

Name: _____ ID: _____ Start Date: Thursday, August 19, 2021
Due Date: Saturday, August 21, 2021

Software and Hardware Co-Design with Zybo, Summer 2021 HUST

Lab #9 VGA, Image Creation and Boot Loader on Zybo

This is an individual lab. Each student must perform it and demonstrate this lab to obtain credit for it. Late lab submission will be accepted with a grade reduction of 10% for each day that it is late.

The main objective of this project is to implement VGA display with Zybo and axi IPs. The idea of this project is from the Zybo Base System Design from Digilent Inc. Another task of this lab is to create a boot loader for QSPI.

1 Available Source Programs and Documents

zybo_base_system.zip is provided by the instructor. It is also available under Design Resources from this website: <https://reference.digilentinc.com/zybo:zybo>. It contains both hardware modules as well as example software programs for various applications.

Lab 5 Configuration and Booting, Advanced Embedded System Design on Zynq using Vivado from Xilinx University Program on how to configure a boot loader from SD card. This is a tutorial to implement Part 2 of this lab.

2 Objectives

- 1) Implement VGA part of Zybo Base System Design from Digilent.
- 2) Create and display one image of those from Tweetable Mathematical Art competition. (<http://codegolf.stackexchange.com/questions/35569/tweetable-mathematical-art>.)
- 3) Create and display one new image of your own creation. It could be a modification of those from the Tweetable mathematical Art. However, you are encouraged to be more creative.
- 4) Create the boot image of the above system, store it in the QSPI flash memory, start the above system from the QSPI memory.

3 Demonstration and Report by Saturday, August 21, 2021

Submit a pdf copy of your image displaying code. Demonstrate your system can be booted from the QSPI flash memory and display at least two images, one of them is your own creation.

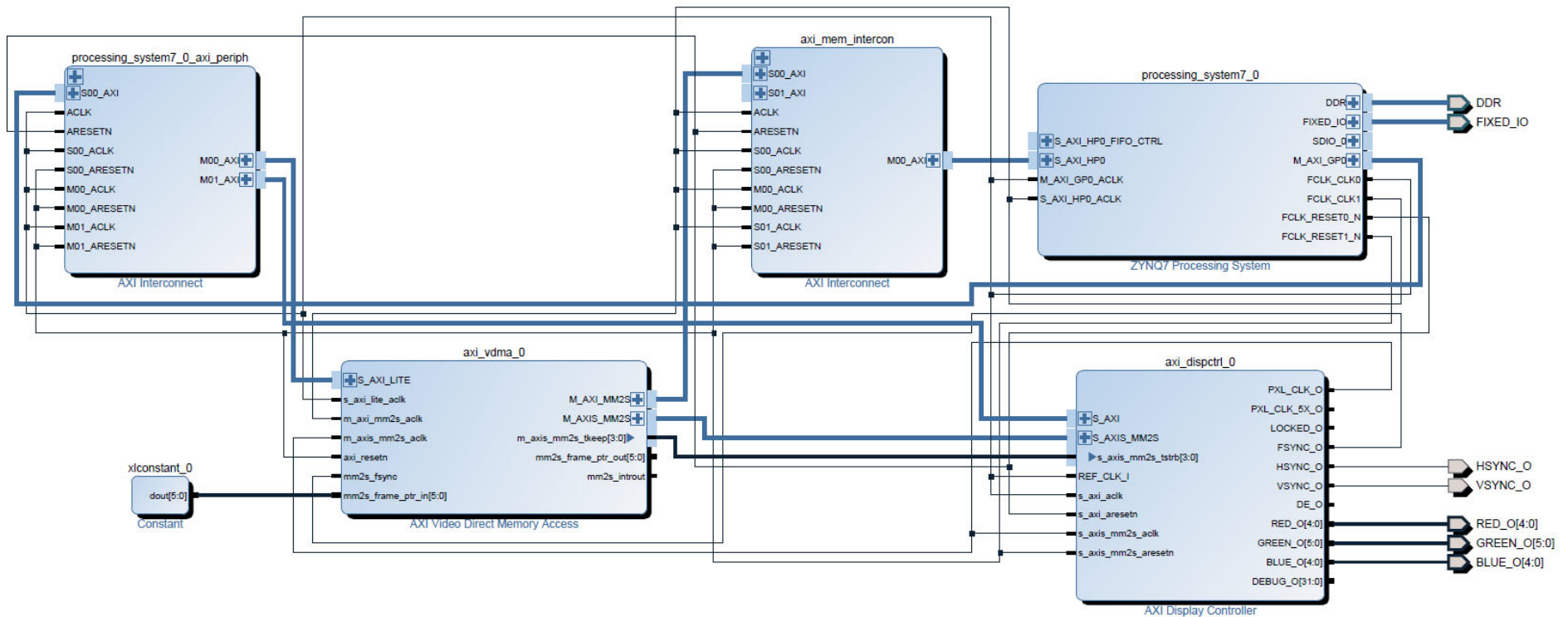
4 Resources

- 1) zybo_base_system.zip, Zybo Base System Design, <https://reference.digilentinc.com/zybo:zybo>.
- 2) Lab 5 Configuration and Booting, Advanced Embedded System Design on Zynq using Vivado, <http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-advanced-design-zynq.html>.
- 3) AXI Video Direct Memory Access v6.2, LogiCORE IP Product Guide, Vivado Design Suite, PG020, November 18, 2015, http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf.
- 4) Tweetable Mathematical Art, <http://codegolf.stackexchange.com/questions/35569/tweetable-mathematical-art>.

