Intoduction to Mini Project - Part 1

Maryam Hemmati

Department of Electrical, Computer and Software Engineering
University of Auckland

email: m.hemmati@auckland.ac.nz

COMPSYS 305-Digital Systems Design

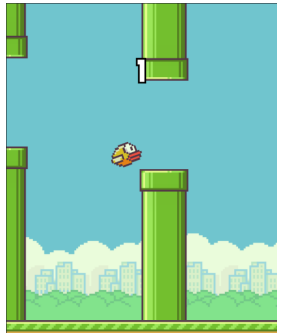
1 April 2021

Content

Mini Project Objective

The goal of the mini project is to design a simple game console.

- The game is Flappy Bird.
- The game is implemented on DE0 board.
- The game is controlled and played using
  - A PS/2 mouse
  - DIP switches on the DE0 board
  - Push-buttons on the DE0 board
- The game is displayed on a VGA screen.
  - with a resolution of **640x480** pixels



Mini Project Objective

Game Description

- The bird can move up or down
  - It is controlled by a PS/2 mouse.
  - If the bird is not flapping, it will free-fall towards the ground.
  - The bird must not touch anything when flying, otherwise, it will lose life points.
- The game may consist of different types of obstacles and gifts
  - pipes
  - dollars, medicine boxes, special flying abilities
- The screen must be kept in motion from the right-hand side to the left-hand side.
  - The speed increases with the game level
- The level of difficulty can be controlled by other criteria
  - The types of obstacles

Mini Project Objective

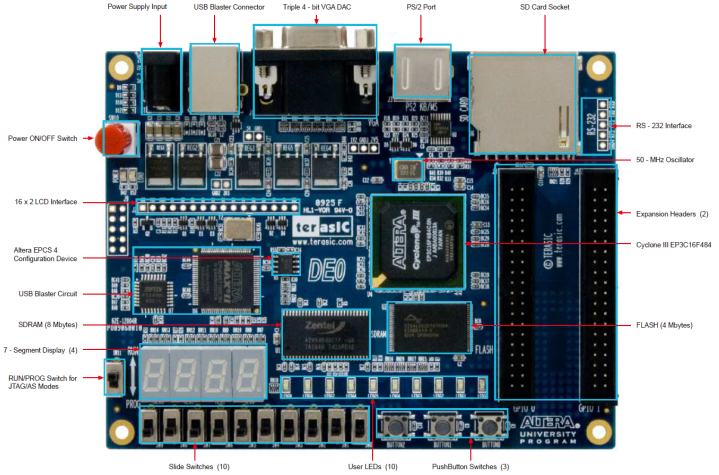
Game Modes

- **Training Mode**
  - Allows the player to practice at the lowest game level.
  - Will continue for a specific time.
- **Single-player Game Mode**
  - The game will proceed to more advanced levels following certain criteria.
  - The time, distance, or the number of obstacles passed could be used as such criteria.

The game mode can be determined by using a DIP switch on the console or through a selection on the welcome screen.

DE0 Board

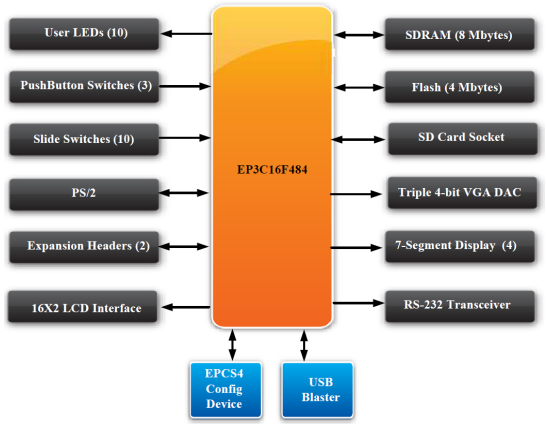
The hardware platform that you will use for implementing the game console is Terasic DE0 board.



