EEE 102 - Introduction to Digital Circuit Design

Object Tracking by FPGA

Final Report

Ege Aydın

Abstract

In this project, an object tracker is designed using an entry-level FPGA Basys3. By using background subtraction algorithm, the position of the object that is moved is be determined. Then that position data is mapped to servo position. Using that, a laser is directed into the moving object. For this project, that object is determined to be a human or an equally sized object. By making use of the FPGA's parallelization characteristics, high speed object tracking device is constructed. It's seen that the only practical speed limitation is caused by camera's speed. For this project, that speed is limited to 30 actions per second due to the camera's maximum speed of 30 fps.

Introduction

Object tracking is one of the concepts that has many areas of application whether it is robotics or automation or defense industry. The way of tracking may change depending on the physical constraints of the application. For example, long-range surface to air missiles use radar-based tracking while in short-range applications such as in self-driving cars generally optical tracking is used. For both of these examples, speed is a key factor. If the aim is a little off in a missile defense system or if a self-driving car couldn't recognize a car that suddenly appeared the results would be fatal. As a solution, first it is necessary to consider what types of methods could be used for object tracking. One method would be using machine learning for an object than extract the position of that object from the frame using the trained model. Even though this method would almost always properly be able to detect the object without much external limitations, it would be extremely math-heavy and would require expensive hardware to reach the desired operating speed. Such expensive hardware would also mean there is a large power consumption of such a system. Another method would be an algorithm called background subtraction. This algorithm simply finds the Euclidean distance of every pixel between the background image and the data from the camera. This means that camera position cannot be changed or this algorithm won't work. Even though calculations are simple, using a microcontroller may not give the desired speed since even every basic instruction needs approximately 4 clock cycles. However, an FPGA will make the process much faster since it can do all the pixel subtraction parallelly. Moreover, using an entry-level FPGA with background subtraction algorithm would make it cheaper and more power efficient than using other AI based methods. Project video at https://youtu.be/_-Oj5pEGLbw.

Methodology

While designing the system, first the main physical constraints were determined. These are as stated in the project proposal:

> The tracked person will be around 4 meters away from the turret and can go sideways for around 1m, giving a total horizontal tracking angle of around 28°. The vertical tracking angle can be less since only moving humans will be detected. The precision of the aim will be around ±15cm, which is adequate for tracking a human torso. The lighting constraints will be daylight or equivalent lighting.

Then there are secondary constraints due to the limitations of BASYS3. First and foremost, is the memory limitation. Assuming we use grayscale image which has 8-bits of luminance value per pixel, for keeping even a VGA sized image, we would require a total of $640 \times 480 \times 8 = 2.5$ Mbit. Since we need to keep 2 frames simultaneously (one frame keeping background for the algorithm) we actually need double that value which is near 5Mbit. Since Basys3 has only 1899Kbit block memory, using all the data is not possible. This problem is solved by saving only one of two horizontal lines we get from the camera (no need for extra precision in vertical tracking) and decreasing pixel bit-size to 3. This would require $640 \times 240 \times 3 \times 2 = 921Kbit$ which is well within the maximum BRAM capacity of Basys3.

Proposed system consists of 4 parts:

1) VGA
2) OV7670 Camera
3) Pan&Tilt system and a laser for targeting
4) Basys3 FPGA

1. VGA

VGA output has 5 main parts:

- VSYNC
- HSYNC
- Clock
- RGB data

VSYNC and HSYNC are created in a relative way to the clock. HSYNC represents end of the line and VSYNC represent the end of the frame. Their timings are set by universal standards. For this project, standard 60fps 640x480 output is used. The timing values can be found at http://tinyvga.com/vga-timing/640x480@60Hz. Greyscale data inputted and RGB equivalent is outputted from the VGA module. That RGB data has the property of $R = G = B = data$. Since only 240 lines are captured from the camera, every line is drawn twice. VGA module also outputs the x and y coordinates of the current pixel that is displayed and whether that pixel is displayed for the first time. These are later used in other modules.

2. Camera

The OV7670 camera has 18 pins in total:

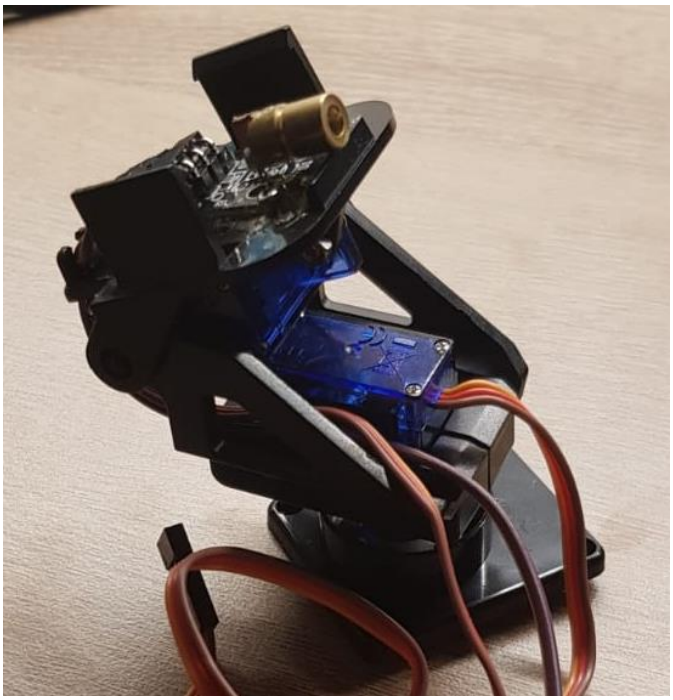
- Vdd(3v3)
- GND
- SIOC/SIOD pins
- VSYNC
- HREF
- PCLK
- XCLK
- 8 data pins
- RESET
- PWDN (active low)

SIOC and SIOD pins are used for I^2C communication with the camera for setting it up. The module that is responsible for setting up the camera has been retrieved from an GitHub repository at https://github.com/Tom-Zheng/OV7670_to_VGA_FPGA. However, the data that is send to the camera is replaced completely. It is modified for camera to output YUV with a constant gain. Since the Y value of the YUV data is the luminance value, it is used as grayscale data. VSYNC, HREF and PCLK is almost the same signals that are used for driving VGA. PCLK is the pixel clock and VSYNC is the same as the VSYNC in VGA. HREF is similar to HSYNC but it's high only when there is a pixel data transmitted not in the beginning of a new line. XCLK is an input of the camera and it determines the frequency of

the PCLK. XCLK is 24MHz which is the maximum input frequency of OV7670. RESET and PWDN pins are tied to locked signal of the clock generator to start the camera after the clock stabilizes.