5 The Final Block Design Diagram for Part 1.

Following this block design diagram to create our block design for Part 1 of this lab. If the steps do not agree with this block design diagram, assume this block design is correct. A larger pdf copy of this block design diagram is available for you to see more details.

You may get a lot of warnings during the process to build a bit stream file, you can ignore most of them.



Part 1: A Tutorial on How to Implement VGA part of the Zybo Base System Design

1 Create a Vivado RTL project for VGA Demonstration

Create a Vivado RTL project, entitled lab9VGAdemoJJS where JJS is your name initials, and choose Zybo board. Create a block design of the same name: lab9VGAdemoJJS.

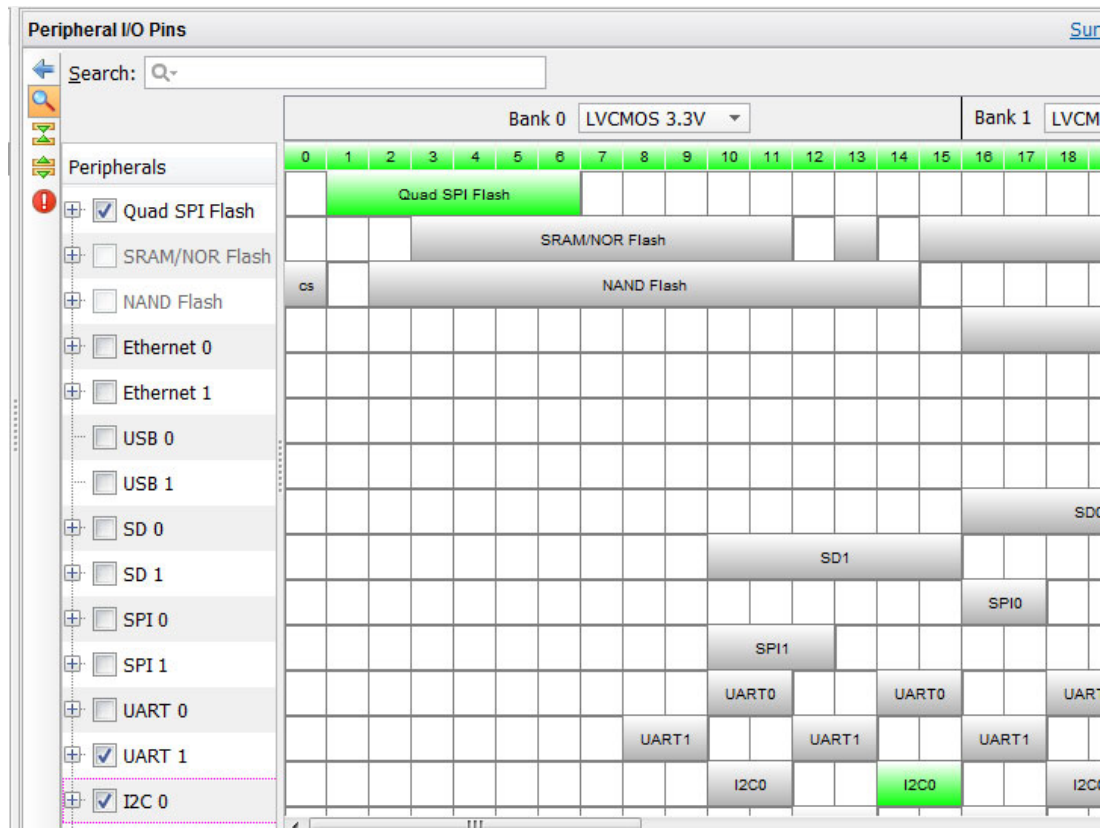
To disconnect a pin from a net, open the IP block where the pin is from to select the pin. Right click on the pin to choose “Disconnect Pin”. Press “Esc” key to deselect a pin.

1.1 Add Zynq7 Processing System

Add ZYNQ7 Processing System and “Run Block Automation” before anything else to make sure that UART connection for SDK to work. Set the following configurations.

