ECE 368 Digital Design - Spring 2016

Laboratory #3: (100pts)

Lab date: February 19th,2016

Lab Report Due: Friday February 26th,2016

Overview and Objectives:

In this laboratory assignment, you will be building upon from your previous laboratory assignments and dive deeper into VHDL. You will be able to understand IBM's Personal System 2 (PS/2) protocol. You will learn how to convert from a PS/2 input signal directly into an ascii value. Next you will be learning the process of outputting data to a VGA display. With the VGA display, you will then learn how to create a buffer to hold characters to display on the VGA for a basic terminal.

Objectives for this lab:

- Understand the PS/2 Protocol and handle keyboard input.
- Learn how to convert from PS/2 **keycode** to **ascii** with lookup tables.
- Learn the process of testing designs in simulation.
- Experience in using a Finite State Machine (FSM) design concepts.
- Understand the process of displaying to a VGA display.
- Able to change the VGA display color on the screen.
- Learn how to create a memory buffer for a VGA display.
- Understand how to generate block RAM with Xilinx® logicore utility.
- Able to assemble previous concepts together.

Lab sections:

1. Introduction with a keyboard.
2. VGA Display concepts part 1.
3. VGA Display concepts part 2.

1. Introduction with a keyboard:

In this section of the lab you will go though the concepts of a PS/2. You will be able to build a PS/2 keyboard interface and have it output the ascii character code on the 8 LED's on the Nexys board. You will also be analyzing a test bench of the keyboard controller to gain a better understanding of the concept.

PS/2 Keyboard Concepts

The PS/2 controller described in this lab can be used as a bi-directional communication device to receive and transmit information between the keyboard and the PS/2 controller. For the basics on the PS/2 protocol, the protocol uses one clock line and one data line. The data can transmit back and forth between the keyboard and the controller in serial. When the clock line goes from high to low, it tells the controller there is data available. The controller will grab the data and wait for the next bit to arrive. The data is transmitted in a 11 bit packet with a start bit, 8 bit message, parity, and a stop bit. The controller will check if the data is valid though the parity bit.

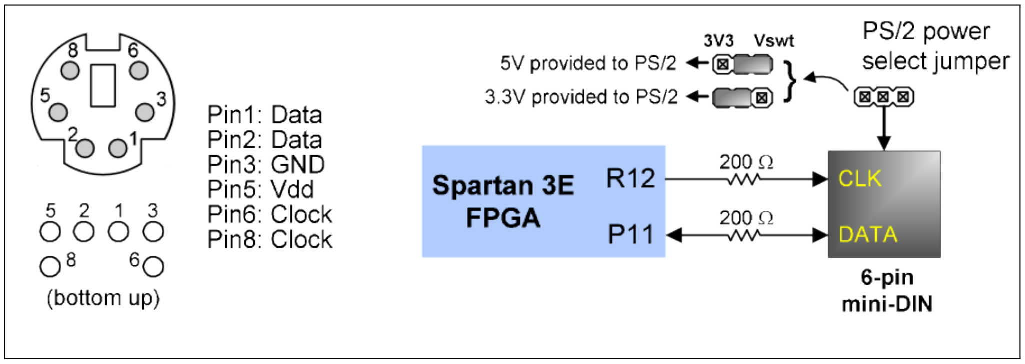


Figure 1: PS/2 Layout

The method in which the data being transmitted from the PS/2 keyboard is through scan codes. When you press any key on the keyboard, a unique code will be sent. The keyboard will send a **X"F0"** before the scan code when a key is released. When a extended key is pressed(arrow keys, right ctrl/Alt) on the keyboard, the keyboard will send out three scan codes. The scan codes go in the order **X"E0" X"F0"** followed by the scan code. For example if you press the letter **a**, the keyboard will send out **X"F0"**,**X"1C"**.