DE0 Board

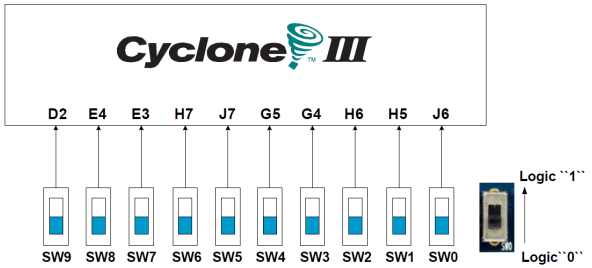
DE0 board includes Altera **Cyclone III 3C16** FPGA device

- All the connections are made through the Cyclone III FPGA device.
  - Gives the flexibility to the user to configure the FPGA to implement any system design.
- The DE0 board includes a **50 MHz** clock signal.



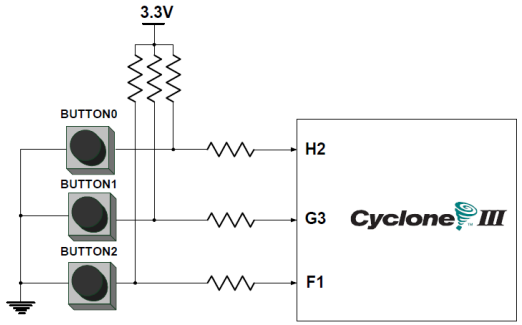
DE0 Board - Switches

- There are 10 slide switches (sliders) on the DE0 board.
- These switches are used as level-sensitive data inputs to a circuit.
  - When a switch is in the **DOWN** position it provides a **low logic level**.
  - When the switch is in the **UP** position it provides a **high logic level**.
- The FPGA pins connected to these switches (and any other I/O) should be defined in the project through pin assignment.



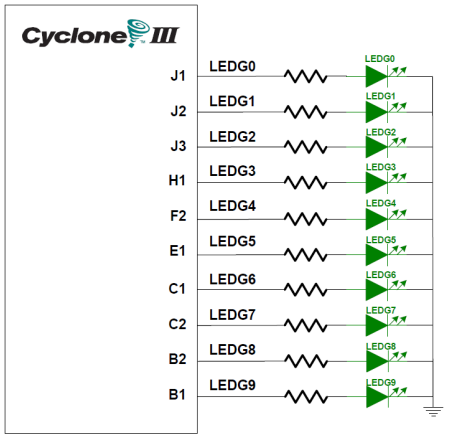
DE0 Board - Pushbuttons

- The DE0 board provides three pushbutton switches.
- BUTTON0, BUTTON1, and BUTTON2 are connected directly to the Cyclone III FPGA as an input.
  - Each button provides a **high** logic level when it is **not pressed**.
  - The button provides a **low** logic level when it is **pressed**.



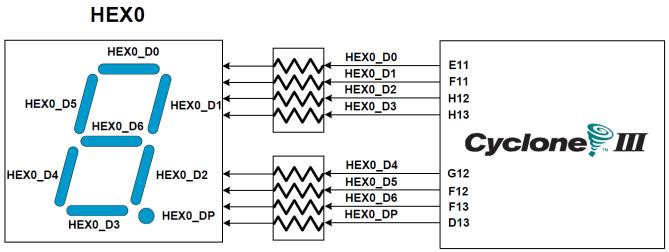
DE0 Board - LEDs

- There are 10 user-controllable LEDs on the DE0 board.
- Each LED is driven directly by a pin on the Cyclone III FPGA.
  - Driving associated pin to a **high** logic level turns the LED **on**.
  - Driving the pin **low** turns it **off**.



DE0 Board - Seven Segments

- The DE0 board has four 7-segment displays.
- They are connected to pins on the Cyclone III FPGA.
  - Applying a **low** logic level to a segment causes it to **light up**.
  - Applying a **high** logic level turns it **off**.
- Each segment in a display is identified by an index from 0 to 6.



