Kepad-HDMI Interface

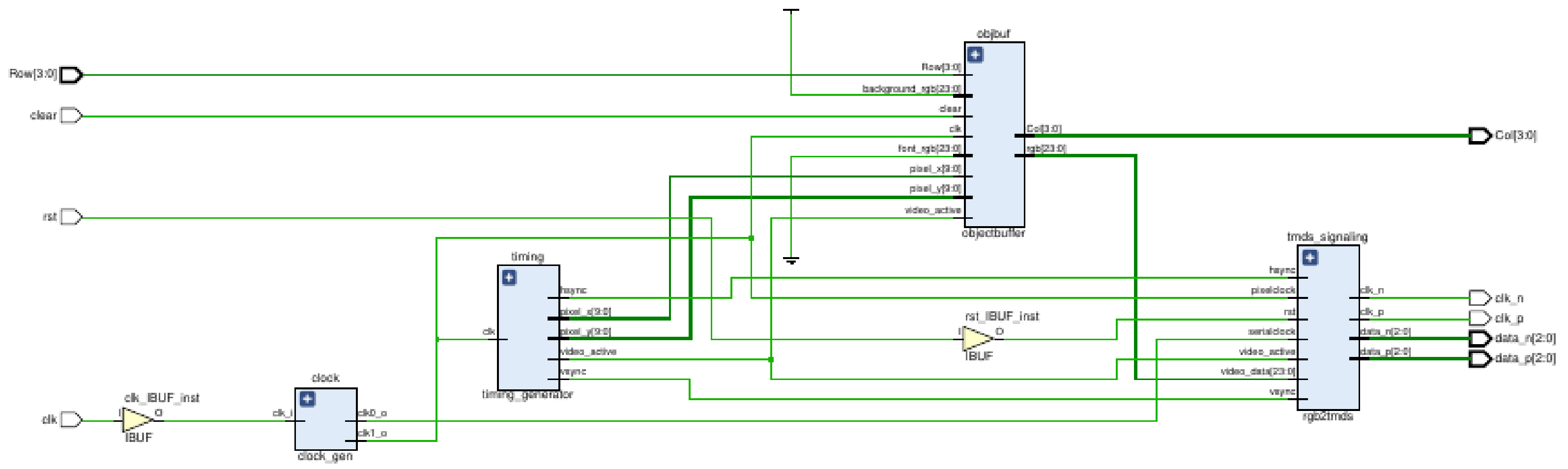
1) Problem Description

This project aims at building a HDMI and keypad interface using Digilent Zybo Board. Keypad input is given using **Pmod Keypad through Pmod port on Zybo.** HDMI Output on the board is connected to a **5 inch 800x480 screen**.

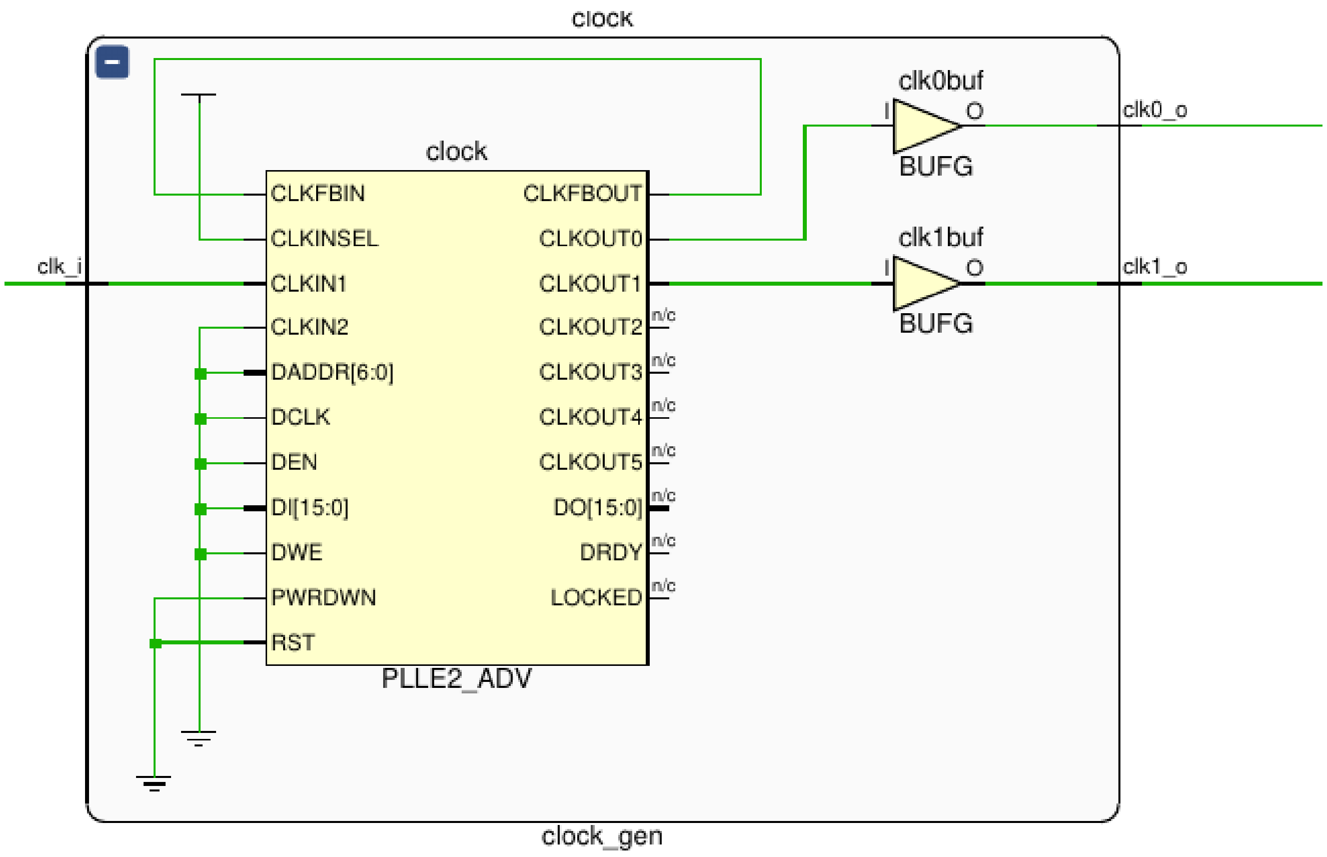
Keypad has 4 rows and 4 columns having 16 keys representing hex characters from **0 to F.** HDMI Display Screen is divided into 20 x 6 display keeping Font Size of 40 x 80 for each character. Initially, screen is blank with white background. On each key press, a character is added to the right of last charcater, Text font color is black.

Reset button resets the screen. Clear button clears the sreen.

