**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 or even allowing for a musician to count how many “clicks” or beats that have occurred in the single measure bar to know when to change parts of a song. Although it was originally design by mechanical means, 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 to 120 in increments of 5 BPM per switch adjustment.



**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 will take the FPGA hardware inputs and drive the outputs through the FPGA hardware outputs. The inputs include five dip switches and externally generated clock signal of 100 MHz. Alternatively, the outputs include the three 4-bit bus VGA color signals, the horizontal sync pin, and vertical sync pin which drive the VGA port. The Metronome Clock module converts the 100 MHz clock to the BPM input configured frequencies of 1 – 2 Hz. Using this new clock, a beat counter tracks which beat number it is in the measure of 4 beats. The VGA module divides the 100 MHz clock into a 50 MHz single for the VGA Timing module to use as a pixel counter. It also includes the VGA image which produces the necessary signals on each of the VGA pins to drive the LCD monitor display.

1. **Source Code**

**Top Module Source Code**

module Top\_Module

(

input CLK, RST,

input [3:0] BPM\_Select,

output VGA\_H\_Sync, VGA\_V\_Sync,

output [3:0] LED\_Met, VGA\_Red, VGA\_Green, VGA\_Blue

);

// Instantuate Metronome Clock module to compute BPM, generate Beat number, and Beat signal for other modules

Metronome\_Clock Met\_Clock(.CLK(CLK),.RST(RST),.BPM\_Select(BPM\_Select),.Beat\_Out(LED\_Met));

// Display Image module to translate metronome singal into image for display

VGA Metronome\_Display(.CLK\_100MHz(CLK),.RST(RST),.Metronome\_Beat(LED\_Met),.H\_Sync(VGA\_H\_Sync),.V\_Sync(VGA\_V\_Sync),.Red(VGA\_Red),.Green(VGA\_Green),

.Blue(VGA\_Blue));

Endmodule

**Metronome Clock Source Code**

module Metronome\_Clock

(

input CLK, RST,

input [3:0] BPM\_Select,

output reg [0:3] Beat\_Out

);

// Met\_Bit\_Size sets allocates the necessary 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 = 5;

// Configure the maximum counter value for the metronome clock

// Depending on input 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;

assign Metronome\_Counter\_Limit = 100000000 - (BPM\_Select \* 3125000);

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,

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\_25MHz),.RST(RST),.Beat(Metronome\_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,

output H\_Sync, V\_Sync,

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

);

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

wire Draw\_Flag;

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

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

always @ \*

begin

if(RST)

begin

Red <= 4'h0;

Green <= 4'h0;

Blue <= 4'h0;

end

else if(Draw\_Flag)

begin

if(Beat!=4'b0000)

begin

Red <= 4'h0;

Green <= 4'hF;

Blue <= 4'h0;

end

else

begin

Red <= 4'h0;

Green <= 4'hA;

Blue <= 4'hA;

end

end

end

endmodule

**VGA Timing Source Code**

module VGA\_Timing

(

input Pixel\_CLK, RST,

output reg H\_Sync, V\_Sync, 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;

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

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

else 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

// Draw Image Set

if(X\_Pos <= H\_Display\_End && Y\_Pos <= V\_Display\_End) Draw\_Image <= 1'b1; // Draw image flag high during

// Display seciton

else Draw\_Image <= 1'b0; // Draw image flag low outside of display section

end

end

endmodule