DE0 Board - FPGA Configuration

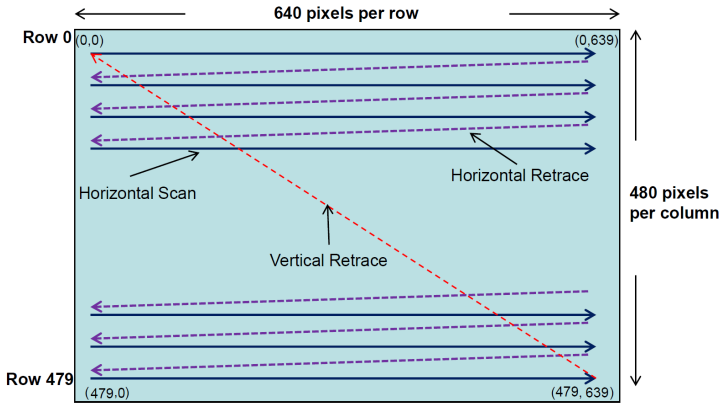
There are two different modes for configuring the **Cyclone III FPGA** on DE0 board.

- **JTAG programming**
  - The configuration bit stream is downloaded **directly** into the **Cyclone III FPGA**.
  - The FPGA will retain this configuration as long as power is applied to the board.
  - The configuration is lost when the power is turned off.
- **Active Serial programming**
  - The configuration bit stream is downloaded into the Altera **EPCS4 serial EEPROM chip**.
  - It provides non-volatile storage of the bit stream.
  - When the board is turned on, the configuration data in the EPCS4 device is automatically loaded into the Cyclone III FPGA.

Configuration bit stream is downloaded into the board through the USB Blaster.

VGA Interface

- **VGA (Video Graphics Array)** is a popular display standard developed by IBM and introduced in 1987.
- The resolution of the VGA screen can vary but a standard default size is **640x480 pixels**.
- The screen refreshes the display from left to right, top to bottom.



VGA Interface

Image on VGA screen is displayed by turning the pixels ON and OFF.

- Video signal must redraw the entire screen 60 times per sec (**60Hz**) to avoid flickers.
  - Human eyes detect flickers at refresh rate less than 30Hz.
- We will use the common VGA display standard at **25MHz** pixel rate with **640x480** resolution.
  - Each pixel takes 40ns at 25MHz pixel rate.

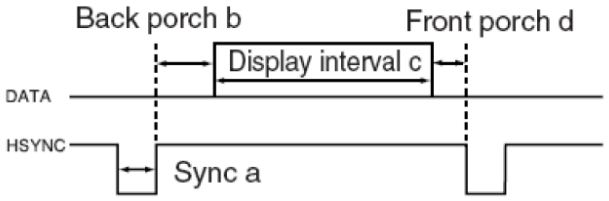
VGA Interface

VGA video standard contains 5 active signals:

- **Horizontal** and **vertical synchronisation** signals.
- Three analog signals for **red**, **green** and **blue** (**RGB**) colours formation.
  - By changing the analog voltage levels of the RGB signals, different colours can be produced.
  - Depending on the number of bits supported by the development board, different amount of colours can be represented.

VGA Interface - Horizontal Synchronisation

- **Horizontal sync** signifies the end of one row of data (i.e. 640 pixels) and the start of the next.
  - The data (RGB) inputs on the monitor must be off for a time period called the **back porch** (b) after the hsync pulse occurs.
  - During the data **display interval** (c) the **RGB** data drives each pixel in turn across the row being displayed.
    - ★ When data display interval is finished, the beam returns from the right most position to left most. During the return process, no pixel data is displayed.
  - **Front porch** (d) is a time period where the RGB signals must again be off before the next hsync pulse can occur.



VGA Interface - Horizontal Synchronisation

- **Horizontal sync** (a) corresponds to 96 pixels.
- **Back porch** (b) corresponds to 48 pixels.
- **Display interval** (c) corresponds to 640 pixels.
- **Front porch** (d) corresponds to 16 pixels.

VGA horizontal timing specification

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(Mhz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25 (640/c)

