Digital Circuit Lab — Final Project Tutorial

PaperBand

Date: 06/30/2017

Team 05 Member: 郭子生 b03901032、曾耕森 b03901154、楊其昇 b03901101

Github: https://github.com/awinder0230/Digital-Circuit-Design-Lab

- **Introduction**

Human Computer Interaction (HCI) is a popular field aim at exploring the way how humans interact with machines. In this project, we design a system for users to create a mixture of music tracks by drawing simple geometry shapes on white board or a piece of paper. Each geometry shape represents an unique sound/track of music. For example, a circle may generate a sound track of snare drum, and a triangle may add on another sound track of bass. By drawing different geometry shapes, users may create their own unique piece of music mix.

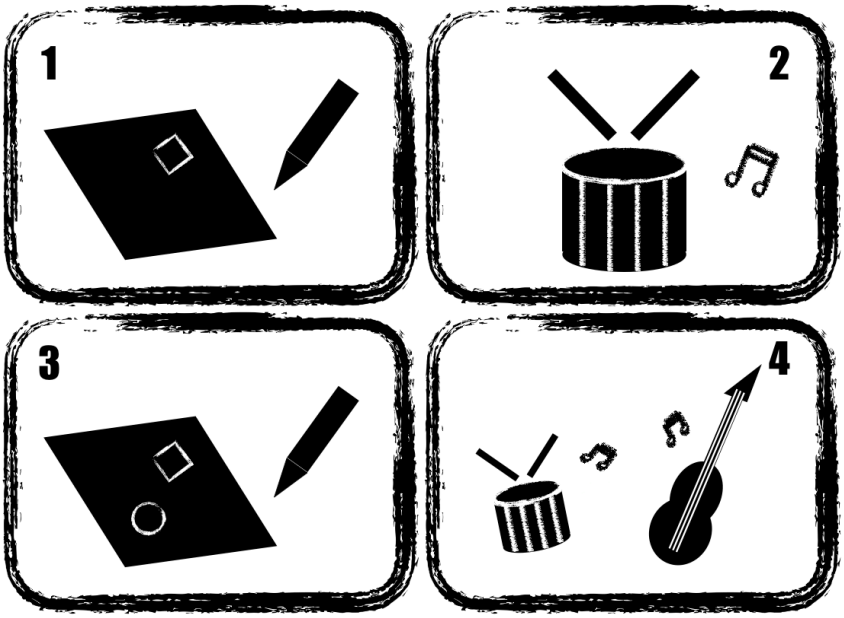


Figure 1. Illustration of the system

- **Tools**: Altera DE2-115 FPGA, Quartus II, 5 Mega Pixel Digital Camera (D5M), Speaker, Screen

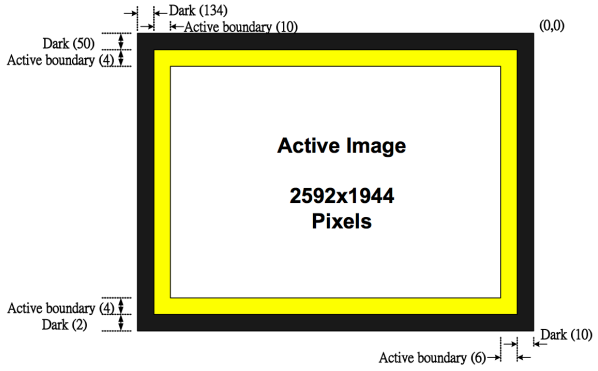
- **Usage**
  - Make sure the camera is able to capture your drawings on a white board or a piece of paper.
  - Draw simple geometry shapes to create your own piece of music!

- **Overview**

Here we briefly give an overview of how our system works, and later dig into more details of each component in our system. Beginning with the D5M camera, which captures images at about 30 frames per second, sends the captured CCD raw data to camera controller, and the camera controller then converts CCD raw data into a black-and-white binary image. The image is stored both in SDRAM and SRAM. The former is for the VGA controller to display images on the screen in real-time, and the latter is for computational purpose of Hough Transform, which is an algorithm for geometry shapes recognition. After shapes recognition, states of the system will be set if it recognize a particular shape, and the system will play the corresponding sound track to generate a music mix created by the user.

