☑ Include configuration settings
☑ Include run results
OK Cancel

- A message box will appear after a successful workspace archiving


Archive Project

The project has been successfully archived.

OK

- The archive file size will be a few megabytes (MB) in size. In case it is hundreds of MB in size, please **delete any simulation waveforms that you may have saved in the simulation folder**, and try to archive the workspace again
- If you need to test the archived workspace, please extract the folder from the archive before opening the project in Vivado. Errors will happen if you do not extract the folder

Figure 9. Instruction on archiving a vivado project

5.2 ITEM B: PROJECT BRIEF SUBMISSION

- Include the following in your project brief:
  - Name, matriculation number, official lab session (Monday | Tuesday | Wednesday | Thursday | Friday)
  - User guide which consists of:
    - Features that you have designed and implemented

  - Instructions on how these features can be operated by the user, either through flowcharts or by using a table format as indicated below:

No.	Feature	Input	Description	Output Display
1	Hello Playback	SW[3]	Select this switch to playback "Hello" continuously.	Playback frequency
		+PB_C	Description: Pause / Resume	Pause / Resume indicator

  - Describe the key features that have been designed and implemented, while appending code segments to aid in your description if necessary
  - Clearly indicate the lead designer(s) each feature. You are not allowed to indicate both students of the team
- ➢ Describe the key features that have been designed and implemented, while appending code segments to aid in your description if necessary
- ➢ Feedback:
  1) What did you like most ☺ / least ☹ about the project?
  2) How would you suggest the overall project assignment be improved?
  3) Any other constructive feedback / suggestions are welcome.
  4) Please include your experience on what made the project doable and enjoyable, and which parts caused unnecessary pressure during your learning journey.
  - Note that feedbacks, whether positive or negative, DO NOT have any effects on your grades ☺
- ➢ References: Include references to sources that you have obtained template codes / ideas from

- Name your report submission in the format indicated below to avoid losing marks (The one team member name MUST be the same as that indicated for the Project Archive):

  Official lab day_Name of any one team member as indicated on the matriculation card_Matriculation number 1_Matriculation number 2_Report
  Example: Monday_Claude_Shannon_A0300416Z_A0131086Y_**Report**.pdf

5.3 SUBMISSION DEADLINES:

- ITEM A and ITEM B should be submitted to IVLE in the correct folders by the deadlines as stated in the table below:

Official Session	ITEM A: Project Archive Submission	ITEM B: Report Submission
Monday	07th April 2018 Saturday 11:59 P.M.	08th April 2018 Sunday 11:59 P.M.
Tuesday		
Wednesday		
Thursday		
Friday		

Table 3. Project archive and report submission deadlines

- Excuses about technical issues during upload will not be entertained, hence you are required to avoid uploading at the last minute
- Penalties apply for late submissions

6. Project Submission

- The demonstration of your submitted project will be held in Week 12. Further details will be provided closer to that week

7. Appendix

7.1. 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.

7.2. 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 10 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 10, a **sampling frequency**[1] of f_s = 10Hz is employed. This means 10 samples (green dots in Figure 10) of data from the sine wave are taken every second. This results in a discrete-time signal as shown in Figure 11.

