Adding Audio to your Project (SD Card)

Why use an SD card?

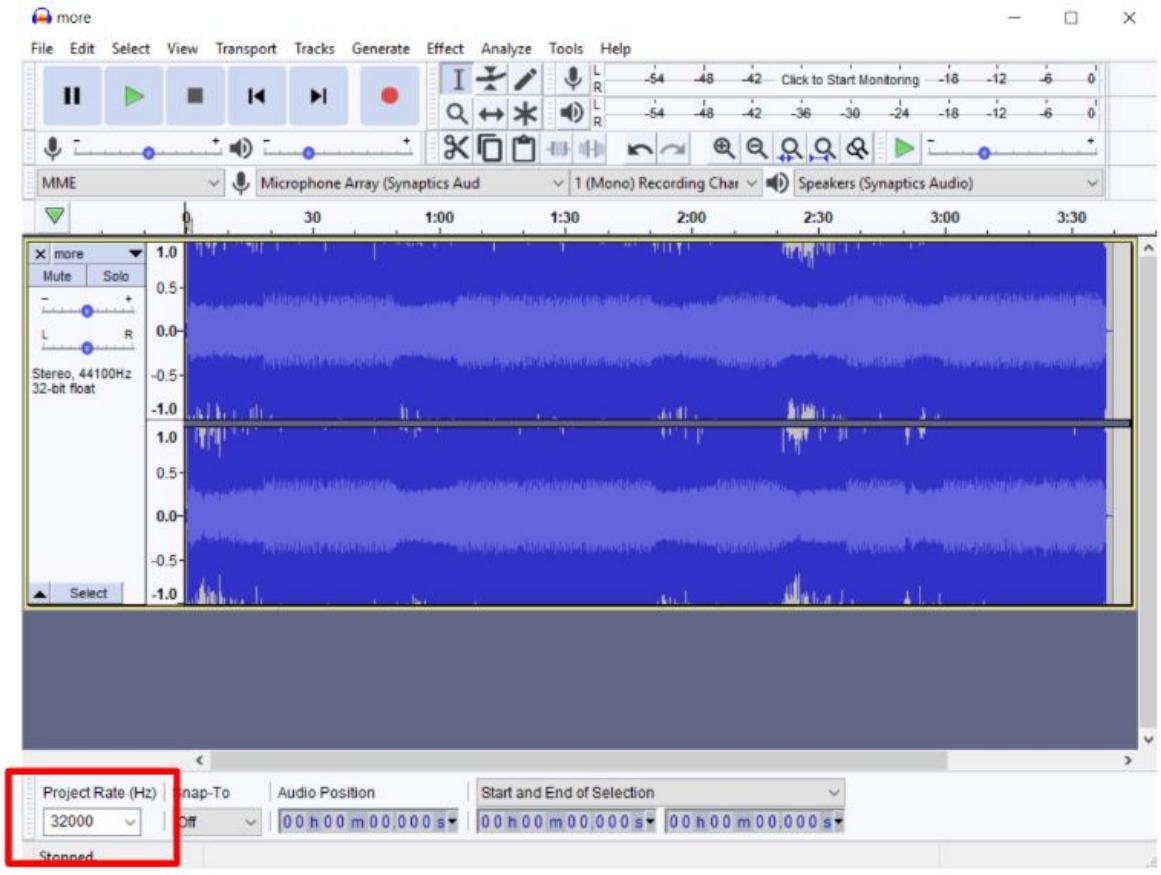
- Ability to hold a lot of samples (ones in lab are 2GB)
    - Can hold many full songs
    - Higher sampling rate = more samples = higher quality audio

