# EE314 Digital Electronics Laboratory Term Project FPGA Based Point of Sale Terminal

Ataberk Öklü, Alper Soysal, Ahsen Topbaş, Göksu Uzuntürk, Erkan Doğan
*B.S. Electrical Electronics Engineering*
*Middle East Technical University*

***Abstract*—Implementing a point of sale (POS) terminals on field-programmable gate array (FPGA) is a challenging task due to the limitations in memory of FGPA. Additionally, one needs to consider the hardware while creating software when dealing with FGPA because every code block may not be meaningful or synthesized by FGPA. Therefore, the software should be reasonable in terms of speed, memory and synthesizability. Our approach to the problem takes these restrictions into a consideration and comes up with new solutions.**

***Index Terms*—FPGA, Verilog, Logic Design**

## I. Introduction

The usage of point of sale POS terminals have increased with digitalization in markets, restaurants. POS terminals are a combination of hardware and software which enables cashiers or waiters to list down the order list. In this project, we have designed a POS terminal by using a field-programmable gate array (FPGA), which is an integrated circuit with programmable logic blocks. FGPA is configured by using a hardware description language (HDL). In this project, we have used Verilog as HDL. We have designed a 800x600 resolution POS terminal screen by using VGA interface. The project consists of six main blocks, namely VGA controller, button controller, hover controller, basket controller, text controller and a state machine which combines these controllers. In this paper, our different techniques are provided in the "Approach To The Problem" part. The implementation of the project is technically explained with the main blocks. There is a "Demonstration Photos" part which shows the photos of the project. Our findings are given in the "Conclusion and Comments" part. Finally, Verilog codes are shared as a link in the "Appendix" part.

## II. Approach To The Problem

First of all, we have learned the basics of FPGA and VGA and then determined the layout of our screen. We have decided to use a 600x800 screen instead of 480x600 to be more flexible using the area. For using a 600x800 screen, a 40 MHz clock signal is required instead of a 25 MHz clock signal, so the frequency of the FPGA's clock (50 MHz) was decreased to 40 MHz using PLL. Also, we have used a 24-bit color format for our screen. We have converted our fruit pictures to .mbp files using our homemade MATLAB code. For writing texts on the screen, we have used ASCII characters instead of memorizing the pictures of words, which allows us to write anything we want on the screen. On the other hand, we have used the 7-segments to see the barcode or quantity entered and LEDs to follow which state is active.
While doing these, we have always been careful about our resource usage. We have used the minimum number of registers and logic units as much as we can. Hence, our final resource usage had become 6%, although our sales terminal "SMARKT I/O" supports twelve products instead of six, which is the minimum number determined by the project constraints.

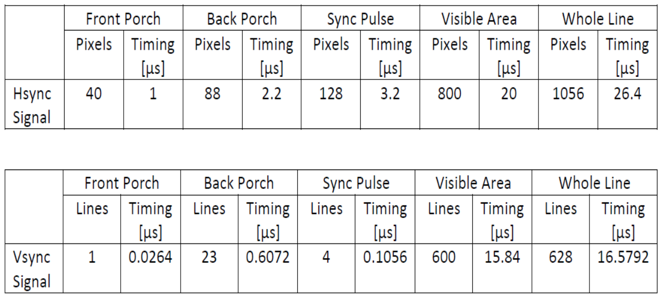
## III. VGA Controller

Video graphics array (VGA) is widely used for video transmission tasks. VGA requires three connections, namely R (Red channel), G (Green channel), B (Blue channel) and these R-G-B data is digitally provided to VGA port. In the project, we have used 800x600 resolution frame at 60 Hz for POS terminal screen by using VGA interface. 800x600 resolution means that there are 600 lines and each line consists of 800 pixels. Each frame is started to displayed from the first line and the order is going from left to right. There are two synchronization signals, namely hsync (horizontal synchronization) and vsync (vertical synchronization) which controls the construction of a frame. Hsync tells the monitor when a line is completed whereas vsync tells the monitor when a frame is completed. In hsync and vsync signals, some of pixels are used for displaying while some of which creates blanking time specified by front porch and back porch around sync pulse. We have determined these pixel values from an external source [1], as shown in Figure 1. In addition to vsync and hsync signals, we have implemented two counters (CounterX and CounterY) to understand currently available pixel coordinates.