VGA Interface - Vertical Synchronisation

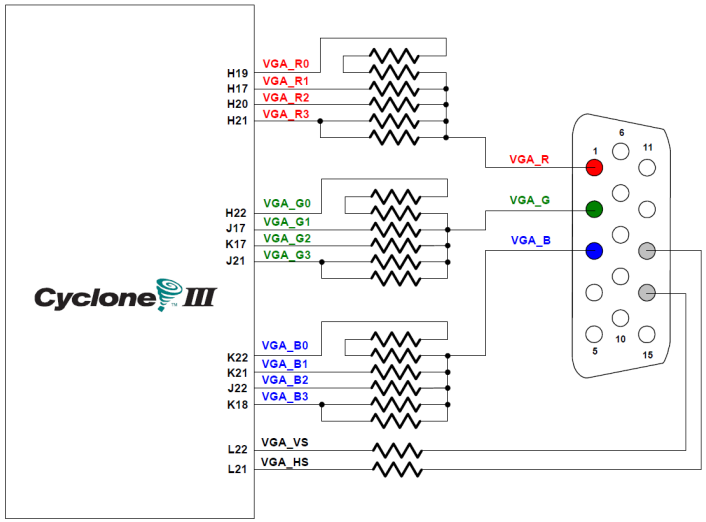
Vertical sync pulse signifies the end of one frame and the start of the next, and the data refers to the set of rows in the frame.

- **Vertical sync** (a) corresponds to 2 lines.
- **Back porch** (b) corresponds to 33 lines.
- **Display interval** (c) corresponds to 480 lines.
- **Front porch** (d) corresponds to 10 lines.

VGA vertical timing specification

VGA mode		Vertical Timing Spec			
Configuration	Resolution (HxV)	a(lines)	b(lines)	c(lines)	d(lines)
VGA(60Hz)	640x480	2	33	480	10

VGA Interface - DE0 Board



VGA Interface - VGA_Sync Component

We need a component to drive the control signals to the display and provide pixel values at the right rate.

- In order to generate the VGA signal at 25 MHz, the clock signal provided by DE0 (50MHz) needs to be halved.
- 25MHz clock signal can be used by counters to generate the horizontal and vertical sync signals.
- The counters also represent row and column address of a pixel, which can be used by other components to retrieve pixel information.

VGA Interface - VGA_Sync Component

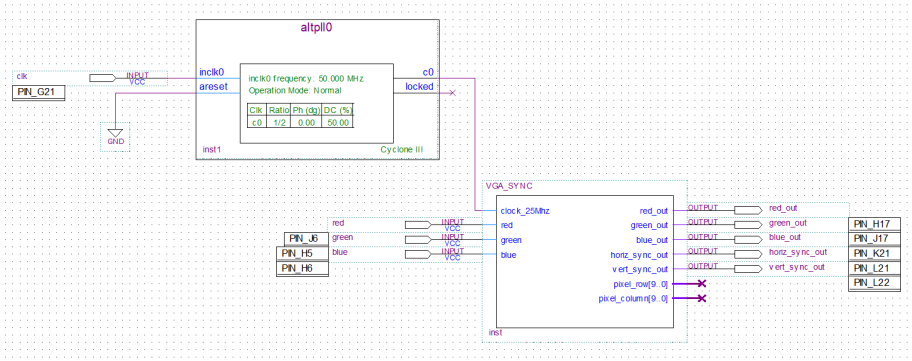
```
ENTITY VGA_SYNC IS
    PORT(   clock_25Mhz, red, green, blue       : IN    STD_LOGIC;
            red_out, green_out, blue_out, horiz_sync_out, vert_sync_out : OUT   STD_LOGIC;
            pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END VGA_SYNC;

ARCHITECTURE a OF VGA_SYNC IS
    SIGNAL horiz_sync, vert_sync : STD_LOGIC;
    SIGNAL video_on, video_on_v, video_on_h : STD_LOGIC;
    SIGNAL h_count, v_count :STD_LOGIC_VECTOR(9 DOWNTO 0);

BEGIN

-- video_on is high only when RGB data is displayed
video_on <= video_on_H AND video_on_V;



PROCESS
BEGIN
    WAIT UNTIL(clock_25Mhz'EVENT) AND (clock_25Mhz='1');

--Generate Horizontal and Vertical Timing Signals for Video Signal
-- H_count counts pixels (640 + extra time for sync signals)
--
--  Horiz_sync  ------------------------------------_________--------
--  H_count       0                640             659       755    799
--
    IF (h_count = 799) THEN
        h_count <= "0000000000";
    ELSE
        h_count <= h_count + 1;
    END IF;

--Generate Horizontal Sync Signal using H_count
    IF (h_count <= 755) AND (h_count >= 659) THEN
        horiz_sync <= '0';
    ELSE
        horiz_sync <= '1';
    END IF;
```