Part 1: Audio Formatting

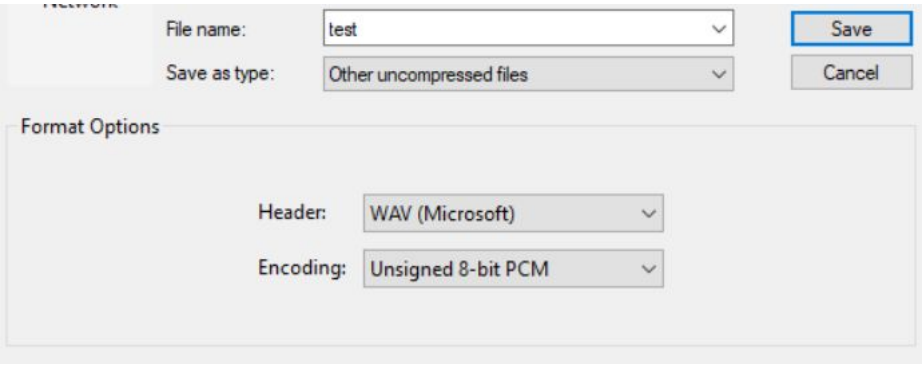
The given sd_controller module is compatible with an unsigned 8-bit .WAV file. The method given uses Audacity, an audio-editing software, but this can probably be done with other software as well.

Given a file with the desired audio:

1. Launch Audacity, and open the audio file you want to convert.
    a. Select "Make a copy of the files before editing"
2. Change the sampling rate to 32kHz (bottom left hand corner)



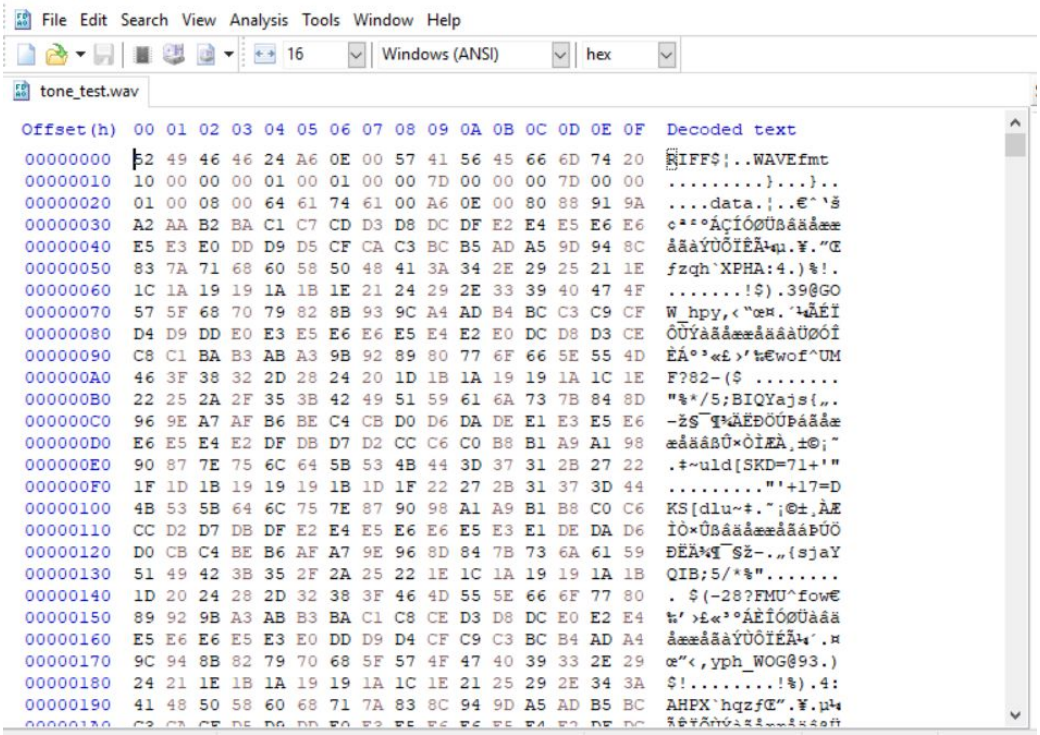
3. File > Export > Export as WAV
    a. **Make sure the file is exported as an unsigned 8-bit .WAV file, as it will not necessarily work with the given Verilog otherwise.**