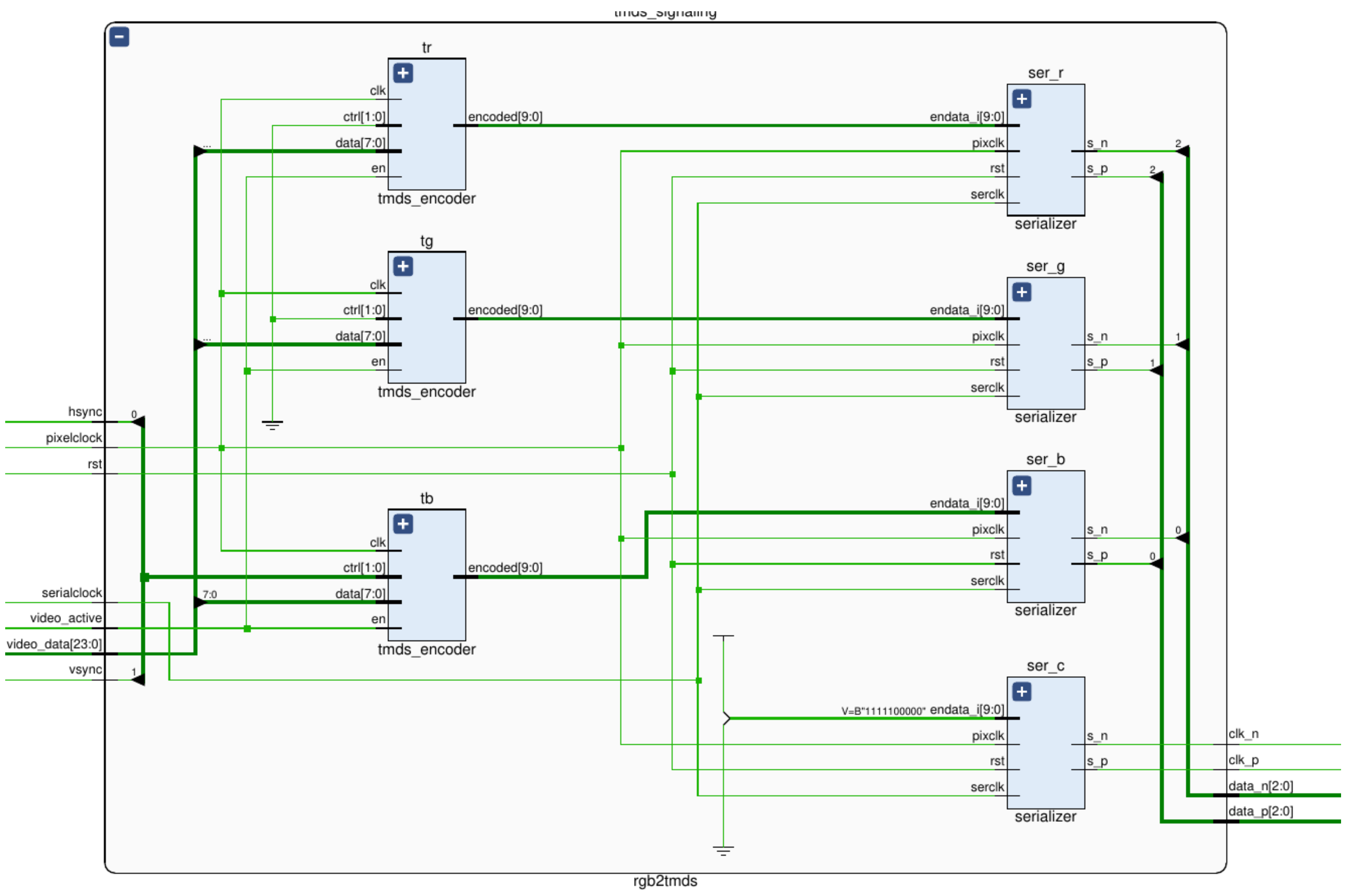
2) Functional Diagram (Architecture)



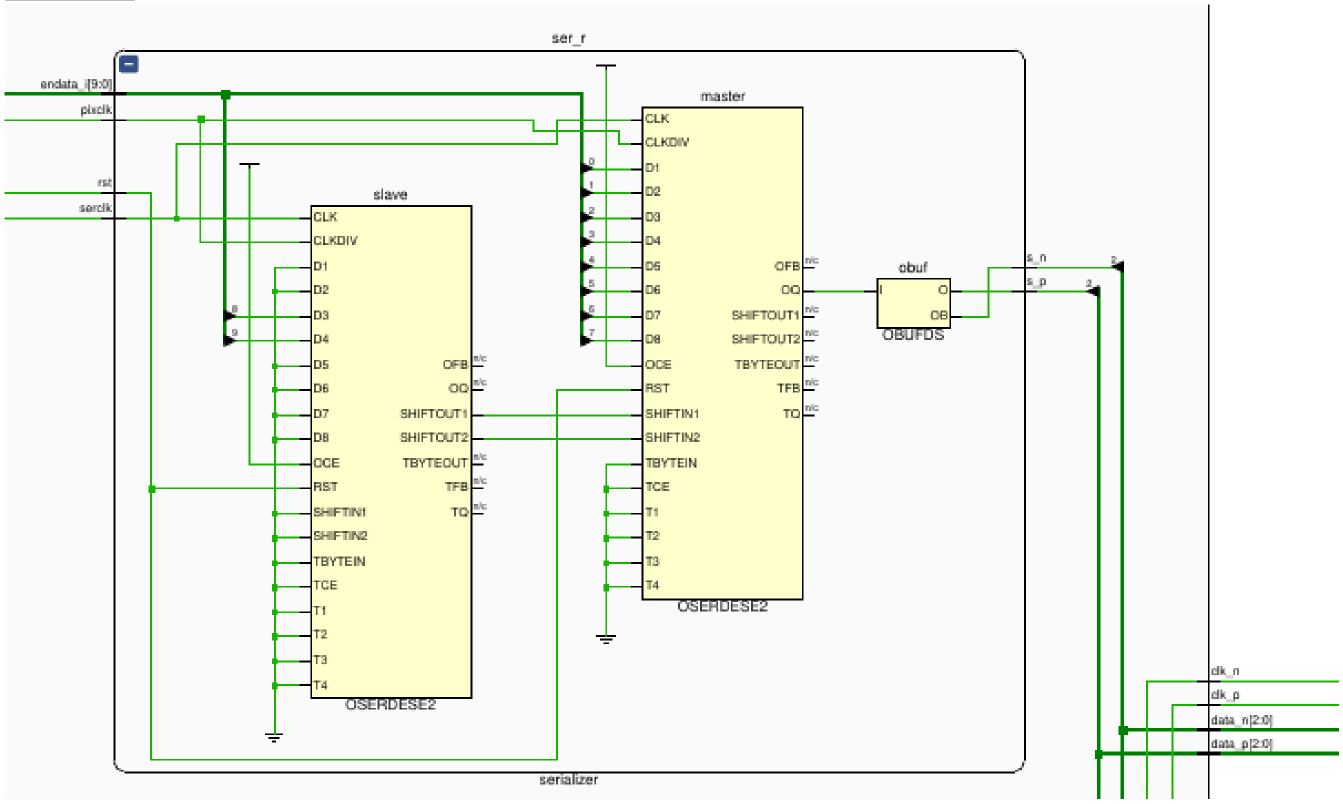
Clock (Pixel Clock and Serial Clock) Generation using PLL



