**VGA Configurable Metronome**

**CompE-470 Lab**

Designed by Tristan Richmond

821070777

1. **Introduction**

A metronome is an oscillatory tool used by musicians to consciously play their music to a consistent beat. It has the use of coordinating a whole band to a uniform tempo when performing live or even allowing musicians to count how many “clicks” or beats that have occurred in the single bar measure. This serves he purpose of allowing the musicians to know how to transition to new sections of the song. Although it tool was originally created mechanically with a pendulum, for decades digital metronomes have been the popular choice among musicians. Typically, these time keeping tools consist of a LED or buzzer speaker to indicate the beat. Additionally, there exists a control input, typically a knob encoder or push buttons, that allows the musician to change the tempo at which the beats occur. The measurement used for indicating tempos is BPM which stands for beats-per-minute.

The following design is inspired off the standard features of a metronome, but instead of utilizing a buzzer or LED as the primary indicator, it will display an animated indicator on LCD monitor over VGA protocol. The entire design is tested and supported on a Basys3 Discovery FPGA Board due to the included VGA port and wide variety of embedded I/O controls. Using 4 binary-encoded switches, the user may select a BPM between 60 and 135 in increments of 5 BPM per switch adjustment. Some additional features are a 4-digit 7-segment display that shows the current BPM and indicators for an accent beat.



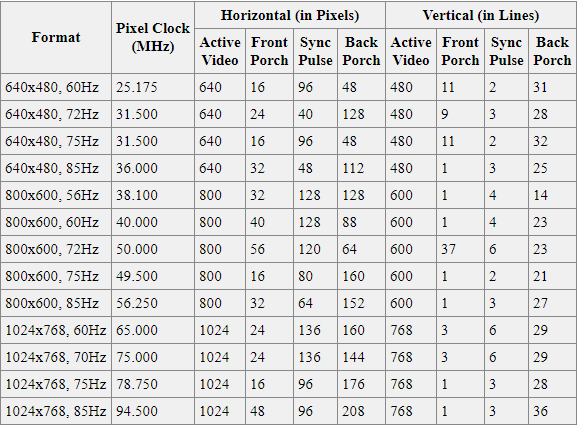
**Figure 1 – Block Diagram of Metronome**

1. **Module Descriptions**

In Figure 1, each module is displayed to show the relationships between each module and the LCD monitor. The top module has the role of organizing and connecting the major modules together through wires. Also, it takes the FPGA hardware inputs and drives the outputs through the FPGA hardware outputs. The inputs include 9 dip switches and externally generated clock signal of 100 MHz. Alternatively, the outputs include: 7-bit digit value, 4-bit digit select, 4-bit accent decimal, three 4-bit bus VGA color signals, four beat LEDs, the horizontal sync pin, and vertical sync pin which drive the VGA port. In total there are 3 major modules that handle separate features for the device.

The “Metronome Clock” module converts the 100 MHz clock to the BPM input configured frequencies of 1 – 2.25 Hz (60 – 135 BPM). These frequencies are determined by the external binary-encoded “BPM Select” input where the scale is 1-digit to 5 BPM. Using this new clock, a beat counter tracks which beat number it is in the measure of 4 beats. This is driven outward to the “VGA” module and the four beat LEDs. Since the new clock for the metronome is set at duty cycle of 10%, it gives a flash effect to better indicate the rising edge of the beat. Typically clock signals are a 50% duty cycle; however, in this application it may cause the user to play slightly slower than the desired tempo.

The “VGA” module has the task of configuring the parameters for a VGA signal and image to display on an LCD monitor. First, it divides the 100 MHz clock into a 50 MHz single with a 50% duty cycle for the “VGA Timings” module to use as a pixel clock. The pixel clock determines the frequency at which the display moves to the next pixel of the entire VGA 2-D matrix. The pixel clock frequency is a predetermined characteristic to each resolution size and refresh rate. In this case, a 50 MHz pixel clock was required to output 800x600 video with a 72 MHz refresh rate.