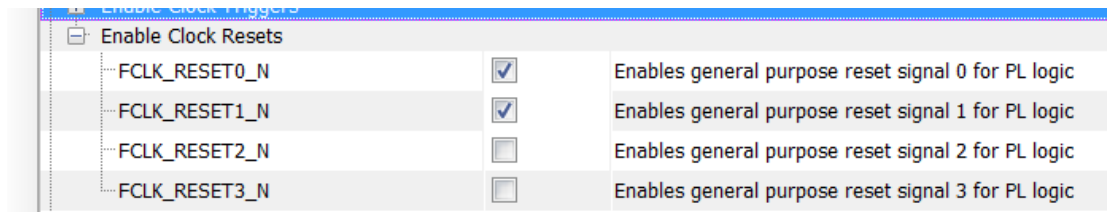
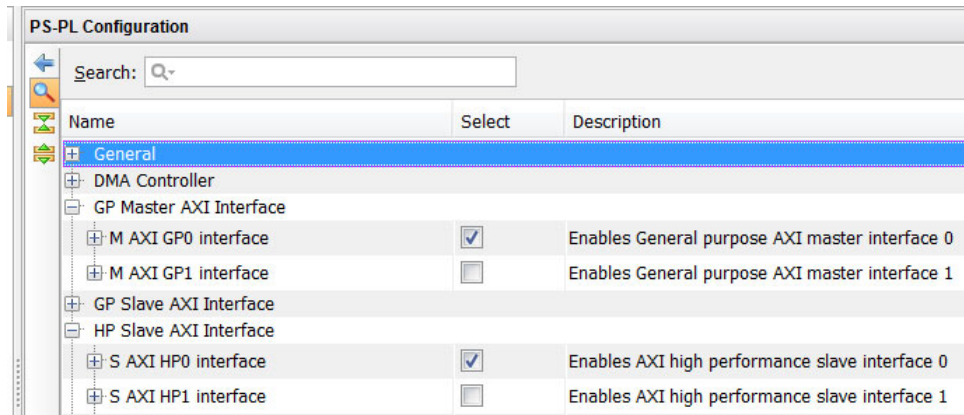
1.1.1 Peripheral IO Pins

Keep Quad SPI Flash, UART 1 and I2C 0 only. SPI module is active so that the final project loadable image can be stored on the flash memory. Choose M14 and M15 for I2C 0. This I2C port is used by the audio demo of the Digilent demo program. Although we will not demonstrate the audio function for this lab, keep this port will make it easier to compile the demo program without revising the demo program to remove the audio demo.



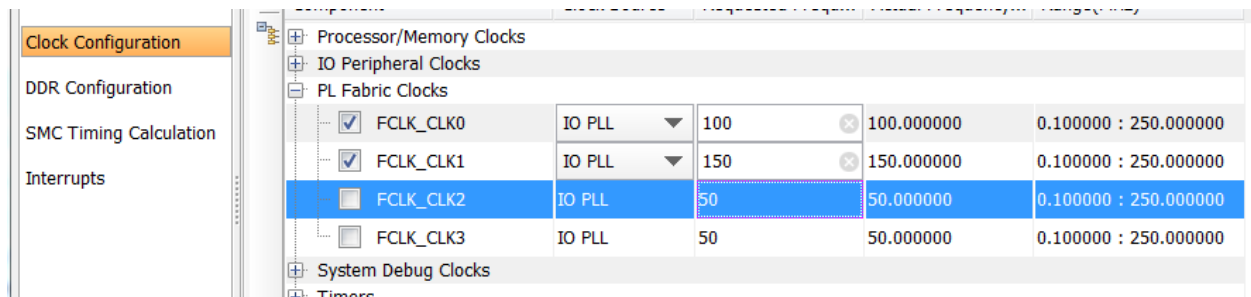
1.1.2 PS-PL Configuration

Enable M AXI GP0, general purpose master interface 0 and S AXI HP0, high performance slave interface 0. In “General” menu, enable FCLK_RESET_0_N and FCLK_RESET_1_N.

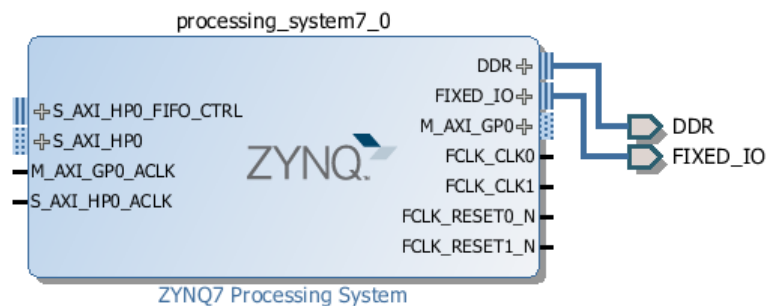


1.1.3 Clock Configuration

Enable FCLK_CLK0, FCLK_CLK01 and FCLK_CLK02 as following. Set FCLK_CLK0 to 100MHz, FCLK_CLK1 to 150MHz



