Driving a VGA and keyboard with Nexys 4 DDR FPGA BOARD

02/07/2019

Luca Caronti, Simone Ruffini

University of Trento

Supervisor: Prof. Roberto Passerone

Overview

The project consists in managing a keyboard and display datas on a monitor through a VGA. The specifications of the VGA resolution are 640 x 480, and the US layout was chosen for the keyboard. Entire project was written in VHDL and simulated on Vivado software.

All project codes are available on github at https://github.com/lucacaronti/project_vhdl.

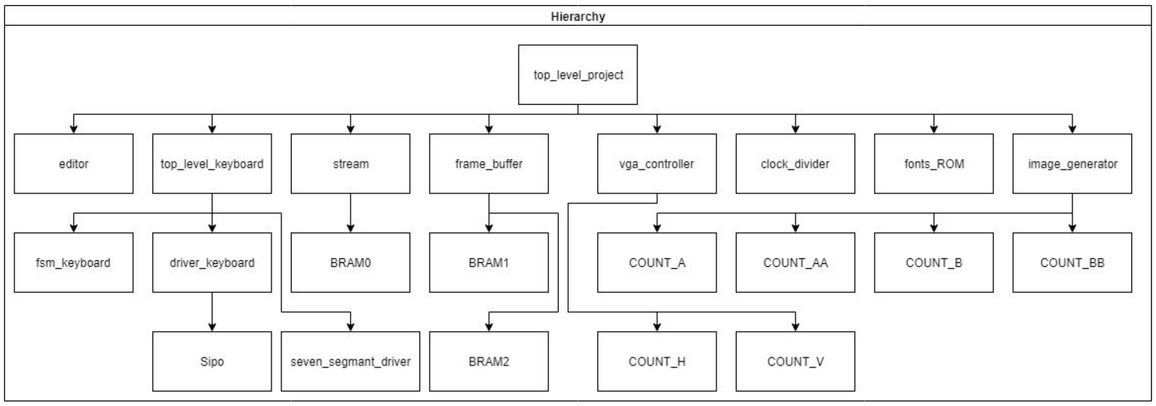
The features of the project are the possibility to write all the available letters, up to five at the same time, to move the cursor with arrows keys through max 10 pages, to change background and text color (one is the opposite of the other) through the switches (switches from 11 to 8 are for red color, from 7 to 4 are for green and from 3 to 0 are for blu).

There are also four 7-segment display that print the entered character code.

Project specifics

Architecture specification

The architecture is divided into 22 entities like as shown in the graph below.



Top level project

This entity has the only purpose of connecting all other entities.

Moreover, a constraints file is related to this entity for physical pin connections.

Image generator

General description

Image generator is the bridge between frame buffer and the physical vga pins on the Nexys 4 DDR board. The main function is to manage requests,decoding and updates from frame buffer values into fonts rom ones that will be displayed on monitor.

Port Description

- pixel_clk: master clock of the whole unit at 25MHz
- resetn: main reset signal
- disp_en: see vga controller disp_en
- horizontal_active: see vga controller horizontal_active
- vertical_active: see vga controller vertical active
- cursor_rel_pos: see editor cursror_rel_pos
- cursor_blink_time: see editor cursor_blink_time
- data_from_frame_buffer: see frame buffer output_char_value
- address_for_frame_buffer_data: is the address sent to frame_buffer where the next char to be displayed is stored
- data_from_fonts_rom: see data_out fonts rom
- address_for_fonts_rom_data: is the address sent to fonts rom where the bit values for that specific character are stored
- VGA_R/G/B: 4 bits vectors that represent the intensity values of RGB channels that will be displayed on screen
- SW: this signal comes from the built in deep switches on the Nexys Board and represents the value of the background color to be displayed such that 0 to 3, 4 to 7 and 8 to 11 are deep switches encoding such RGB value.

Processes

- COUNTERS_MNGMNT

  This process manages input signals for 4 counters: INIT (initialization of the counter to a fixed value) and CE (count enable).

  When vga controller is in display time COUNTER_A can count. Since COUNTER_A is activated, subsequent counters can count too since their CE signals are bounded together in cascade using terminals counts.

- SAVE_FONTS_ROM_DATA

Every time COUNTER_A reaches the end (TC_A = 1) the value of fonts_rom is stored in pixel_values_to_VGA. This is necessary because pixel values for character n are retrieved when character n-1 is displayed.

- MEM_EDITOR_SIGNALS

  The function of this process is to make a copy of all signals that could change during the output of a frame stored in the frame buffer.

- INV_COL

  This process checks if the current character is the one under the cursor, if so the next CHARACTER_WIDTH pixels will have the colours inverted. This is done with the help of a flag: invert_colors_MEM.

- IMAGE_GEN

  Displays the values stored in pixels_value_to_VGA and if the inverted_clolors_MEM flag is on the colors are inverted.

- COUNT_A

  This counter keeps track of the horizontal position in a character. So its value changes from 0 to CHARACTER_WIDTH -1.

- COUNT_AA

  This counter increases every time a character is completed, so its function is to track in which character column we are. This counter gets initialized at 1 and not 0 because during the display of character n we are requesting character n+1 so we are counting in advance for requesting to frame buffer the right values.

- COUNT_B

  COUNTER_B counts the vertical position in a character. Every character is made of CHARACTER_HEIGHT*CHARACTER_WIDTH pixels, so in our project CHARACTER_WIDTH pixels is a character and every CHARACTER_HEIGHT characters, a character is printed.