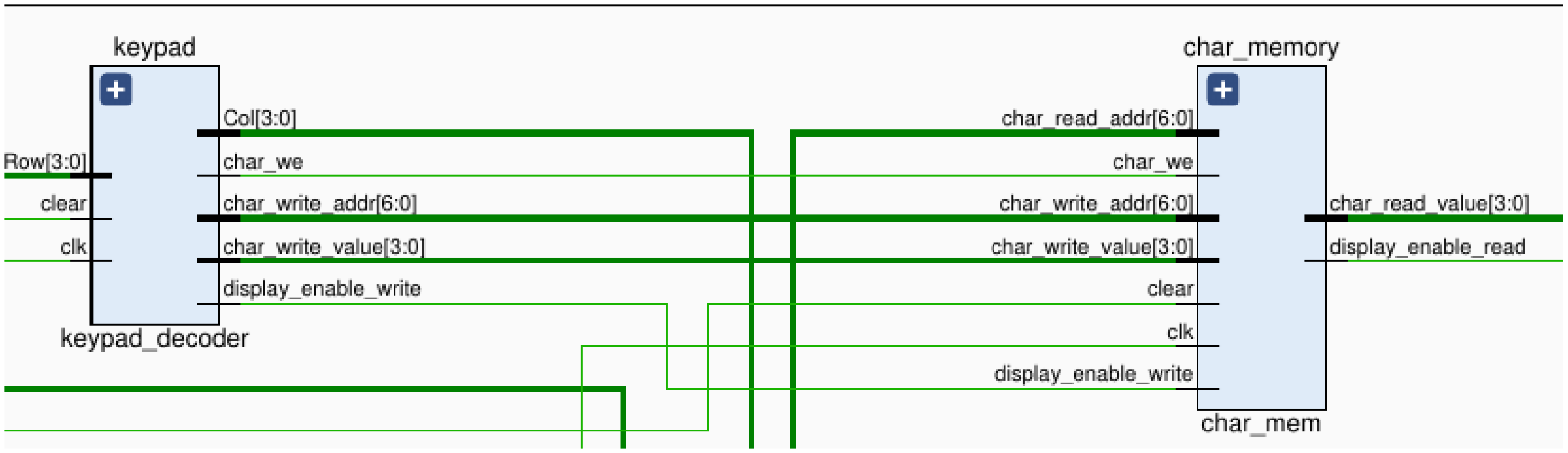
TMDS Encoder with Serialiser



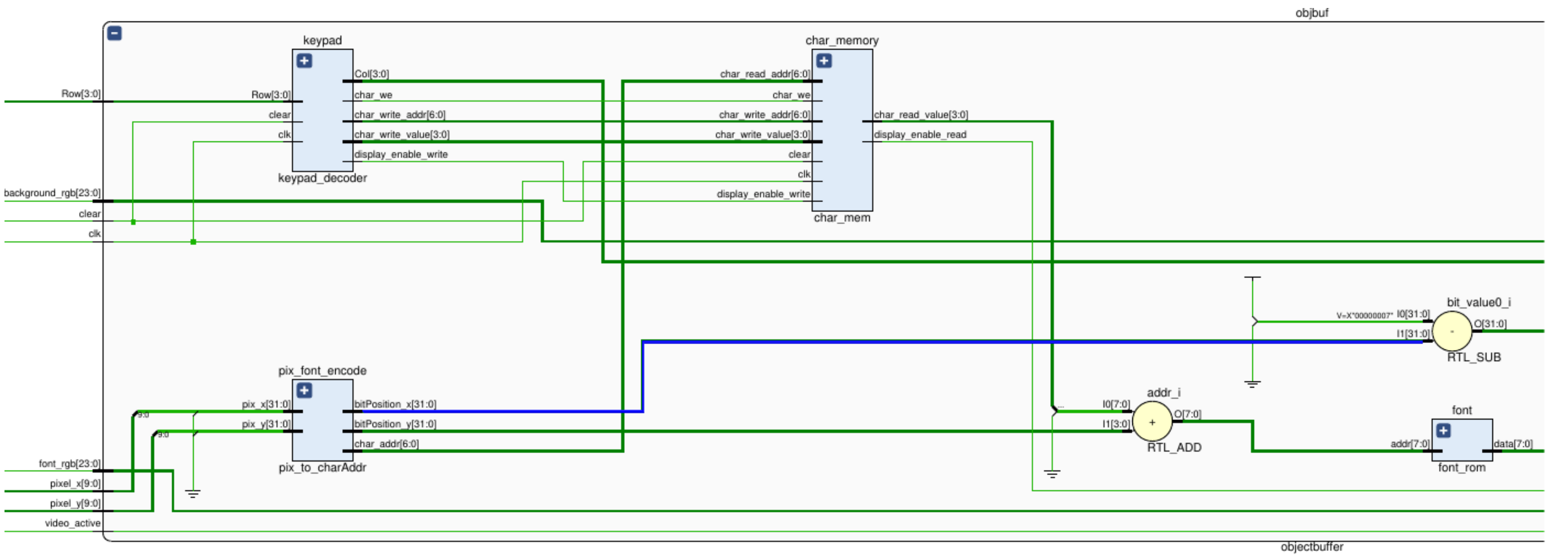
Serialiser



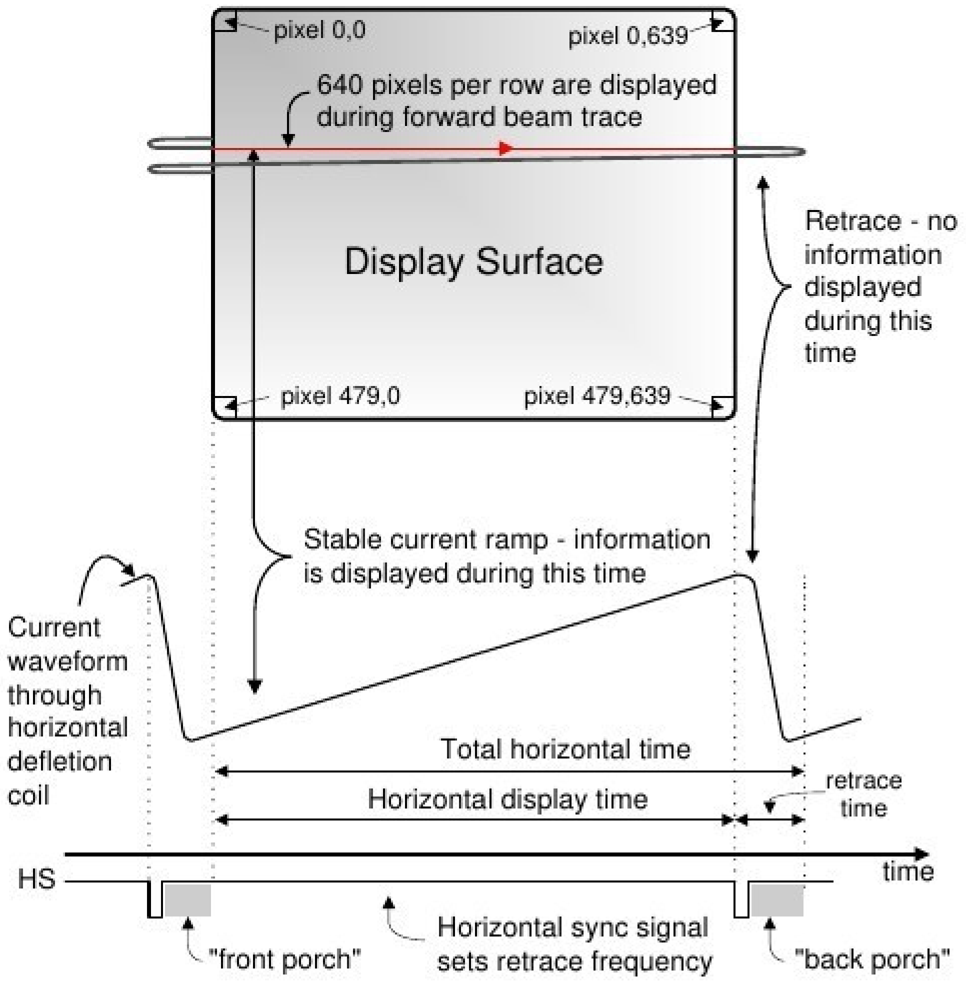
Keypad input written to Character Memory

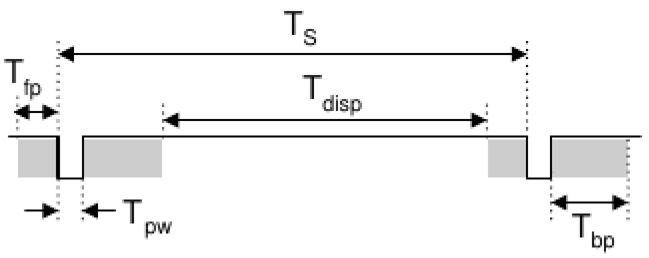


Data mapping to each pixel address generated by timing generator at each pixel (after each pixel clock)



3) Timing Diagrams


pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