As required, there should be twelve item images and the pixel values of these item images are kept in ROM. We have defined a product id for each item. The top left item has a product id of zero, on the left the product id becomes one, and it goes like that. By a simple math with counters and product id, we have accessed the ROM addresses of the item images and designed a POS terminal screen, as indicated in Figure 2.

## IV. Button Controller

The DE-01 SoC Development Board has multiple manual inputs, including buttons and switches, to provide an interac-



| | Front Porch | | Back Porch | | Sync Pulse | | Visible Area | | Whole Line | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Pixels | Timing [µs] | Pixels | Timing [µs] | Pixels | Timing [µs] | Pixels | Timing [µs] | Pixels | Timing [µs] |
| Hsync Signal | 40 | 1 | 88 | 2.2 | 128 | 3.2 | 800 | 20 | 1056 | 26.4 |

| | Front Porch | | Back Porch | | Sync Pulse | | Visible Area | | Whole Line | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Lines | Timing [µs] | Lines | Timing [µs] | Lines | Timing [µs] | Lines | Timing [µs] | Lines | Timing [µs] |
| Vsync Signal | 1 | 0.0264 | 23 | 0.6072 | 4 | 0.1056 | 600 | 15.84 | 628 | 16.5792 |

Fig. 1. Horizontal and Vertical Timing

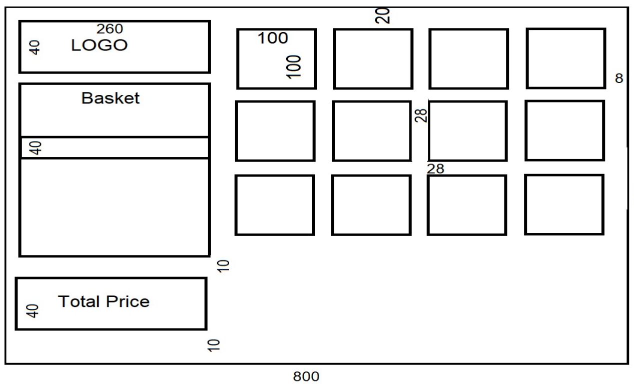


Fig. 2. POS Terminal Screen Layout

tive interface. In this project, four of four push-button inputs and three of nine switches are used. The SW1 and SW2 are the mode selection switches, while SW0 is the Button Management switch, explained in this section. This controller aims to manage debouncing and synchronization of the inputs and provide necessary state and pulse outputs used in other sub-blocks and the state machine.

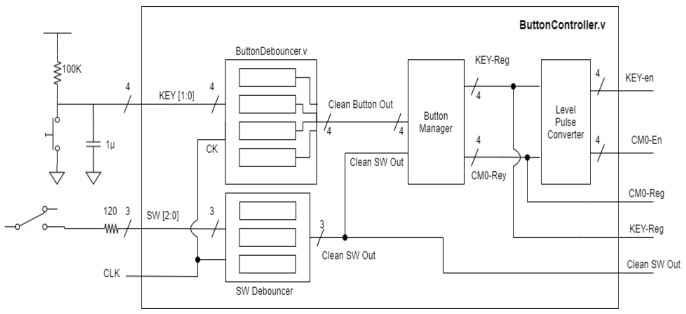


Fig. 3. Button Controller Block Diagram

## A. Board Input Schematics

To design a proper controller, first, the input characteristics should be analyzed carefully. The schematic file is provided in a *GitHub Repository*. The four push-button sections is shown below.

There are multiple essential points that should be extracted from the above schematic. The first property of the input is