Command Codes:	Description:
X"F0"	Key has been released
X"E0"	Extended Key code is detected
X"FF"	Tell the keyboard to reset
X"ED"	Set keyboard LED Lock Indicator: Num(bit 0), Caps(bit 1), Scroll(bit 2)
X"FA"	Keyboard Acknowledge for X"ED"

Table 1: Common Keyboard Command Codes

One the next few pages are the tables for the keyboard scan codes and a ASCII Table for reference. For this lab, you will be testing the keyboard controller provided in this lab. A test bench is provided for you to gain a better understanding of what is happening in the code. I would strongly recommend analyzing the code in the **keycode_to_ascii.vhd** source file. In this file there are two lookup tables which will convert the keyboard scan codes directly to ASCII(upper/lower case). The file also houses a finite state machine. This state machine monitors the inputs from the keyboard driver and updates the state of the ASCII based on the order of keycode. The keyboard can only send data to the FPGA if both clock and data lines are set high which indicate the line is idle.

Keyboard Scan Codes

All values are in Hexadecimal.

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[	54	F0,54
B	32	F0,32	‘	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	F0,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24		5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARRW	E0,75	E0,F0,75
I	43	F0,4C	CAPS	58	F0,58	L ARRW	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	F0,12	D ARRW	E0,72	E0,F0,72
K	42	F0,42	L CTRL	14	F0,14	R ARRW	E0,74	E0,F0,74
L	4B	F0,4B	L GUI	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0,F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01	]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	’	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	58	F0,46	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14, 77,E1, F0,14, F0,77	-NONE-			

Table 2: Keyboard Scan Code table
(Source: computer-engineering.org)

ASCII Code Table

Decimal	Octal	Hex	Binary	Value	Description
000	000	00	00000000	NUL	Null Char.
001	001	01	00000001	SOH	Start of Header
002	002	02	00000010	STX	Start of Text
003	003	03	00000011	ETX	End of Text
004	004	04	00000100	EOT	End of Transmission
005	005	05	00000101	ENQ	Enquiry
006	006	06	00000110	ACK	Acknowledgment
007	007	07	00000111	BEL	Bell
008	010	08	00001000	BS	Backspace
009	011	09	00001001	HT	Horizontal Tab
010	012	0A	00001010	LF	Line Feed
011	013	0B	00001011	VT	Vertical Tab
012	014	0C	00001100	FF	Form Feed
013	015	0D	00001101	CR	Carriage Return
014	016	0E	00001110	SO	Shift Out
015	017	0F	00001111	SI	Shift In
016	020	10	00010000	DLE	Data Link Escape
017	021	11	00010001	DC1	Device Control 1 (XON)
018	022	12	00010010	DC2	Device Control 2
019	023	13	00010011	DC3	Device Control 3 (XOFF)
020	024	14	00010100	DC4	Device Control 4
021	025	15	00010101	NAK	Negative Acknowledgment
022	026	16	00010110	SYN	Synchronous Idle
023	027	17	00010111	ETB	End of Trans. Block
024	030	18	00011000	CAN	Cancel
025	031	19	00011001	EM	End of Medium
026	032	1A	00011010	SUB	Substitute
027	033	1B	00011011	ESC	Escape
028	034	1C	00011100	FS	File Separator
029	035	1D	00011101	GS	Group Separator
030	036	1E	00011110	RS	Request to Send (Rec. Sep.)
031	037	1F	00011111	US	Unit Separator
032	040	20	00100000	SP	Space
033	041	21	00100001	!	Exclamation Mark
034	042	22	00100010	”	Double Quote
035	043	23	00100011	#	Number Sign
036	044	24	00100100	$	Dollar Sign
037	045	25	00100101	%	Percent
038	046	26	00100110	&	Ampersand
039	047	27	00100111	’	Single Quote
040	050	28	00101000	(	Left/Opening parenthesis
041	051	29	00101001	)	Right/Closing Parenthesis
042	052	2A	00101010	*	Asterisk
043	053	2B	00101011	+	Plus
044	054	2C	00101100	,	comma
045	055	2D	00101101	-	Minus/Dash
046	056	2E	00101110	.	Dot
047	057	2F	00101111	/	Forward Slash
048	060	30	00110000	0	
049	061	31	00110001	1	
050	062	32	00110010	2	