Retrace - no information displayed during this time
pixel 479,0
pixel 479,639
Stable current ramp - information is displayed during this time
Current waveform through horizontal defletion coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"



For 800 x 480 screen,
For **Hsync,**
T_{fp} = 40 clock cycles ;
T_{pw} = 48 clock cycles ;

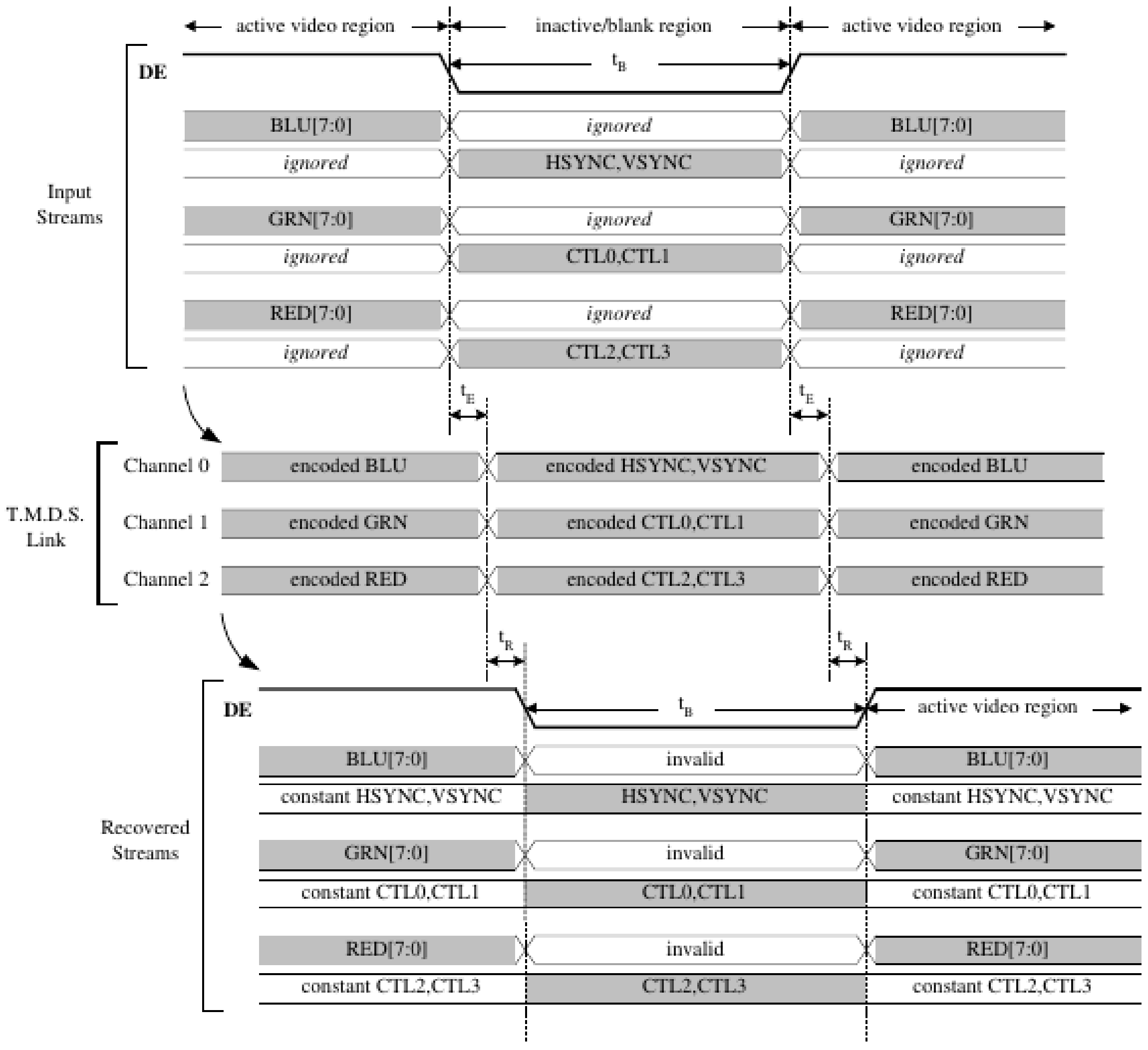
T_s = 928 clock cycles;
T_{disp} = 800 clock cycles;
T_{bp}= 40 clock cycles;