- **TRDB-D5M Camera module**

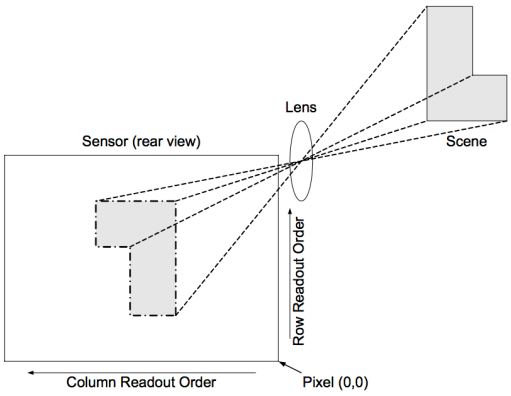
From capturing images to converting raw data into a binary image, it is necessary to understand the speculation and the protocol of D5M camera in first hand. TRDB-D5M is a camera module provided by Terasic for FPGA board. It can be connected to FPGA via GPIO ports on FPGA. D5M pixel array consists of a matrix with 2,004-row by 2,752-column and is addressed by row and column. The address (column 0, row 0) represents the upper-right corner of the entire array as shown in the figure down below.



Column	Pixel Type
0–9	Dark (10)
10–15	Active boundary (6)
16–2,607	Active image (2592)
2,608–2,617	Active boundary (10)
2,618–2,751	Dark (134)

Figure 2. Pixel informations of D5M camera

Boundary regions can be used to avoid edge effects when performing color processing to achieve a image with 2,592 x 1,944 resolution. Optically black columns and rows can be used to monitor the black level. Readout order as well as output data format is described in the figures below:



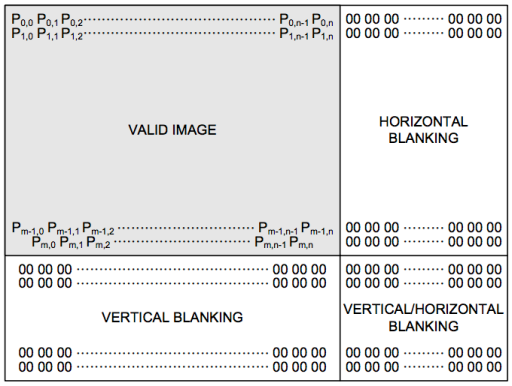


Figure 3. Illustration of imaging a scene and spatial illustration of image readout

Some other useful informations about TRDB-D5M is listed down below just for references:

Resolution	Frame Rate	Sub-sampling Mode	Column _Size (R0x04)	Row_Size (R0x03)	Shutter_ Width_ Lower (R0x09)	Row_Bin (R0x22 [5:4])	Row_Skip (R0x22 [2:0])	Column _Bin (R0x23 [5:4])	Column _Skip (R0x23 [2:0])
2592 x 1944 (Full Resolution)	15.15	N/A	2591	1943	<1943	0	0	0	0
2,048 x 1,536 QXGA	23	N/A	2047	1535	<1535	0	0	0	0
1,600 x 1,200 UXGA	35.2	N/A	1599	1199	<1199	0	0	0	0
1,280 x 1,024 SXGA	48	N/A	1279	1023	<1023	0	0	0	0
	48	skipping	2559	2047		0	1	0	1
	40.1	binning	2559	2047		1	1	1	1
1,024 x 768 XGA	73.4	N/A	1023	767	<767	0	0	0	0
	73.4	skipping	2047	1535		0	1	0	1
	59.7	binning	2047	1535		1	1	1	1
800 x 600 VGA	107.7	N/A	799	599	<599	0	0	0	0
	107.7	skipping	1599	1199		0	1	0	1
	85.2	binning	1599	1199		1	1	1	1
640 x 480 VGA	150	N/A	639	479	<479	0	0	0	0
	150	skipping	2559	1919		0	3	0	3
	77.4	binning	2559	1919	3	3	3	3	3