051	063	33	00110011	3	
052	064	34	00110100	4	
053	065	35	00110101	5	
054	066	36	00110110	6	
055	067	37	00110111	7	
056	070	38	00111000	8	
057	071	39	00111001	9	
058	072	3A	00111010	:	Colon
059	073	3B	00111011	;	Semi-Colon
060	074	3C	00111100	<	Less Than
061	075	3D	00111101	=	Equal Sign
062	076	3E	00111110	>	Greater Than
063	077	3F	00111111	?	Question Mark
064	100	40	01000000	@	AT Symbol
065	101	41	01000001	A	
066	102	42	01000010	B	
067	103	43	01000011	C	
068	104	44	01000100	D	
069	105	45	01000101	E	
070	106	46	01000110	F	
071	107	47	01000111	G	
072	110	48	01001000	H	
073	111	49	01001001	I	
074	112	4A	01001010	J	
075	113	4B	01001011	K	
076	114	4C	01001100	L	
077	115	4D	01001101	M	
078	116	4E	01001110	N	
079	117	4F	01001111	O	
080	120	50	01010000	P	
081	121	51	01010001	Q	
082	122	52	01010010	R	
083	123	53	01010011	S	
084	124	54	01010100	T	
085	125	55	01010101	U	
086	126	56	01010110	V	
087	127	57	01010111	W	
088	130	58	01011000	X	
089	131	59	01011001	Y	
090	132	5A	01011010	Z	
091	133	5B	01011011	[	Left/Opening Bracket
092	134	5C	01011100		Back Slash
093	135	5D	01011101	]	Right/Closing Bracket
094	136	5E	01011110	ˆ	Caret/Circumflex
095	137	5F	01011111	_	Underscore
096	140	60	01100000	‘	
097	141	61	01100001	a	
098	142	62	01100010	b	
099	143	63	01100011	c	
100	144	64	01100100	d	
101	145	65	01100101	e	
102	146	66	01100110	f	
103	147	67	01100111	g	
104	150	68	01101000	h	

105	151	69	01101001	i	
106	152	6A	01101010	j	
107	153	6B	01101011	k	
108	154	6C	01101100	l	
109	155	6D	01101101	m	
110	156	6E	01101110	n	
111	157	6F	01101111	o	
112	160	70	01110000	p	
113	161	71	01110001	q	
114	162	72	01110010	r	
115	163	73	01110011	s	
116	164	74	01110100	t	
117	165	75	01110101	u	
118	166	76	01110110	v	
119	167	77	01110111	w	
120	170	78	01111000	x	
121	171	79	01111001	y	
122	172	7A	01111010	z	
123	173	7B	01111011	{	Left/Opening Brace
124	174	7C	01111100	—	Vertical Bar
125	175	7D	01111101	}	Right/Closing Brace
126	176	7E	01111110		Tilde
127	177	7F	01111111	DEL	Delete

Table 3: Keyboard Scan Code table
(Source: computer-engineering.org)