- COUNT_BB

  The counterpart of COUNT_AA but counts in multiples of VERTICAL_CHARS (the max value of characters that fit in a line) this because this value is used to make requests to frame buffer.

Combinational logic

The address sent to frame buffer is calculated by summing COUNTER_BB and COUNTER_AA. When frame buffer responds with a value this one is sent to fonts rom padded with the value of COUNTER_B (this value represents which character of a character we are printing)

Frame buffer

General description

The frame buffer deals with the management of the frame to be printed on the screen. It contains BRAM_1 that is used to save a snapshot of a frame from stream and BRAM_2 that is used to synchronize the start scanning signal that comes from VGA_controller. In fact when there is a transition from 0 to 1 of v_sync a process starts scanning and save the data from stream (BRAM_0) to frame buffer. Frame buffer stores ASCII int values and not pixels this makes less memory used. Since frame buffer is used with different clocks (master clk for write and pixel clk for read) use of a true dual port BRAM is necessary to overcome timing and clock phase problems.

Port description

- clk: master clock 100 MHz
- clk_vga: pixel clock 25 MHz
- resetn: general reset
- vertical_sync: see v_sync from vga controller
- stream_start_addr_for_scan: see frame_start_addr from editor
- output_data_from_stream: see output_char_value from stream
- addr_to_stream_for_data: is the address sent to stream where the next char to be saved is stored
- addr_request_char: see address_for_frame_buffer_data from image generator
- output_char_value: is the value returned after requesting data from address addr_request_char
- is_writing: optional signal used for future components that signals busyness of frame buffer, if is_writing is on and read requests are made, data could be not in sync with the actual value stored

Processes

- FSM_STARTER

  This process creates a pulse when senses a rising_edge in vertical_sync_mem, this pulse is assigned to start_pulse the signal responsible for starting the fsm. vertical_sync_mem is a signal in direct correlation with verical_sync but since the last one is out of sync with the master clk so sample of it must be taken.

- SEQUENTIAL, FUT_CALC, OUTPUTS

  This fsm switches between two states: wait and start, when in start state all subsequent processes are triggered else a sample of stream_start_address is taken for data consistency during stream scanning.

- MANAGE_EN_SIGNALS

  The only purpose of this process is to manage counter signals.

- ADDR_CALC

  This is a combinational process uses the value of counter and calculates address values for both stream requests and for storage in frame buffer itself. This process takes in count that output data from stream comes with 2 clocks delay.

- COUNTER

  Counts from NUMBER_OF_CHARS_IN_AFRAME times.