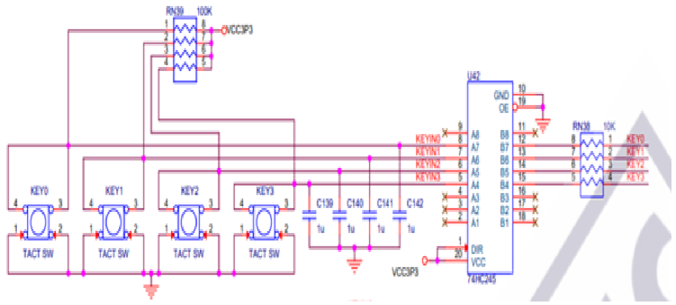


Fig. 4. The DE-01 SoC Development Board Push Button Schematic

ACTIVE LOW since a button press pulls the KEYIN# to GND. In addition, the IDLE state of the buttons is HIGH, due to 100K pull-up resistors. Another critical point is having a 1uF bypass capacitor connected to KEYIN# for each pin, preventing high-frequency content or noise from affecting the output state of the button. A simple calculation of $\tau = R*C = 100ms$ shows that there is a strong bounce filter. However, an addition of software debouncing applied to satisfy more generic design, which can be used other board not having these bypass capacitors connected, still runs correctly. 74HC245 IC is the octal-bus having tri-state buffers and direction selection for general purpose design; but, as it can be seen from the above figure, this 74HC245 is fixed direction and has no High-Z output feature, only transfers left side to right side combinationally.

Switch connections have no external capacitors, only serial resistors to limit current; therefore, these inputs are vulnerable to mechanical switch noises. Therefore, these inputs also cleaned using debounce modules.

## B. Button Manager Module

The DE-01 SoC Development Board has four user buttons to interact with; however, the project requires more than four distinct triggers for digit inputs and direction inputs besides command inputs like select, cancel, etc. Therefore, this module allows the user to switch between push button roles between informative inputs and command inputs using the SW0 switch. When SW0 is high, push-buttons are in the command mode representing select, cancel, backspace, or other state-depended actions. While SW0 is low, push-buttons are representing either digits or direction keys depending on the shopping state. The active high current status of both KEY and CMD information is kept in KEY_Reg and CMD_Reg registers, respectively, to inform other blocks of the current statuses of input in the level form. The active-high level form can be used in trigger or condition for logic circuits.

## C. Button Level Pulse Convertor

An active-high level status of a button can represent whether the button is pressed or not at the time instant when the check is done. However, if an enable-like input needed to be driven according to the time instant when the debounced clean button pressed action is received, a pulse should be generated for one clock period. This pulse signal can trigger shift register modules to take only the specified input by the state machine to save. Since the one-clock-period pulses are helpful for such operations, Button Controller must provide these for each CMD_Reg and KEY_Reg statuses.

## V. Hover Controller

The Hover Controller block is responsible for highlighting both products listed in the basket section and the product images. The product images have a square mark on the left upper side to indicate the product is selected. There are two conditions where the products are highlighted; first, every barcode digit entered triggers matching items; second, when the user is in the interactive selection mode. On the other hand, when Basket Edit mode is activated, only the product list is hovered, indicating the currently selected product. Hence, this controller consists of two distinct sub-blocks, and one is responsible for barcode-related hovering, the other is for selected item hovering depending on the operation mode.

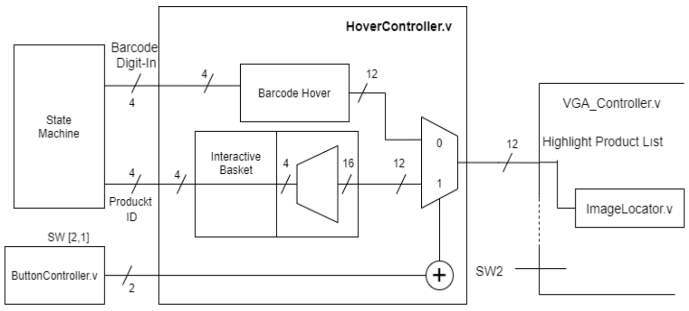


Fig. 5. Hover Controller Block Diagram

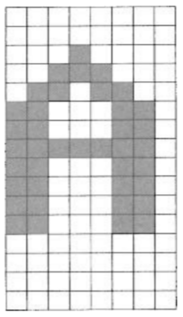
The selected item hovering process is simplified since all other blocks are compatible with Product IDs. Either Basket Edit or Interactive Select Mode, only one selected item has hovered; therefore, we use either array ID or Product ID in our algorithm. On the other hand, the matching barcode highlighting process is more complicated and resource-intensive. The Barcode Hover Sub-block should be able to identify possible products according to the current state of the barcode entered. Although the task seems like a database search structure in the programming aspect, it can be more efficient when it is considered a hardware description language. Therefore, we manage to generate a possible product list with no register used in this sub-block. This optimization resulted in freeing significant available resources that we can use for other operations.