Part 2: SD Card Formatting

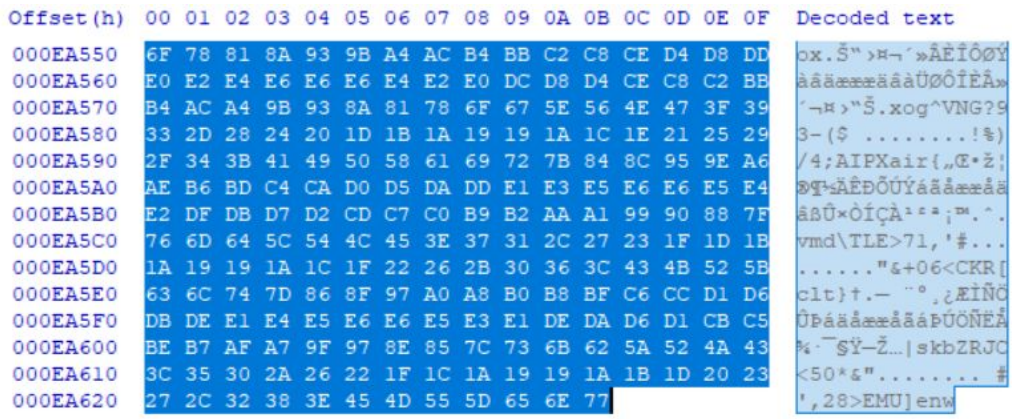
In order for the audio data to be read correctly, it needs to be written *directly* to the SD card. This needs to be done using a hex editor, such as HxD (which is only compatible with Windows). Download the portable version of HxD here. Other hex editors will work, too.

1. Insert the SD card in your computer.
2. Launch the hex editor (run it as an administrator), and open the converted .WAV file. It will look something like this.



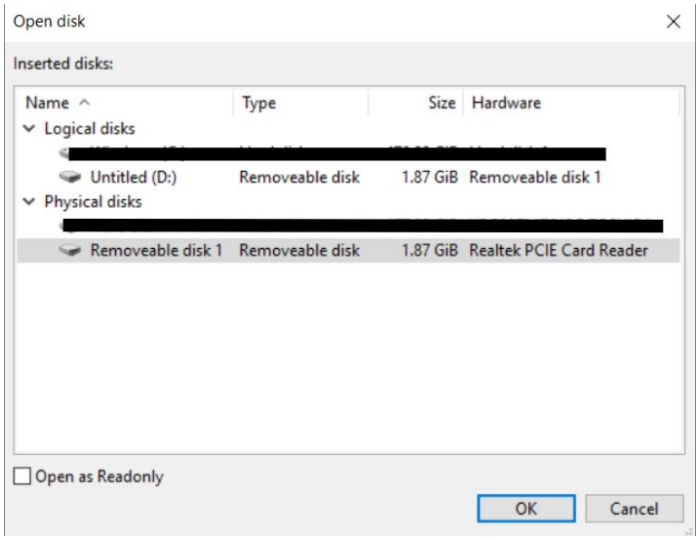
    a. Each byte is a sample from the audio file. There are 16 samples/row The offset column keeps track of the number of rows.

3. To copy the audio samples to the SD card:
    a. *Select all* of the audio samples (Edit > Select All) and *copy* them.
        i. Take note of the total number of samples in the audio file by looking at the final offset (all the way at the bottom… you should be routed here when you select all) and adding the number of samples in that row. This could be an important parameter in your Verilog code.
        ii. In this example, there are 000EA620 + 12 = 960048 total samples.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000EA550	6F	78	81	8A	93	9B	A4	AC	B4	BB	C2	C8	CE	D4	D8	DD	ox.Š"›¤¬´»ÂÈÎÔØÝ
000EA560	E0	E2	E4	E6	E6	E6	E4	E2	E0	DC	D8	D4	CE	C8	C2	BB	àâäæææäâàÜØÔÎÈÂ»
000EA570	B4	AC	A4	9B	93	8A	81	78	6F	67	5E	56	4E	47	3F	39	´¬¤›"Š.xog^VNG?9
000EA580	33	2D	28	24	20	1D	1B	1A	19	19	1A	1C	1E	21	25	29	3-($ ....!%)
000EA590	2F	34	3B	41	49	50	58	61	69	72	7B	84	8C	95	9E	A6	/4;AIPXair{„Œ•ž¦
000EA5A0	AE	B6	BD	C4	CA	D0	D5	DA	DD	E1	E3	E5	E6	E6	E5	E4	®¶½ÄÊÐÕÚÝáãåææåä
000EA5B0	E2	DF	DB	D7	D2	CD	C7	C0	B9	B2	AA	A1	99	90	88	7F	âßÛ×ÒÍÇÀ¹²ª¡™.ˆ.
000EA5C0	76	6D	64	5C	54	4C	45	3E	37	31	2C	27	23	1F	1D	1B	vmd\TLE>71,'#...
000EA5D0	1A	19	19	1A	1C	1F	22	26	2B	30	36	3C	43	4B	52	5B	......"&+06<CKR[
000EA5E0	63	6C	74	7D	86	8F	97	A0	A8	B0	B8	BF	C6	CC	D1	D6	clt}†.— ¨°¸¿ÆÌÑÖ
000EA5F0	DB	DE	E1	E4	E5	E6	E6	E5	E3	E1	DE	DA	D6	D1	CB	C5	ÛÞáäåææåãáÞÚÖÑËÅ
000EA600	BE	B7	AF	A7	9F	97	8E	85	7C	73	6B	62	5A	52	4A	43	¾·¯§Ÿ—Ž…|skbZRJC
000EA610	3C	35	30	2A	26	22	1F	1C	1A	19	19	1A	1B	1D	20	23	<50*&"..... #
000EA620	27	2C	32	38	3E	45	4D	55	5D	65	6E	77					',28>EMU]enw