VGA Interface - VGA_Sync Component

```
--V_count counts rows of pixels (480 + extra time for sync signals)
--
--  Vert_sync      ----------------------------------------------_______------------
--  V_count         0                                           480    493-494          524
--
    IF (v_count >= 524) AND (h_count >= 699) THEN
        v_count <= "0000000000";
    ELSIF (h_count = 699) THEN
        v_count <= v_count + 1;
    END IF;

-- Generate Vertical Sync Signal using V_count
    IF (v_count <= 494) AND (v_count >= 493) THEN
        vert_sync <= '0';
    ELSE
        vert_sync <= '1';
    END IF;

-- Generate Video on Screen Signals for Pixel Data
    IF (h_count <= 639) THEN
        video_on_h <= '1';
        pixel_column <= h_count;
    ELSE
        video_on_h <= '0';
    END IF;

    IF (v_count <= 479) THEN
        video_on_v <= '1';
        pixel_row <= v_count;
    ELSE
        video_on_v <= '0';
    END IF;

-- Put all video signals through DFFs to elminate any delays that cause a blurry image
        red_out <= red AND video_on;
        green_out <= green AND video_on;
        blue_out <= blue AND video_on;
        horiz_sync_out <= horiz_sync;
        vert_sync_out <= vert_sync;

END PROCESS;
END a;
```


VGA Interface - Example

Try this simple example and see how you can change the background colour on your VGA screen by using three switches on the DE0 board:



Graphics Display on VGA Screen

- Red, Green, and Blue values for all the pixels on the screen should be generated.
- Each pixel is identified by its row and column values.
- These values are generated by VGA_Sync component.
- Horizontal and vertical sync signals are generated by VGA_Sync component.
  - 25MHz clock signal is required.

That means we can draw any object on the screen if we know the RGB values and pixel information of the object.

Draw a 4x4 blue square on the top right of the screen

For pixels within the **0<= row <=3 and 636<= column <=639**, the **Blue** signal should be driven to **'1'**.

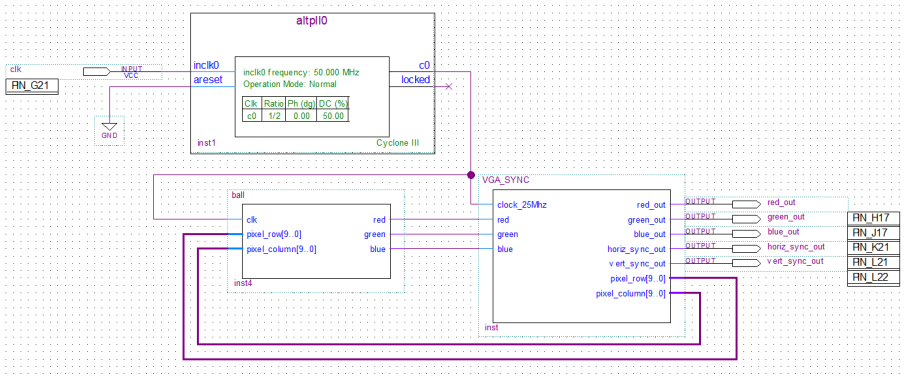
Graphics Display - Ball Example

- (x,y) position of the square are set to some constant values.
- Background colour and ball colour are defined as white and red respectively.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY ball IS
    PORT
        (SIGNAL clk                         : IN std_logic;
        SIGNAL pixel_row, pixel_column      : IN std_logic_vector(9 DOWNTO 0);
        SIGNAL red, green, blue             : OUT std_logic);
END ball;

architecture behavior of ball is

SIGNAL ball_on                  : std_logic;
SIGNAL size                     : std_logic_vector(9 DOWNTO 0);
SIGNAL ball_y_pos, ball_x_pos   : std_logic_vector(9 DOWNTO 0);