## VI. Basket Controller

The basket controller is the module that adds selected products to the basket, calculates the price of each selected product and the total price every time, cancels the selected product, and decreases the total price accordingly. The state machine drives it. It takes six inputs: clock, enable, active-low reset, product ID, quantity, and cancel. When enable is high, it takes product ID and quantity inputs and saves them into the internal registers to remember them afterward. At the same time, it calculates the price of each selected product and saves these prices. Also, it updates the total price and the number of products in the basket after each product addition. The basket controller can hold a maximum of twelve products. When cancel is high, the basket controller takes product ID and deletes this ID, quantity, and price from the registers, decreasing the total price and the number of products in the basket. It gives thirty-eight outputs: two of them are the number of products and the total price of the products in the basket. Other thirty-six outputs are twelve product IDs, twelve quantities, and twelve prices. The basket controller continuously gives these outputs, so other modules do not need to remember the data related to the products in the basket. Finally, when the reset input is low, the basket controller sets all internal registers and outputs to their default values.

## VII. Text Controller

Text controller is a module that writes the basket information to the screen. The module takes the basket outputs and VGA controller counters as input and gives one-bit output. In this controller, we add the all characters in the ASCII table to the font ROM as 8x16 bits. These bits represent the pixels on the screen. In the figure 6, you can see that each character is represented zeros and ones. If the bit is 1, the pixel will paint a constant color. If the bit is 0, the pixel will paint the background color. Thus, any word can be written to the screen. To achieve this, we used two sub-systems.



```
//code x41
11'h410: data = 8'b00000000; //
11'h411: data = 8'b00000000; //
11'h412: data = 8'b00010000; //    *
11'h413: data = 8'b00111000; //   ***
11'h414: data = 8'b01101100; //  ** **
11'h415: data = 8'b11000110; // **   **
11'h416: data = 8'b11000110; // **   **
11'h417: data = 8'b11111110; // *******
11'h418: data = 8'b11000110; // **   **
11'h419: data = 8'b11000110; // **   **
11'h41a: data = 8'b11000110; // **   **
11'h41b: data = 8'b11000110; // **   **
11'h41c: data = 8'b00000000; //
11'h41d: data = 8'b00000000; //
11'h41e: data = 8'b00000000; //
11'h41f: data = 8'b00000000; //
```

Fig. 6. Character pixel explanation and example from font ROM

The first part consists of presets of the items and their corresponding quantity and price values. Since the basket controller module stores the data as a combination of the item name, quantity, and price, this preset uses a conditional structure to use only the necessary part in the second part. This part only created outputs of the first rows of the characters, and a counting method is used in the second part for size optimization. For item names, twelve case options are created for product id inputs and stored in these options. Each option is set to save 9 characters and create an output register of 63 bits long to simplify the storage. A similar method with item names used and the output register creates 11 characters with

77 bits long for quantity and price. Since the quantity and price inputs are 1 BCD and 4 BCD long, respectively, their position is fixed on the output register. A dot sign is placed between the second and third BCD values after multiplying the price by 100 to handle fractional components. An enable method is used not to create a new module for the total price. For this particular case, the structure of the output register is changed according to the enable value since the total price can be 5 BCD long.

The second part's goal is the send the pixel information to the VGA controller. In the font ROM, each character is represented with 7 bits since there is 128 character in the ASCII table. To make easy writing and reading pixels, each character consists of 8 bit (columns) x 16 bit (rows). Therefore, inputs of the font ROM take 11 bit where the first 7 bit represents a character, last 4 bit represents the row number of the character. While sending the pixel information to VGA, there are four counters which are column, character, row, and basket. Since the VGA counters count horizontally, the column counter selects the pixels from the first row of the first character of the first product in the basket. When it reaches 8, it returns to 0 and increases the character counter, and starts to write the top of the second character. When the characters finish, the character counter returns to zero and the row counter is increasing and start to write the second row of the first product in the basket. When the row counter reaches 16, the first product in the basket is finished and the basket counter is increased by 1. Then, the same procedure is applied to other products. Thus, all products and their prices are written on the screen.