For V**sync,**
T_{fp} = (13 * 928) clock cycles ;
T_{pw} = (3 * 928) clock cycles ;
T_s = (525 * 928) clock cycles;
T_{disp} = (400 *928) clock cycles;
T_{bp}= (29 *928) clock cycles;

At refresh rate of 60 Hz, clock cycles required is 928x525x60 = 29232000;
So, pixel clock is set to 30 Mhz.
Each pixel has 24 bits of data which is encoded into 30 bits of data. This is serialised to 3 different channels. So, serial clock should be able to support 10 times data rate and pixel clock. So, 5 times the pixel clock is chosen as serial clock frequency along with DDR (double data rate).

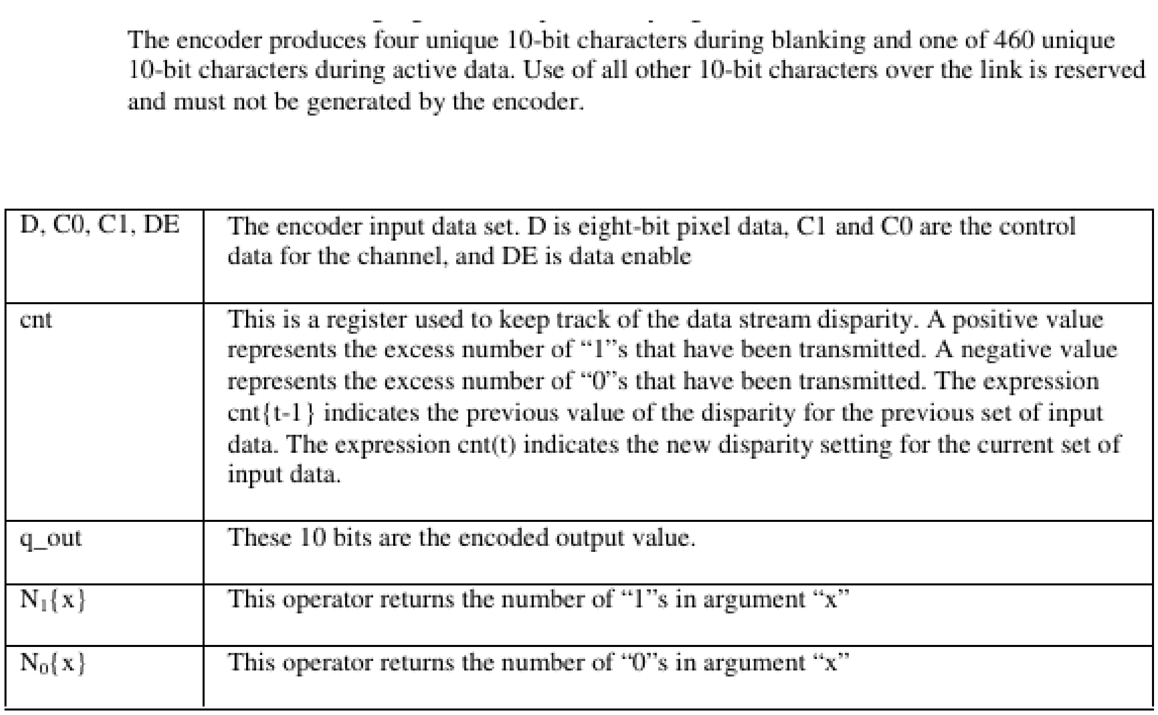
TMDS Link Timing Requirements

The maximum time for encode and serialization and decode is specified in order to bound latency across the interface. Figure 3-7 with Table 3-2 specifies these parameters.



Symbol	Description	Value	Unit
t_B	Minimum duration blanking period required to ensure character boundary recovery at the receiver. Blanking periods of this duration must occur at least once every 50 mS (20 Hz).	128	T_{pixel}
t_E	Maximum encoding/serializer pipeline delay	64	T_{pixel}
t_R	Maximum recovery/de-serializer pipeline delay. Recovery timing includes inter-channel skew, and is measured from the earliest DE transition among the data channels.	64	T_{pixel}