3. Servo motor modules

Since there is a need to aim the laser at a moving object, a system that can move the laser's direction is required. This system consists of two servo motors which are physically configured in a pan&tilt position. Over the tilt servo, a laser module is present. Which requires 5V input however, since it is too bright at 5V, 3.3V from Basys3 is fed into it.



Servos are called SG90 and it is 180 degrees rotatable cheap motors. According to a dubious source, they have a 10µs dead bandwidth, which results in a precision of 1.8°. This precision decreases with lower voltages like 3.3V therefore 6V is fed to the motors from the voltage supply. Driving servos are quite simple: A PWM signal is fed to the signal pin of the servo. That PWM signal has a 20ms period and has a pulse width of between 1ms and 2ms. Pulse width of 1ms turns the servos all the way left and 2ms pulse width turns them all the way right. However, those are not exact values and vary for each servo motor. Therefore, if there is a need to use exact degrees as position data, modules must be calibrated using minimum and the maximum values of the servos before using. However, in this project every value is relative to each other as explained later therefore there is no need for servo calibration.

4. Main computation modules

Basys3 retrieves data from the camera and saves it into its block memory. This part of the memory will be called frame buffer. Activated by a switch, the data from the camera is also saved into another part of the block memory. This part of the memory will be called initial frame.

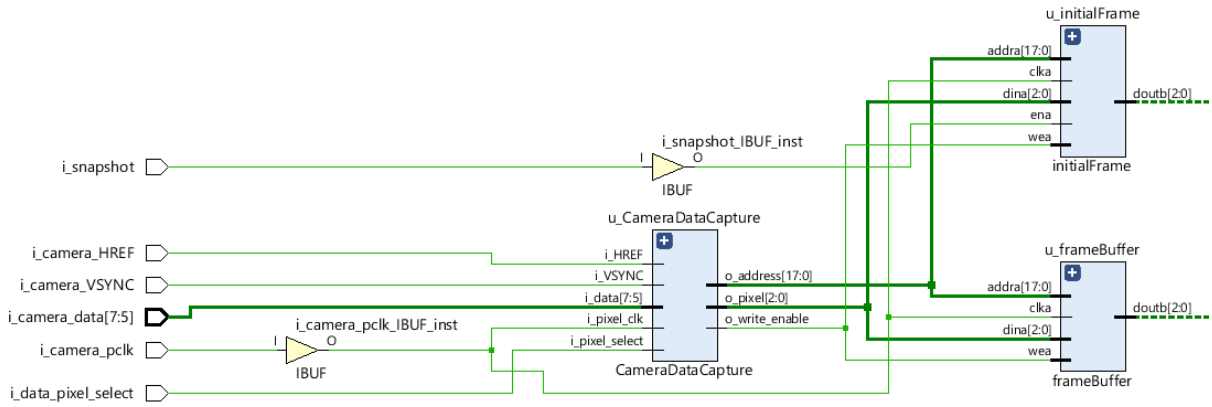


Figure 1. Data from camera is distributed to initial frame and frame buffer (Some pins are removed for a clearer representation)

The BRAM modules are in simple dual port mode, meaning that it is always updated from the camera while the data inside is read. The reason for that is the outputted clock frequency from the camera is 24MHz while VGA frequency is 25MHz. The read address is changed when there is a need to output a pixel to VGA. The data outputs are subtracted then their absolute values are taken as the pixel data.

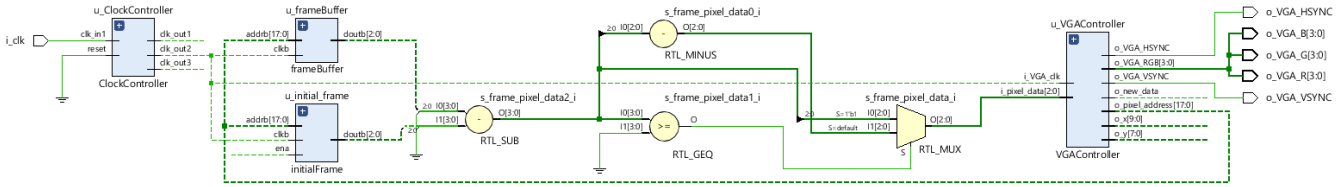


Figure 2. Data flow from memory to VGA (Some pins are removed for a clearer representation)

The next module is the module that is responsible from finding the middle point of all the pixels that have been changed. To find that, it simply calculates the arithmetic average of all the position vectors of changed pixels. However, since this requires all frame to be scanned, this calculation happens only once in a frame. Also, in order to show the location on the screen, the output is connected to VGA controller. VGA controller simply makes all the pixels in the line and the row whose index is the same as the middle point coordinates x and y white.

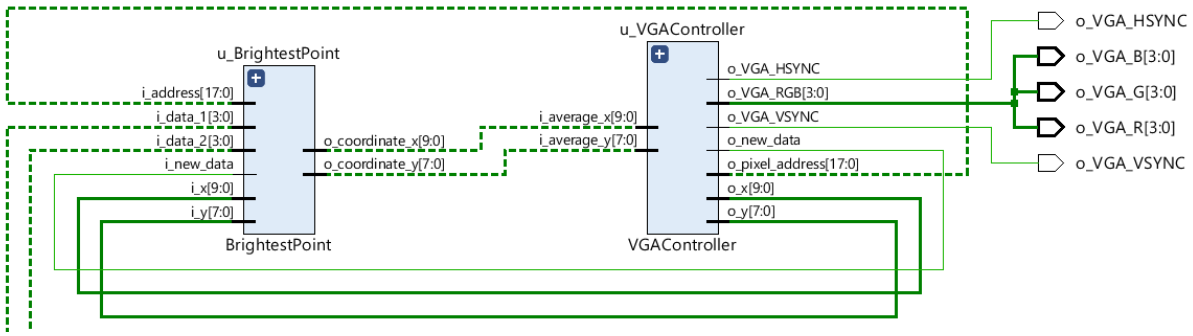


Figure 3. Calculating the position the middle point of every pixel that is changed (Some pins are removed for a clearer representation)

After finding the average point's pixel coordinates, it's necessary to convert it into real life coordinates such that it's sent into servo motors. To do this, a simple linear mapping formula is used:

$$x_{mapped} = (x - in_{min}) * (out_{max} - out_{min}) / (in_{max} - in_{min}) + out_{min}$$

Here, in_{min} and in_{max} are corresponding to 0 and 640 for x; 0 and 240 for y. However, output limits are more complicated. In order to obtain output limits, it's necessary to get servo position data when the laser is pointing at the edges of the camera's sight. This requires a pre-calibration before the actual system operation. After the limits of the servo are determined, they are passed to mapping module and the system is ready for object tracking. This is the reason why there is no need for extra servo calibration, every value given to the servos are relative to out_{max} and out_{min} values and their corresponding physical positions.