## VIII. State Machine

The state machine is responsible for managing other modules depending on the operation mode or state. It sends activation, enable, and reset pulses when needed to operate correctly. There are seven states, namely, "Start", "Idle", "Barcode", "Interactive", "Quantity", "BasketEdit", "EndShopping". These states are based on the nature of the design. "Start" state is for preparing the modules for new customers and resetting previous information. When reset is done "Start" state directly goes to "Idle" state, where the operation mode is decided and redirected. There are three operation modes, "Barcode Selection Mode", "Interactive Selection Mode", and "Basket Edit Mode". According to mode prerequisites, the "Idle" state redirects the state to the proper mode. The "Idle" state handles the transition between operations. Basket Edit operation mode is superior to other operation modes; when activated, it has priority. The condition for every operation mode is checked inside its state; hence, a transition between operations can be catch and handled by again "Idle" state. After any product selection by either Interactive or Barcode, the state is transited to "Quantity" state, where the machine waits for quantity input from the user or the cancel action. When quantity is entered, the state machine handles the other controllers in order to add the product to the basket and display it. Then, it again forwards the state to "Idle" state where the operation mode is decided at the next clock pulse, reaching either "Barcode", "Interactive" or "BasketEdit" states.

On the other hand, if "BasketEdit" state is reached, the state waits for direction input from the user. If the user wants to select the item which is wanted to be deleted, "Select Command Button" is pressed. Finally, when the user wants to end the shopping, "End Shopping Command Button" is pressed, which leads to "EndShopping" state. The "EndShopping" state waits for the user for confirmation by pressing "End Shopping Command Button" again. When the confirmation is done, it directs the state to "Start" state, where a new shopping environment is prepared. In Figure 7, the State Machine Viewer result generated from Quartus Prime can be seen. For a detailed state flowchart, please refer to Appendix I.

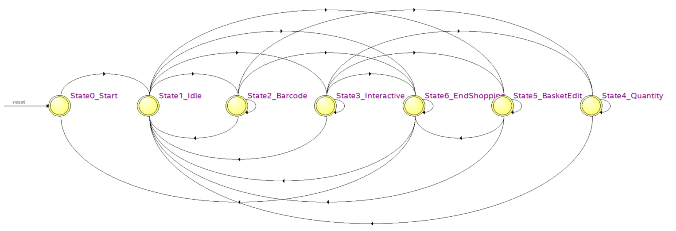


Fig. 7. The State Machine Viewer result generated from Quartus Prime

## IX. Demonstration Photos

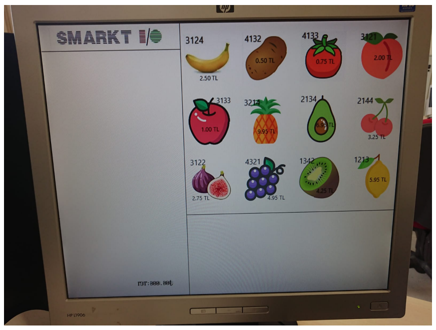


Fig. 8. Starting view of the screen

In this section, we are showing the some photos of the demonstration. In the figure 9, you can see that first two seven segment display shows the product ID which is represented by increasing left to the right. For example, pineapples product ID is 05 and babanas product ID is 00. Last 4 segment represents the product barcode. Also, leds on the FPGA board shows the states of the machiene. So, we can check the current state from FPGA board.