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

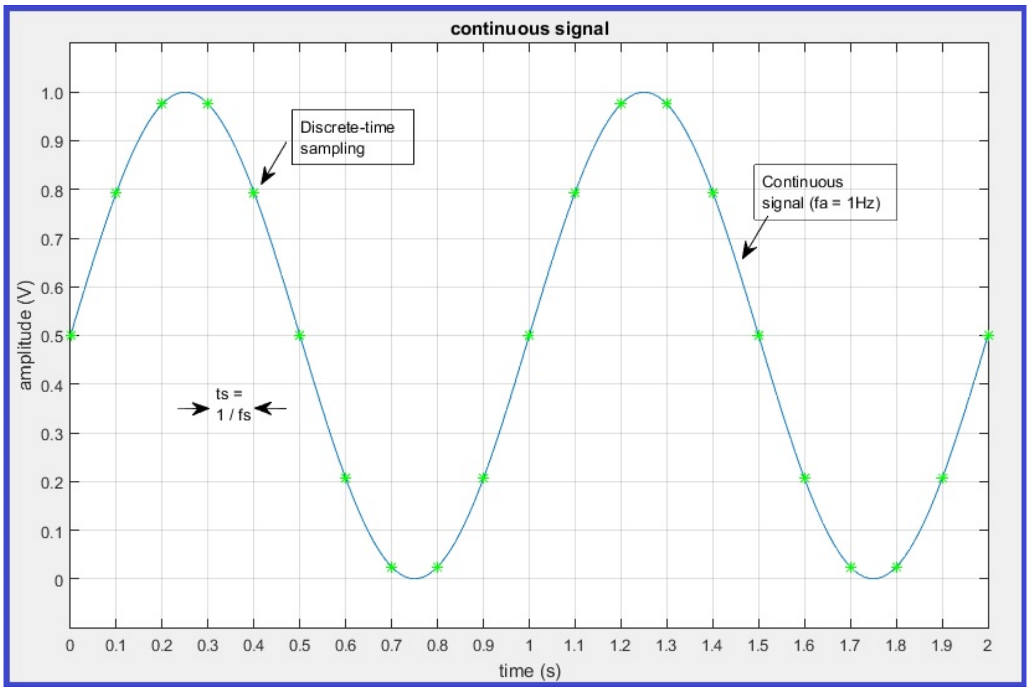


Figure 10. An analog waveform of 1Hz. It is to be sampled at f_s = 10Hz, i.e. 10 samples/second.

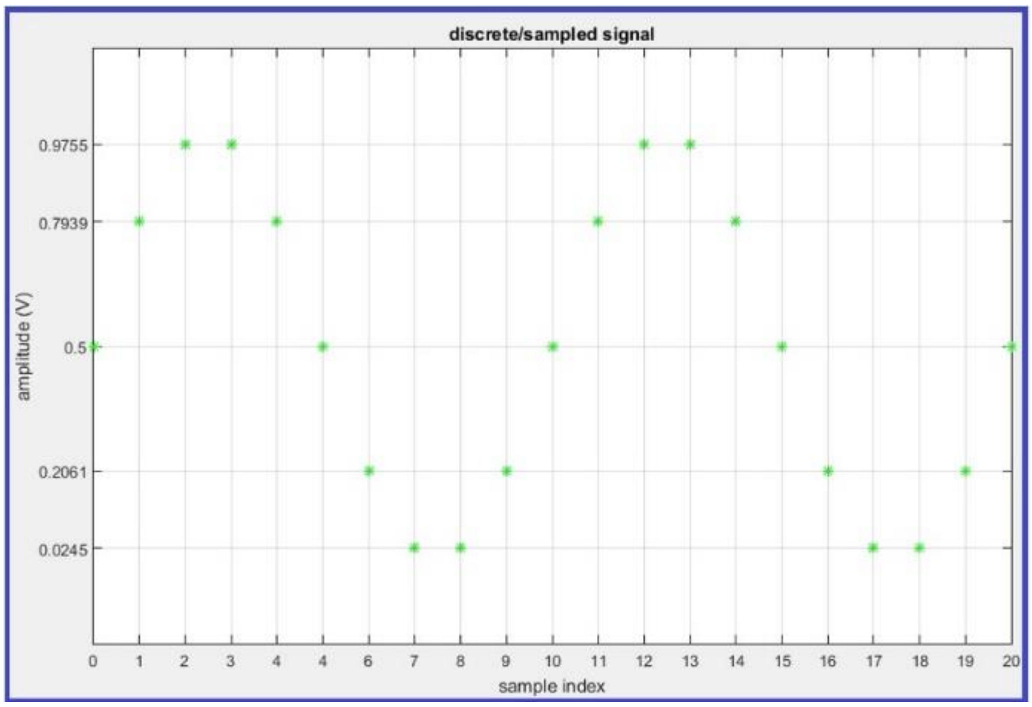


Figure 11. Discrete-time signal at sampling rate of f_s = 10Hz

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 12.

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.