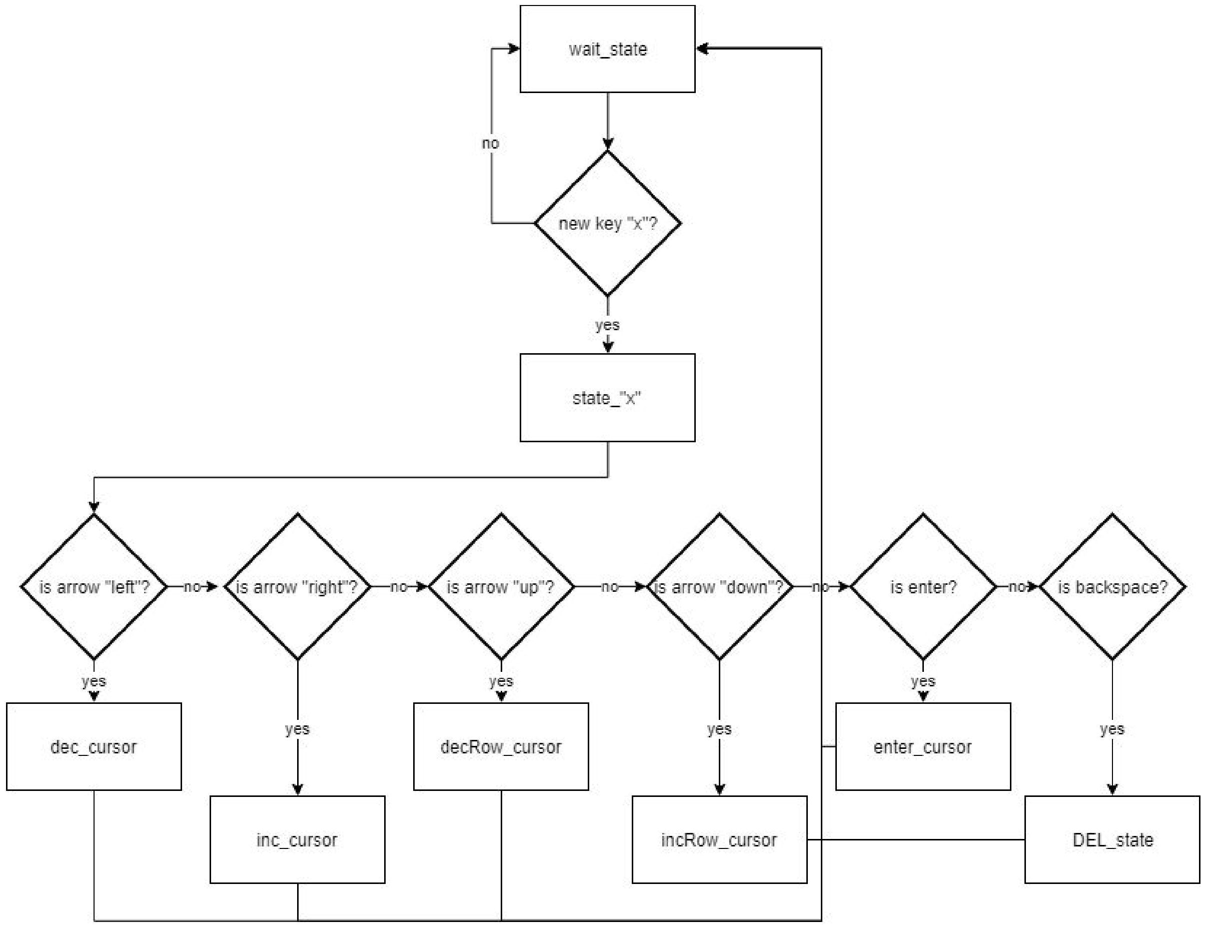
Editor

General description

Editor entity is used to manage:

1. Saving order if multiple keys are pressed at the same time.
2. Cursor position.
3. Special keys like arrows, backspace and enter.
4. Blinking of cursor timing.

Fsm description



Variable “x” goes from 1 to 5 and represents multiple key digit.

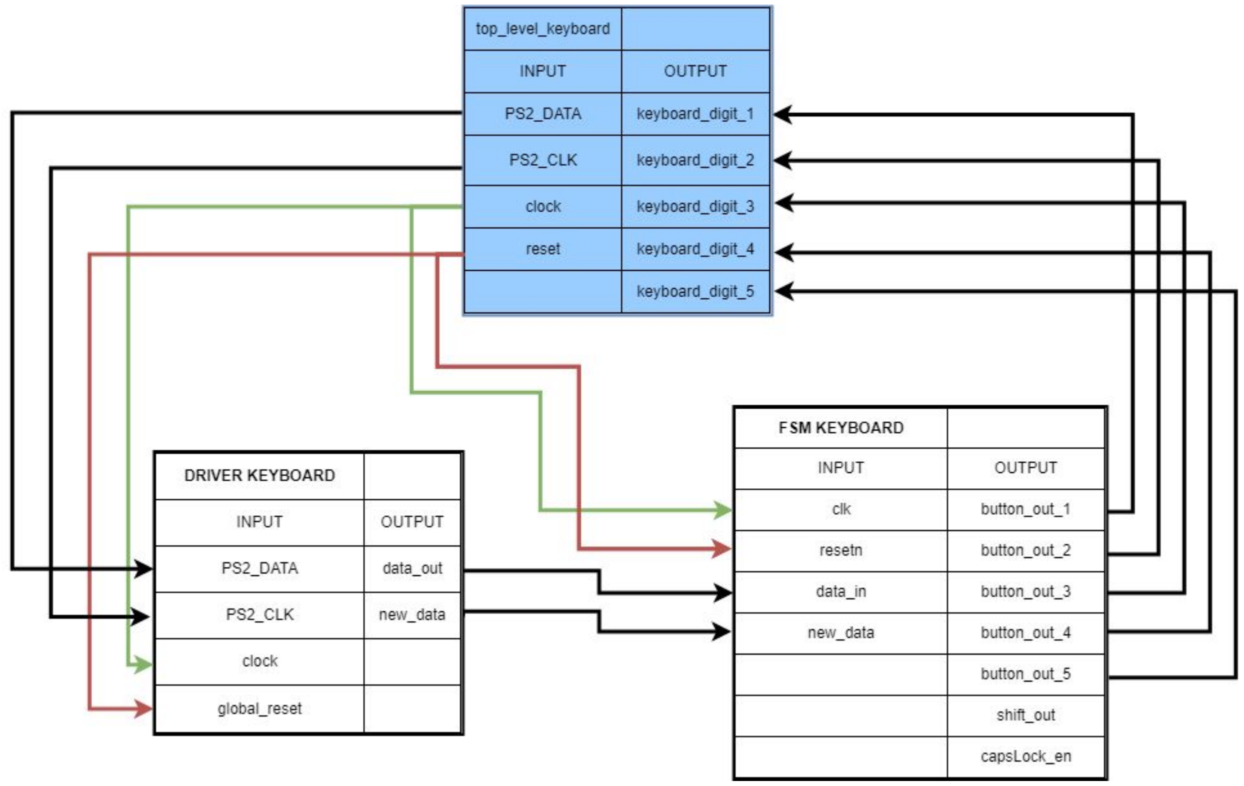
Port description

- sys_clk : system clock
- resetn : reset active low
- keyboard_digit_1 ... 5 : keyboard digit (5 lines) from top_level_keyboard
- cursor_pos_abs : absolute cursor potition
- cursor_rel_pos : cursor position relative to the page printed
- frame_start_addr : address of frame first position
- char_to_write : char to send into stream
- cursor_blinking_time : cursor blinking port

Top level keyboard

General description

His task is to connect all keyboard entity, i.e. fsm_keyboard and keyboard_driver.



In the upper diagram is shown how fsm_keyboard and driver_keyboard are linked together and with top_level_keyboard. Others tasks of this entity are the conversion from keyboard key format at ASCII format, the management of shift key and of caps lock.

Port description

- PS2_DATA : PS2 data line.
- PS2_CLK : PS2 clock line.
- clock : system clock.
- reset : reset active low.