TMDS Encoding Algorithm

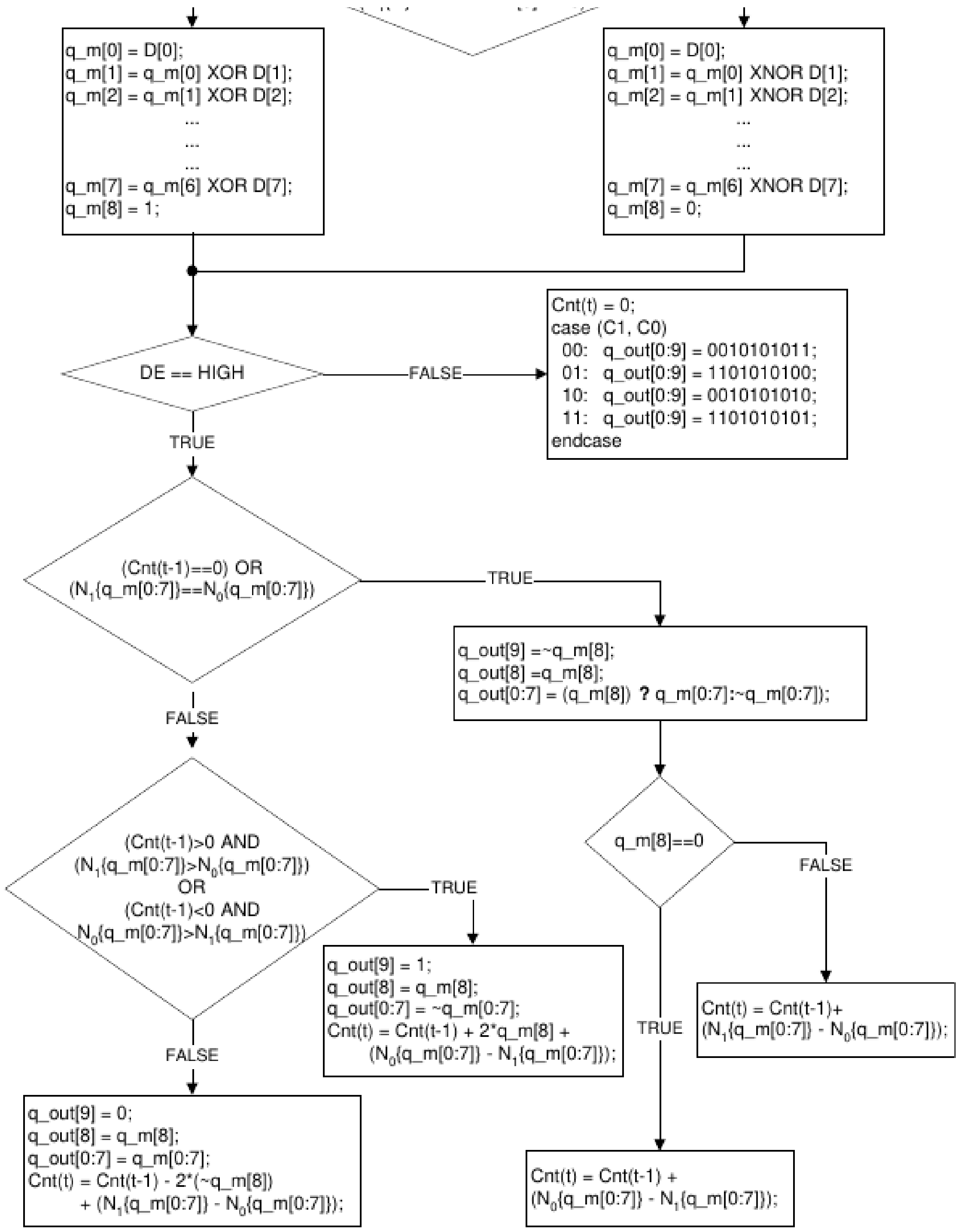


The encoder produces four unique 10-bit characters during blanking and one of 460 unique 10-bit characters during active data. Use of all other 10-bit characters over the link is reserved and must not be generated by the encoder.

D, C0, C1, DE	The encoder input data set. D is eight-bit pixel data, C1 and C0 are the control data for the channel, and DE is data enable
cnt	This is a register used to keep track of the data stream disparity. A positive value represents the excess number of “1”s that have been transmitted. A negative value represents the excess number of “0”s that have been transmitted. The expression cnt{t-1} indicates the previous value of the disparity for the previous set of input data. The expression cnt(t) indicates the new disparity setting for the current set of input data.
q_out	These 10 bits are the encoded output value.
$N_1\{x\}$	This operator returns the number of “1”s in argument “x”
$N_0\{x\}$	This operator returns the number of “0”s in argument “x”

Table 3-1 Encoding Algorithm Definitions

The stream of T.M.D.S. characters produced by the encoder is serialized for transmission on the T.M.D.S. data channel. The least significant bit of each character (q_out[0]) is the first bit to be transmitted.


q_m[0] = D[0];
q_m[1] = q_m[0] XOR D[1];
q_m[2] = q_m[1] XOR D[2];
...
q_m[7] = q_m[6] XOR D[7];
q_m[8] = 1;
q_m[0] = D[0];
q_m[1] = q_m[0] XNOR D[1];
q_m[2] = q_m[1] XNOR D[2];
...
q_m[7] = q_m[6] XNOR D[7];
q_m[8] = 0;
DE == HIGH
FALSE
TRUE
Cnt(t) = 0;
case (C1, C0)
00: q_out[0:9] = 0010101011;
01: q_out[0:9] = 1101010100;
10: q_out[0:9] = 0010101010;
11: q_out[0:9] = 1101010101;
endcase
(Cnt(t-1)==0) OR (N1{q_m[0:7]}==N0{q_m[0:7]})
TRUE
FALSE
q_out[9] =~q_m[8];
q_out[8] =q_m[8];
q_out[0:7] = (q_m[8]) ? q_m[0:7]:~q_m[0:7]);
q_m[8]==0
FALSE
TRUE
(Cnt(t-1)>0 AND (N1{q_m[0:7]}>N0{q_m[0:7]}) OR (Cnt(t-1)<0 AND N0{q_m[0:7]}>N1{q_m[0:7]}))
TRUE
FALSE
q_out[9] = 1;
q_out[8] = q_m[8];
q_out[0:7] = ~q_m[0:7];
Cnt(t) = Cnt(t-1) + 2*q_m[8] + (N0{q_m[0:7]} - N1{q_m[0:7]});
Cnt(t) = Cnt(t-1)+ (N1{q_m[0:7]} - N0{q_m[0:7]});
q_out[9] = 0;
q_out[8] = q_m[8];
q_out[0:7] = q_m[0:7];
Cnt(t) = Cnt(t-1) - 2*(~q_m[8]) + (N1{q_m[0:7]} - N0{q_m[0:7]});
Cnt(t) = Cnt(t-1) + (N0{q_m[0:7]} - N1{q_m[0:7]});

Font ROM: The appearance of the characters on the screen is determined by a "font ROM". The font ROM contains the pattern of pixels that should be displayed on the screen when a particular character needs to be displayed. The bits within the font ROM indicate which pixels of a 8 x 16 bit tile should be displayed in the 'foreground' and which pixels on the display should be displayed in the background. A '1' in the font ROM indicates the corresponding pixel should be displayed in the foreground (i.e., white in our case) while a '0' in the font ROM indicates that the corresponding pixel should be blanked or put in the background (black in our case). The text below demonstrates the contents of the font ROM for the upper-case character 'A':

```
"00000000", -- 0
"00000000", -- 1
"00010000", -- 2    *
"00111000", -- 3   ***
"01101100", -- 4  ** **
"11000110", -- 5 **   **
"11000110", -- 6 **   **
"11111110", -- 7 *******
"11000110", -- 8 **   **
"11000110",--  9  **   **
"11000110",--  a  **   **
"11000110",--  b  **   **
"00000000", -- c
"00000000", -- d
"00000000", -- e
"00000000", -- f
```

Character Memory: In addition to the font ROM, we use a "character memory" that stores the character value at each of the 20x6 character locations on the display. The minimum size of this memory is 20x6x4 bits to provide enough room to store characters (one nibble each) for each of the 20 columns and 6 rows.