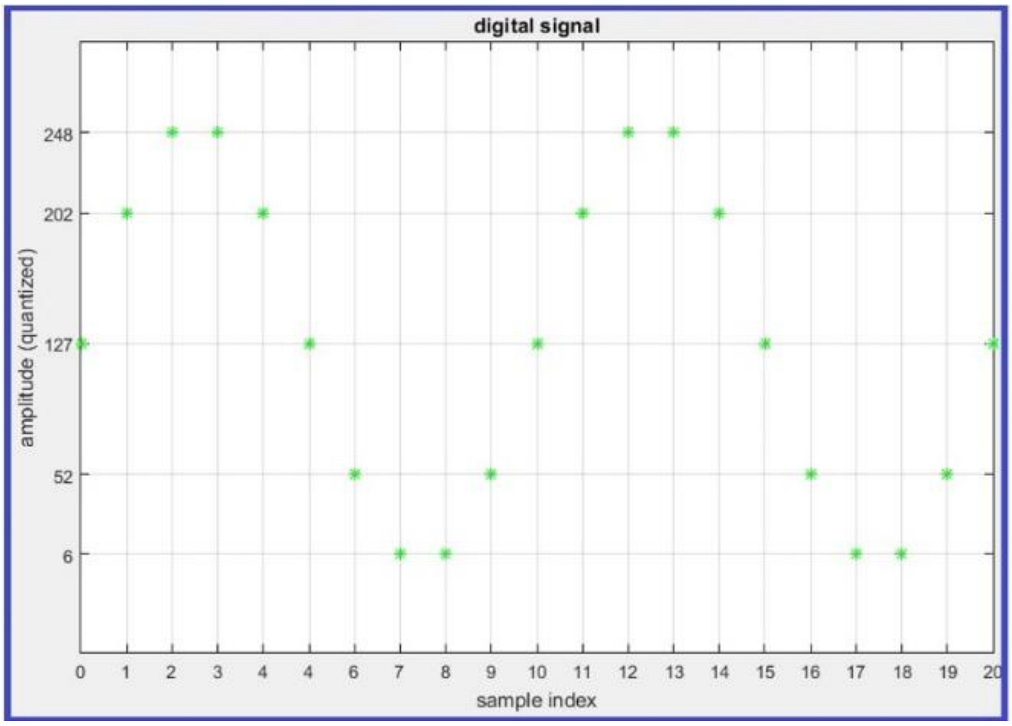


Figure 12. Digital signal quantized by 8 bits at sampling rate of f_s = 10Hz

Figures 13 and 14 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.