BEGIN

size <= CONV_STD_LOGIC_VECTOR(8,10);
-- ball_x_pos and ball_y_pos show the (x,y) for the centre of ball
ball_x_pos <= CONV_STD_LOGIC_VECTOR(590,10);
ball_y_pos <= CONV_STD_LOGIC_VECTOR(350,10);


ball_on <= '1' when ( ('0' & ball_x_pos <= pixel_column + size) and ('0' & pixel_column <= ball_x_pos + size)
                    and ('0' & ball_y_pos <= pixel_row + size) and ('0' & pixel_row <= ball_y_pos + size) )  else
           '0';


-- Colours for pixel data on video signal
-- Keeping background white and square in red
Red <=  '1';
-- Turn off Green and Blue when displaying square
Green <= not ball_on;
Blue <=  not ball_on;


END behavior;
```


Graphics Display - Ball Example

Try this example to see the red square on white background. You may change the colour and position of the square in ball component.



Graphics Display - Bouncy Ball Example

The motion feature is added to our simple object to make it bounce off the edges.

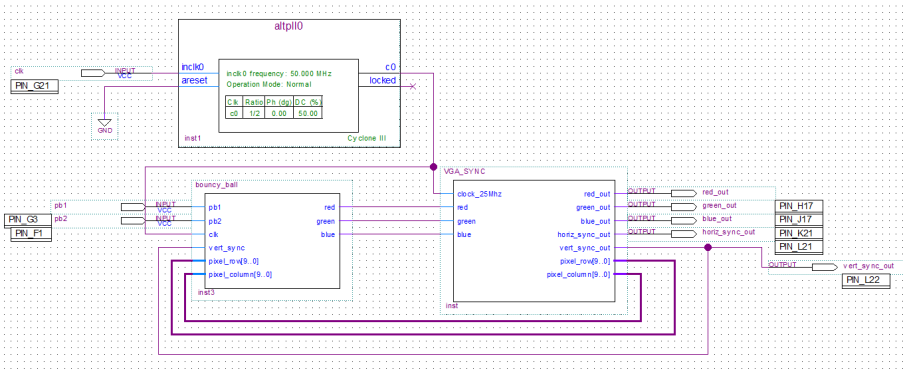
- The new position of the ball should be updated once for each frame.
  - One update per each vertical sync.
- Ball position is calculated by adding its current Y position and its vertical motion.
- Screen boundaries are checked; ball speed is changed once it reaches the boundaries at row 0 and 479.
- Two pushbuttons are used to change the background and ball colour.

Graphics Display - Bouncy Ball Example

```
ENTITY bouncy_ball IS
    PORT
        (SIGNAL pb1, pb2, clk, vert_sync    : IN std_logic;
        SIGNAL pixel_row, pixel_column      : IN std_logic_vector(9 DOWNTO 0);
        SIGNAL red, green, blue             : OUT std_logic);
END bouncy_ball;

architecture behavior of bouncy_ball is

SIGNAL ball_on                  : std_logic;
SIGNAL size                     : std_logic_vector(9 DOWNTO 0);
SIGNAL ball_y_pos               : std_logic_vector(9 DOWNTO 0);
SIGNAL ball_x_pos               : std_logic_vector(10 DOWNTO 0);
SIGNAL ball_y_motion            : std_logic_vector(9 DOWNTO 0);

BEGIN

size <= CONV_STD_LOGIC_VECTOR(8,10);
-- ball_x_pos and ball_y_pos show the (x,y) for the centre of ball
ball_x_pos <= CONV_STD_LOGIC_VECTOR(590,11);

ball_on <= '1' when ( ('0' & ball_x_pos <= '0' & pixel_column + size) and ('0' & pixel_column <= '0' & ball_x_pos + size)
                    and ('0' & ball_y_pos <= pixel_row + size) and ('0' & pixel_row <= ball_y_pos + size) )  else
           '0';


-- Colours for pixel data on video signal
-- Changing the background and ball colour by pushbuttons
Red <=  pb1;
Green <= (not pb2) and (not ball_on);
Blue <=  not ball_on;


Move_Ball: process (vert_sync)
begin
    -- Move ball once every vertical sync
    if (rising_edge(vert_sync)) then
        -- Bounce off top or bottom of the screen
        if ( ('0' & ball_y_pos >= CONV_STD_LOGIC_VECTOR(479,10) - size) ) then
            ball_y_motion <= - CONV_STD_LOGIC_VECTOR(2,10);
        elsif (ball_y_pos <= size) then
            ball_y_motion <= CONV_STD_LOGIC_VECTOR(2,10);
        end if;
        -- Compute next ball Y position
        ball_y_pos <= ball_y_pos + ball_y_motion;
    end if;
end process Move_Ball;