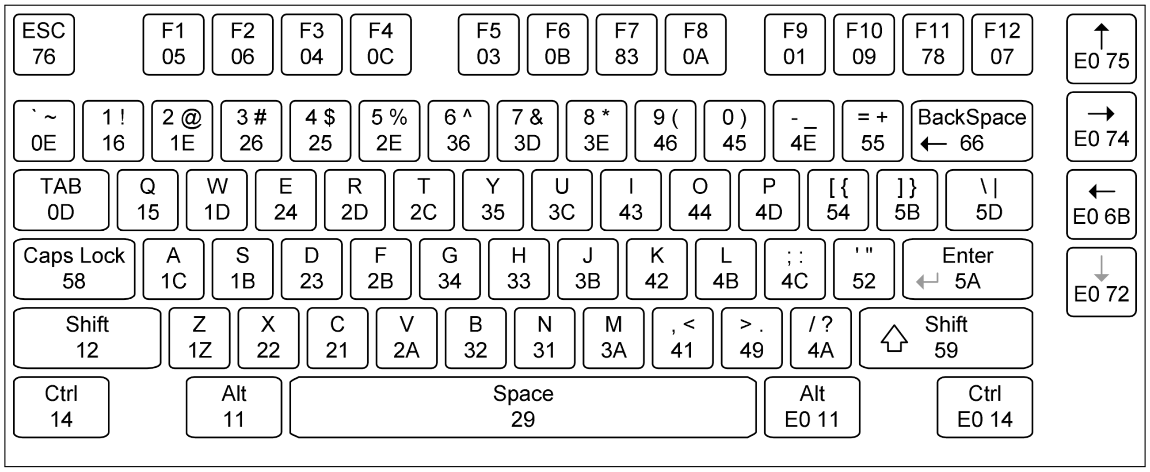


Figure 2: PS/2 Keyboard scan codes

Breakdown of the Keyboard Controller

With the experience you gain from the previous lab, you now know how to setup a project and grab the vhd files. For the Keyboard Controller, you need to get all six files(vhd,ucf) in the Keyboard folder for this section of the lab. When your looking at each file you can see what they are designed for. I recommend reading the files from the top down to give a perspective of the device. You don't need to look at the PS2 Driver but everything else is a must.