**Figure 2 -VGA Timing Parameters**

Furthermore, the “VGA Image” module generally handles what size and color the metronome image will appear onto the LCD monitor. This is done by a state machine determining which beat the metronome is one and whether that beat is accented. The accented beats draw a blue square, non-accented beats draw a green square, and if it is between two beats it displays a purple square. The square that is drawn is 200x200 pixels but may appear slightly stretched or shrunk due to the monitor’s receiving characteristics of the signal. Additionally, the “VGA Timing” module is required to determine the VGA signal’s position and generate the vertical and horizontal synchronization flags. This is necessary since the VGA signal requires a larger 2-D counter than just the display resolution values. For example, this metronome used an 800x600 resolution size but requires 1040x666 pixels to function as a VGA signal.

The extra non-displayed pixels serve the purpose as a time buffer for the monitor to travel down a row or back to the origin point. These off-screen sections are referred to as the front and back porches. The horizontal sync flag is triggered when the signal has reached the last pixel in row while the vertical sync flag is triggered at the last pixel in the column. It is through these flags that a monitor can synchronize its displaying procedure with the signal’s video. The v-sync and h-sync signals are outputted to the LCD monitor alongside the 4-bit red, green, and blue buses to create a stable VGA signal.

The “Metronome Information” module handles interpreting the selected BPM value and identifying which beats are accented. It then takes these data points to display them onto a 4-digit 7-segment display. Through a multiplexer on the “BPM Select” input, the module decides which values to assign each digit of the 7-segment display. The display is organized to display a 0 on the leftmost digit and the BPM value on the remaining three digits. Furthermore, each digit contains its own decimal point which is utilized as the accent indicator. Based on the “Accent Select” input, each decimal whose corresponding beat is accented will be turned on while the non-accented ones will be off. The leftmost decimal point represents the first beat of the measure while the rightmost digit is the fourth beat.

To drive a 7-segment display, a proper refresh rate clock is required to function correctly. Inside the “Metronome Information” module, another clock divider is utilized to convert the 100 MHz system clock into a 250 Hz 50% duty clock. This signal serves as the refresh rate of the 7-segment display to transition to through each digit at a possible rate for the hardware.

1. **Design Choices**

This project had multiple ideas on its features and how to implement them. Along the process unseen bugs arose leading to a slightly different implementation. First, the initial requirement for the resolution was 640x480 at 60 Hz since it is the smallest VGA resolution at a common monitor refresh rate. Unfortunately, the monitor that was used for testing the metronome was unable to sync correctly with the VGA signal. Alternatively, changing to a resolution of 800x600 with a 72 Hz refresh rate supported the signal resulting in a colored square being displayed.

Upon displaying a square that originally would have fit the entire resolution, large amounts of screen tearing were apparent. This was due to the metronome image changing faster than the display could draw onto the monitor. To solve this issue, the draw image was reduced in pixel size to a 200x200 square which could be drawn quicker than the fastest metronome setting.

For input controls of the metronome there are three main sections that all utilize different parts of the Basys3 switch array. The leftmost switch was selected as the reset switch since it is easily identifiable in the case the user wishes to reset the board. The next section handles switches 13 through 10 on the Basys3. These are one-hot encoded as the accent select since each one uniquely controls its corresponding beat. Additionally, these switch locations were chosen since the almost align vertically with the decimal points on the 7-segment display. Finally, the 4 rightmost switches on the Basys3 serve as the binary-encoded BPM select input. Like the RST switch, they are placed on the edge for ease of accessibility to the user. They are binary encoded with each value representing 5 BPM to minimize the amount of inaccuracy in the timers.

1. **Conclusion**

The VGA metronome project displays what can be achieved with a display protocol and internal timers. Through understanding clock dividers various timers were implemented to create a consistent metronome signal, 7-segment display signal, and a VGA display output. Allowing the user of a device to control settings externally provides the benefit or multiple features that are easy to access. In hardware, state-machines are the backbone to efficient user input controls as it supports ideas such as multiple BPM speeds and accented beats. It is by learning and incrementally using testbenches that large projects are achievable to a hardware engineers since improves the rate at which bugs are solved. Utilizing these tools and techniques is what allows for ideas such as the VGA metronome to be built.