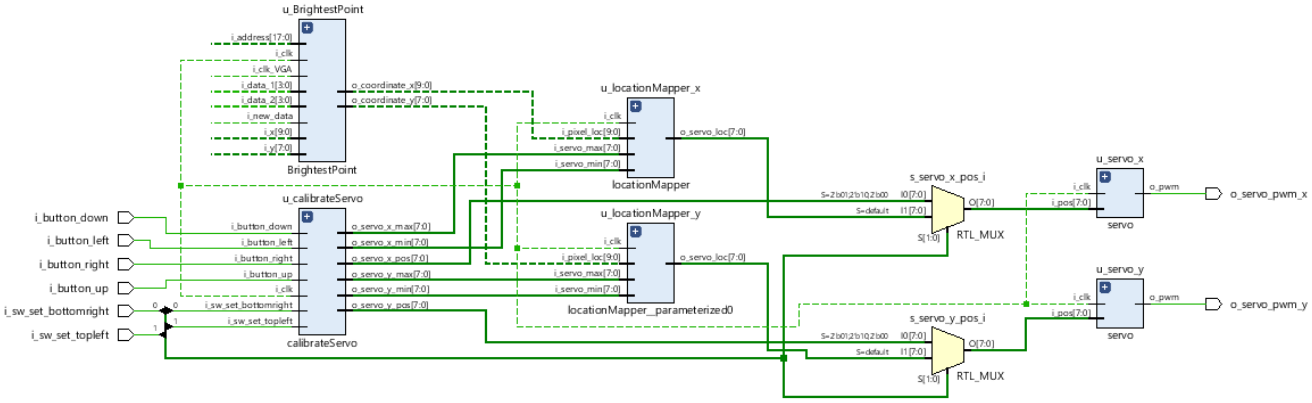


Figure 4. Calibration and screen coordinates to servo coordinates mapping (Some pins are removed for a clearer representation)

As can be seen in figure 4, the system's calibration state is determined by two switches. When one of them is zero, the corresponding min/max values are changing and servo position is determined by calibration module. The servo positions are changed with four buttons on Basys3 board then the values are set with switches. When both of the calibration switches are on, servos are driven directly from location mapper module which means system exits calibration mode and enters tracking mode.

Borrowed code and libraries:

- Xilinx Clocking Wizard IP
  This IP is used for creating necessary clock frequencies for the system.
- Xilinx Block Memory Generator IP
  This IP is used for creating frame buffer and initial frame memory blocks.
- Xilinx Divider Generator IP
  This IP is used for high-speed division.
- Xilinx Multiplier IP
  This IP is used for high-speed multiplication.
- Xilinx ODDR IP
  This IP is used for telling Vivado to use clocking resources while outputting XCLK to camera.
- Tom Zheng's OV7670 I^2C code. Retrieved from https://github.com/Tom-Zheng/OV7670_to_VGA_FPGA . *(I2C_OV7670_RGB565_Config2, I2C_Controller2 and ov7670_init modules)*
  This code is used to initialize desired camera settings. Those settings are changed to completely different values for it to satisfy project's conditions. The code isn't written from the scratch due to the possibility of output conflict on inout pin SIOD. Therefore, it's determined to be safer to modify a sure-to-work code.

Results

The system is tested with stationary and moving targets with different speeds. For each case scenario, it's seen that the moving object is determined and aim on the screen successfully. However, due to used servos' imprecision and low resolution, there are dead zones where laser do not aim at. However, this is not a problem because the target was determined to be a human and the dead zones are less than 5cm at near 4m distance. This is well within our constraints of ±15cm. The camera has a viewing angle of 25 degrees which is a little less than desired 28 degrees however, extra few cm from that 3 degrees are determined as inconsequential.

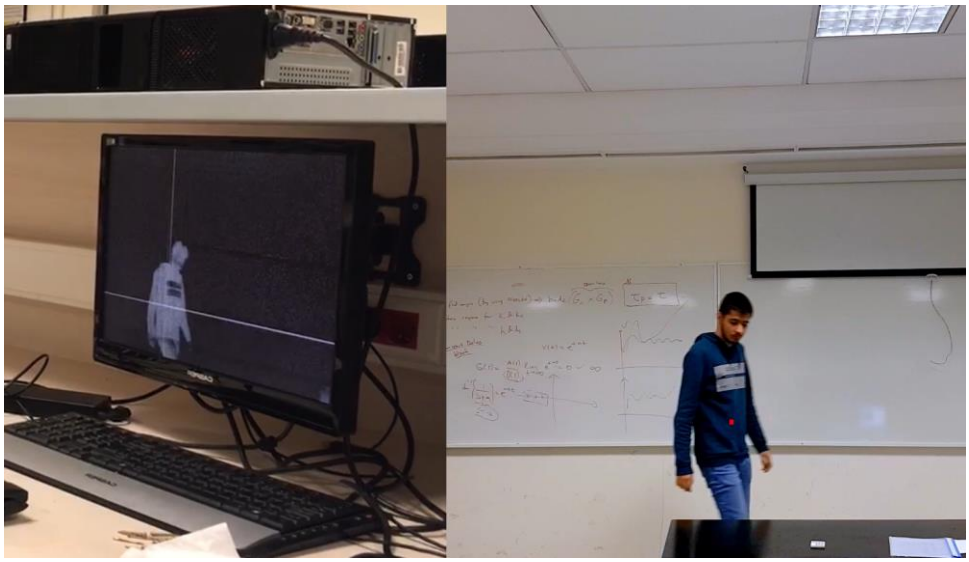


Figure 5. Result of the system (Red dot is enlarged by editing)

For moving object test, the systems ability to response sudden movements and constant speed such as walking and running is tested. For walking, the target is aimed at successfully. For running, target is hit by laser multiple times however since servos couldn't react instantly, it wasn't a continues aim.