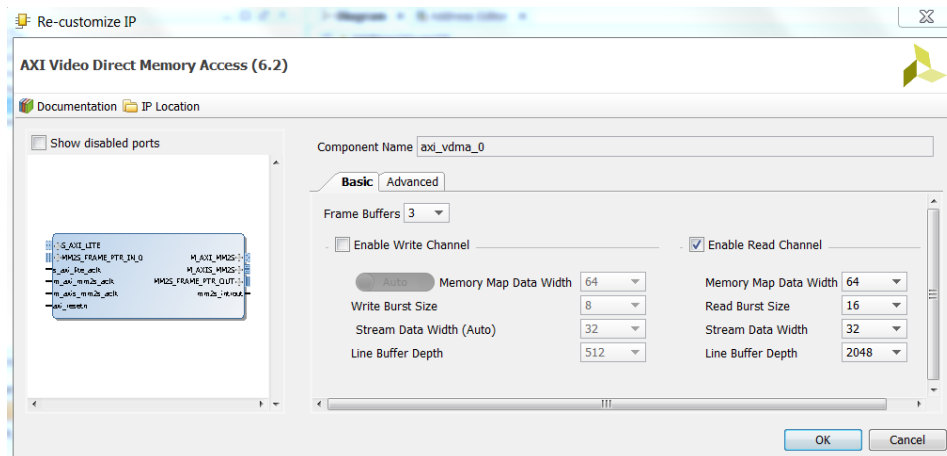
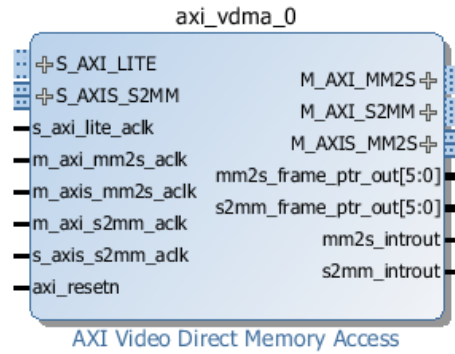
Your ZYNQ system should look like the following.



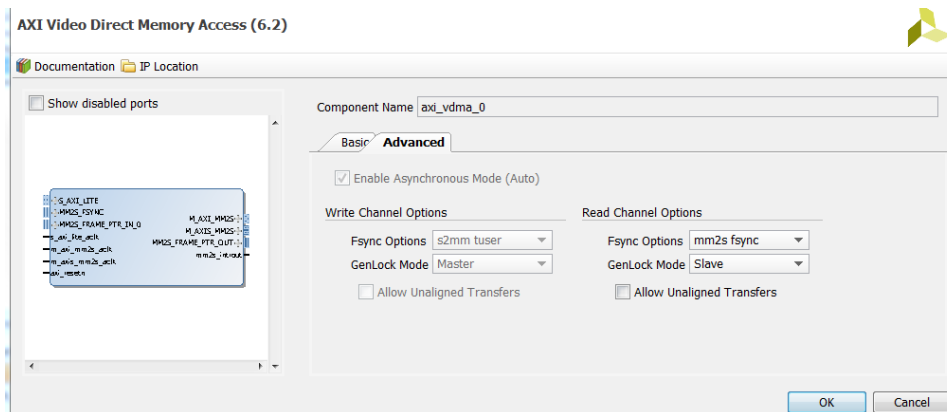
1.2 Add an AXI Video DMA IP

Add axi video direct memory access IP as follows. Disable Write Channel and set Read Channel as shown.

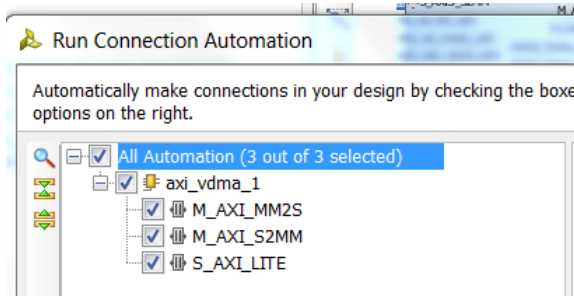
Basic Settings.



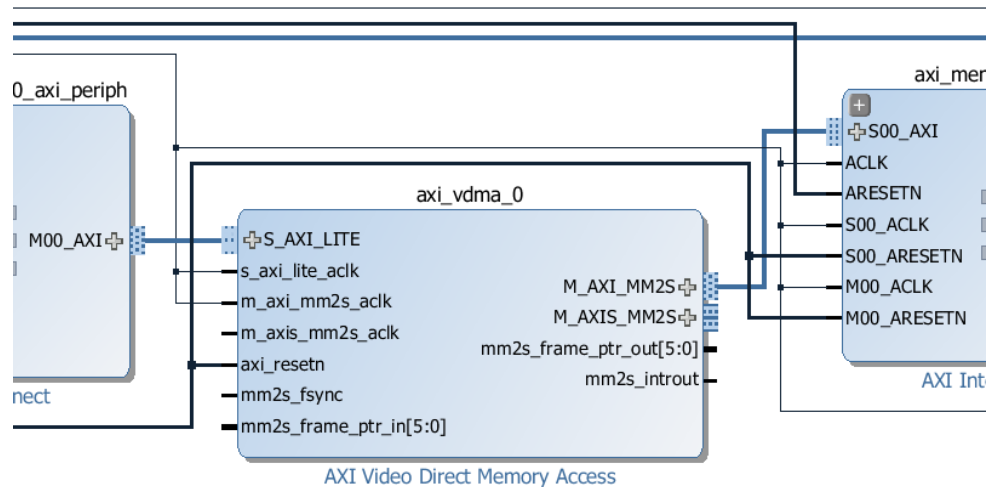
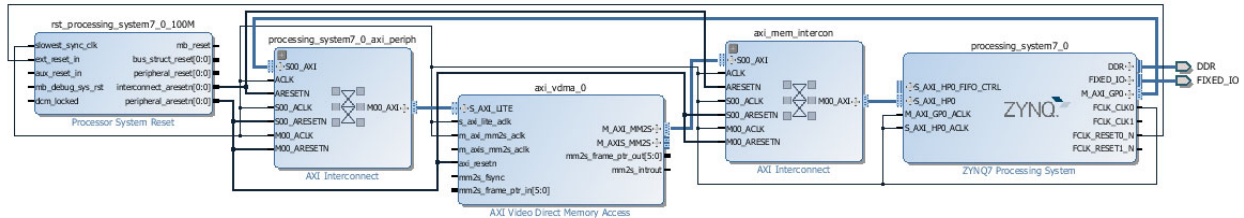
Advanced Settings.