1. **Source Code**

**Top Module Source Code**

module Metronome\_Info

(

input CLK\_100MHz, RST,

input [3:0] BPM\_Select, Accent\_Indicator,

output [7:0] Digit\_Cathode,

output [3:0] Select\_Anode

);

reg [19:0] Segment\_Counter = 500000;

wire Segment\_CLK;

reg [3:0] BPM\_Value\_A, BPM\_Value\_B, BPM\_Value\_C;

// Generate 250Hz clock for refreshing 7segment display

Clock\_Divider #(19,50) Segment\_Refresh(.CLK(CLK\_100MHz),.RST(RST),.Counter\_Limit(Segment\_Counter),.CLK\_Slow(Segment\_CLK));

always @(posedge Segment\_CLK or posedge RST)

begin

if(RST)

begin

BPM\_Value\_C <= 0;

BPM\_Value\_B <= 0;

BPM\_Value\_A <= 0;

end

else

begin

//BPM\_Value\_C <= 4'b0001;

//BPM\_Value\_B <= 4'b0001;

//BPM\_Value\_A <= 4'b0001;

// When the BPM is odd display 5 in A segment

// Otherwise display 0 in A segment

if(BPM\_Select[0]==1) BPM\_Value\_A <= 5;

else BPM\_Value\_A <= 0;

// When the BPM is greater than 100, display 1 in C segment

// Otherwise display 0 in C segmnet

if(BPM\_Select[3]==1) BPM\_Value\_C <= 1;

else BPM\_Value\_C <= 0;

// When the BPM is greater than 100, the 2 bits represent the tenths value of post 100 BPM

// For example, 0->100 1->110 2->120 etc.

// Otherwise, the 2 bits represent the tenths value of post 60 BPM

// For example, 0->60 1->70 2->80 etc.

if(BPM\_Value\_C==1) BPM\_Value\_B <= {2'b00,BPM\_Select[2:1]};

else BPM\_Value\_B <= BPM\_Select[2:1] + 3'b110;

end

end

Seven\_Segment Metronome\_Info\_Display(.CLK(CLK\_100MHz),.RST(RST),.Refresh\_Rate(Segment\_CLK),.Data\_A(BPM\_Value\_A),.Data\_B(BPM\_Value\_B),.Data\_C(BPM\_Value\_C)

,.Data\_D(0),.Dec\_A(Accent\_Indicator[3]),.Dec\_B(Accent\_Indicator[2]),.Dec\_C(Accent\_Indicator[1])

,.Dec\_D(Accent\_Indicator[0]),.Digit\_Cathode(Digit\_Cathode),.Select\_Anode(Select\_Anode));

endmodule

**Metronome Clock Source Code**

module Metronome\_Clock

(

input CLK, RST,

input [3:0] BPM\_Select, Accent\_Select,

output reg [0:3] Beat\_Out,

output [Met\_Bit\_Size-1:0] Metronome\_Counter\_Limit

);

// Met\_Bit\_Size sets allocates the necesary amount of bits for the counter variable

// Met\_Duty\_Cycle sets the duty cycle of metronome waveform; supports options 0%-100%

// Example: 5 -> 5%, 20 -> 20%, ... 100 -> 100%

parameter Met\_Bit\_Size = 27, Met\_Duty\_Cycle = 10;

// Configure the maximumn counter value for the metronome clock

// Depending on inputed BPM selection, the BPM will multiply to a constant value which is equivalent to 5 BPM

// This product will be subtracted from the base limit which is the equivalent for 60 BPM

//wire [Met\_Bit\_Size-1:0] Metronome\_Counter\_Limit; 3125000

assign Metronome\_Counter\_Limit = 100000000 - (BPM\_Select \* 4166667);

reg [1:0] Beat\_Counter;

initial Beat\_Counter = 2'b0;

Clock\_Divider #(Met\_Bit\_Size,Met\_Duty\_Cycle) Metronome\_Clock\_Divider(.CLK(CLK),.RST(RST),.Counter\_Limit(Metronome\_Counter\_Limit)

,.CLK\_Slow(Met\_CLK));

// State Machine that adds to the Beat Counter each time a beat is reached

// 4 beats total before it resets to the 1st beat

always @(posedge Met\_CLK or posedge RST)

begin

if(RST)

begin

Beat\_Counter <= 2'b0;

end

else

begin

Beat\_Counter <= Beat\_Counter + 1;

end

end

// State Machine that displays the beat indication on the corresponding beat to LED