Parameter	Name	Equation	Default Timing at EXTCLK = 96 MHz
fps	Frame Rate	1/tFRAME	15
^{t}FRAME	Frame Time	(H + max(VB, VBMIN)) × tROW	66ms
^{t}ROW	Row Time	2 × tPIXCLK x max(((W/2) + max(HB, HBMIN)), (41 + 208 x (Row_Bin+1) + 99))	33.5µs
W	Output Image Width	2 × ceil((Column_Size + 1) / (2 × (Column_Skip + 1)))	2592 PIXCLK
H	Output Image Height	2 × ceil((Row_Size + 1) / (2 × (Row_Skip + 1)))	1944 rows
SW	Shutter Width	max(1, (2*16 × Shutter_Width_Upper) + Shutter_Width_Lower)	1943 rows
HB	Horizontal Blanking	Horizontal_Blank + 1	1 PIXCLK
VB	Vertical Blanking	Vertical_Blank + 1	26 rows
HBMIN	Minimum Horizontal Blanking	208 × (Row_Bin + 1) + 64 + (WDC/2)	312 PIXCLK
VBMIN	Minimum Vertical Blanking	max(8, SW - H) + 1	9 rows
^{t}PIXCLK	Pixclk Period	1/fPIXCLK	10.42ns

Figure 4. Useful informations of D5M camera module

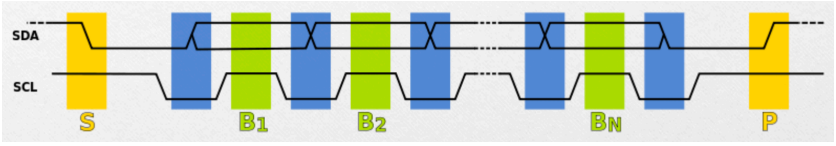


Figure 5. I2C Protocol

Same as the case of WM8731 which we had used in Lab 3, we need to initialize the camera module by I2C protocol. As shown in figure 5, when SCL=1, SDA would be recognized as valid data (green part). Exceptions would occur at the beginning and the end (brown part) of data transferring. Besides, SDA sets the transferred bit while SCL is low (blue part). After every 8 data bits complete transferring in one direction, an "acknowledge" bit (0) would be sent in the reverse direction. There are many registers which can be set according to user preference, such as registers for Row Start, Column Start, Horizontal and Vertical Blanking, Exposure and so on. Beginners may refer to the settings in the examples provided on Terasic website and revised it to meet your needs.

```
    0   :   LUT_DATA    <=  24'h000000;
    1   :   LUT_DATA    <=  24'h20c000;             //  Mirror Row and Columns
    2   :   LUT_DATA    <=  {8'h09,senosr_exposure};//  Exposure
    3   :   LUT_DATA    <=  24'h050000;             //  H_Blanking
    4   :   LUT_DATA    <=  24'h060019;             //  V_Blanking
    5   :   LUT_DATA    <=  24'h0A8000;             //  change latch
    6   :   LUT_DATA    <=  24'h2B0013;             //  Green 1 Gain
    7   :   LUT_DATA    <=  24'h2C009A;             //  Blue Gain
    8   :   LUT_DATA    <=  24'h2D019C;             //  Red Gain
    9   :   LUT_DATA    <=  24'h2E0013;             //  Green 2 Gain
    10  :   LUT_DATA    <=  24'h100051;             //  set up PLL power on
`ifdef VGA_640x480p60
    11  :   LUT_DATA    <=  24'h111f04;             //  PLL_m_Factor<<8+PLL_n_Divider
    12  :   LUT_DATA    <=  24'h120001;             //  PLL_p1_Divider
`else
    11  :   LUT_DATA    <=  24'h111805;             //  PLL_m_Factor<<8+PLL_n_Divider
    12  :   LUT_DATA    <=  24'h120001;             //  PLL_p1_Divider
`endif
    13  :   LUT_DATA    <=  24'h100053;             //  set USE PLL
    14  :   LUT_DATA    <=  24'h980000;             //  disble calibration
    15  :   LUT_DATA    <=  24'hA00000;             //  Test pattern control
    16  :   LUT_DATA    <=  24'hA10000;             //  Test green pattern value
    17  :   LUT_DATA    <=  24'hA20FFF;             //  Test red pattern value
    18  :   LUT_DATA    <=  sensor_start_row    ;   //  set start row
    19  :   LUT_DATA    <=  sensor_start_column ;   //  set start column
    20  :   LUT_DATA    <=  sensor_row_size;        //  set row size
    21  :   LUT_DATA    <=  sensor_column_size;     //  set column size
    22  :   LUT_DATA    <=  sensor_row_mode;        //  set row mode in bin mode
    23  :   LUT_DATA    <=  sensor_column_mode;     //  set column mode  in bin mode
    24  :   LUT_DATA    <=  24'h4901A8;             //  row black target
    default:LUT_DATA    <=  24'h000000;
    endcase
```