Run “Run Connection Automation” and choose to connect all pins. The block design should look like the following.



(This does not match the steps above it)



Make sure the memory address assignment for this module is done as shown below under Address Editor. If the address is not assigned, right click the address table and select auto assign address.

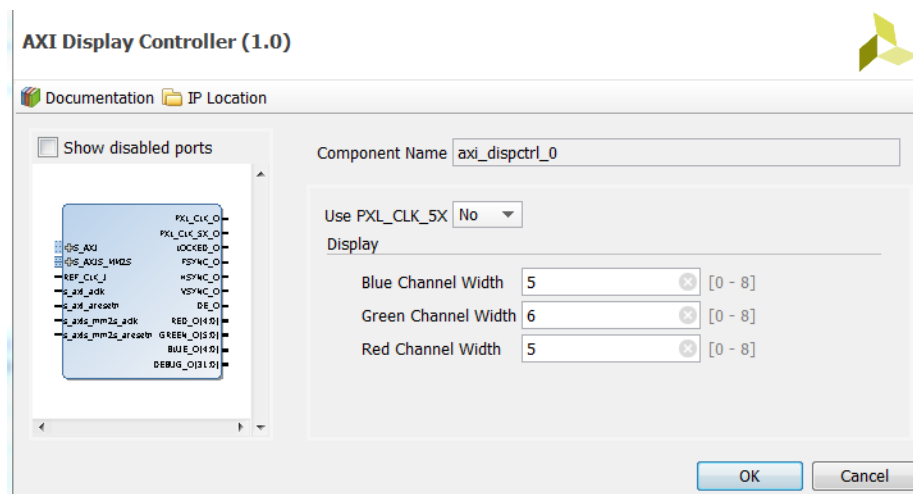
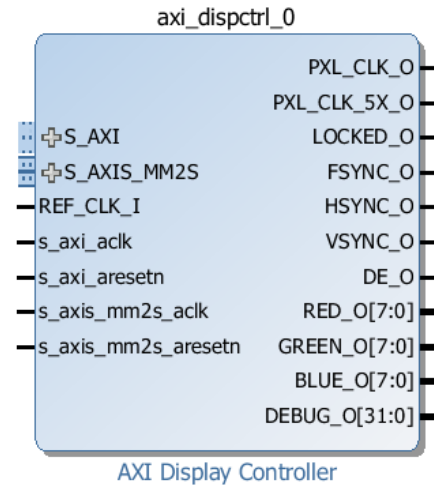
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_vdma_0	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
axi_dispctrl_0	S_AXI	S_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
axi_vdma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_L...	0x0000_0000	512M	0x1FFF_FFFF
Data_S2MM (32 address bits : 4G)					
Data_SG (32 address bits : 4G)					

1.3 Add an axi display controller IP from Digilent Inc.

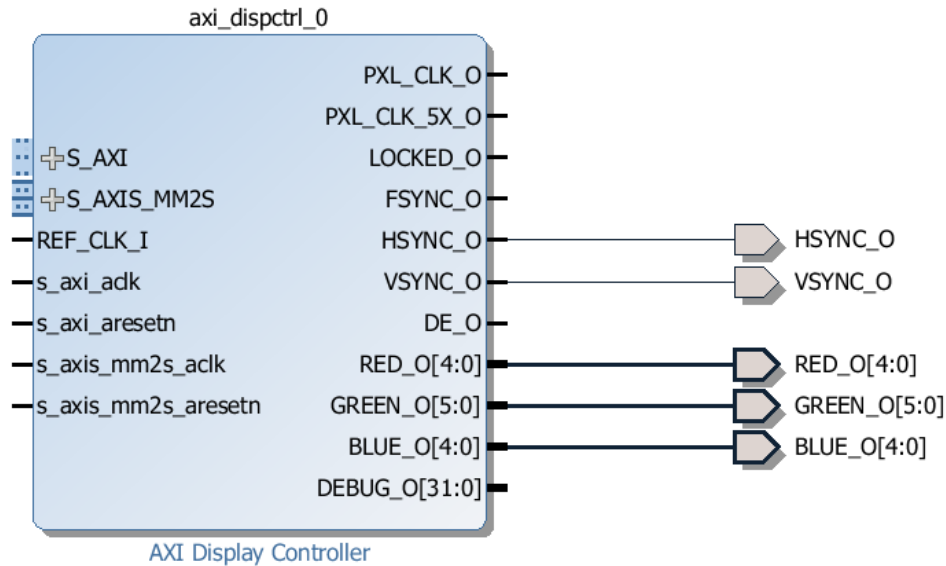
The axi display controller IP is called axi_dispctrl_1.0 from Digilent Inc. It is available as part of Zybo Base Design zip file that can be downloaded from Digilent. The IP is in a folder called lib folder under zybo_base_system\zybo_base_system\source\vivado\hw. This lib folder needs to be added to the IP Repository of your project. You may want to copy this lib folder in your project folder for easy access.