- keyboard_digit_1 ... 5 : are the signal that indicate which key is pressed. Up to 5 keys could be pressed at the same time. Data are in ASCII format.
- CA, CB, CC, CD, CE, CF, CG, DP : cathodes for 7 segment display.
- AN : anode for 7 segment display.0

VGA controller

General description

This entity handles the VGA synchronization signals for a 640x480 @ 60Hz monitor.

Port description

- pixel_clk: vga clock that depends on the resolution of the display and framerate
- res: master reset signal
- h_sync: horizontal sync
- v_sync: vertical sync
- disp_en: display enable (screen time)
- horizontal_active: on when not in either back porch, sync pulse or front porch horizontal
- vertical_active: on when not in either back porch, sync pulse or front porch vertical

Fonts ROM

General description

Fonts ROM contains all bits that compose the 128 character. Each font is made of 8 bits for width and 16 for height. The process that gives you the data is synchronous. Every address of this fonts rom represents a CHARACTER that is a line of pixels that composes a entire CHARACTER. A character is made of 16 characters.

Port description

- clk: master clock at 25 Mhz (pixel_clock)
- addr: see address_for_fonts_rom_data of image generator
- data_out: is the 8 bit vector value that represents a character of a character

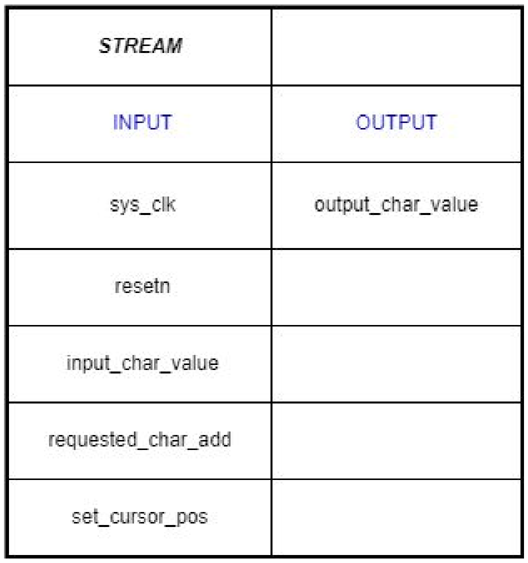
Stream

General description

Stream takes care of managing the writing and reading procedure from BRAM.

Stream is composed by 2 process; writing_into_BRAM and save_char_val that manage the relative tasks. See the code for more information.

The entity has the following ports.



STREAM	
INPUT	OUTPUT
sys_clk	output_char_value
resetn	
input_char_value	
requested_char_add	
set_cursor_pos	

Port description

- sys_clk : it’s the system clock.
- resetn : it’s the reset port, active low.
- input_char_value : This port is used to save data into BRAM. The procedure for saving datas are the following:

  Input_char_value must be for default at 0, then when there is a change on the port, data will be saved into BRAM in 1 clock cycle. It’s a asynchronous process so the port must be stable for minimum 1 clock cycle. Until the port doesn’t change value only one data is saved. To save two consecutive equal data input_char_value must goes to 0 for one clock cycle and then must be set again with the desired value. When the port is 0 no data are saved.

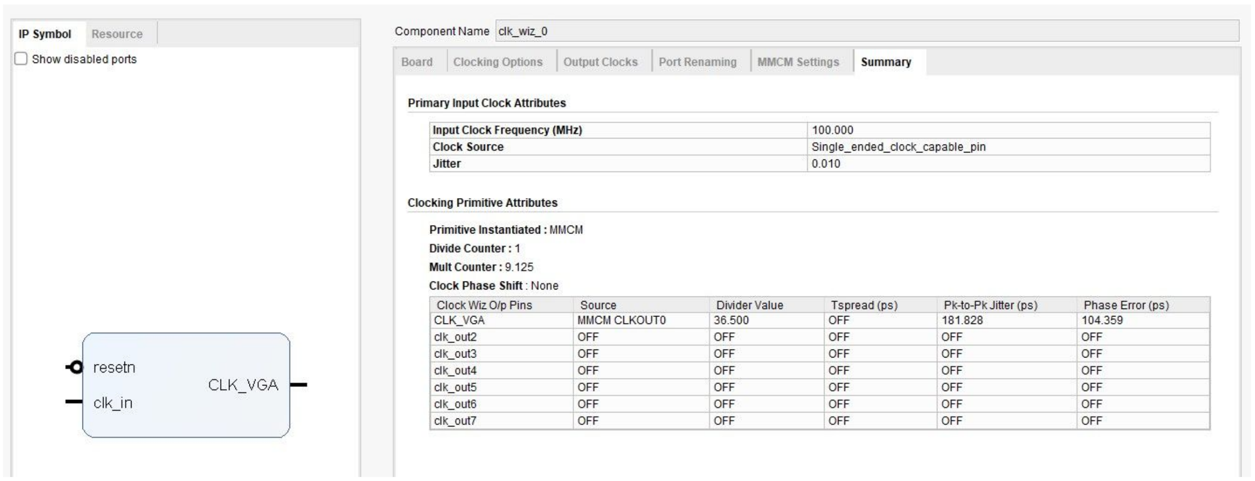
- requested_char_add : It’s used to set request char address.
- set_cursor_pos : It’s used to change cursor position.
- output_char_value : If new data is requested, after 2 clock cycles will be available on this port.

Clock wizard

General description

Clock wizard is internal IP of Vivado that allow to manage the clock. In this project is used in MMCP mode to generate clock for VGA at 25 MHz (optimal would be at 25.175 MHz).

The summary of clock wizard is shown in the figure below.