    b. Open the SD card in the hex editor.
        i. Tools > Open Disk



        ii. Choose the removable disk under “Physical disks” (your SD card, unless you have other removable disks in your computer) and uncheck “open as Readonly”. - **don’t select your hard drive; make sure you selected your SD card.**
            1. The SD card under “Physical Disks” will have the same size as the (D:) drive under “Logical Disks” -- since they are the same drive.

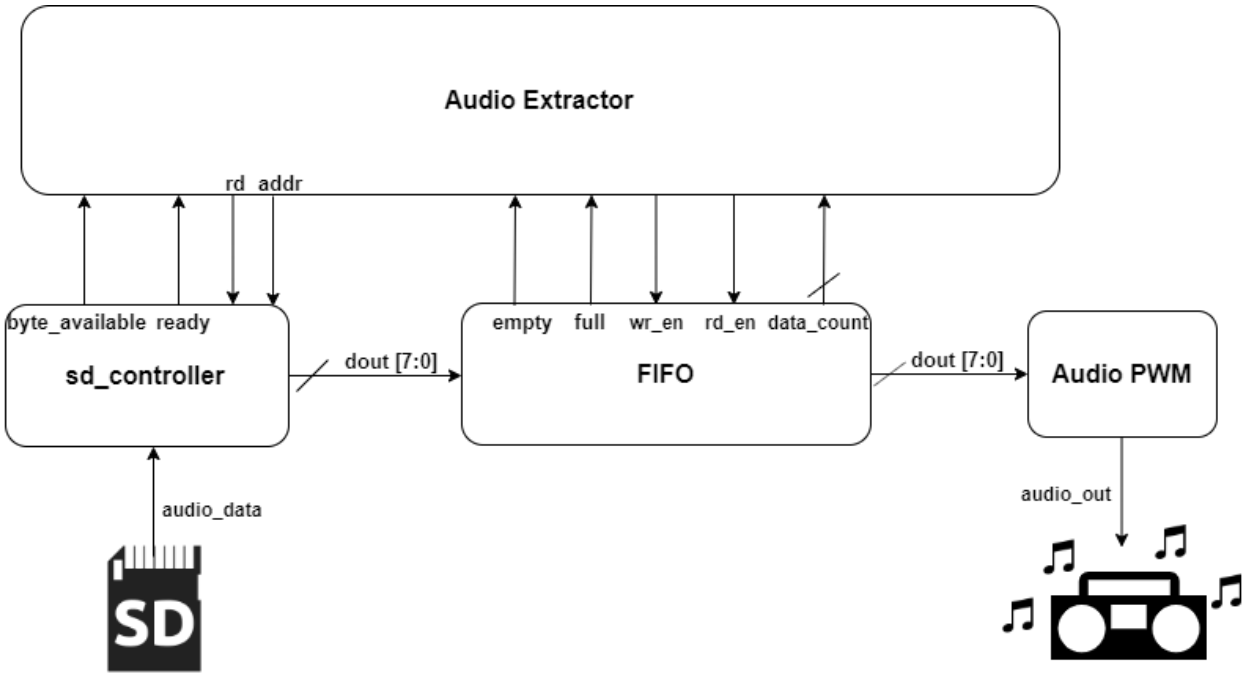
iii. Accept the warning about making your disk unreadable - just maybe double check that you chose the correct drive and more importantly not your hard drive.