// Four LEDs going left to right

// Example: Beat 2 XOXX

// X is OFF o is ON

always @(posedge CLK or posedge RST)

begin

if(RST)

begin

Beat\_Out <= 4'b0;

end

else

begin

Beat\_Out[Beat\_Counter] <= Met\_CLK;

Beat\_Out[Beat\_Counter+1] <= 1'b0;

Beat\_Out[Beat\_Counter+2] <= 1'b0;

Beat\_Out[Beat\_Counter+3] <= 1'b0;

end

end

endmodule

**Clock Divider Source Code**

module Clock\_Divider

#(parameter CLK\_Size = 27, Duty\_Cycle = 50)

(

input CLK, RST,

input [CLK\_Size-1:0] Counter\_Limit,

output reg CLK\_Slow

);

reg [CLK\_Size-1:0] Counter;

initial Counter = 0;

initial CLK\_Slow = 1;

// State Machine that adds to a counter that counts until a specifed limit

// Once the computed duty cycle value of the maximumn is reached, the output signal turns 0

// Once the maximumn counter value is reached, the output signal turns 1

always @(posedge CLK or posedge RST)

begin

if(RST)

begin

Counter <= 0;

CLK\_Slow <= 1;

end

else

begin

if(Counter >= Counter\_Limit - 1)

begin

Counter <= 0;

CLK\_Slow <= 1;

end

else if(Counter == (Counter\_Limit\*Duty\_Cycle/100-1))

begin

CLK\_Slow <= 0;

Counter <= Counter + 1;

end

else Counter <= Counter + 1;

end

end

endmodule

**VGA Source Code**

module VGA

(

input CLK\_100MHz, RST,

input [3:0] Metronome\_Beat, Accent\_Beat,

output H\_Sync, V\_Sync,

output [3:0] Red, Green, Blue

);

wire CLK\_50MHz;

reg [1:0] CLK\_50MHz\_Limit = 2;

Clock\_Divider #(2,50)Pixel\_CLK(.CLK(CLK\_100MHz),.RST(RST),.Counter\_Limit(CLK\_50MHz\_Limit),.CLK\_Slow(CLK\_50MHz));

VGA\_Image VGA\_Display(.Pixel\_CLK(CLK\_50MHz),.RST(RST),.Beat(Metronome\_Beat),.Accent(Accent\_Beat),.H\_Sync(H\_Sync),.V\_Sync(V\_Sync),

.Red(Red),.Green(Green),.Blue(Blue));

endmodule

**VGA Image Source Code**

module VGA\_Image

(

input Pixel\_CLK, RST,

input [3:0] Beat, Accent,

output H\_Sync, V\_Sync,

output reg [3:0] Red, Green, Blue

);

localparam Left\_Square = 300, Right\_Square = 500,

Top\_Square = 200, Bottom\_Square = 400;

wire [10:0] X\_Pos, Y\_Pos;

wire Draw\_Display;

wire New\_Frame;

VGA\_Timing Display\_Timing(.Pixel\_CLK(Pixel\_CLK),.RST(RST),.H\_Sync(H\_Sync),.V\_Sync(V\_Sync),.Draw\_Image(Draw\_Display)

,.X\_Pos(X\_Pos),.Y\_Pos(Y\_Pos));

always @ \*

begin

if(RST)

begin

Red <= 4'hF;

Green <= 4'hF;

Blue <= 4'hF;

end

// Will only enter block when in display section of X & Y grid

else if(Draw\_Display)

begin

// If cursor is inside 200x200 square, then display purple or green square

// Green if beat, purple otherwise

if(X\_Pos >= Left\_Square && X\_Pos < Right\_Square && Y\_Pos >= Top\_Square && Y\_Pos < Bottom\_Square)

begin

// If the selected beat doesn't have the accent switch indicated, display green

if((Beat == 4'b1000 && Accent[0] == 1'b0)||(Beat == 4'b0100 && Accent[1] == 1'b0)

||(Beat == 4'b0010 && Accent[2] == 1'b0)||(Beat == 4'b0001 && Accent[3] == 1'b0))

begin

Red <= 4'h0;