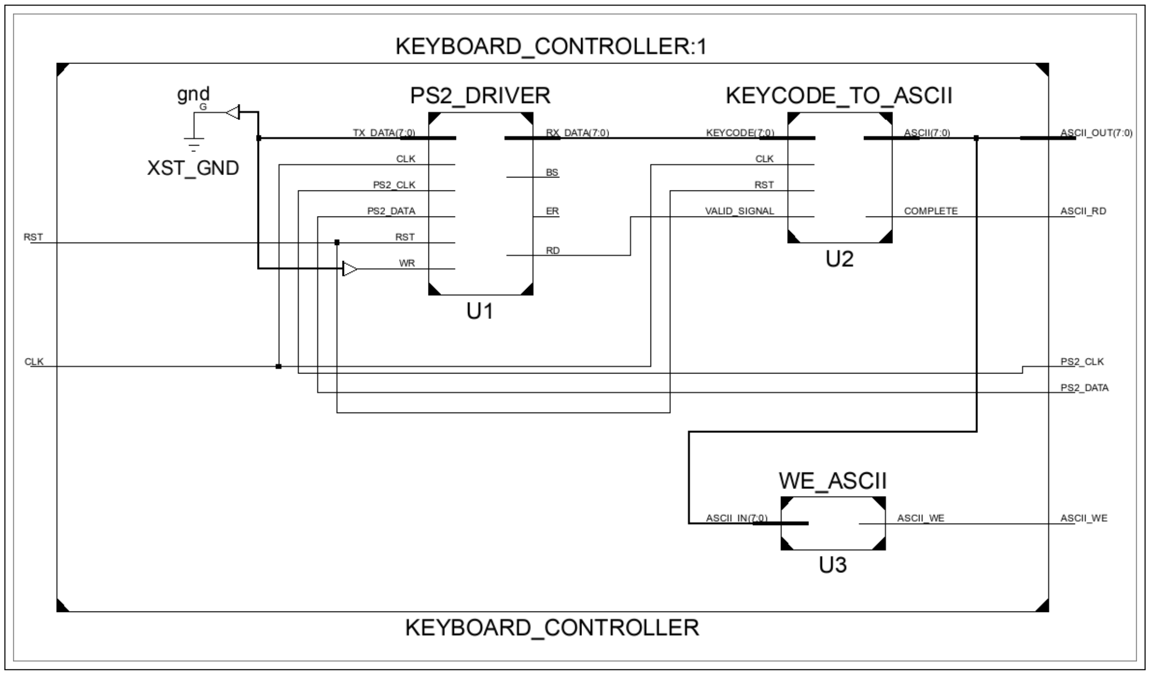


Figure 3: Keyboard Controller RTL Diagram

Testing the Keyboard Controller

After you looked at each file for the keyboard controller. You are now ready to test it out. First you should simulate the design to see how it outputs the ASCII data. There is already a test bench create for this test. All you have to do is have the simulation go from **1us** to **2ms**. This can be done on the top right bar on the ISim utility. Go and look at the simulation and capture the wave form when the ASCII value updates to the letter "**a**". When you zoom into the simulation, you can see **ASCII_RD** rise high for one clock cycle. This line is used to tell other devices that there is a new ASCII code on the bus.

After you simulated the keyboard controller, you will now need to build and flash it on the Neyxs 2 board in order to test it out. Just like in the real world, you will be needing to test you device throughly. This is normally done with an hardware acceptance test plan(ATP). For this section of the lab you are going to be creating a test plan for the keyboard controller. You will be testing the basic ASCII codes(a-z,A-Z,0-9) to verify that they are outputting correctly. In the keyboard folder there is a Test Plan document to help with testing the basic inputs and outputs of the device. You will be needing to test each output to see what is the result and trace it to the cause of the problem.

Keyboard Objectives

For the Keyboard Controller Section you will be asked several questions and demonstrate a working keyboard controller to the TA or Professor. As stated before, write down what you did during the lab so you can easily put them into your lab report.

1. What break codes does the keyboard send when you type **key**?
2. What are lookup tables used in the **keycode_to_ascii.vhd** file?
3. What is the state of the PS2 clock and data to indicate a idle line?
4. During your hardware acceptance test plan on the Keyboard, what keys were incorrect?
5. Obtain the printouts of your simulation results in you lab report.
6. Demonstrate the keyboard controller with correct outputs.

2. VGA Display concepts part 1:

In this section of the lab you will be learning about the basic concepts of a VGA Display. On the Nexys 2 dev board you have a VGA port with 8-bit color plus two sync signals for a horizontal sync and a vertical sync. With the 8-bit color you are able to create 256 different types of color. The pin layout for the VGA can be found on the next figure and in the ucf listed in the VGA Part 1 folder.

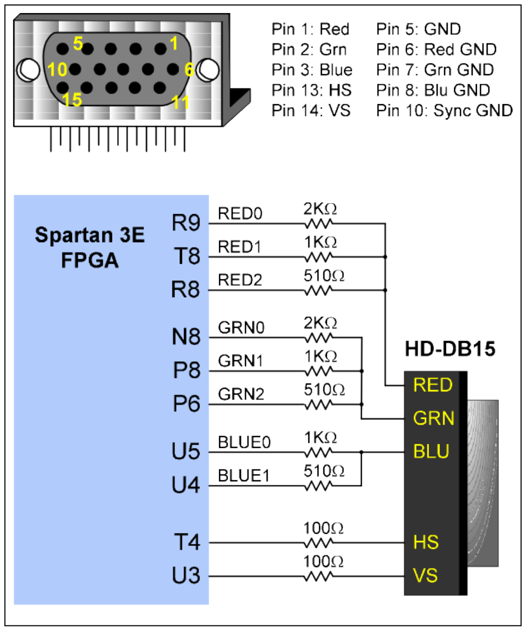


Figure 4: VGA Port Pin Layout (source: digilent inc.)