In Flow Navigator, select Project Setting -> IP -> IP manager. Add the “lib” folder which contain the AXI Display Controller IP. Add AXI Display Controller to the block diagram as follows.

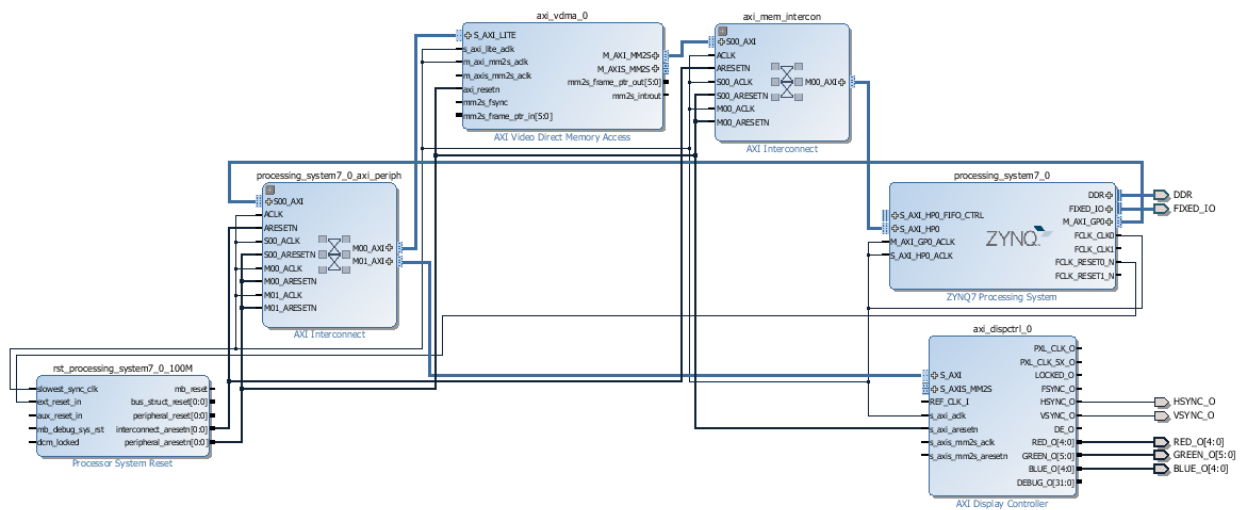
Double click the IP and change red channel to 5 bits, green channel to 6 bits and blue channel to 5 bits.



Make the RGB pins and h sync and v sync pins external. The block should like this.

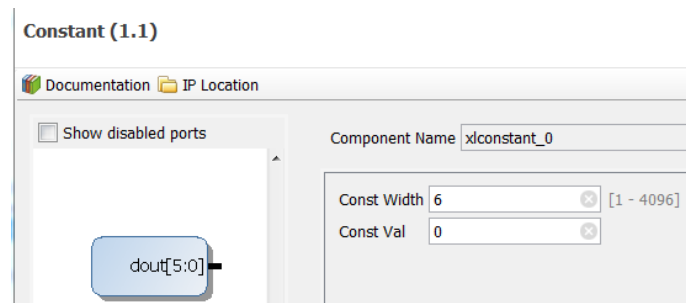


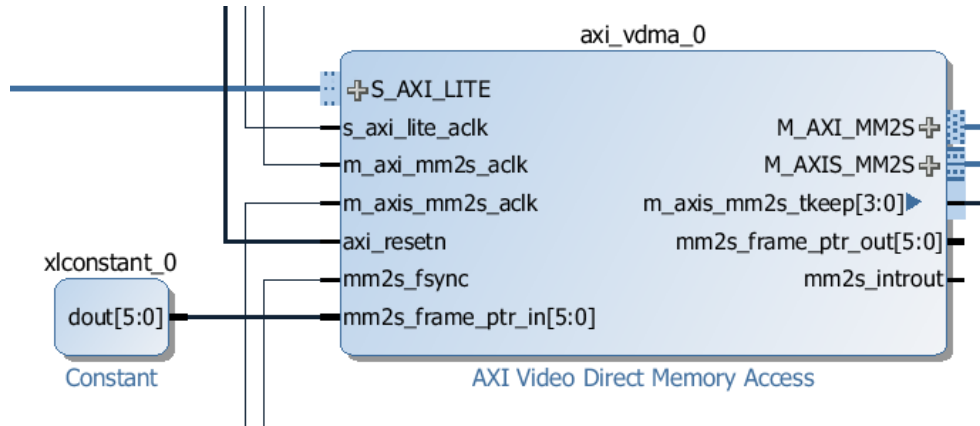
Run “Run Connection Automation” to obtain the following block design.



1.4 Add a 6-bit Constant Number IP

Add a constant IP, called Constant. Make it 6-bit in width and constant zero value, as the starting address of the frame buffers. Connect it to mm2s_frame_ptr_in[5:0].