Component Name clk_wiz_0

Primary Input Clock Attributes

Input Clock Frequency (MHz)	100.000
Clock Source	Single_ended_clock_capable_pin
Jitter	0.010

Clocking Primitive Attributes

Primitive Instantiated : MMCM
Divide Counter : 1
Mult Counter : 9.125
Clock Phase Shift : None

Clock Wiz O/p Pins	Source	Divider Value	Tspread (ps)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
CLK_VGA	MMCM CLKOUT0	36.500	OFF	181.828	104.359
clk_out2	OFF	OFF	OFF	OFF	OFF
clk_out3	OFF	OFF	OFF	OFF	OFF
clk_out4	OFF	OFF	OFF	OFF	OFF
clk_out5	OFF	OFF	OFF	OFF	OFF
clk_out6	OFF	OFF	OFF	OFF	OFF
clk_out7	OFF	OFF	OFF	OFF	OFF

Port description

- clk_in : connected with system clock at 100 MHz.
- resetn : reset active low.
- clk_VGA : output clock for VGA management.

BRAM_0

General description

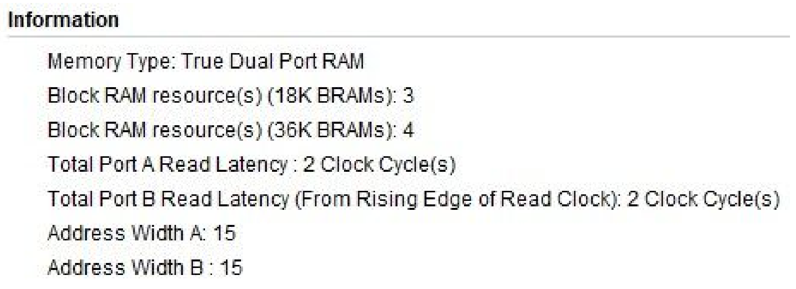
BRAM_0, that stay for Block of RAM is the entity that contains RAM memory. It's generated by internally Vivado IP, therefore some specifications are protected by copyright and cannot be published.

The entity has the following ports.

BRAM	
INPUT	OUTPUT
clka	douta
rsta	doutb
ena	rsta_busy
wea	rstb_busy
addra	
dina	
clkb	
rstb	
enb	
web	
addrb	
dinb	

The memory type is TRUE DUAL PORT RAM that means that there there are two different ports which could access at the same data also at the same time. In this project the PORTA is used to write into BRAM and PORTB is used to read. Write width is of 7 bits and the write depth is of 168000 bits. Since one character occupies 7 bits the stream could contain up to 24000 chars.

The following image rappreset BRAM summary.



Information

Memory Type: True Dual Port RAM
Block RAM resource(s) (18K BRAMs): 3
Block RAM resource(s) (36K BRAMs): 4
Total Port A Read Latency : 2 Clock Cycle(s)
Total Port B Read Latency (From Rising Edge of Read Clock): 2 Clock Cycle(s)
Address Width A: 15
Address Width B : 15

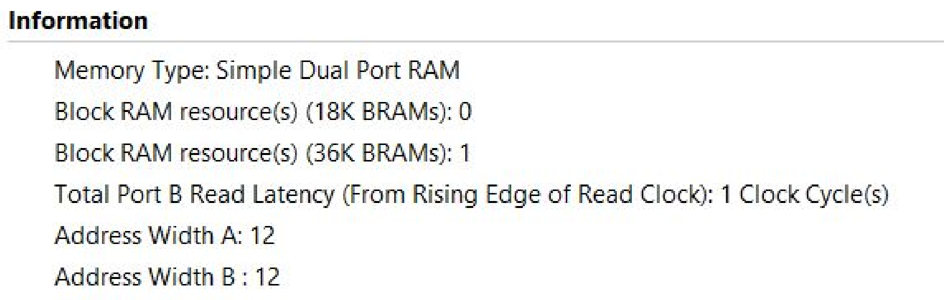
Port description

- clka, clkb : Are connected to the system clock.
- rsta, rstb : Are connected to reset (active low).
- ena, enb : Are used to enable the respective port.
- wea, web : If hight port is in writing mode, else in reading.
- addra, addrb : Indicate the port address.
- dina, dinb : Are the data input ports.
- douta, doutb : Are the data output ports.
- rsta_busy, rstb_busy : Are not used ports.

BRAM_1

General description

This BRAM is used to storage the current frame values, so it contains 680 x 480 = 2400 data. It is used in simple dual port RAM mode because the writing clock is at 100MHz and the reading clock is at 25.175 MHz, therefore it also performs a work of connection and synchronization between the two circuits with different clocks.



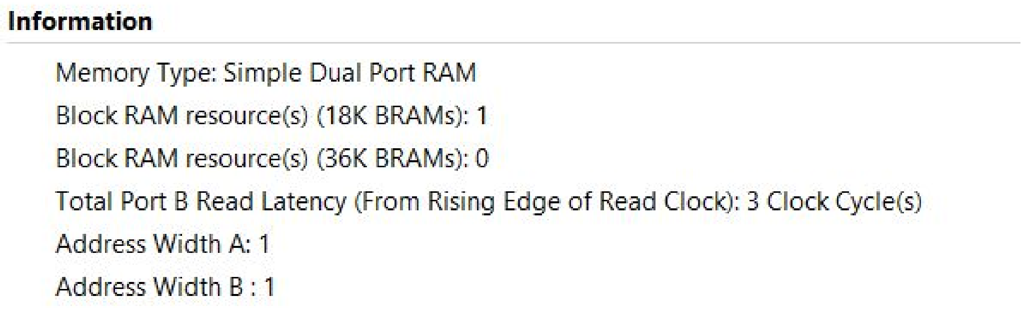
Information

Memory Type: Simple Dual Port RAM
Block RAM resource(s) (18K BRAMs): 0
Block RAM resource(s) (36K BRAMs): 1
Total Port B Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 12
Address Width B : 12