VGA Concepts

For a VGA display, signal timing is extremely vital in order to display each pixel to the screen orderly. The VGA display system consist of a horizontal and vertical sync. Both are used to setup the display surface. Further detail on this can be found in **VGA_port_discussion.ppt** under the class dir.

Testing the VGA RGB Displayer

For this section you will be building the RGB Color project. After you have built and flash the Nexys 2. Test out the displayer unit and change the color to each group members favorite. Save your color code so you can implement it in the VGA RGB file when you are on the next section.

VGA Part 1: Objectives

1. Demonstrate to the TA or Professor the RGB display terminal
2. What is the resolution of the VGA display in the RGB display terminal?
3. What is the purpose of the **hcount** and **vcount** in the VGA controller?

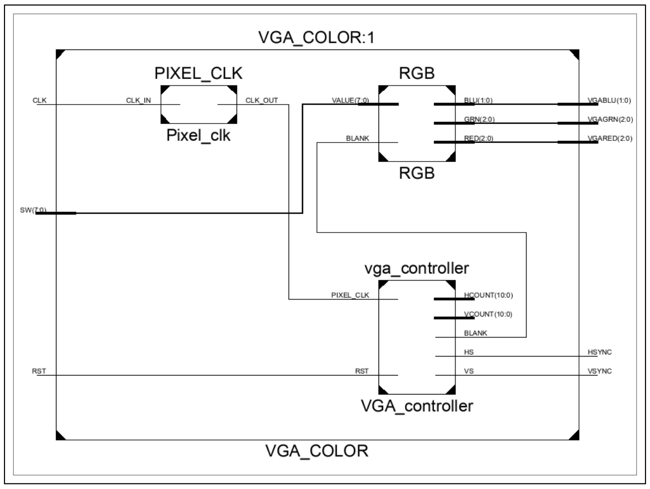


Figure 5: VGA Color RTL Diagram

3. VGA Display concepts part 2:

In this section, you will be building from the previous section on how a VGA display works. You will be using this concept and the example code provided to create a VGA Display with a ASCII Terminal. During this section you will be learning how to generate memory through Xilinx® logicore utility. This will allow you to use the memory provided on the FPGA as a buffer for you terminal.

VGA Setup

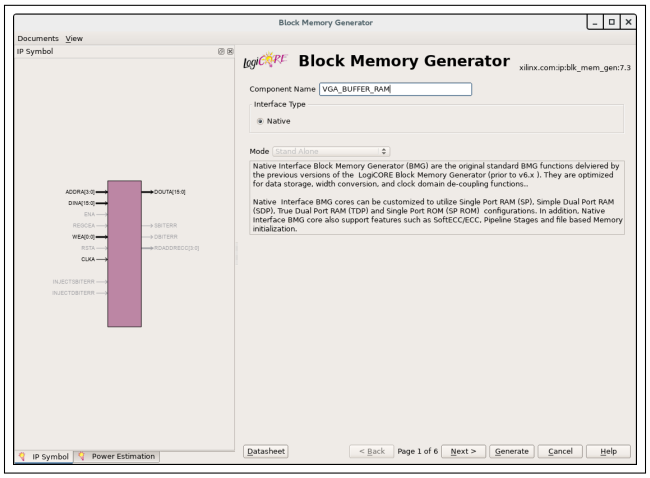
Create a new project or empty the files in the current file. Include all the files in the VGA Part 2 folder and the Keyboard folder. When you open up the design view, you will notice a file missing(U7). U7 is the memory for the VGA. This file you will be needing to generate through logicore in the next subsection.

Generate Memory

To generate the memory for the VGA display, you first need to open up logicore. This can be done under **Tools ->Core Generator** or Alt+t,c. When the Core Generator opens up, you will now need to create a new project.