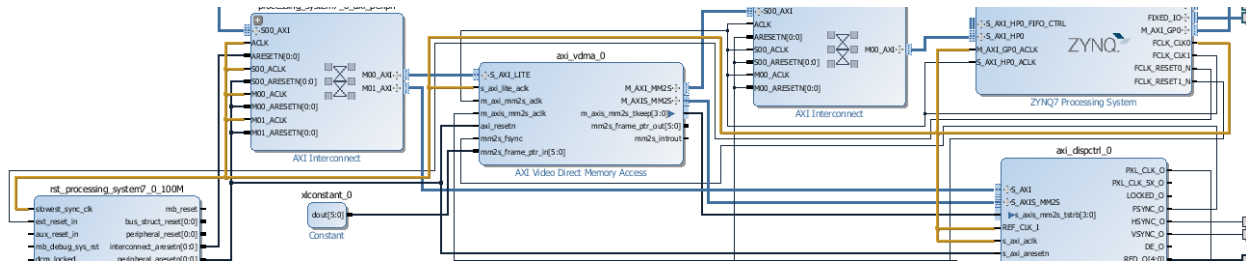


1.5 Clean Up Connections Manually

Check all clock connections to make sure the following. FCLK_CLK0 is used for AXI GP, (general purpose) and FCLK_CLK1 is used for AXI HP (high performance).

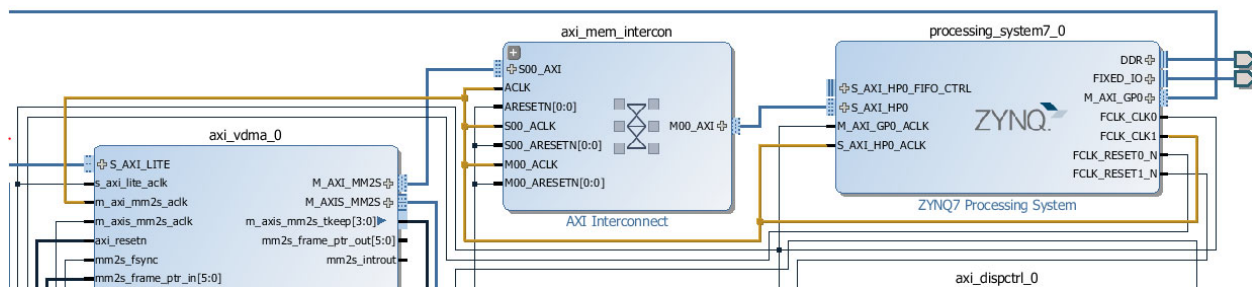
- 1) FCLK_CLK0 is connected to the following:

ACLK, S00_ACLK, M00_ACLK, on processing_system7_0_axi_periph,
s_axi_lite_aclck on axi_vdma_0,
M_AXI_GP0_ACLK on ZYNQ,
REF_CLK_1 and s_axi_aclck on axi_dispctrl_0.



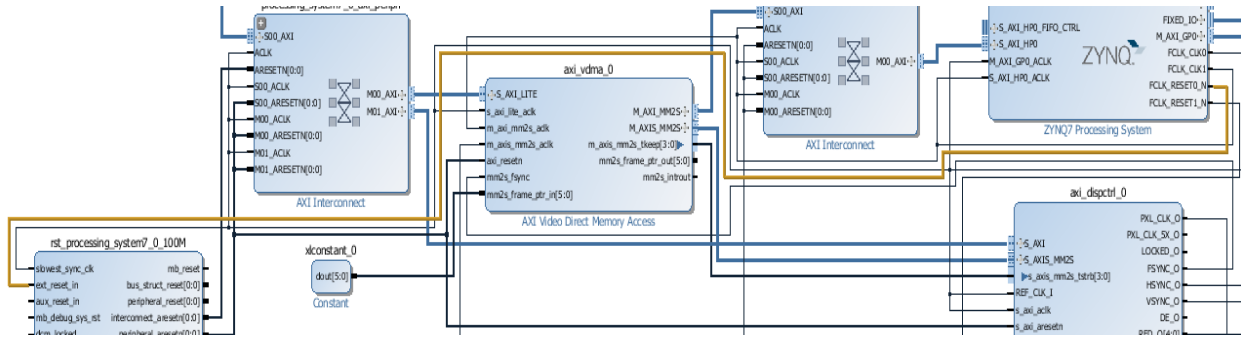
- 2) FCLK_CLK1 is connected to the following:

AXI_HP0_ACLK on ZYNQ,
m_axi_mm2s_aclck on axi_vdma_0,
ACLK, S00_ACLK, M00_ACLK on axi_mem_intecon, as shown below.