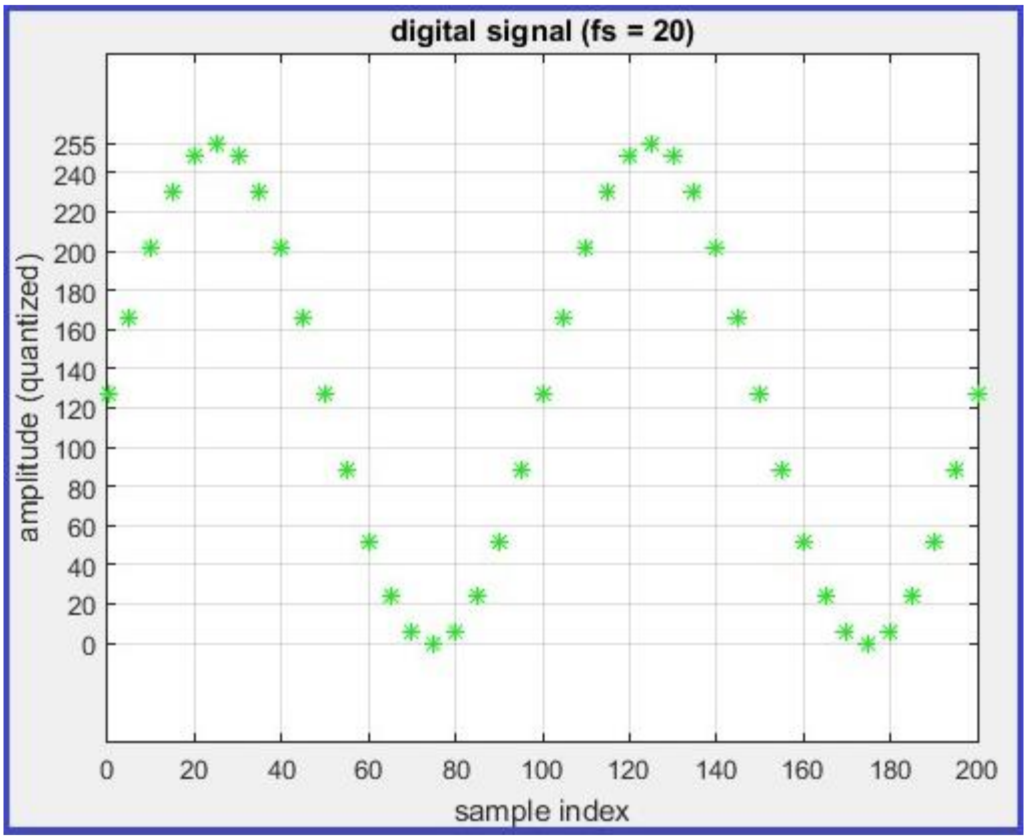


Figure 13. Digital signal quantized by 8 bits at sampling rate of f_s = 20Hz

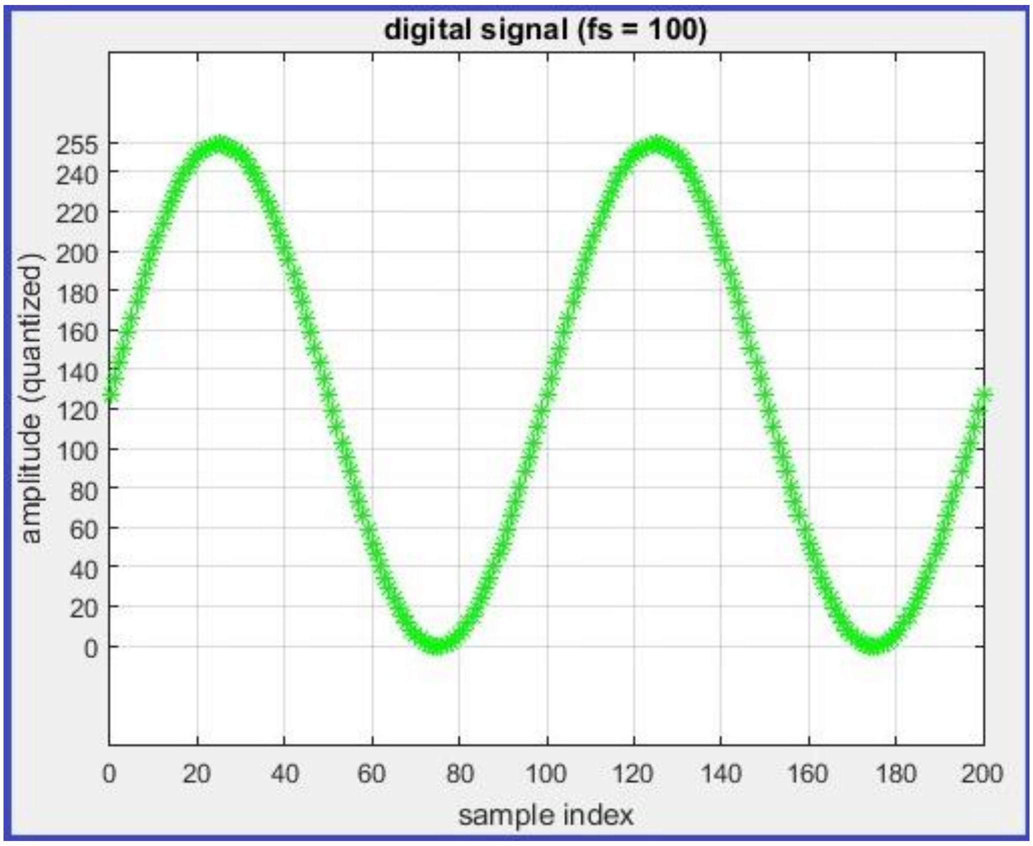


Figure 14. Digital signal quantized by 8 bits at sampling rate of f_s = 100Hz

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.

7.3. 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 15. It consists of a digital decoder and an analog voltage divider which converts a 3-bit binary number into an analog value.