Green <= 4'hF;

Blue <= 4'h0;

end

// If the selected beat has the accent switch indicated, display blue

else if((Beat == 4'b1000 && Accent[0] == 1'b1)||(Beat == 4'b0100 && Accent[1] == 1'b1)

||(Beat == 4'b0010 && Accent[2] == 1'b1)||(Beat == 4'b0001 && Accent[3] == 1'b1))

begin

Red <= 4'h3;

Green <= 4'h0;

Blue <= 4'hF;

end

else

begin

Red <= 4'hF;

Green <= 4'h0;

Blue <= 4'hF;

end

end

// Displays black if cursor is not within the metronome square

else

begin

Red <= 4'h0;

Green <= 4'h0;

Blue <= 4'h0;

end

end

end

endmodule

**VGA Timing Source Code**

module VGA\_Timing

(

input Pixel\_CLK, RST,

output reg H\_Sync, V\_Sync,

output Draw\_Image,

output reg [10:0] X\_Pos, Y\_Pos

);

localparam H\_Display\_End = 799, H\_Sync\_Start = 855, H\_Sync\_End = 975, H\_Period = 1039,

V\_Display\_End = 599, V\_Sync\_Start = 636, V\_Sync\_End = 642, V\_Period = 665;

// Displaying is enabled when X & Y position is within 800x600 display grid

assign Draw\_Image = 1'b1 ? (X\_Pos <= H\_Display\_End && Y\_Pos <= V\_Display\_End) : 1'b0;

always @(posedge Pixel\_CLK or posedge RST)

begin

if(RST)

begin

X\_Pos <= 11'b0;

Y\_Pos <= 11'b0;

H\_Sync <= 1'b0;

V\_Sync <= 1'b0;

//Draw\_Image <= 1'b0;

end

else

begin

// Horizontal Timings

if(X\_Pos == H\_Period)

begin

X\_Pos <= 1'b0; // X position is reset to 0

if(Y\_Pos == V\_Period) Y\_Pos <= 1'b0; // Maximum Screen Timing reached

else Y\_Pos <= Y\_Pos + 1; // Increment Y position because of the X position reset

end

else if(X\_Pos >= H\_Sync\_Start && X\_Pos <= H\_Sync\_End)

begin

X\_Pos <= X\_Pos + 1; // Increment X position and Output high for H\_Sync only when

H\_Sync <= 1'b1; // X counter is in the porch sections of the screen

end

else

begin

X\_Pos <= X\_Pos + 1; // Increment X position and Output low for H\_Sync only when

H\_Sync <= 1'b0; // Y counter is in displaying or syncing sections of the screen

end

// Vertical Timings

if(Y\_Pos >= V\_Sync\_Start && Y\_Pos <= V\_Sync\_End)

begin

V\_Sync <= 1'b1; // Output high for V\_Sync only when Y counter is in the porch sections of the screen

end

else

begin

V\_Sync <= 1'b0; // Output low for V\_Sync only when Y counter is displaying or syncing of the screen

end

end

end

endmodule

**Metronome Info Source Code**

module Metronome\_Info

(

input CLK\_100MHz, RST,

input [3:0] BPM\_Select, Accent\_Indicator,

output [7:0] Digit\_Cathode,

output [3:0] Select\_Anode

);

reg [19:0] Segment\_Counter = 500000;

wire Segment\_CLK;

reg [3:0] BPM\_Value\_A, BPM\_Value\_B, BPM\_Value\_C;

// Generate 250Hz clock for refreshing 7segment display

Clock\_Divider #(19,50) Segment\_Refresh(.CLK(CLK\_100MHz),.RST(RST),.Counter\_Limit(Segment\_Counter),.CLK\_Slow(Segment\_CLK));

always @(posedge Segment\_CLK or posedge RST)

begin

if(RST)

begin

BPM\_Value\_C <= 0;

BPM\_Value\_B <= 0;

BPM\_Value\_A <= 0;

end

else

begin

// When the BPM is odd display 5 in A segment

// Otherwise display 0 in A segment

if(BPM\_Select[0]==1) BPM\_Value\_A <= 5;

else BPM\_Value\_A <= 0;

// When the BPM is greater than 100, display 1 in C segment

// Otherwise display 0 in C segmnet

if(BPM\_Select[3]==1) BPM\_Value\_C <= 1;

else BPM\_Value\_C <= 0;

// When the BPM is greater than 100, the 2 bits represent the tenths value of post 100 BPM

// For example, 0->100 1->110 2->120 etc.