1. Enter the parameters for the new project:
   Family : **Spartan3E**
   Device : **XC3S500E**
   Package : **FG320**
   Speed Grade : **-4**

The next part you need to do is generate the memory with the **Block Memory Generator**. This can be done by going in the IP Catalog and selecting **Memories & Storage Elements ->RAMSs & ROMs ->Block Memory Generator**. For the Memory you will need to name it **VGA_BUFFER_RAM**. The memory will be a simple Dual Port Ram with **8** bit width while having **4096** depth. On page 4 you will need to initialize the memory. This can be done by loading the coe file in the VGA Part 2 folder. I recommend filling in the remaining memory locations with spaces(**X"20"**). Follow the next four figures to double check the configurations. After you finished with the configuration, the core generator will generate the xco files you need for the project. Don't forget to check your design properties for the preferred language **VHDL**.



Block Memory Generator xilinx.com:ip:blk_mem_gen:7.3

Component Name VGA_BUFFER_RAM

Interface Type

[✓ Native]

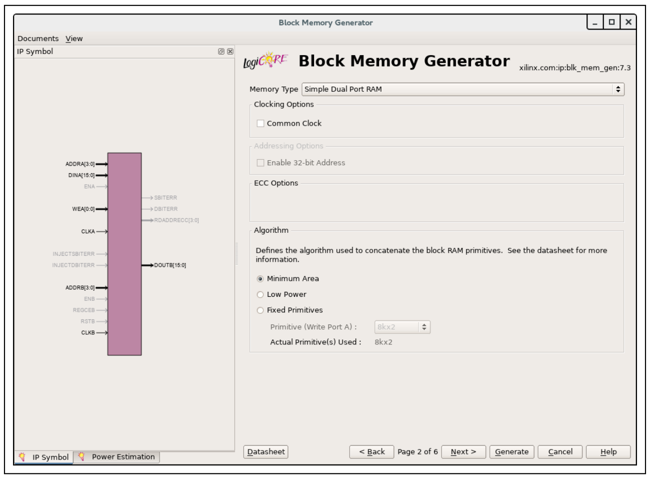
Mode Stand Alone

Native Interface Block Memory Generator (BMG) are the original standard BMG functions delviered by the previous versions of the LogiCORE Block Memory Generator (prior to v6.x). They are optimized for data storage, width conversion, and clock domain de-coupling functions..

Native Interface BMG cores can be customized to utilize Single Port RAM (SP), Simple Dual Port RAM (SDP), True Dual Port RAM (TDP) and Single Port ROM (SP ROM) configurations. In addition, Native Interface BMG core also support features such as SoftECC/ECC, Pipeline Stages and file based Memory initialization.

Page 1 of 6

Figure 6: Generate Memory Wizard Page 1



Block Memory Generator xilinx.com:ip:blk_mem_gen:7.3

Memory Type Simple Dual Port RAM

Clocking Options

☐ Common Clock

Addressing Options

☐ Enable 32-bit Address

ECC Options

Algorithm

Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.

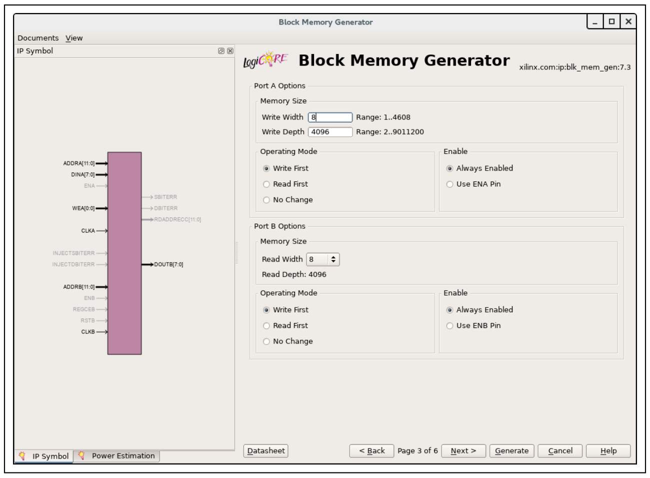
[✓ Minimum Area] Low Power Fixed Primitives