BRAM_2

General description

This BRAM is used only to synchronize a signal that is read at 100MHz and write at 25.175 MHz. The size is the smallest possible, in fact it's of 2 bits and only one is used. The BRAM is always turned on and the delay from the input to the output is of 3 clock cycles, this because if the input signal change exactly during the rising edge of the clock the flip flop could go to metastability status and it would take some clock cycles to get back to normal.

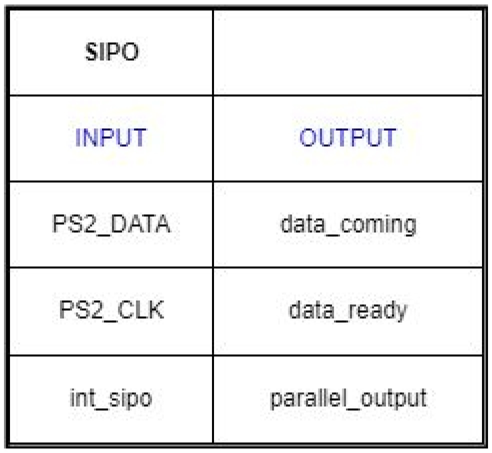

Information

Memory Type: Simple Dual Port RAM
Block RAM resource(s) (18K BRAMs): 1
Block RAM resource(s) (36K BRAMs): 0
Total Port B Read Latency (From Rising Edge of Read Clock): 3 Clock Cycle(s)
Address Width A: 1
Address Width B : 1

Sipo

General description

Sipo is the acronym of Serial Input Parallel Output, in fact this entity works as register. It takes serial data from PS2_DATA line which comes from keyboard. Keyboard PS2 protocol use 11-bit words that include a start bit (0), data byte (LSB first), odd parity, and stop bit (1). When all data arrives data_ready signal in set and you can read data from parallel_output signals, then you must reset sipo.



SIPO	
INPUT	OUTPUT
PS2_DATA	data_coming
PS2_CLK	data_ready
int_sipo	parallel_output

Port description

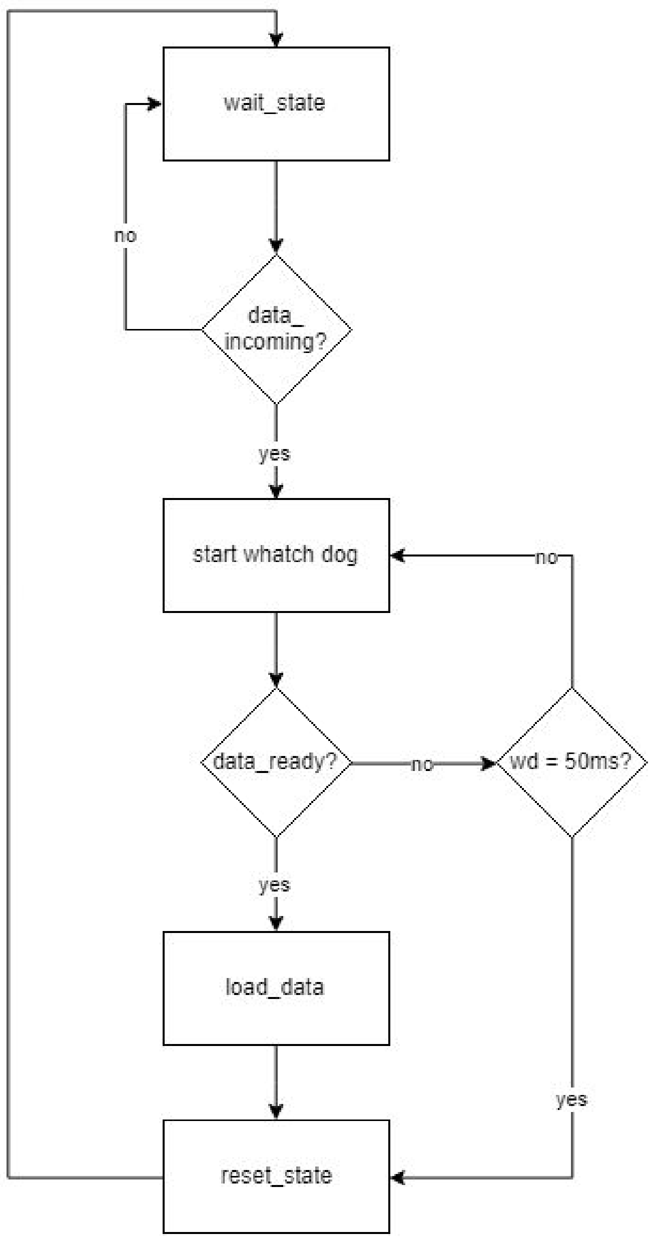
- PS2_DATA : PS2 data line.
- PS2_CLK : PS2 clock line.
- init_sipo : if low reset to 0 the register.

- data_coming : is high when data are coming from PS2_DATA port.
- data_ready : is set when all 11 bits are arrived.
- parallel_output : parallel data output.