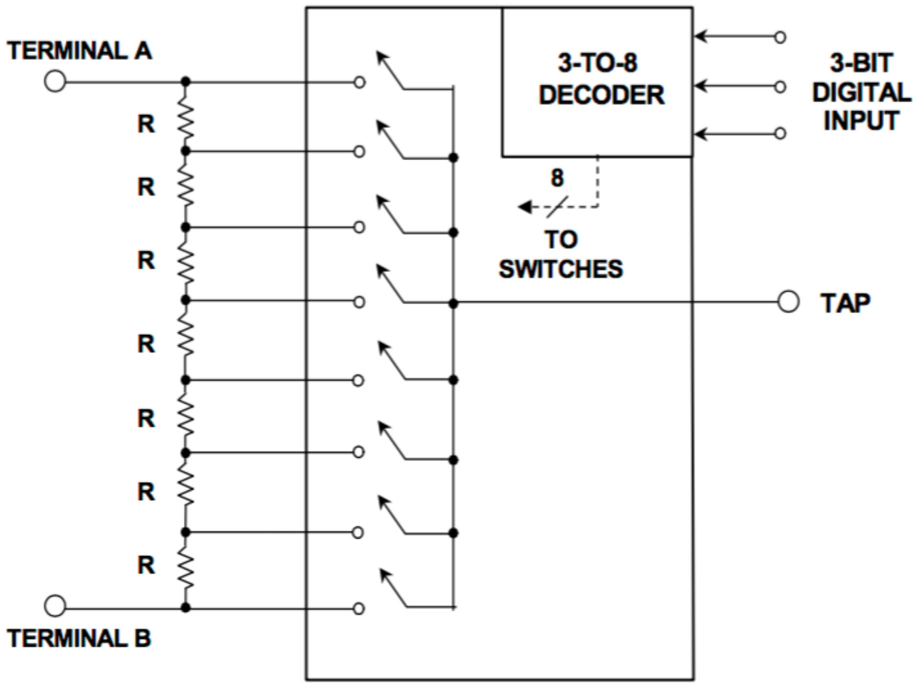


Figure 15. Configuration of a 3-bit DAC, consisting of a 3-to-8 decoder and a voltage divider

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 12 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 16.

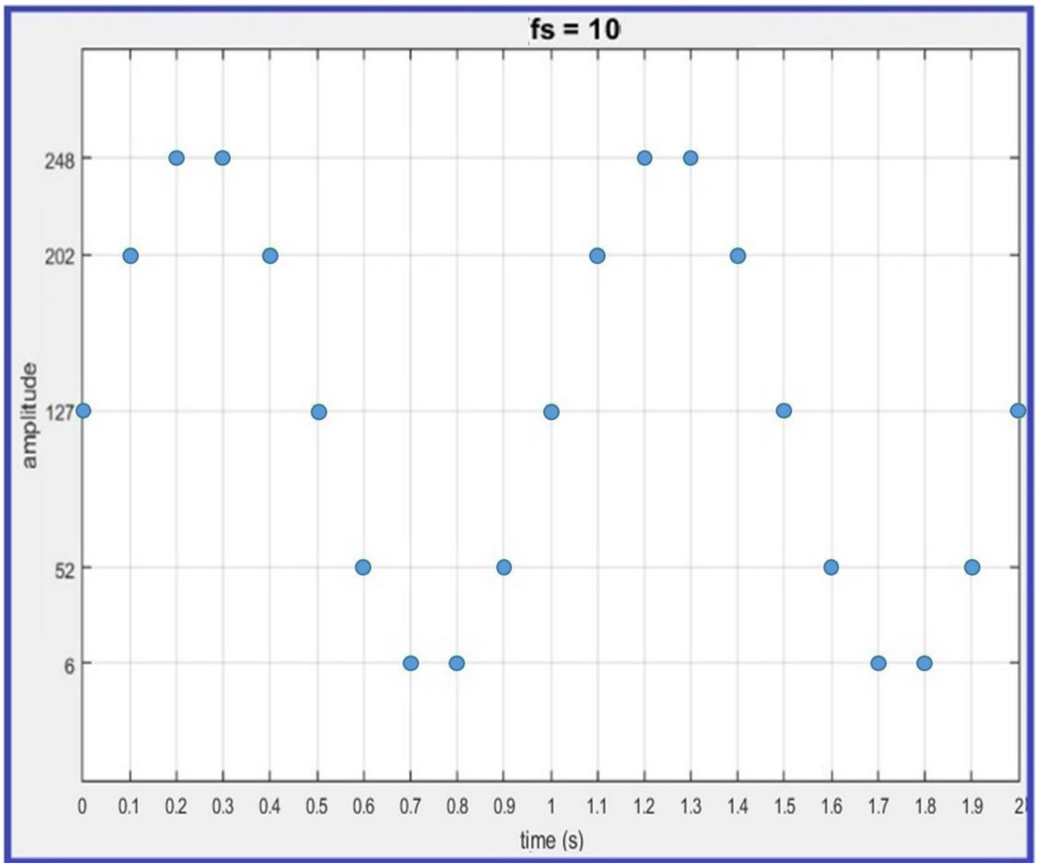


Figure 16. DAC capture inputs (10Hz) at sampling frequency fs=10Hz

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 12. 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 17), while the other eight data (green dots in Figure 17) are missing due to the difference between the input and DAC sampling frequencies.