END behavior;
```

Graphics Display - Bouncy Ball Example

Try this example and see how you can change the colour of background and bouncy ball by using **pushutton 1** and **pushbutton 2** on DE0 board:



Graphics Display - Bouncy Ball Example

- When **no button** is pressed:
  - Pixels showing the ball has R='1', G='0', B='0': **Red ball**
  - Background pixels has R='1', G='0', B='1': **Magenta background**
- When only **pushbutton 1** is pressed:
  - Pixels showing the ball has R='0', G='0', B='0': **Black ball**
  - Background pixels has R='0', G='0', B='1': **Blue background**
- When only **pushbutton 2** is pressed:
  - Pixels showing the ball has R='1', G='0', B='0': **Red ball**
  - Background pixels has R='1', G='1', B='1': White background
- When both **pushbutton 1 and 2** are pressed:
  - Pixels showing the ball has R='0', G='0', B='0': **Black ball**
  - Background pixels has R='0', G='1', B='1': **Cyan background**

Text Display

If we want to put a text on the screen, we need to know the pattern of characters.

- Based on the character pattern, pixel row, and column information, we decide on RGB values to be sent to the VGA_Sync component.
- The following lines of code can put **H** on the screen:

```
if (((8<row<18) and (col = 8)) or ((8<row<18) and (col = 13))
  or ((row=13) and (8<col<13))) then
    red <= '1';
else
    red <= '0';
end if;
```

We can store the display pattern of characters in a memory and access the memory for writing text on the screen.

Text Display

A group of characters are stored in a memory block in the FPGA.

- This memory is instantiated in the ***char_rom.vhd***
- The memory should be initialized with the information of character patterns.
  - A *.mif file is used to initialize the memory.
  - TCGROM.mif is the memory initialization file that contains the patterns of 64 characters.
  - Each character in a .mif file is described through 8 lines of memory address and is translated to a block of 8x8 pixels.

Address	Font Data
000001000 :	00011000 ;
000001001 :	00111100 ;
000001010 :	01100110 ;
000001011 :	01111110 ;
000001100 :	01100110 ;
000001101 :	01100110 ;
000001110 :	01100110 ;
000001111 :	00000000 ;

8 x 8 Font Pixel Data

0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0

Text Display

char_rom.vhd gets an instance of ***altsyncram*** component which is a memory IP core.

```
ENTITY char_rom IS
    PORT
    (
        character_address   :   IN STD_LOGIC_VECTOR (5 DOWNTO 0);
        font_row, font_col  :   IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        clock               :   IN STD_LOGIC ;
        rom_mux_output      :   OUT STD_LOGIC
    );
END char_rom;

ARCHITECTURE SYN OF char_rom IS

    SIGNAL rom_data     : STD_LOGIC_VECTOR (7 DOWNTO 0);
    SIGNAL rom_address  : STD_LOGIC_VECTOR (8 DOWNTO 0);

    COMPONENT altsyncram
    GENERIC (
        address_aclr_a          : STRING;
        clock_enable_input_a    : STRING;
        clock_enable_output_a   : STRING;
        init_file               : STRING;
        intended_device_family  : STRING;
        lpm_hint                : STRING;
        lpm_type                : STRING;
        numwords_a              : NATURAL;
        operation_mode          : STRING;
        outdata_aclr_a          : STRING;
        outdata_reg_a           : STRING;
        widthad_a               : NATURAL;
        width_a                 : NATURAL;
        width_byteena_a         : NATURAL
    );
    PORT (
        clock0      : IN STD_LOGIC ;
        address_a   : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
        q_a         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
    END COMPONENT;
```


Text Display

We only need to provide ***rom_address*** and extract one bit of ***rom_data*** as an output for each pixel.