Figure 6. Registers of D5M camera module

After initializing the camera module, we can adjust exposure and switch to zoom-in mode by changing the settings of sensor_exposure, location of the start row and start column, and the size of the rows and columns with input signals from key buttons or switches on FPGA board.

In order to convert from the output raw data of camera module to a RGB format for image processing, first we will have check out the output data format of the camera module. Here is an example of output data timing diagram, with P0, P1, P2… represent pixels of a valid image:

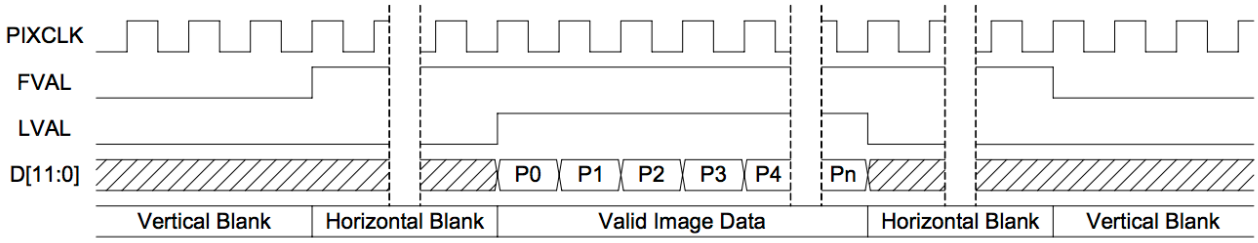


Figure 7. Data transmission protocol of D5M camera module

Here we can see that output data D[11:0] is valid only if signals FVAL and LVAL are both high. When the condition is met, the value of D will be a valid pixel value and will be updated for each PIXCLK clock for transmissions of different pixels.

Since the output data of camera module (i.e. value of D) is not a complete RGB information of a pixel, we could use a line buffer to store previous values. Once the buffer collects complete RGB data informations, we are able to output RGB values of one pixel simultaneously. The main function of the line buffer module is shifting, and the operation of shifting is executed by a module generated by Quartus II.

```
altshift_taps   altshift_taps_component (
            .clken (clken),
            .clock (clock),
            .shiftin (shiftin),
            .taps (sub_wire0),
            .shiftout (sub_wire5)
            // synopsys translate_off
            ,
            .aclr ()
            // synopsys translate_on
            );
defparam
    altshift_taps_component.lpm_hint = "RAM_BLOCK_TYPE=M9K",
    altshift_taps_component.lpm_type = "altshift_taps",
    altshift_taps_component.number_of_taps = 3,
    altshift_taps_component.tap_distance = 800,
    altshift_taps_component.width = 12;
```

Figure 8. Shifting function of line buffer

In figure 8. altshift_taps module is generated by RAM-based shift register (ALTSHIFT_TAPS) megafunction IP core provided by Altera. The ALTSHIFT_TAPS IP core implements a shift register with taps and offers several additional features, including selectable RAM block type, a wide range of widths for shiftin and shiftout ports, selectable distance between taps, and so on. Figure below shows a traditional 12-word-depth shift register.