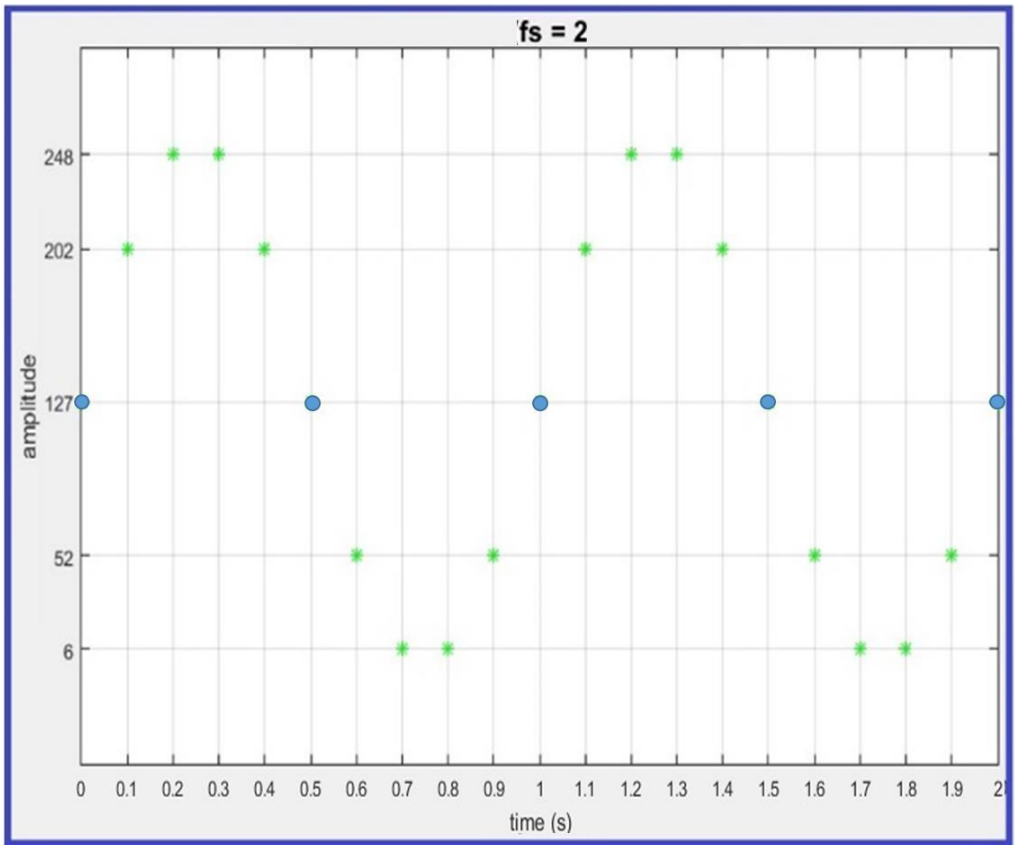


Figure 17. DAC capture inputs (10Hz) at sampling frequency fs=2Hz

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 18 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).