```
BEGIN

    altsyncram_component : altsyncram
    GENERIC MAP (
        address_aclr_a => "NONE",
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "tcgrom.mif",
        intended_device_family => "Cyclone III",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 512,
        operation_mode => "ROM",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "UNREGISTERED",
        widthad_a => 9,
        width_a => 8,
        width_byteena_a => 1
    )
    PORT MAP (
        clock0 => clock,
        address_a => rom_address,
        q_a => rom_data
    );

    rom_address <= character_address & font_row;
    rom_mux_output <= rom_data (CONV_INTEGER(NOT font_col(2 DOWNTO 0)));

END SYN;
```


Text Display

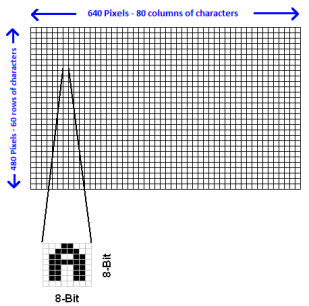
The following table shows the contents of the CharROM which is initialized through TCGROM.mif file.

- Memory depth is **512**.
- Memory width is **8**. The content of memory for each address is an 8-bit value.
- Notice that the address is in **Oct** format.

CHAR	ADDRESS	CHAR	ADDRESS	CHAR	ADDRESS	CHAR	ADDRESS
@	00	P	20	Space	40	0	60
A	01	Q	21	!	41	1	61
B	02	R	22	"	42	2	62
C	03	S	23	#	43	3	63
D	04	T	24	$	44	4	64
E	05	U	25	%	45	5	65
F	06	V	26	&	46	6	66
G	07	W	27	'	47	7	67
H	10	X	30	(	50	8	70
I	11	Y	31	)	51	9	71
J	12	Z	32	*	52	A	72
K	13	[	33	+	53	B	73
L	14	Dn Arrow	34	,	54	C	74
M	15	]	35	-	55	D	75
N	16	Up Arrow	36	.	56	E	76
O	17	Lft Arrow	37	/	57	F	77

Text Display

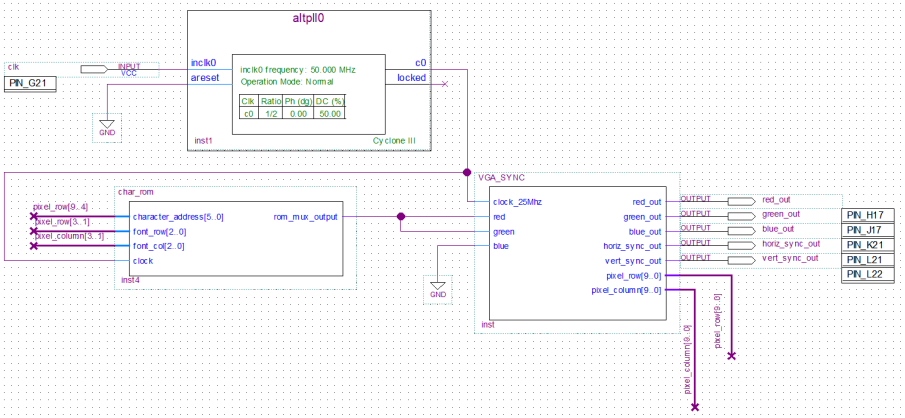
- The way we use part of pixel-row and pixel-column value as the address to the CharROM defines the size of the text.
  - If we use 3 lower bits of the pixel-row address, we will get the text in its original size of 8x8.
- To make characters larger, each dot in the font should map to several pixels.
  - To double the size, each dot should map to a 2x2 pixel block.
  - pixel-row[3 downto 1] and pixel-column[3 downto 1] are used as the font row and font column.



Character Address	Font Row
0	000
0	001
0	010
0	011
0	100
0	101
0	110
0	111
1	000
1	001
1	010
1	011
1	100
1	101
1	110
1	111
2	000
2	001
2	010
2	011
2	100
2	101
2	110
2	111

Text Display Example

Try this example and see how you can fill the screen with rows of different characters:



Summary

- We discussed about mini project and its objective.
- We talked about DE0 board and its interfaces.
- We looked at VGA interface and discussed how to show graphics and text on the VGA screen through several examples.

Acknowledgment

- Some figures/notes are taken from or inspired by the
  - CS305 Lecture notes by Muhammad Nadeem, 2019