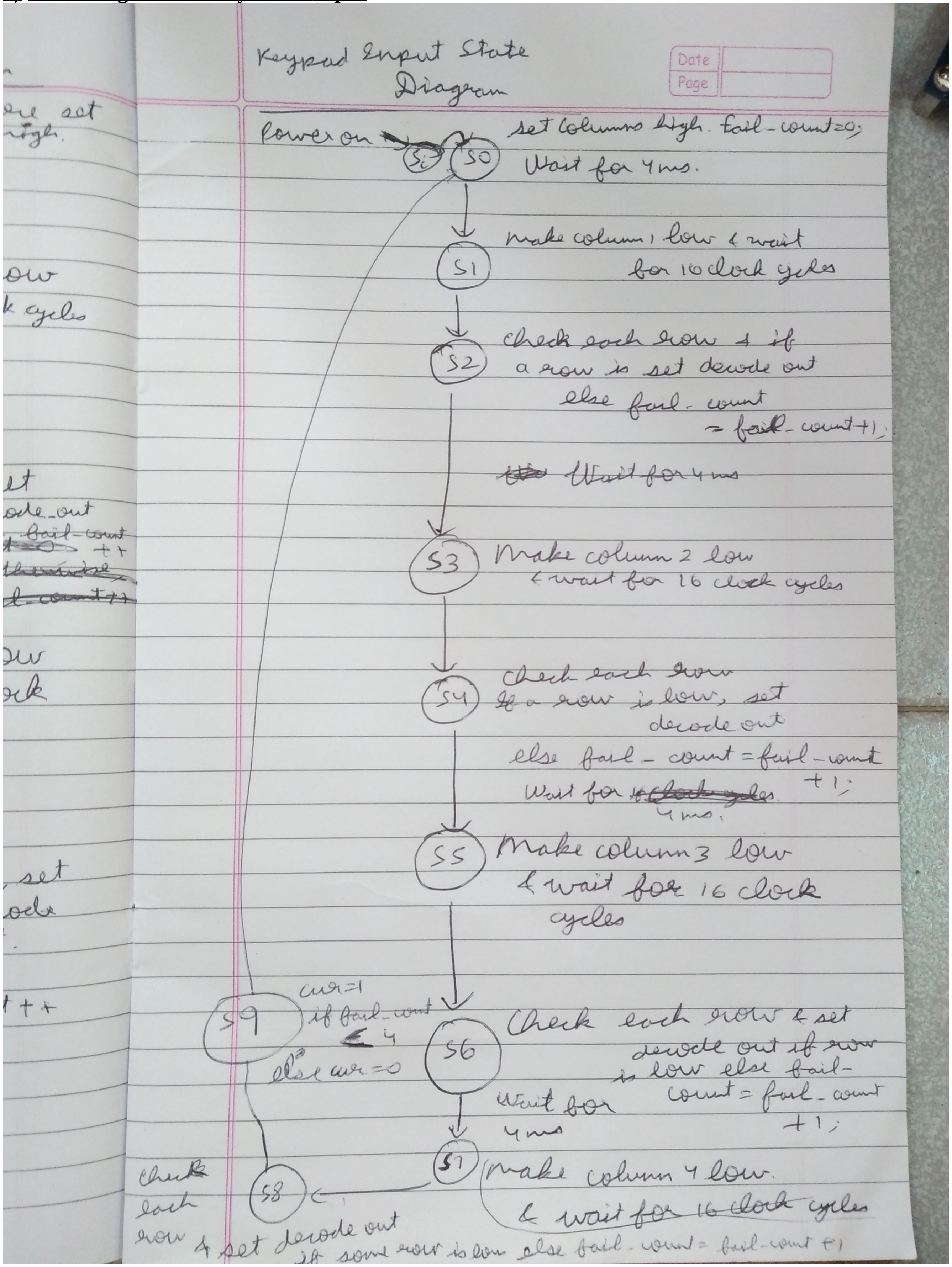
This memory has two memory ports: a character read port and a character write port. The read port is needed by the character generator to read the character value of the current character location. The write port is used to update the contest of the character display.

The **char_read_addr** port is a 7 bit signal used to determine which location in the character memory to read. The result of the read is available on the **char_read_value**. Like the font ROM, this character ROM will provide the character result one clock cycle after the address is provided.

The second memory port is the character write port. This write port is used to update the contents of the character memory so the display can be changed. This port operates simultaneously with the read port so the character display can access characters while the character memory is being updated. Three signals are used for this port. The **char_write_addr** signal is used to indicate the address within the character memory that will be updated. The **char_write_value** is the value that will be written at this address. The **char_we** signal is the control signal to enable a write into the character memory. When the char_we signal is asserted, the value on the char_write_value signal will be written at address char_write_addr at the next clock edge.

Character Memory is used to give video_data to TMDS signal generator.

4) State Diagram for Keyboard Input


Keypad Input State Diagram
Power on
Si
S0
set columns high. fail-count=0; Wait for 4ms.
S1
make column 1 low & wait for 16 clock cycles
S2
check each row & if a row is set decode out else fail-count = fail-count+1;
~~Wait for 4ms~~
S3
Make column 2 low & wait for 16 clock cycles
S4
Check each row If a row is low, set decode out else fail-count = fail-count +1; Wait for ~~16 clock cycles~~ 4ms.
S5
Make column 3 low & wait for 16 clock cycles
S6
Check each row & set decode out if row is low else fail-count = fail-count +1;
Wait for 4ms
S7
make column 4 low. & wait for 16 clock cycles
S8
Check each row & set decode out if some row is low else fail-count = fail-count +1
S9
cur=1 if fail-count ≤ 4 else cur=0

After each 4 ms, a column is made low. Then, each row is checked for low signal. Corresponding after each 16 ms, it is recorded whether a key was pressed. Then, cur state is recorded. If last state was not key pressed and cur state is key pressed, then keypad writes to character memory.

<u>5)Timing Analysis</u>

<u>System Clock</u>
System Clock (Time Period) = 8 ns;