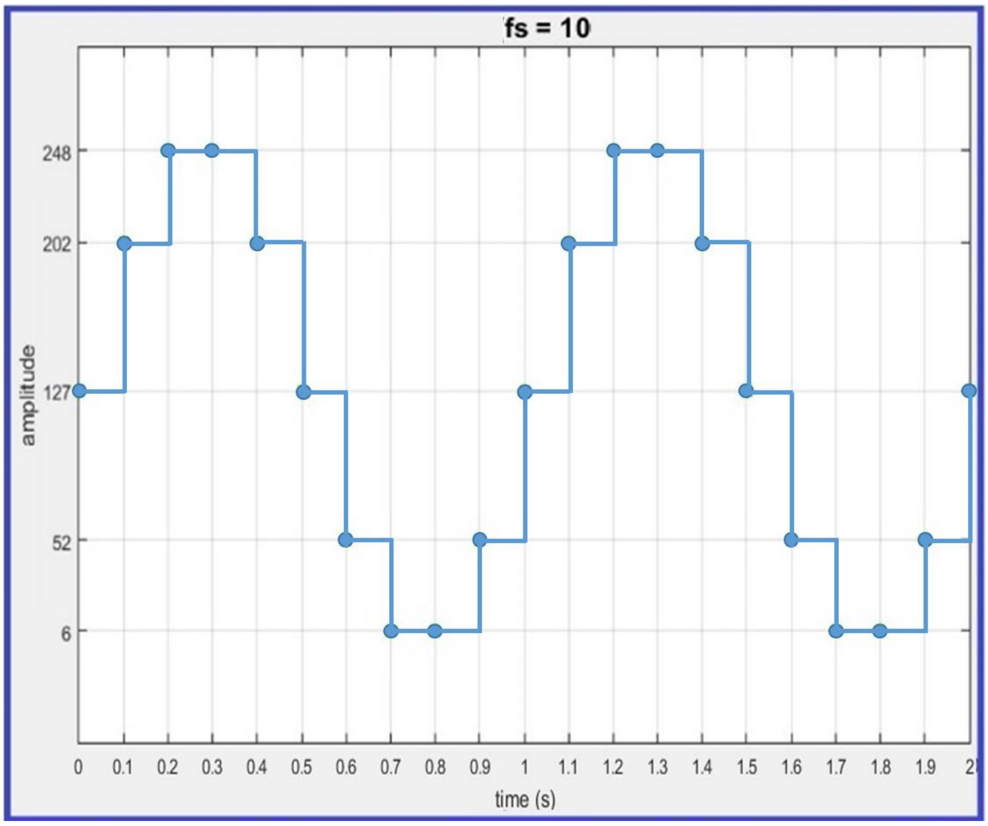


Figure 18. DAC recovered signal (sampling at 10Hz) of a 1Hz sinosoidal analog signal

The recovered signal output from DAC may not be as smooth as the original sinusoidal curve as shown in Figure 12. 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 13 and the DAC runs at 20Hz. It can be further improved by increasing the input and conversion rate at 100Hz according to Figure 14.

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.

7.4. 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 19). They are for connecting and communicating with external devices. The Basys 3 Pmod pin assignments are as shown in Figure 19.

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.

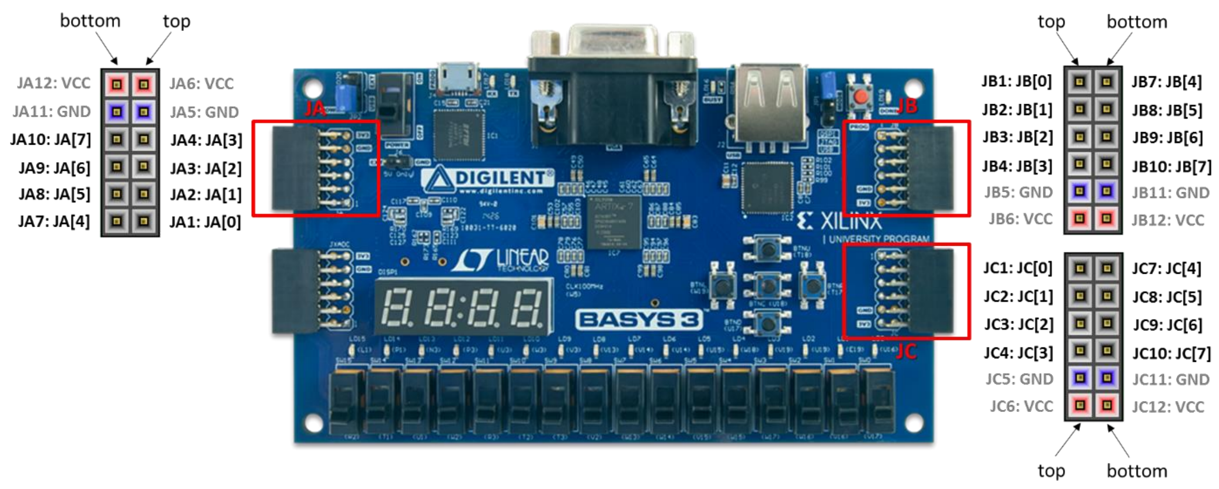


Figure 19. Basys 3 Pmod Pins Assignment

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

7.5 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 20.

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`.