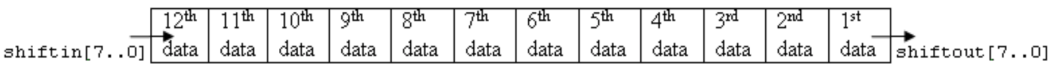


Figure 9. 12-word-depth shift register

Figure down below shows how datas in the shift register chain are being tapped at even spaces (1st, 4th, 7th, and 10th) at the output taps of the ALTSHIFT_TAPS IP core (TAP_DISTANCE = 3 and NUMBER_OF_TAPS = 4):

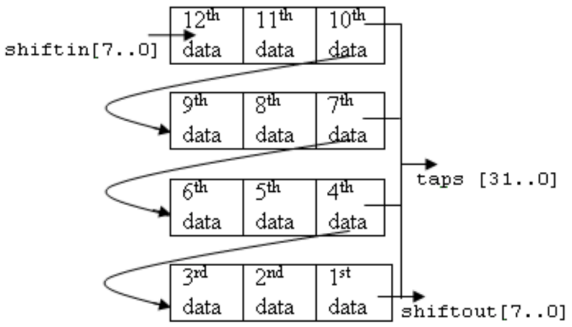


Figure 10. Illustration of shift register chain

Finally, tables down below are some parameters that we can adjust:

Name	Type	Required	Description
NUMBER_OF_TAPS	Integer	Yes	Specifies the number of regularly spaced taps along the shift register.
TAP_DISTANCE	Integer	Yes	Specifies the distance between the regularly spaced taps in clock cycles. This number translates to the number of RAM words that will be used. TAP_DISTANCE must be at least 3.
WIDTH	Integer	Yes	Specifies the width of the input pattern.
POWER_UP_STATE	String	No	Specifies the shift register contents at power-up. Values are CLEARED and DONT_CARE. If omitted, the default is CLEARED.

Value	Description
CLEARED	Zero content. For Stratix and Stratix II device families, you must use M512 or M4K RAM blocks.
DONT_CARE	Unknown contents. M-RAM blocks can be used with this setting.

Table of input ports:

Name	Required	Description
shiftin[]	Yes	Data input to the shifter. Input port WIDTH bits wide.
clock	Yes	Positive-edge triggered clock.
clken	No	Clock enable for the clock port. clken defaults to V_{CC}.
aclr	No	Asynchronously clears the contents of the shift register chain. The shiftout outputs are cleared immediately upon the assertion of the aclr signal.

Table of output ports:

Name	Required	Description
shiftout[]	Yes	Output from the end of the shift register. Output port WIDTH bits wide.
taps[]	Yes	Output from the regularly spaced taps along the shift register. Output port WIDTH * NUMBER_OF_TAPS wide. This port is an aggregate of all the regularly spaced taps (each WIDTH bits) along the shift register.

Figure 11. Parameter tables

In this project, we use images captured by the camera, and turn it into binary images by first converting RGB data into Grey scale data with a formula, and set a threshold for gray scale pixel values to determine the pixel values of a binary image being either 0 or 1.

- **VGA Controller**

As we get a binary image captured by a camera, we could project it onto a screen with VGA signal. Here, the first problem is that the output of VGA signal is serial, which implies that we have to send pixel datas of images in order. Also, the clock frequency of D5M camera is 25 MHz, where the clock frequency of VGA signal is 40 MHz. Therefore, we would need a buffer between the output of camera and the input of VGA controller module. In this project, we use SDRAM on FPGA as a image buffer due to its larger memory size comparing to other memories on FPGA board.