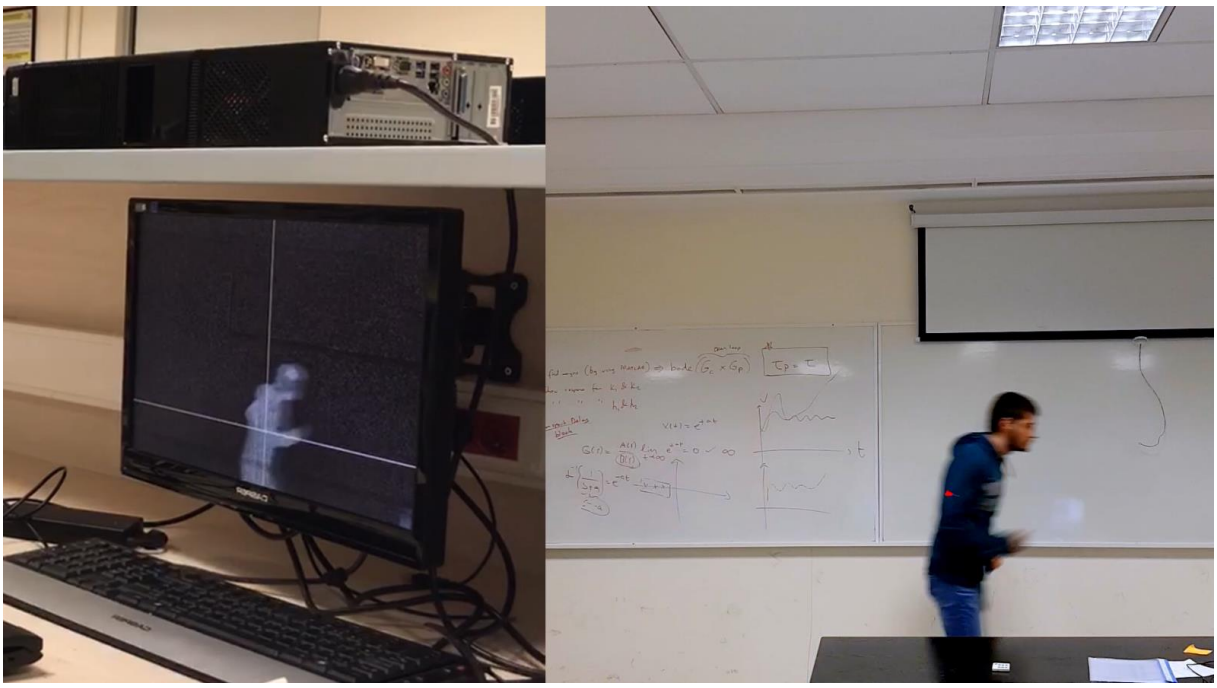


Figure 6. Servos are lagging behind in high speed movement even though VGA calculates the position correctly. (Red dot is enlarged by editing)

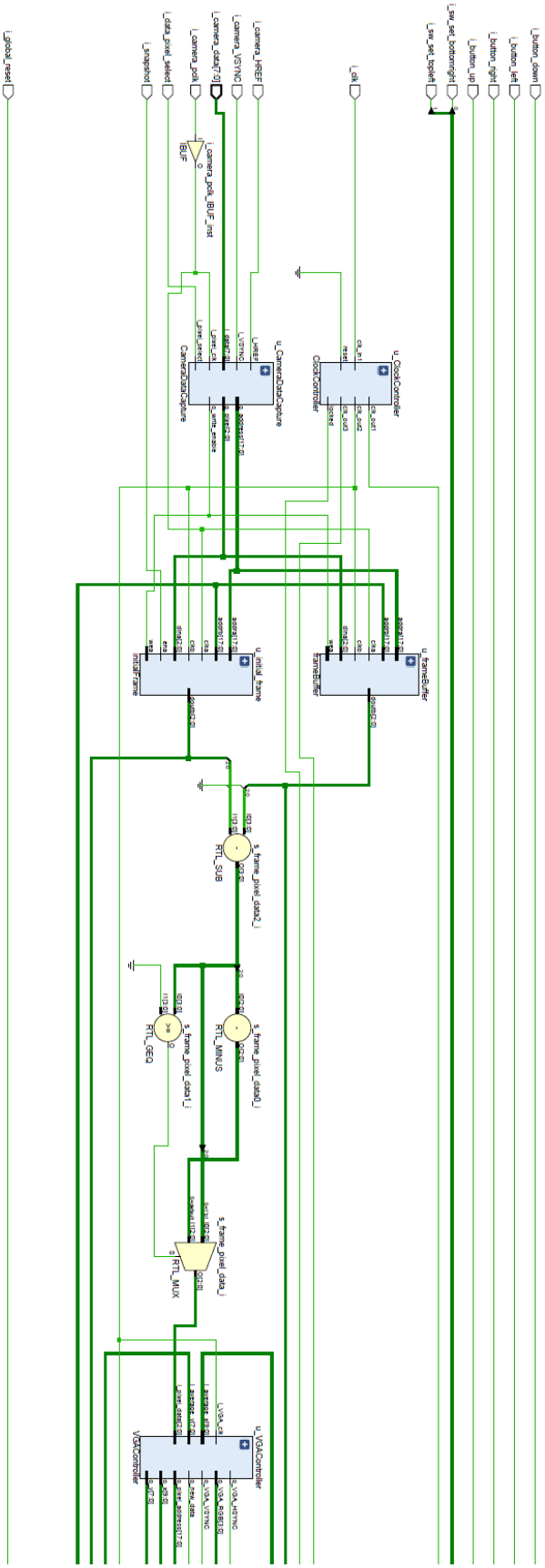
Discussion

It's seen that proposed system works perfectly at 30fps. However, this design does not run at its maximum speed. The speed is limited by primarily camera speed and VGA speed. For the camera, even if one that can output more than 30fps is used, it still wouldn't fully make use of parallelization of FPGA if the data is outputted from camera pixel by pixel. Also, higher resolutions are impossible to use with Basys3 due to memory constraints. The low viewing angle of the camera also constraints the trackable area. The most problematic part would be servo motors. For higher range(20-40m) the servo needs to have a less than 0.1° resolution. Such servos are only used for industrial purposes and very expensive.

For future work, VGA module may be removed or designed to work independently to not create a bottleneck at 6ofps. It's required to use higher quality camera with higher speed, wider viewing angle and higher quality for increased performance. To hold such amount of data, a better FPGA with more memory should be chosen. Instead of servo motors, stepper motors with microstepping capability should be used to get such high resolution. Also, for changing light conditions, an extra algorithm for updating the background frame is necessary. For the camera noise, it may be better to implement a Gaussian filter. Hardcoding calibration data may also be useful for a static camera and servo design.