Worst Negative Slack (Pulse width) = 2 ns

<u>Pixel Clock</u>
Pixel Clock (Time Period= 33.33 ns

Worst Negative Slack (Setup) = 5.44 ns

Worst Negative Slack (Hold) = 0.177 ns

Worst Negative Slack (Pulse width) = 16.167 ns

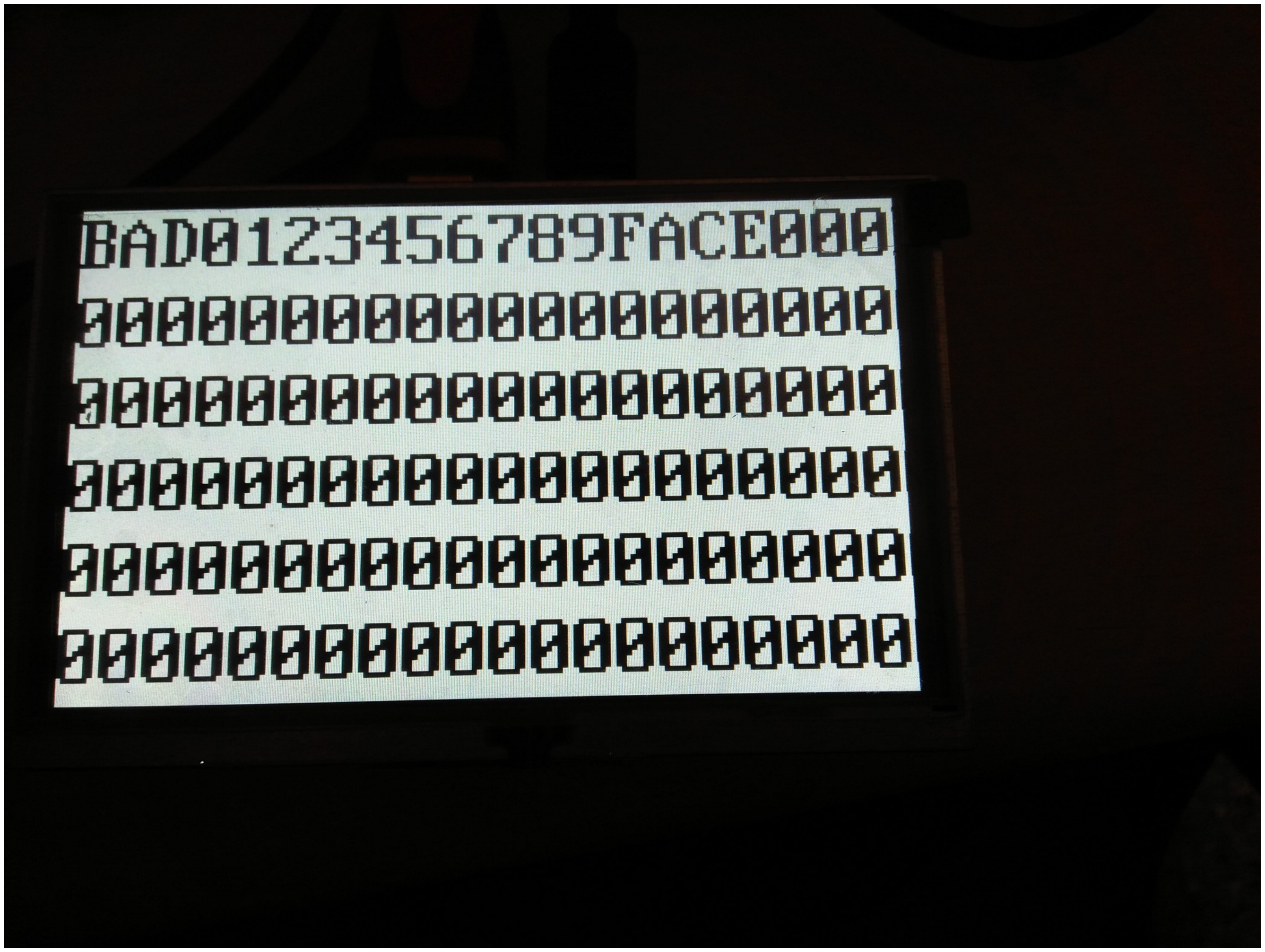
<u>Serial Clock</u>
Serial Clock (Time Period) = 6.66 ns;

Worst Negative Slack (Pulse width) = 4.511 ns

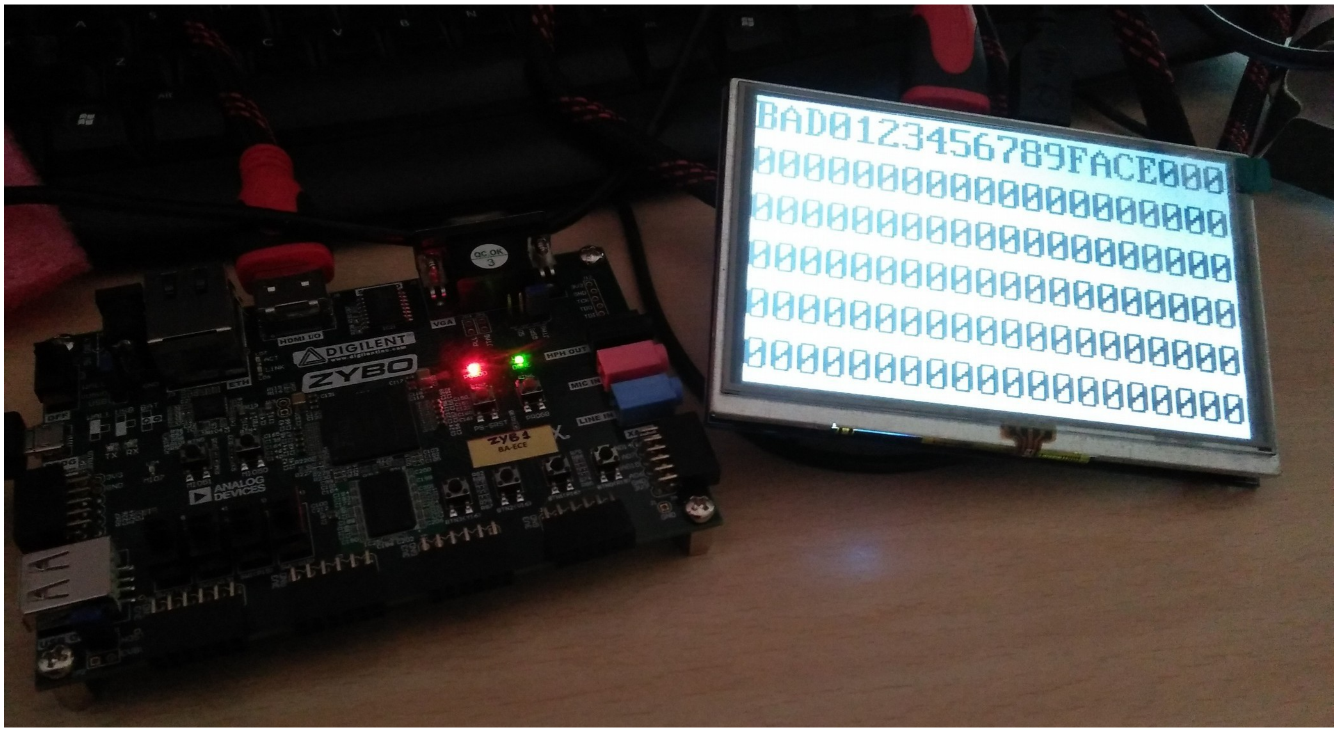
<u>Resource Utilisation</u>

LUTs = 1189
FFs = 690

6) Results on Board (HDMI Display)



```
BAD0123456789FACE000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```



Conclusion

20 x 6 Text Displaying hex characters was implemented using Zybo board HDMI Output printing characters on screen taking input from Pmod Keypad.

References

1) Digital Visual Interface – www.ddwg.org
2) VGA Text Generator- https://ece320web.groups.et.byu.net/labs/VGATextGeneration/VGA_Terminal.html