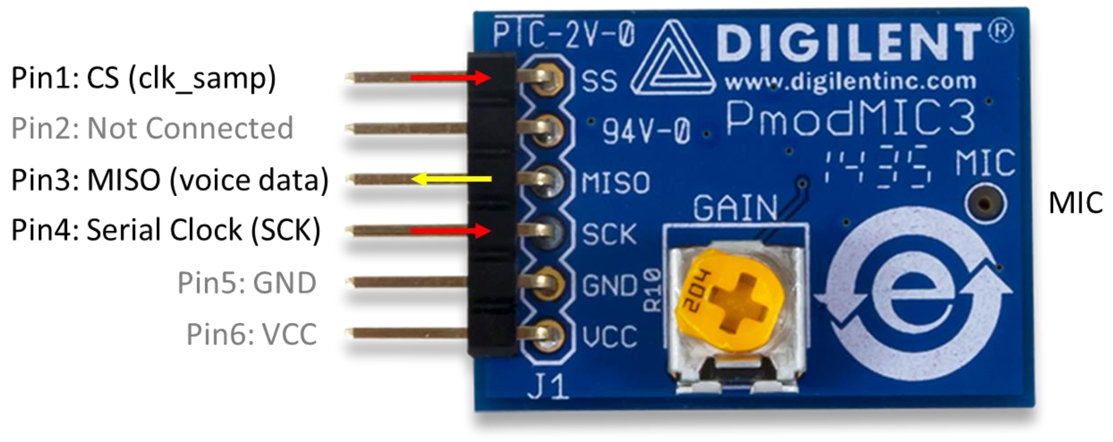


Figure 20. Digilent PmodMIC3 pin configuration

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

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

7.6 PmodDA2

The Digilent PmodDA2 board as shown in Figure 21 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).

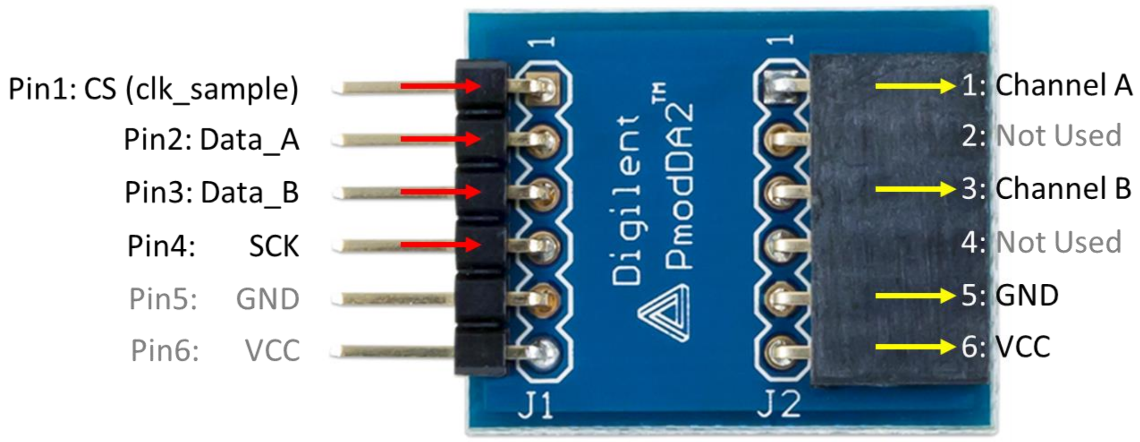


Figure 21. Digilent PmodDA2 pin configuration

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 21).

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

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

7.7 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 22 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.

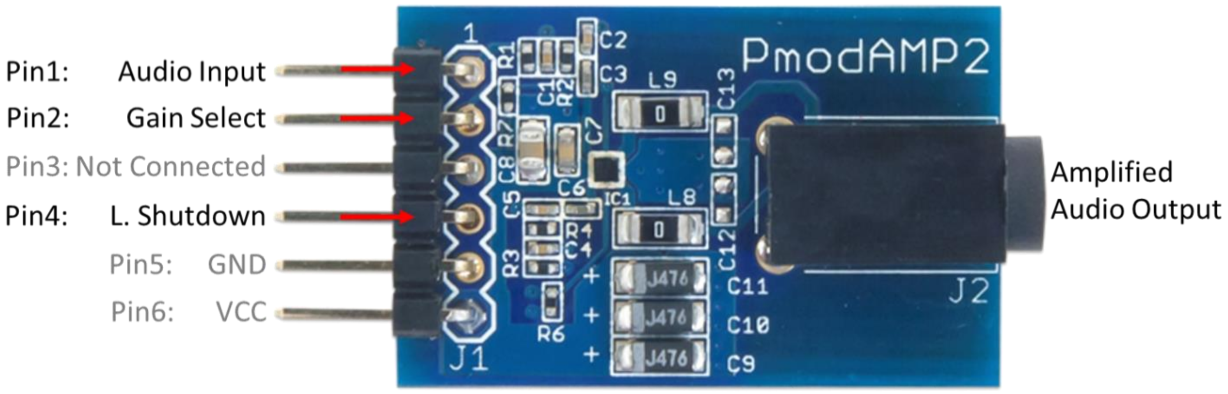


Figure 22. Digilent PmodAMP2 pins configuration

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

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

7.8 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