After reseting the VGA controller, it will keep sending requesting signal if the horizontal and vertical coordinates are in the range of the screen. Sdram_Control module would receive the requesting signal and send pixel informations in the same order as stored in the SDRAM (i.e. FIFO). Output signals of VGA not only contain the RGB information of one pixel but also contain the corresponding coordinate information to ensure that the image can be displayed on screen in the correct way.

- **Hough transform**

Hough transform is first invented to detect straight lines in a given picture. The basic idea is pretty simple: for every point recognized as edge point in the original picture, we project the point to a parameter space that represents possible lines which go through the point. For the case of straight lines, the parameter space is 2-D: x-intercept and slope. Each point in the parameter space corresponds to a straight line in image space; and each edge point in the image space corresponds to a straight line in parameter space. For each edge point we draw a line in parameter space, and finally the points in parameter space that have many line intercepts are recognized as straight lines. In practice, we discretize the parameter space and vote on the grids instead of drawing lines. Apparently, there is a trade-off between the step of discretization and accuracy.

Hough transform can not only detect lines but also being applied to geometry shapes such as polygons and circle. General Hough transform could further detect any kinds of shape according to user preference. In our project, we design a module for each geometry shape representing the corresponding sound track. Here we demonstrate the Hough transform algorithm to detect a circle in an image. Similar algorithms for detecting other shapes could be find easily by simply googling Hough transform.

- **Circular Hough Transform**

For Circle detection, we project each edge point to a 3-D parameter space: x-coordinate, y-coordinate and radius. For each edge point, we vote on every possible circle (which corresponds to a point in parameter space) that passes the edge point. Finally, choose the most voted point as the recognized circle. The algorithm is summarized down below:

1. Quantize the parameter space
   $P[x_{0_{min}}, \ldots, x_{0_{max}}, y_{0_{min}}, \ldots, y_{0_{max}}, r_{min}, \ldots, r_{max}]$.
2. For each edge point (x, y) do
   For $(r = r_{min}, r \leq r_{max}, r++)$
   $x_0 = x - r\cos\theta$
   $y_0 = y - r\sin\theta$
   $P[x_0, y_0, r] = P[x_0, y_0, r] + 1$.
3. Find the local maxima in the parameter space.

Figure 12. Hough Transform algorithm pseudocode

- **Circular Hough transform for FPGA**

Hough transform is powerful but often consume a lot of resources. It contains nested for loop that can significantly increase calculation time. Also, it requires a big memory space to store the multi-dimension parameter space for voting data. The performance can be enhanced by hardware design techniques, using parallel computing and some approximations.

Here we aim to implement our circular hough transform in pure Verilog / SystemVerilog. Since the original algorithm contains sin and cos calculations, which is obviously not supported in hardware language, we must do some kind of approximation to calculate the result. After doing some research, we adjusted the algorithm to a new one that contains only simple additions and shift operations. We set the step of θ to 2^{-6} (≈0.8952°) and thus $\sin(\Delta\theta) \approx \Delta\theta$ and $\cos(\Delta\theta) \approx 1$, so that we can calculate next position to be voted according to the previous position:

$$x_{n+1} = x_n - \Delta\theta(y - y_n)$$

$$y_{n+1} = y_n + \Delta\theta(x - x_n)$$

Since Δθ is set to 2^{-6}, calculations now removes the multiplications and contains only shift operations and addition / subtraction. This feature is crucial to hardware applications since it not only increases the speed but also avoids large multipliers and consumes less area.