NOTE: The SD card is split into “sectors,” with each sector being 512 bytes. Always paste the audio at the beginning of an empty sector. As you add more audio files, it would be smart to keep track of the addresses where audio is stored so you don’t accidentally overwrite any relevant data.

c. Find the first empty sector in the SD card (where all of the samples are “00”), and paste the selected samples (Edit > Paste Write).
   i. Take note of the beginning address and the ending address (can just add the number of samples to the beginning address to find this), as these will be important parameters in Verilog.

Part 3: Verilog

High-Level Overview



- The SD card is being read from at 25MHz, which is much faster than any reasonable audio sample rate.
  - FIFO (first in, first out) module stores bytes read from the sd card and outputs them to the PWM module in the same order that it receives them

    - Bytes can be read from the FIFO with the same sampling rate as the audio file.
        - So, the audio will sound exactly the same as the original file.
- The overall system should run at 100MHz, but the sd_controller module MUST operate on a 25MHz clock.

Part A: sd_controller

Given: sd_controller module

- Do not need to edit the module, but need to adjust inputs to achieve desired behavior

```
output reg cs, // Connect to SD_DAT[3].
output mosi, // Connect to SD_CMD.
input miso, // Connect to SD_DAT[0].
output sclk, // Connect to SD_SCK.
             // For SPI mode, SD_DAT[2] and SD_DAT[1] should be held HIGH.
             // SD_RESET should be held LOW.

input rd,   // Read-enable. When [ready] is HIGH, asseting [rd] will
             // begin a 512-byte READ operation at [address].
             // [byte_available] will transition HIGH as a new byte has been
             // read from the SD card. The byte is presented on [dout].
output reg [7:0] dout, // Data output for READ operation.
output reg byte_available, // A new byte has been presented on [dout].

input wr,   // Write-enable. When [ready] is HIGH, asserting [wr] will
             // begin a 512-byte WRITE operation at [address].
             // [ready_for_next_byte] will transition HIGH to request that
             // the next byte to be written should be presentaed on [din].
input [7:0] din, // Data input for WRITE operation.
output reg ready_for_next_byte, // A new byte should be presented on [din].

input reset, // Resets controller on assertion.
output ready, // HIGH if the SD card is ready for a read or write operation.
input [31:0] address,   // Memory address for read/write operation. This MUST
                        // be a multiple of 512 bytes, due to SD sectoring.
input clk,  // 25 MHz clock.
```

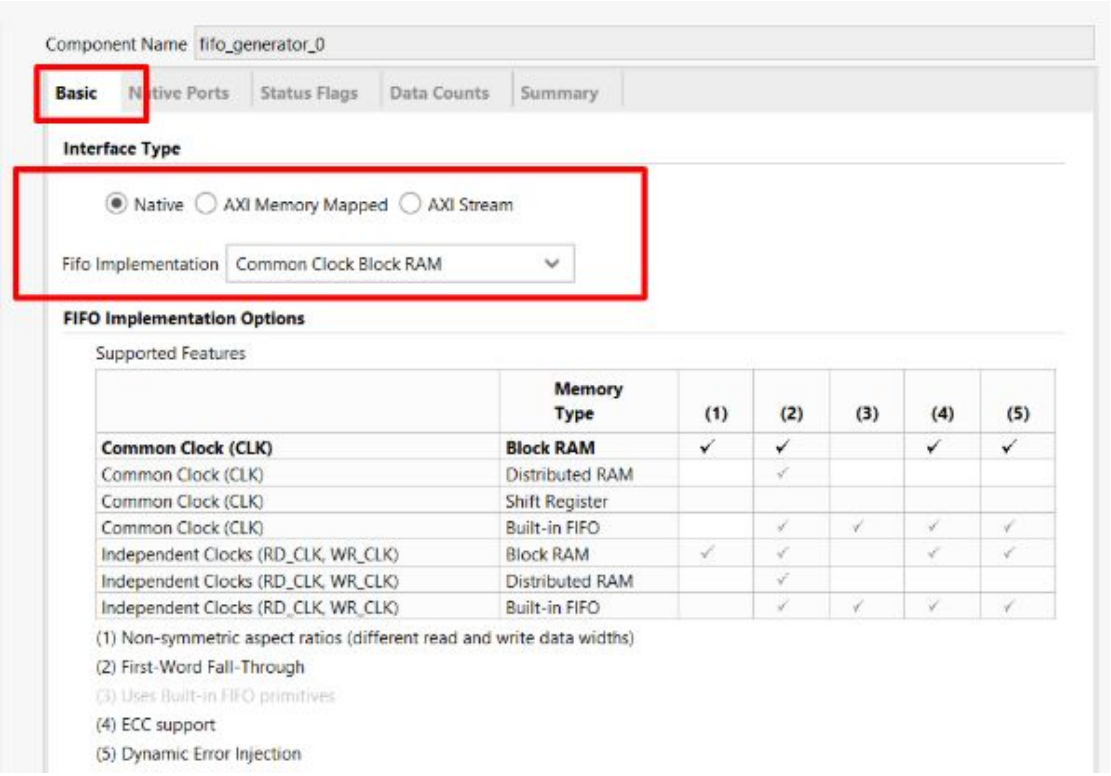
Overview:

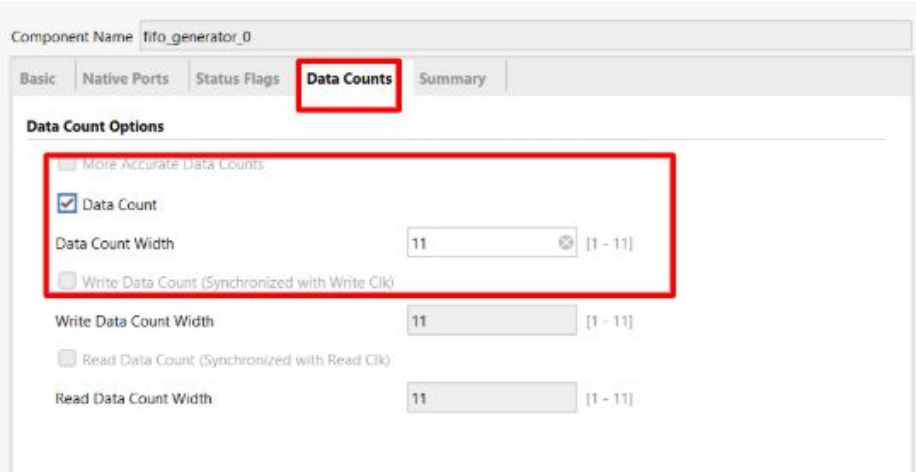
- In the .xdc file, uncomment `sd_reset, sd_cd, sd_sck, sd_cmd,` and `sd_dat[3:0]`
- `sd_controller` must take in a 25MHz clock
- A read (or write) operation begins when both `rd` and `ready` are asserted
    - `sd_controller` will output `ready` at the end of each read/write operation
    - Keep `rd` high, unless you don't want to be reading from the sd card
- Once a read operation begins, `sd_controller` will present 512 bytes on `dout`, one at a time, beginning with the byte that is stored at `addr`
    - When a new byte is available, `byte_available` will be asserted