Primitive (Write Port A) : 8kx2

Actual Primitive(s) Used : 8kx2

Page 2 of 6

Figure 7: Generate Memory Wizard Page 2



Block Memory Generator xilinx.com:ip:blk_mem_gen:7.3

Port A Options

Memory Size

Write Width 8 Range: 1..4608

Write Depth 4096 Range: 2..9011200

Operating Mode: [✓ Write First] Read First No Change

Enable: [✓ Always Enabled] Use ENA Pin

Port B Options

Memory Size

Read Width 8

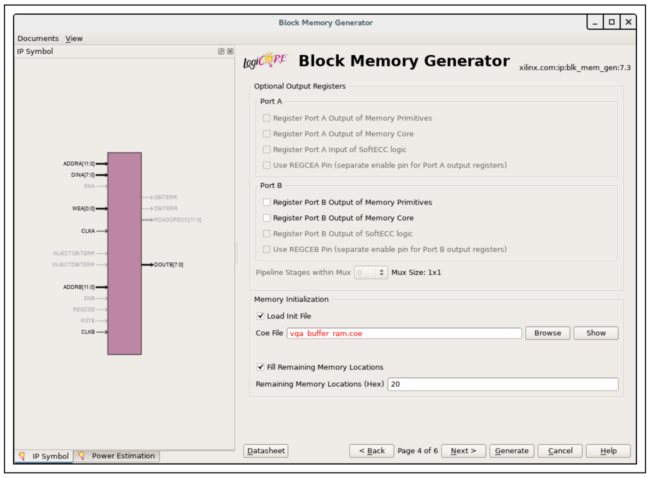
Read Depth: 4096

Operating Mode: [✓ Write First] Read First No Change

Enable: [✓ Always Enabled] Use ENB Pin

Page 3 of 6

Figure 8: Generate Memory Wizard Page 3



Block Memory Generator xilinx.com:ip:blk_mem_gen:7.3

Optional Output Registers

Port A

- [ ] Register Port A Output of Memory Primitives
- [ ] Register Port A Output of Memory Core
- [ ] Register Port A Input of SoftECC logic
- [ ] Use REGCEA Pin (separate enable pin for Port A output registers)

Port B

- [ ] Register Port B Output of Memory Primitives
- [ ] Register Port B Output of Memory Core
- [ ] Register Port B Output of SoftECC logic
- [ ] Use REGCEB Pin (separate enable pin for Port B output registers)

Pipeline Stages within Mux 0 Mux Size: 1x1

Memory Initialization

- [✓] Load Init File

Coe File vga_buffer_ram.coe

- [✓] Fill Remaining Memory Locations

Remaining Memory Locations (Hex) 20

Page 4 of 6

Figure 9: Generate Memory Wizard Page 4

Testing the VGA Display

After you generate the Logicore file for the memory. The next step you need to do is test the VGA display. First you need to generate the bit file and flash it to the Nexys 2. After you flash it, test how the terminal works. If you want, change the display controls to suite you.

VGA Part 2: Objectives

1. Demonstrate to the TA or Professor the ASCII display terminal
2. Why is a 8 to 1 MUX needed?
3. What is the reason for a blinker?

Lab 3 Grade

Section	Description	Value	Score
Section 1		**-30-**	
1.1. Keyboard Demo	Demonstrate Keyboard Controller	20	
1.2. Questions	Answer Questions about the Keyboard Controller	10	
Section 2		**-20-**	
2.1. VGA Demo 1	Demonstrate RGB VGA Display	10	
2.2. Questions	Answer Questions about the VGA Display	10	
Section 3		**-10-**	
3.1. VGA Demo 2	Demonstrate VGA Terminal Display	5	
3.2. Questions	Answer Questions about the VGA Terminal Display	5	
Lab Report		**-50-**	
Total		100	

Table 4: Lab Grade Breakdown Table