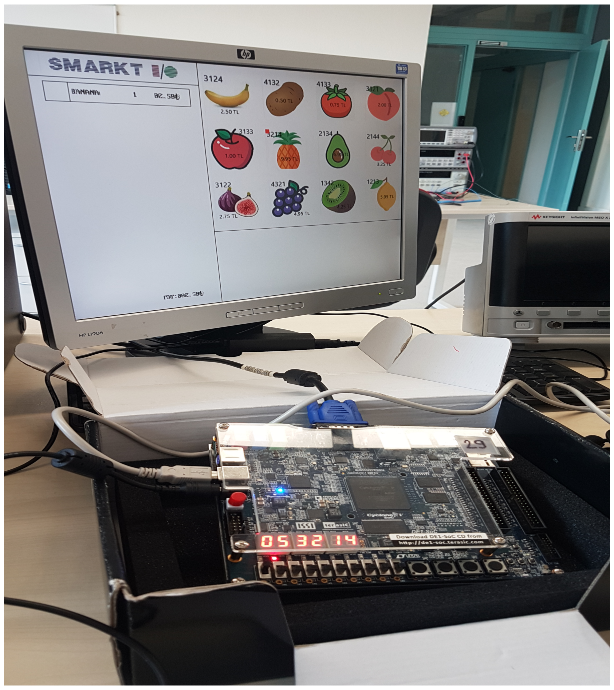


Fig. 9. Seven segment and led usage

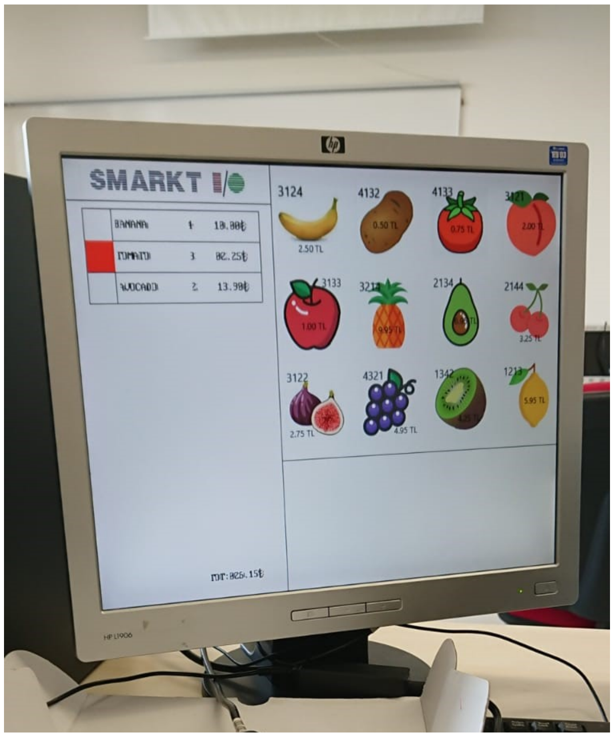
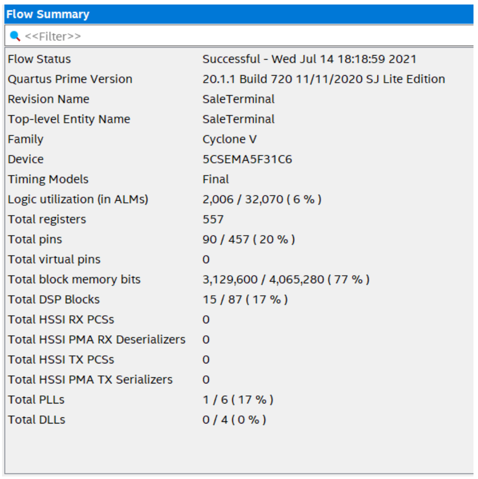


Fig. 10. Basket screen

## X. Conclusion And Comments

In this project, a POS terminal is created with FPGA. The entire design consists of five controllers, namely VGA controller, button controller, hover controller, basket controller and text controller. VGA controller is responsible from creating a POS terminal screen with 800x600 resolution at 60 Hz. Button controller adjusts switches and buttons to provide an interactive movement in the screen. Hover controller highlights the related images in the screen during the processes of entering barcode digits and interactive selection modes. Basket controller adds selected items to the basket, calculates the prices and it also allows to delete item from a basket. Text controller writes the basket information (i.e names of items in the basket and their prices) to the POS terminal screen. Finally, there is a state machine which combines and governs these five controllers. To be able to create this project, we have benefited from our knowledge within the concepts of sequential and combinational logic blocks and gates.


Flow Summary

<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Wed Jul 14 18:18:59 2021 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Lite Edition |
| Revision Name | SaleTerminal |
| Top-level Entity Name | SaleTerminal |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 2,006 / 32,070 ( 6 % ) |
| Total registers | 557 |
| Total pins | 90 / 457 ( 20 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 3,129,600 / 4,065,280 ( 77 % ) |
| Total DSP Blocks | 15 / 87 ( 17 % ) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 1 / 6 ( 17 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

Fig. 11. Flow summary of the project

## XI. References

[1] (n.d.). Retrieved from http://tinyvga.com/vga-timing/800x600@60Hz

## XII. Appendix

Whole project file can be found with following link. https://github.com/AtaberkOKLU/SaleTerminal