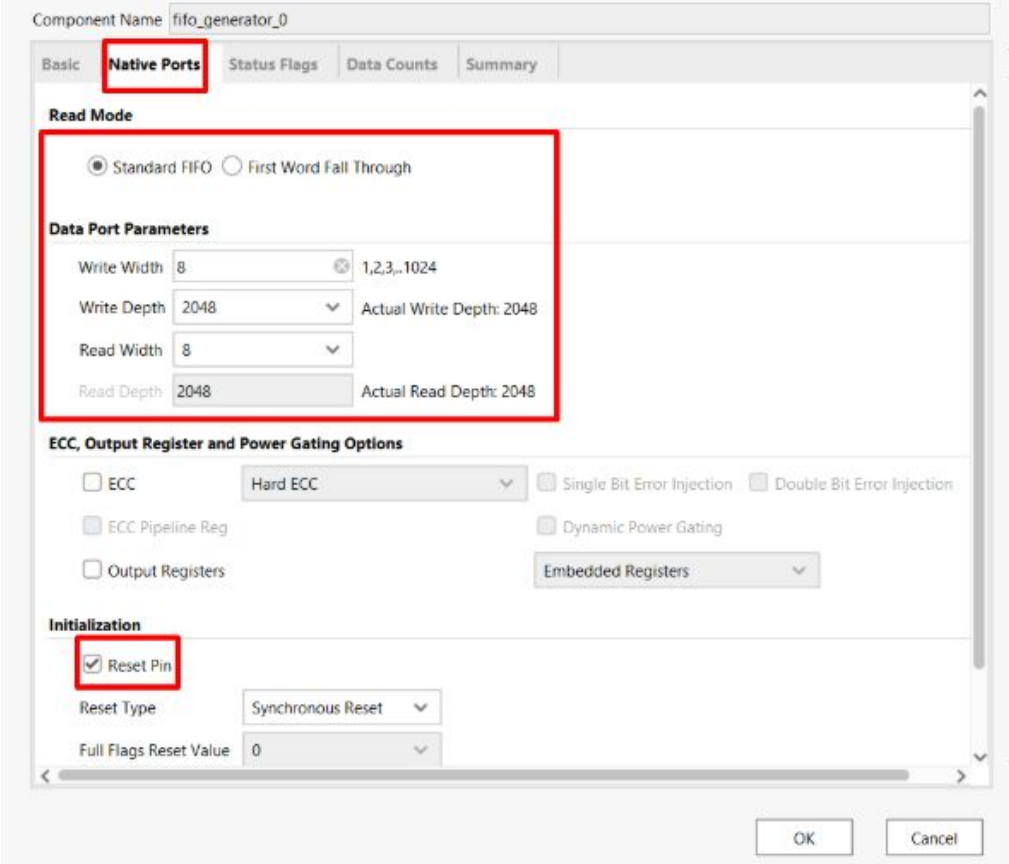
        - Note: The same byte could be presented on `dout` for multiple clock cycles, so check for the rising edge of byte_available (to prevent multiple reads of the same byte)
    - It will be helpful to keep track of the number of bytes read (either from the SD card or from the FIFO) during the read operation.
    - Make sure to initialize `addr` to the start address of the audio file you want to read.
- At the end of a read operation (once 512 bytes have been read)
    - Increment the address (`addr`) by 512
    - sd_controller will assert ready, and another read operation will take place (as long as `rd` is still asserted)

<u>Part B: FIFO (First In, First Out)</u>

- There is a FIFO Generator IP in the IP Catalog (use the settings shown)







- Feel free to use a different write depth, but it should be at least 2048 bytes deep.
    - Use the default data_count width (it depends on the write depth)

- `full/empty:` high when the fifo is full or empty.
  - When full, nothing can be written
  - When empty, nothing can be read
- `din:` data written to the FIFO
- `dout:` data read from the FIFO
- `wr_en:` FIFO write-enable
- `rd_en:` FIFO read-enable
- `clk:` use system clock
- `srst:` link to system reset
- `data_count:` keeps track of the number of bytes stored in the FIFO

Overview

- FIFO can run on system clock (it is not restricted to 25 MHz clock from sd_controller)
- On the rising edge of `byte_available` (from sd_controller)
  - Assert `wr_en` to write the byte from `dout` to the FIFO
    - NOTE: `wr_en` must be a pulse -- you only want to write each byte to the FIFO once!
- Make sure the FIFO doesn't become full (as this will disable any reads) by adjusting `rd` (from the sd_controller module) to stop/start reading from the SD card.
  - One solution is to set `rd` low if `data_count` passes a certain threshold
- Read from the FIFO at the same rate as your audio's sampling rate.
  - Keep track of the number of cycles in order to determine when to assert the FIFO's `rd_en`
    - NOTE: `rd_en` must be a pulse -- same reasoning as making wr_en a pulse.
  - To assure yourself of the actual sampling rate of the song (the hex editor might change the number of samples from what was set before).

$$sampling\ rate\ (Hz) = \frac{number\ of\ samples\ in\ the\ audio\ file}{time\ of\ the\ audio\ file\ (sec)}$$

Part C: PWM

- Use the given PWM module

Appendix: Additional (Useful) Information

- sd_controller Tutorial (Fall 2015)