Keyboard driver

General description

Keyboard driver entity has the task of managing the sipo entity, in particular there is a fsm that takes care of it. The following image represents the specifics of fsm.



Fsm description

- wait_state : In this state sipo initialization sipo is done. The state doesn't change until port "data_coming" is set.
- start_watch_dog : In this state a watchdog counter stars and is increased. The state change only if data from sipo has arrived or if watchdog counter reaches 50ms. In this case there was an error because the data took too long to arrive.
- load_data : makes data available from "data_out" ports and set new_data signal.
- reset_state : reset the sipo, the watchdog counter and new_data signal.

Port description

- PS2_DATA : PS2 data line.
- PS2_CLK : PS2 clock line.
- clock : It's connected with the system clock.
- global_reset : It's connected with the reset signal, active low.
- data_out : It's the data out port.
- new_data : Indicates that new data arrived. It's set for one clock cycle only.

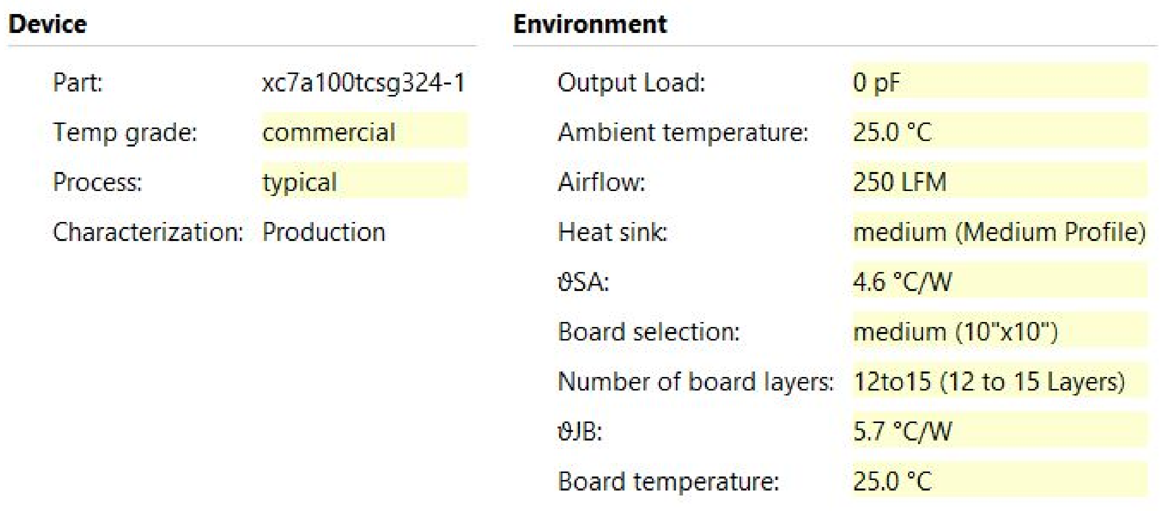
Seven segment driver

General description

This entity drives 7 segment display that print the ASCII code of keyboard pressed keys

Power summary

The graphics below indicates the power consumptions setting.



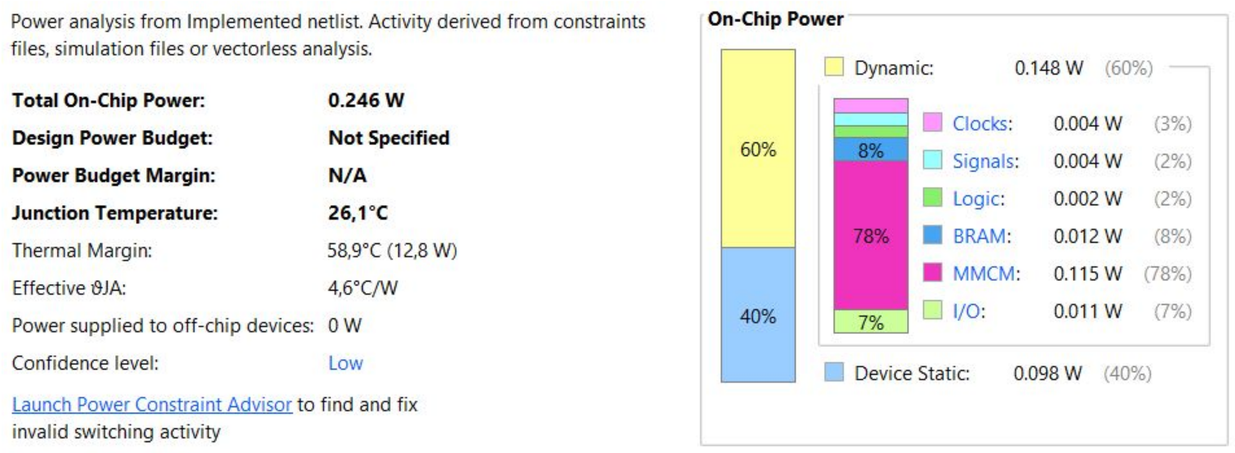
Device

Part:	xc7a100tcsg324-1
Temp grade:	commercial
Process:	typical
Characterization:	Production

Environment

Output Load:	0 pF
Ambient temperature:	25.0 °C
Airflow:	250 LFM
Heat sink:	medium (Medium Profile)
ϑSA:	4.6 °C/W
Board selection:	medium (10"x10")
Number of board layers:	12to15 (12 to 15 Layers)
ϑJB:	5.7 °C/W
Board temperature:	25.0 °C

These settings produced the following results.