Appendices

Appendix A. Full RTL Schematic



u_VGAController
VGAController
u_BrightestPoint
BrightestPoint
u_calibrateServo
calibrateServo
u_locationMapper_x
locationMapper
u_locationMapper_y
locationMapper_parameterized0
u_DebounceReset
DebounceReset
u_CameraController
ov7670_init
u_servo_x
u_servo_y
servo
u_oddr
RTL_MUX
RTL_INV
OBUF
o_VGA_VSYNC
o_VGA_R[3:0]
o_VGA_G[3:0]
o_VGA_B[3:0]
o_VGA_HSYNC
o_servo_pwm_y
o_servo_pwm_x
o_camera_pwdn
o_camera_reset
o_camera_xCLK
o_camera_SIO_C
io_camera_SIO_D

Appendix B. Original VHDL code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity BrightestPoint is
    Port (
           i_clk : in STD_LOGIC;
           i_clk_VGA : in STD_LOGIC;
           i_address : in STD_LOGIC_VECTOR (17 downto 0);
           i_data_1 : in STD_LOGIC_VECTOR (3 downto 0);
           i_data_2 : in STD_LOGIC_VECTOR (3 downto 0);
           i_new_data : in STD_LOGIC;
           i_x : in STD_LOGIC_VECTOR (9 downto 0);
           i_y : in STD_LOGIC_VECTOR (7 downto 0);
           o_coordinate_x : out STD_LOGIC_VECTOR (9 downto 0);
           o_coordinate_y : out STD_LOGIC_VECTOR (7 downto 0)
           );
end BrightestPoint;

architecture Behavioral of BrightestPoint is
    COMPONENT divider
      PORT (
        aclk : IN STD_LOGIC;
        s_axis_divisor_tvalid : IN STD_LOGIC;
        s_axis_divisor_tdata : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
        s_axis_dividend_tvalid : IN STD_LOGIC;
        s_axis_dividend_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(55 DOWNTO 0)
      );
    END COMPONENT;


    signal s_count: unsigned(17 downto 0) :=  (others => '0');
    signal s_result_count: std_logic_vector(23 downto 0) :=  (others =>
'1');
    signal s_totalX : unsigned(25 downto 0) := (others => '0');
    signal s_result_totalX, s_result_totalY : std_logic_vector(31 downto 0)
:= (others => '0');
    signal s_totalY : unsigned(25 downto 0) := (others => '0');
    signal s_coordinate_x: std_logic_vector(55 downto 0);
    signal s_coordinate_y: std_logic_vector(55 downto 0);
begin
    o_coordinate_x <= s_coordinate_x(33 downto 24);
    o_coordinate_y <= s_coordinate_y(31 downto 24);

    process(i_clk_VGA, i_new_data)
    begin
        if (rising_edge(i_clk_VGA) and i_new_data = '1') then
            if i_address = x"0000"&"00" then
                if s_count > 100 then           --threshold
                    s_result_totalY <= "000000" &
std_logic_vector(s_totalY);
                    s_result_totalX <= "000000" &
std_logic_vector(s_totalX);
                    s_result_count <= "000000" & std_logic_vector(s_count);
                end if;
                s_totalY <= (others => '0');
                s_totalX <= (others => '0');
                s_count <= (others => '0');
```

```
            else
                if abs(signed(i_data_1) - signed(i_data_2)) > 1 then --
threshold
                    s_totalX <= s_totalX + unsigned(i_x);
                    s_totalY <= s_totalY + unsigned(i_y);
                    s_count <= s_count + 1;
                end if;
            end if;

        end if;
    end process;

    u_divider_x : divider
    PORT MAP (
        aclk => i_clk,
        s_axis_divisor_tvalid => '1',
        s_axis_divisor_tdata => s_result_count,
        s_axis_dividend_tvalid => '1',
        s_axis_dividend_tdata => s_result_totalX,
        m_axis_dout_tdata => s_coordinate_x
    );

    u_divider_y : divider
    PORT MAP (
        aclk => i_clk,
        s_axis_divisor_tvalid => '1',
        s_axis_divisor_tdata => s_result_count,
        s_axis_dividend_tvalid => '1',
        s_axis_dividend_tdata => s_result_totalY,
        m_axis_dout_tdata => s_coordinate_y
    );

end Behavioral;----------------------------------------------------------------
----------------------
-- Company:
-- Engineer:
--
-- Create Date: 12/21/2019 01:26:51 PM
-- Design Name:
-- Module Name: calibrateServo - RTL
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity calibrateServo is
    Port ( i_clk : in STD_LOGIC;
           i_button_left : in STD_LOGIC;
           i_button_right : in STD_LOGIC;
           i_button_up : in STD_LOGIC;
           i_button_down : in STD_LOGIC;
           i_sw_set_topleft : in STD_LOGIC;
           i_sw_set_bottomright : in STD_LOGIC;
           o_servo_x_min : out STD_LOGIC_VECTOR (7 downto 0);
           o_servo_x_max : out STD_LOGIC_VECTOR (7 downto 0);
           o_servo_y_min : out STD_LOGIC_VECTOR (7 downto 0);
           o_servo_y_max : out STD_LOGIC_VECTOR (7 downto 0);
           o_servo_x_pos: out STD_LOGIC_VECTOR(7 downto 0);
           o_servo_y_pos : out STD_LOGIC_VECTOR(7 downto 0)
           );
end calibrateServo;

architecture RTL of calibrateServo is
    signal s_pos_x  : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal s_pos_y : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin
    o_servo_x_pos <= s_pos_x;
    o_servo_y_pos <= s_pos_y;

    process(i_clk)
    begin
        if rising_edge(i_clk) then
            if i_sw_set_topleft = '0' then
                o_servo_x_min <= s_pos_x;
                o_servo_y_min <= s_pos_y;
            end if;
            if i_sw_set_bottomright = '0' then
                o_servo_x_max <= s_pos_x;
                o_servo_y_max <= s_pos_y;
            end if;
        end if;
    end process;

    u_button: entity work.button_to_pos(RTL)
    port map(
        i_reset => '0',
        i_clk => i_clk,
        i_button_inc => i_button_left,
        i_button_dec => i_button_right,
        o_pos => s_pos_x
    );

    u_button2: entity work.button_to_pos(RTL)
    port map(
        i_reset => '0',
        i_clk => i_clk,
        i_button_inc => i_button_down,
        i_button_dec => i_button_up,
        o_pos => s_pos_y
    );
```