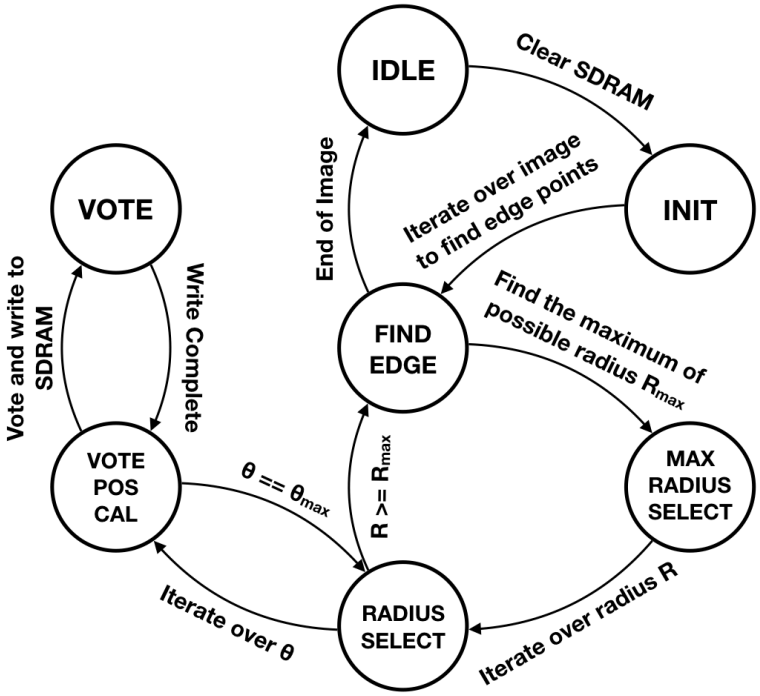


Figure 13. State diagram of Hough Transform for FPGA

- **SRAM**

SRAM is the easiest to use among memory devices in DE2-115. We use SRAM to store our current binary image for shape recognition. The size of image stored is 640 * 480 pixels. However, the resolution of image captured by camera module is originally with 2592 * 1944 pixels, which means we will have to scale the image into 1/4. There're many ways to downsample images, but since we're only using binary image and error of a few pixels doesn't matter, we simply takes one pixel and skip the following 3 pixels. Also we have to skip rows to scale the vertical dimension along with the horizontal dimension.

- **SDRAM**

SDRAM read / write, unlike SRAM, is rather complicated. SDRAM of DE2-115 board concatenates two 64MB SDRAM chips and partitioned into 4 banks. It uses row address and column address to access words, each is 32bit long. Since SDRAM is volatile, the memory must be recharged every once in a while. The read / write operation takes multiple steps to complete, which makes SDRAM generally slower than SRAM. We use SDRAM as buffer of image captured by CCD and output to VGA monitor. The parameter space matrix of Circular Hough Transform is also stored in SDRAM. To make sure the data doesn't conflict, we set an offset much larger than the image size to the address of the Hough parameter matrix.

- **Flash Memory**

Flash memory is non-volatile, that is, the data stored in flash memory will not disappear even if the power is turned off. Thus we decided to store our music in flash memory so we won't have to reload the music every time when the program is turned on. The flash memory on DE2-115 board is organized as 8M addresses * 8 bit word, with 8MB in total. We use Terasic DE2-115 Control Panel, a tool to control DE2-115 board provided by Terasic, to load the music wave file into the flash memory. Note that the .wav files have 46 bytes of header information, so the actual music starts at the 47th byte, with 4 byte per sample, 2 byte per channel.

- **Music Mixer**

In our system, there is a state indicating which sound tracks are enable, meaning that the system has recognizing the corresponding shapes in the image captured by the camera. For example, if the sound effect of a circle is bass, then the state of bass would be set to 1 once the system has detect a circle in the captured image by performing Hough transform.

Each sound track is stored in the flash memory at different addresses but with same spatial size. Also, there is a counter in our system for time synchronization of all the sound tracks. For example, suppose there are 5 sound tracks stored in flash with address beginning from 0x000000, 0x060000, 0xC0000, 0x120000, and 0x180000. When the counter = 1, the system reads data from address 0x000001, 0x60001, 0xC0001, 0x120001, 0x180001, and add it to a temporal register if the corresponding state of the track is set (add the value of 0x000001 to temporal register if the state of sound track #1 is set, and discard the value of 0x60001 if the state of sound track # 2 is not set, although the system has already read its value from the flash memory.) For each audio sampling cycle, read all the values in the address: (beginning of certain sound track) + (counter value), and add it to temporal register according to the states. Finally, send the value of temporal register to Para2Seri module and further being passed WM8731 audio CODEC chips to play the mixture of music. Since Para2Seri module and WM8731 had already being used in Lab 3, please refer to our Lab 3 Tutorial for more information regarding these modules.