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.246 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,1°C**
Thermal Margin:	58,9°C (12,8 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

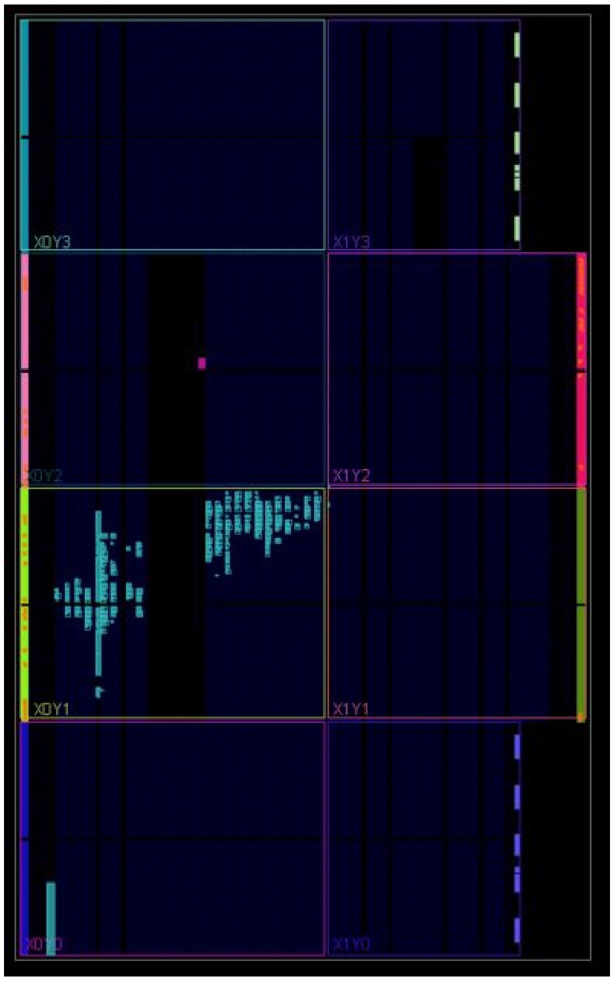
Launch Power Constraint Advisor to find and fix invalid switching activity

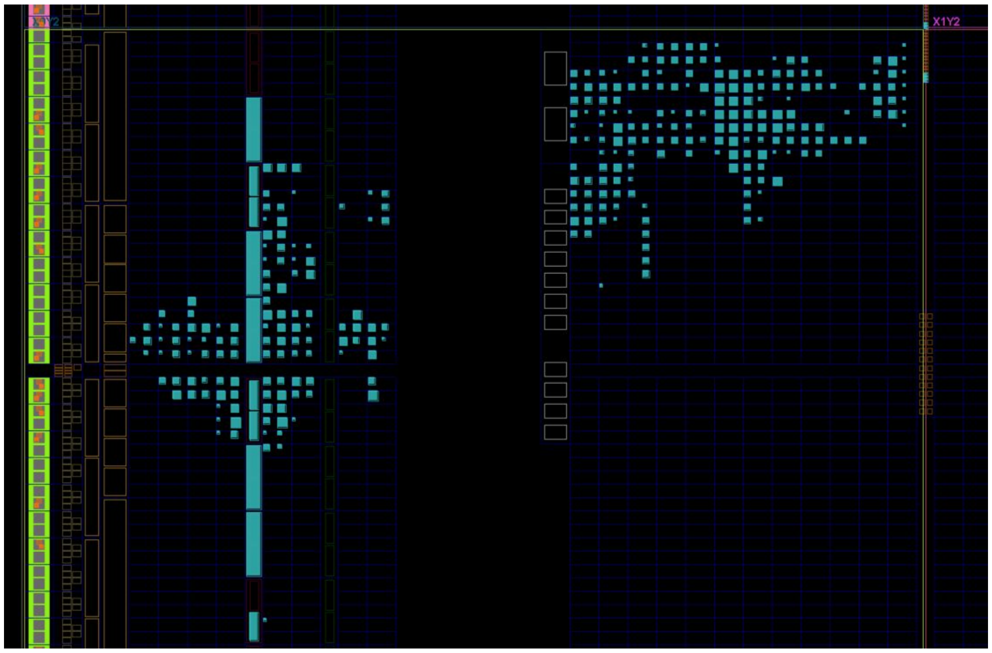
On-Chip Power

🟨	Dynamic:	0.148 W	(60%)
🟪	Clocks:	0.004 W	(3%)
🟦	Signals:	0.004 W	(2%)
🟩	Logic:	0.002 W	(2%)
🟦	BRAM:	0.012 W	(8%)
🟥	MMCM:	0.115 W	(78%)
🟩	I/O:	0.011 W	(7%)
🟦	Device Static:	0.098 W	(40%)

Implementation specifics

The following images represents the internal occupation on FPGA.





Total lookup table = 867

Total Flip Flop = 386

Total BRAMs = 7.5

Total URAM = 0

Total DSP count = 0

Worst negative slack = 2.937 ns

Total negative slack = 0 ns

Worst hold slack = 0.14 ns

Total hold slack = 0 ns

Worst pulse width slack = 3 ns

Total pulse width negative slack = 0 ns

Critical issues and conclusions

Conclusions

The program works correctly! In a future implementation/feature embedded monitor data will be taken in consideration such that the program is dynamic in regards of which max resolution the monitor can handle and could be displayed by the program.