```
end RTL;
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 12/21/2019 12:00:10 AM
-- Design Name:
-- Module Name: locationMapper - RTL
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity locationMapper is
    Generic (
            G_width : STD_LOGIC_VECTOR(31 downto 0)
            );

    Port    (
            i_clk : in STD_LOGIC;
            i_pixel_loc : in STD_LOGIC_VECTOR (9 downto 0);
            i_servo_max : in STD_LOGIC_VECTOR (7 downto 0);
            i_servo_min : in STD_LOGIC_VECTOR (7 downto 0);
            o_servo_loc : out STD_LOGIC_VECTOR (7 downto 0)
            );
end locationMapper;

architecture RTL of locationMapper is
    COMPONENT mult_gen_0
    PORT(
        CLK : IN STD_LOGIC;
        A : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        P : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
    END COMPONENT;
    COMPONENT map_divider
    PORT (
```

```
        aclk : IN STD_LOGIC;
        s_axis_divisor_tvalid : IN STD_LOGIC;
        s_axis_divisor_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        s_axis_dividend_tvalid : IN STD_LOGIC;
        s_axis_dividend_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(63 DOWNTO 0)
      );
    END COMPONENT;

    signal s_difference : std_logic_vector(31 downto 0);
    signal s_product : std_logic_vector(31 downto 0);
    signal s_result : std_logic_vector(63 downto 0);
    signal s_servo_loc : std_logic_vector(31 downto 0);
begin
    s_difference <= std_logic_vector(signed(x"000000" & i_servo_max) -
signed(x"000000" & i_servo_min)); --signed
    o_servo_loc <= s_servo_loc(7 downto 0);
    process(i_clk)
    begin
        if rising_edge(i_clk) then
            s_servo_loc <= std_logic_vector(signed(s_result(63 downto 32))
+ signed(x"000000" & i_servo_min)); --should be unsigned
        end if;
    end process;

    u_multiplier : mult_gen_0
    PORT MAP (
        CLK => i_clk,
        A => i_pixel_loc,  --unsigned
        B => s_difference, --signed
        P => s_product --signed
    );
    u_divider : map_divider
    PORT MAP (
        aclk => i_clk,
        s_axis_divisor_tvalid => '1',
        s_axis_divisor_tdata => G_width,
        s_axis_dividend_tvalid => '1',
        s_axis_dividend_tdata => s_product,
        m_axis_dout_tdata => s_result
      );


end RTL;
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 11/26/2019 10:52:13 PM
-- Design Name:
-- Module Name: TopModule - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
```

```
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TopModule is
    Port ( i_clk : in STD_LOGIC;
           i_snapshot : in STD_LOGIC;
           i_button_left : in STD_LOGIC;
           i_button_right : in STD_LOGIC;
           i_button_up : in STD_LOGIC;
           i_button_down : in STD_LOGIC;
           i_sw_set_topleft : in STD_LOGIC;
           i_sw_set_bottomright : in STD_LOGIC;
           i_camera_VSYNC : in STD_LOGIC;
           i_camera_HREF : in STD_LOGIC;
           i_camera_data : in STD_LOGIC_VECTOR (7 downto 5);
           i_camera_pclk : in STD_LOGIC;
           i_global_reset : in STD_LOGIC;
           i_data_pixel_select : in STD_LOGIC;
           io_camera_SIO_D : inout STD_LOGIC;
           o_camera_SIO_C : out STD_LOGIC;
           o_camera_xCLK : out STD_LOGIC;
           o_camera_reset : out STD_LOGIC;
           o_camera_pwdn : out STD_LOGIC;
           o_VGA_VSYNC : out STD_LOGIC;
           o_VGA_HSYNC : out STD_LOGIC;
           o_VGA_R : out STD_LOGIC_VECTOR(3 downto 0);
           o_VGA_G : out STD_LOGIC_VECTOR(3 downto 0);
           o_VGA_B : out STD_LOGIC_VECTOR(3 downto 0);
           o_servo_pwm_x : out STD_LOGIC;
           o_servo_pwm_y : out STD_LOGIC
           );
end TopModule;

architecture RTL of TopModule is
    COMPONENT oddr_xCLK
    PORT (
        clk_in : IN STD_LOGIC;
        clk_out : OUT STD_LOGIC
    );
    END COMPONENT;
    component ClockController
    port(
      clk_out1          : out    std_logic;
      clk_out2          : out    std_logic;
      clk_out3          : out    std_logic;
      reset             : in     std_logic;
```

```
  locked            : out    std_logic;
  clk_in1           : in     std_logic
  );
end component;

COMPONENT frameBuffer
PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    clkb : IN STD_LOGIC;
    addrb : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END COMPONENT;

COMPONENT initialFrame
PORT (
    clka : IN STD_LOGIC;
    ena : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    clkb : IN STD_LOGIC;
    addrb : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END COMPONENT;

COMPONENT ov7670_init
PORT(
    iCLK : in STD_LOGIC;
    iRST_N : in STD_LOGIC;

    I2C_SCLK : out STD_LOGIC;
    I2C_SDAT : inout STD_LOGIC;
    Config_Done : out STD_LOGIC;
    I2C_RDATA : out STD_LOGIC_VECTOR(7 downto 0)
);
END COMPONENT;

-- CLOCKING SIGNALS BEGIN --
signal s_locked : std_logic;
signal s_clk_24MHz : std_logic;
signal s_clk_25MHz : std_logic;
signal s_clk_100MHz : std_logic;
-- CLOCKING SIGNALS END --

-- DATA CAPTURE SIGNALS BEGIN --
signal s_pixel_address : std_logic_vector(17 downto 0);
signal s_pixel_data : std_logic_vector(2 downto 0);
signal s_write_enable : std_logic_vector(0 downto 0);
-- DATA CAPTURE SIGNALS END --

-- VGA SIGNALS BEGIN --
signal s_frame_pixel_data : std_logic_vector(3 downto 0);
signal s_frame_pixel_data1 : std_logic_vector(3 downto 0) := x"0";
signal s_frame_pixel_data2 : std_logic_vector(3 downto 0) := x"0";
signal s_frame_pixel_address : std_logic_vector(17 downto 0);
signal s_VGA_RGB : std_logic_vector(3 downto 0);
```