Last but not least, we've designed an overflow handler when mixing sound tracks together. Without overflow handler, the temporal register mentioned above may be added over its bits limitation (overflow), which further results in 'crack' sound which is unbearable for our system. For this reason, although the audio data has only 16 bits, the temporal register is designed to have 19 bits. Note that the value of audio datas are 'signed', thus when extending 16 bits data into 19 bits, the MSB should be extended rather than simply adding zeros. After adding all sound tracks together, our system use these additional three bits along with the original MSB to check if overflow has occur. The followings are the conditions indicating whether overflow has occurred:

```
if temp_register[18:15] = 4'b0000 // positive value without overflow
    output_data = temp_register[15:0]
else if temp_register[18:15] = 4'b1111 // negative value without overflow
    output_data = temp_register[15:0]
else if temp_register[18] == 1'b0 && temp_register[17:15] != 3'b0 // positive overflow
    output_data = 16'b0_111_1111_1111_1111 // clip to maximum positive value
else // negative overflow
    output_data = 16'b1_000_0000_0000_0000 // clip to minimum negative value
```

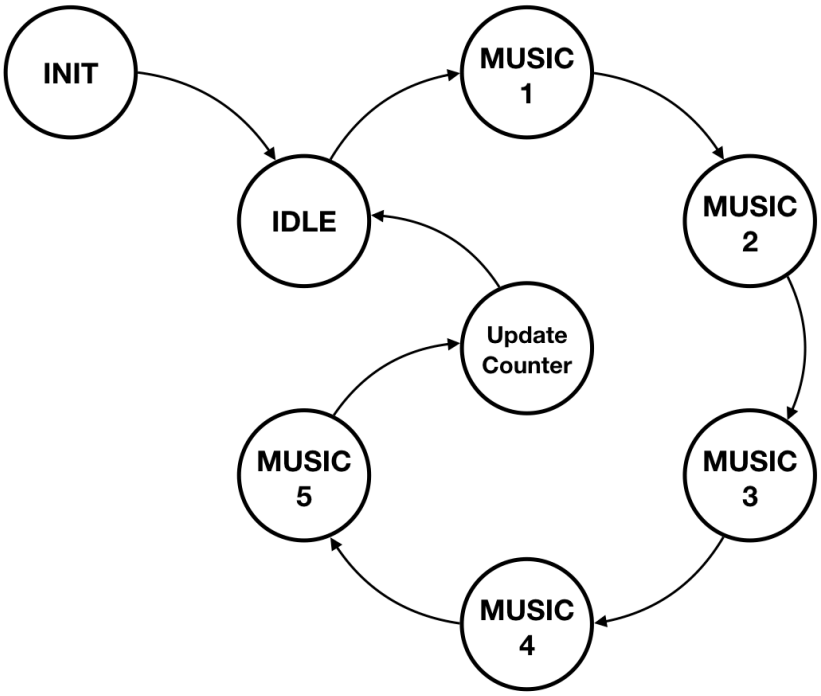


Figure 14. State Diagram of Music Mixer

- **Useful Debugging Tool**

  - **SignalTap logic analyzer**

    Quartus II provides a very useful tool to monitor your signal waveforms when running program on FPGA. It provides instant waveforms and thus is very helpful when debugging run-time errors.

    - Go to Tools → SignalTap II Logic Analyzer.
    - At the Setup section you can select signals you want to monitor, and set the basic clock at Signal Configuration.
    - Save your settings as xxx.stp.
    - Go back to your project and go to Assignments → Settings → SignalTap II Logic Analyzer and specify the stp file.
    - Compile your design. SignalTap basically insert probes in your circuits to monitor the signals specified by the user, so we have to recompile the design to activate them.
    - Program the sof file to your FPGA.
    - After the program start running, press "Autorun Analysis" to view your instant waveforms.