// Otherwise, the 2 bits represent the tenths value of post 60 BPM

// For example, 0->60 1->70 2->80 etc.

if(BPM\_Value\_C==1) BPM\_Value\_B <= {2'b00,BPM\_Select[2:1]};

else BPM\_Value\_B <= BPM\_Select[2:1] + 3'b110;

end

end

Seven\_Segment Metronome\_Info\_Display(.CLK(CLK\_100MHz),.RST(RST),.Refresh\_Rate(Segment\_CLK),.Data\_A(BPM\_Value\_A),.Data\_B(BPM\_Value\_B),.Data\_C(BPM\_Value\_C)

,.Data\_D(0),.Dec\_A(Accent\_Indicator[3]),.Dec\_B(Accent\_Indicator[2]),.Dec\_C(Accent\_Indicator[1])

,.Dec\_D(Accent\_Indicator[0]),.Digit\_Cathode(Digit\_Cathode),.Select\_Anode(Select\_Anode));

Endmodule

**Seven Segment Source Code**

module Seven\_Segment

(

input CLK, RST, Refresh\_Rate,

input [3:0] Data\_A, Data\_B, Data\_C, Data\_D,

input Dec\_A, Dec\_B, Dec\_C, Dec\_D,

output reg [7:0] Digit\_Cathode,

output reg [3:0] Select\_Anode

);

localparam Display\_A = 2'b00, Display\_B = 2'b01,

Display\_C = 2'b10, Display\_D = 2'b11,

Zero = 4'b0000, One = 4'b0001, Two = 4'b0010, Three = 4'b0011,

Four = 4'b0100, Five = 4'b0101, Six = 4'b0110, Seven = 4'b0111,

Eight = 4'b1000, Nine = 4'b1001, A = 4'b1010, B = 4'b1011,

C = 4'b1100, D = 4'b1101, E = 4'b1110, F = 4'b1111;

reg [1:0] Segment\_Select;

always @(posedge Refresh\_Rate or posedge RST)

begin

if(RST) Segment\_Select <= 2'b00;

else Segment\_Select <= Segment\_Select + 2'b01;

end

always @(posedge CLK)

begin

case(Segment\_Select)

Display\_A:

begin

Select\_Anode <= 4'b1110;

if(Dec\_A) Digit\_Cathode[7] <= 1'b0;

else Digit\_Cathode[7] <= 1'b1;

case(Data\_A)

Zero: Digit\_Cathode[6:0] <= 7'b1000000;

One: Digit\_Cathode[6:0] <= 7'b1111001;

Two: Digit\_Cathode[6:0] <= 7'b0100100;

Three: Digit\_Cathode[6:0] <= 7'b0110000;

Four: Digit\_Cathode[6:0] <= 7'b0011001;

Five: Digit\_Cathode[6:0] <= 7'b0010010;

Six: Digit\_Cathode[6:0] <= 7'b0000010;

Seven: Digit\_Cathode[6:0] <= 7'b1111000;

Eight: Digit\_Cathode[6:0] <= 7'b0000000;

Nine: Digit\_Cathode[6:0] <= 7'b0010000;

A: Digit\_Cathode[6:0] <= 7'b0001000;

B: Digit\_Cathode[6:0] <= 7'b0000011;

C: Digit\_Cathode[6:0] <= 7'b1000110;

D: Digit\_Cathode[6:0] <= 7'b0100001;

E: Digit\_Cathode[6:0] <= 7'b0000110;

F: Digit\_Cathode[6:0] <= 7'b0001110;

default: Digit\_Cathode[6:0] <= 7'b0011011;

endcase

end

Display\_B:

begin

Select\_Anode <= 4'b1101;

if(Dec\_B) Digit\_Cathode[7] <= 1'b0;

else Digit\_Cathode[7] <= 1'b1;

case(Data\_B)