```
    -- VGA SIGNALS END --

    -- DEBOUNCER SIGNALS BEGIN --
    signal s_reset : std_logic := '0';
    -- DEBOUNCER SIGNALS END --

    -- BRIGHTES POINT DETECTOR SIGNALS BEGIN --
    signal s_pixel_avg_y : std_logic_vector(7 downto 0);
    signal s_pixel_avg_x : std_logic_vector(9 downto 0);
    signal s_x : std_logic_vector(9 downto 0);
    signal s_y : std_logic_vector(7 downto 0);
    signal s_new_data : std_logic;
    -- BRIGHTES POINT DETECTOR SIGNALS END --

    -- SERVO CALIBRATION SIGNALS BEGIN --
    signal s_servo_x_min : STD_LOGIC_VECTOR (7 downto 0);
    signal s_servo_x_max : STD_LOGIC_VECTOR (7 downto 0);
    signal s_servo_y_min : STD_LOGIC_VECTOR (7 downto 0);
    signal s_servo_y_max : STD_LOGIC_VECTOR (7 downto 0);
    signal s_calibration_x_pos : STD_LOGIC_VECTOR(7 downto 0);
    signal s_calibration_y_pos : STD_LOGIC_VECTOR(7 downto 0);
    -- SERVO CALIBRATION SIGNALS END --

    -- LOCATION MAPPER SIGNALS BEGIN --
    signal s_mapped_x_pos : STD_LOGIC_VECTOR(7 downto 0);
    signal s_mapped_y_pos : STD_LOGIC_VECTOR(7 downto 0);
    -- LOCATION MAPPER SIGNALS END --

    -- SERVO DRIVER SIGNALS BEGIN --
    signal s_servo_x_pos : STD_LOGIC_VECTOR(7 downto 0);
    signal s_servo_y_pos : STD_LOGIC_VECTOR(7 downto 0);
    -- SERVO DRIVER SIGNALS END --

begin
    u_ClockController : ClockController
    port map (
        clk_out1 => s_clk_24MHz,
        clk_out2 => s_clk_25MHz,
        clk_out3 => s_clk_100MHz,
        reset => '0',
        locked => s_locked,
        clk_in1 => i_clk
    );
    o_camera_pwdn <= not s_locked;
    o_camera_reset <= s_locked;

    u_DebouncedReset : entity work.DebouncedReset(RTL)
    port map (
        i_clk => s_clk_100MHz,
        i_global_reset => i_global_reset,
        o_reset => s_reset
    );

    u_oddr : oddr_xCLK
    PORT MAP (
        clk_in => s_clk_24MHz, -- maybe increase later
        clk_out => o_camera_xCLK
    );

    u_CameraController : ov7670_init
    port map (
```

```
        iRST_N => s_reset,
        iCLK => s_clk_100MHz,
        I2C_SDAT => io_camera_SIO_D,
        I2C_SCLK => o_camera_SIO_C
    );



    u_CameraDataCapture : entity work.CameraDataCapture(RTL)
    port map (
        --i_fast_clk => s_clk_100MHz,
        i_pixel_clk => i_camera_pclk,
        i_HREF => i_camera_HREF,
        i_VSYNC => i_camera_VSYNC,
        i_data => i_camera_data,
        i_pixel_select => i_data_pixel_select,
        o_address => s_pixel_address,
        o_pixel => s_pixel_data,
        o_write_enable => s_write_enable(0)
    );

    u_frameBuffer : frameBuffer
    PORT MAP (
        clka => i_camera_pclk,
        wea => s_write_enable,
        addra => s_pixel_address,
        dina => s_pixel_data,
        clkb => s_clk_25MHz,
        addrb => s_frame_pixel_address,
        doutb => s_frame_pixel_data1(2 downto 0)
    );

    u_initial_frame : initialFrame
    PORT MAP (
        clka => i_camera_pclk,
        wea => s_write_enable,
        addra => s_pixel_address,
        dina => s_pixel_data,
        ena => i_snapshot,
        clkb => s_clk_25MHz,
        addrb => s_frame_pixel_address,
        doutb => s_frame_pixel_data2(2 downto 0)
  );
    s_frame_pixel_data <= std_logic_vector(abs(signed(s_frame_pixel_data1)
- signed(s_frame_pixel_data2)));

    u_VGAController : entity work.VGAController(RTL)
    port map(
        i_VGA_clk => s_clk_25MHz,
        i_pixel_data => s_frame_pixel_data(2 downto 0),
        i_average_x => s_pixel_avg_x,
        i_average_y => s_pixel_avg_y,
        o_pixel_address => s_frame_pixel_address,
        o_VGA_RGB => s_VGA_RGB,
        o_VGA_VSYNC => o_VGA_VSYNC,
        o_VGA_HSYNC => o_VGA_HSYNC,
        o_new_data => s_new_data,
        o_x => s_x,
        o_y => s_y
    );
```

```
o_VGA_R <= s_VGA_RGB;
o_VGA_G <= s_VGA_RGB;
o_VGA_B <= s_VGA_RGB;


u_BrightestPoint: entity work.BrightestPoint(Behavioral)
port map(
    i_clk => s_clk_100MHz,
    i_clk_VGA => s_clk_25MHz,
    i_address => s_frame_pixel_address,
    i_data_1 => s_frame_pixel_data1,
    i_data_2 => s_frame_pixel_data2,
    i_x => s_x,
    i_y => s_y,
    i_new_data => s_new_data,
    o_coordinate_x => s_pixel_avg_x,
    o_coordinate_y => s_pixel_avg_y
);

u_calibrateServo : entity work.calibrateServo(RTL)
Port map (
   i_clk => s_clk_100MHz,
   i_button_left => i_button_left,
   i_button_right => i_button_right,
   i_button_up => i_button_up,
   i_button_down => i_button_down,
   i_sw_set_topleft => i_sw_set_topleft,
   i_sw_set_bottomright => i_sw_set_bottomright,
   o_servo_x_min => s_servo_x_min,
   o_servo_x_max => s_servo_x_max,
   o_servo_y_min => s_servo_y_min,
   o_servo_y_max => s_servo_y_max,
   o_servo_x_pos => s_calibration_x_pos,
   o_servo_y_pos => s_calibration_y_pos
);

u_locationMapper_x : entity work.locationMapper(RTL)
Generic map (
    G_width => x"00000280" --640
)
Port map (
    i_clk => s_clk_100MHz,
    i_pixel_loc => s_pixel_avg_x,
    i_servo_max => s_servo_x_max,
    i_servo_min => s_servo_x_min,
    o_servo_loc => s_mapped_x_pos
);
u_locationMapper_y : entity work.locationMapper(RTL)
Generic map (
    G_width => x"000000f0" --240
)
Port map (
    i_clk => s_clk_100MHz,
    i_pixel_loc(9 downto 8) => "00",
    i_pixel_loc(7 downto 0) => s_pixel_avg_y,
    i_servo_max => s_servo_y_max,
    i_servo_min => s_servo_y_min,
    o_servo_loc => s_mapped_y_pos
);
```

```
    with std_logic_vector'(i_sw_set_topleft&i_sw_set_bottomright) select
s_servo_x_pos <=
        s_calibration_x_pos when "01"|"10"|"00",
        s_mapped_x_pos when others;

    with std_logic_vector'(i_sw_set_topleft&i_sw_set_bottomright) select
s_servo_y_pos <=
        s_calibration_y_pos when "01"|"10"|"00",
        s_mapped_y_pos when others;


    u_servo_x : entity work.servo(RTL)
    port map(
        i_clk => s_clk_100MHz,
        i_pos => s_servo_x_pos,
        o_pwm => o_servo_pwm_x
    );

    u_servo_y : entity work.servo(RTL)
    port map(
        i_clk => s_clk_100MHz,
        i_pos => s_servo_y_pos,
        o_pwm => o_servo_pwm_y
    );

end RTL;
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 11/26/2019 10:52:13 PM
-- Design Name:
-- Module Name: VGA - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity VGA is
    Port ( i_clk : in STD_LOGIC;
           o_VSYNC : out STD_LOGIC := '1';
           o_HSYNC : out STD_LOGIC := '1';
           o_active_area : out STD_LOGIC);
end VGA;

architecture RTL of VGA is
    signal s_HCounter : unsigned(9 downto 0) := (others=> '0');
    signal s_VCounter : unsigned(9 downto 0) := (others=> '0');
    signal s_VActive : STD_LOGIC := '1';
    signal s_HActive : STD_LOGIC := '1';
begin
    o_active_area<= s_VActive and s_HActive;


    process(i_clk)
    begin
        if rising_edge(i_clk) then
            s_HCounter <= s_HCounter + 1;
            s_HActive <= '0';

            if s_HCounter = 0 then
                s_VCounter <= s_Vcounter + 1;
                s_VActive <= '0';

                if s_VCounter < 480 then
                    s_VActive <= '1';
                elsif s_VCounter < 490 then
                elsif s_VCounter < 492 then
                    o_VSYNC <= '0';
                elsif s_VCounter < 525-1 then
                    o_VSYNC <= '1';
                else
                    s_VCounter <= (others => '0');
                end if;
            end if;
            if s_HCounter < 640 then --check numbers, may be wrong
                s_HActive <= '1';
            elsif s_HCounter < 656 then
            elsif s_HCounter < 752 then
                o_HSYNC <= '0';
            elsif s_HCounter < 800-1 then
                o_HSYNC <= '1';
            else
                s_HCounter <= (others=> '0');
            end if;
        end if;
    end process;
end RTL;

----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 11/30/2019 05:14:02 PM
-- Design Name:
-- Module Name: VGAController - Behavioral
-- Project Name:
-- Target Devices:
```

```
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity VGAController is
    Port ( i_VGA_clk : in STD_LOGIC;
           i_pixel_data : in STD_LOGIC_VECTOR(2 downto 0);
           i_average_x : in STD_LOGIC_VECTOR(9 downto 0);
           i_average_y : in STD_LOGIC_VECTOR(7 downto 0);
           o_pixel_address : out STD_LOGIC_VECTOR(17 downto 0); --This may
be wrong, simulate!
           o_VGA_RGB : out STD_LOGIC_VECTOR(3 downto 0);
           o_VGA_VSYNC : out STD_LOGIC;
           o_VGA_HSYNC : out STD_LOGIC;
           o_x : out STD_LOGIC_VECTOR(9 downto 0);
           o_y : out STD_LOGIC_VECTOR(7 downto 0);
           o_new_data : out std_logic);
end VGAController;

architecture RTL of VGAController is
signal s_VSYNC : STD_LOGIC := '0';
signal s_HSYNC : STD_LOGIC := '0';
signal s_active_area : STD_LOGIC := '0';
signal s_address : unsigned(17 downto 0) := (others => '1');
signal s_pixelCounter : unsigned(11 downto 0) := x"280";
signal s_RGB : STD_LOGIC_VECTOR(3 downto 0);
signal s_x : unsigned(9 downto 0) := (others => '1');
signal s_y : unsigned(7 downto 0) := (others => '0');
signal s_double_line : std_logic := '1';
begin
    u_VGA : entity work.VGA(RTL)
    port map (
        i_clk => i_VGA_clk,
        o_VSYNC => s_VSYNC,
        o_HSYNC => s_HSYNC,
        o_active_area => s_active_area
    );

    o_new_data <= s_active_area and s_double_line;
    o_VGA_VSYNC <= s_VSYNC;
```

```
    o_VGA_HSYNC <= s_HSYNC;
    s_RGB <= i_pixel_data & '0';
    o_pixel_address <= std_logic_vector(s_address);
    o_x <= std_logic_vector(s_x);
    o_y <= std_logic_vector(s_y);

    process(i_VGA_clk)
    begin
        if rising_edge(i_VGA_clk) then
            if s_active_area = '1' then
                s_address <= s_address + 1; --Might be a good idea to see
whether or not this exceeds the threshold...
                s_x <= s_x + 1;
                o_VGA_RGB <= s_RGB;
                if std_logic_vector(s_x) = i_average_x or
std_logic_vector(s_y) = i_average_y then
                    o_VGA_RGB <= (others => '1');
                end if;
                s_pixelCounter <= s_pixelCounter + 1;
                if s_pixelCounter = 1279 then
                    s_address <= s_address - 639;
                    --s_x <= (others => '1');
                    s_pixelCounter <= (others => '0');
                end if;
            else
                if s_x > 0 then
                    s_x <= (others => '0');
                    s_double_line <= not s_double_line;
                    if s_double_line = '0' then
                        s_y <= s_y + 1;
                    end if;
                end if;
                o_VGA_RGB <= (others => '0');
                if s_VSYNC = '0' then
                    s_address <= (others => '0');
                    s_x <= (others => '0');
                    s_y <= (others => '0');
                end if;
            end if;
        end if;
    end process;


end RTL;
```