Zero: Digit\_Cathode[6:0] <= 7'b1000000;

One: Digit\_Cathode[6:0] <= 7'b1111001;

Two: Digit\_Cathode[6:0] <= 7'b0100100;

Three: Digit\_Cathode[6:0] <= 7'b0110000;

Four: Digit\_Cathode[6:0] <= 7'b0011001;

Five: Digit\_Cathode[6:0] <= 7'b0010010;

Six: Digit\_Cathode[6:0] <= 7'b0000010;

Seven: Digit\_Cathode[6:0] <= 7'b1111000;

Eight: Digit\_Cathode[6:0] <= 7'b0000000;

Nine: Digit\_Cathode[6:0] <= 7'b0010000;

A: Digit\_Cathode[6:0] <= 7'b0001000;

B: Digit\_Cathode[6:0] <= 7'b0000011;

C: Digit\_Cathode[6:0] <= 7'b1000110;

D: Digit\_Cathode[6:0] <= 7'b0100001;

E: Digit\_Cathode[6:0] <= 7'b0000110;

F: Digit\_Cathode[6:0] <= 7'b0001110;

default: Digit\_Cathode[6:0] <= 7'b0011011;

endcase

end

Display\_C:

begin

Select\_Anode <= 4'b1011;

if(Dec\_C) Digit\_Cathode[7] <= 1'b0;

else Digit\_Cathode[7] <= 1'b1;

case(Data\_C)

Zero: Digit\_Cathode[6:0] <= 7'b1000000;

One: Digit\_Cathode[6:0] <= 7'b1111001;

Two: Digit\_Cathode[6:0] <= 7'b0100100;

Three: Digit\_Cathode[6:0] <= 7'b0110000;

Four: Digit\_Cathode[6:0] <= 7'b0011001;

Five: Digit\_Cathode[6:0] <= 7'b0010010;

Six: Digit\_Cathode[6:0] <= 7'b0000010;

Seven: Digit\_Cathode[6:0] <= 7'b1111000;

Eight: Digit\_Cathode[6:0] <= 7'b0000000;

Nine: Digit\_Cathode[6:0] <= 7'b0010000;

A: Digit\_Cathode[6:0] <= 7'b0001000;

B: Digit\_Cathode[6:0] <= 7'b0000011;

C: Digit\_Cathode[6:0] <= 7'b1000110;

D: Digit\_Cathode[6:0] <= 7'b0100001;

E: Digit\_Cathode[6:0] <= 7'b0000110;

F: Digit\_Cathode[6:0] <= 7'b0001110;

default: Digit\_Cathode[6:0] <= 7'b0011011;

endcase

end

Display\_D:

begin

Select\_Anode <= 4'b0111;

if(Dec\_D) Digit\_Cathode[7] <= 1'b0;

else Digit\_Cathode[7] <= 1'b1;

case(Data\_D)

Zero: Digit\_Cathode[6:0] <= 7'b1000000;

One: Digit\_Cathode[6:0] <= 7'b1111001;

Two: Digit\_Cathode[6:0] <= 7'b0100100;

Three: Digit\_Cathode[6:0] <= 7'b0110000;

Four: Digit\_Cathode[6:0] <= 7'b0011001;

Five: Digit\_Cathode[6:0] <= 7'b0010010;

Six: Digit\_Cathode[6:0] <= 7'b0000010;

Seven: Digit\_Cathode[6:0] <= 7'b1111000;

Eight: Digit\_Cathode[6:0] <= 7'b0000000;

Nine: Digit\_Cathode[6:0] <= 7'b0010000;

A: Digit\_Cathode[6:0] <= 7'b0001000;

B: Digit\_Cathode[6:0] <= 7'b0000011;

C: Digit\_Cathode[6:0] <= 7'b1000110;

D: Digit\_Cathode[6:0] <= 7'b0100001;

E: Digit\_Cathode[6:0] <= 7'b0000110;

F: Digit\_Cathode[6:0] <= 7'b0001110;

default: Digit\_Cathode[6:0] <= 7'b0011011;

endcase

end

default:

begin

Select\_Anode <=4'b0000;

Digit\_Cathode <= 8'b10001110;

end

endcase

end

endmodule