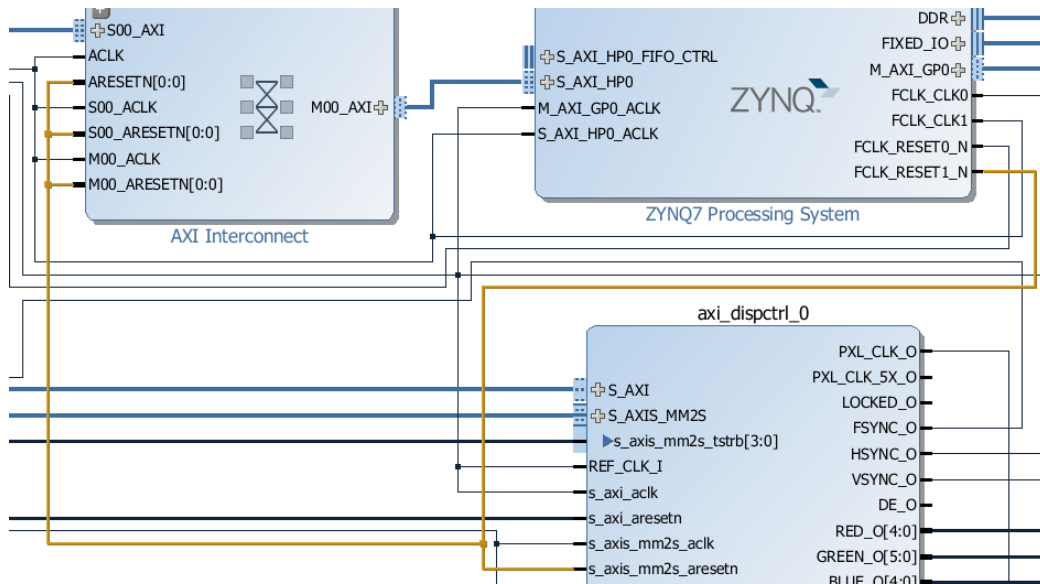


Check all reset connections to make sure resets are as follows:

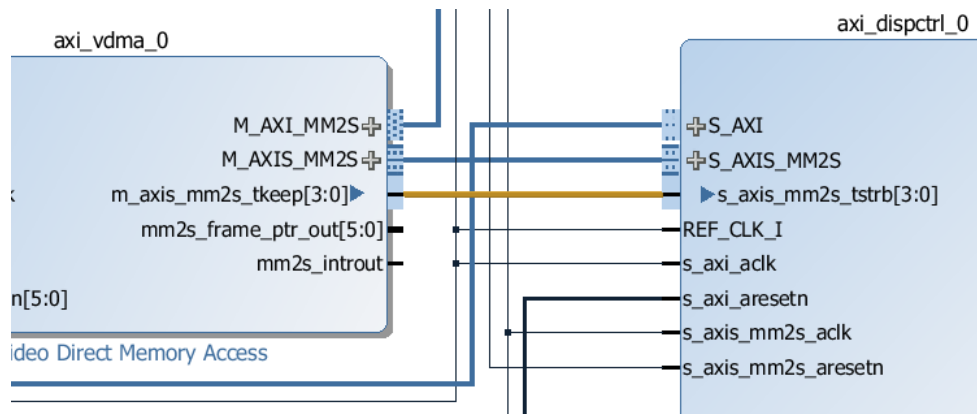
- 3) FCLK_RESET0 is used for AXI bus reset and is connected to s_axi_aresetrn on axi_dispctrl_0, ARESETN on axi_mem_intecon, ARESETN on processing_system7_0_axi_periph, (These may not be correct. 1-19-2017)



- 4) *FCLK_RESET1* is used for AXI peripheral reset and is connected to *s_axis_mm2s_aresetn* on *axi_dispctrl_0*, *S00_ARESETN*, *M00_ARESETN*, on *axi_mem_intercon*, *S00_ARESETN*, *M00_ARESETN*, on *processing_system7_0_axi_periph*, *axi_resten* on *axi_vdma_0*.

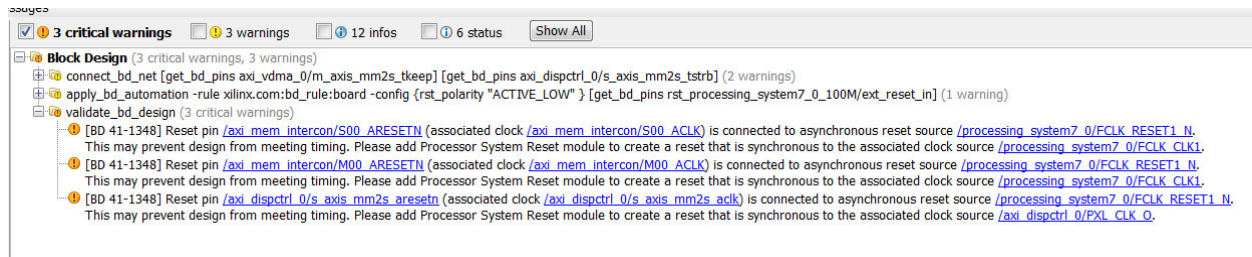


- 5) *M_axis_mm2s_aclk* on *axi_vdma_0* is connected to the *PXL_CLK_O* on *axi_dispctrl_0*.
- 6) *mm2s_fsync* on *axi_vdma_0* is connected to *FSYNC* on *axi_dispctrl_0*.
- 7) Open *M_AXIS_MM2S* and connect *m_axis_mm2s_tkeep[3:0]* of *M_AXI_MM2S* to *s_axis_mm2s_tstrb[3:0]* of *S_AXIS_MM2S* on *axi_dispctrl_0*.
- 8) *M_AXIS_MM2S* on *axi_vdma_0* is connected to *S_AXIS_MM2S* on *axi_dispctrl_0*.



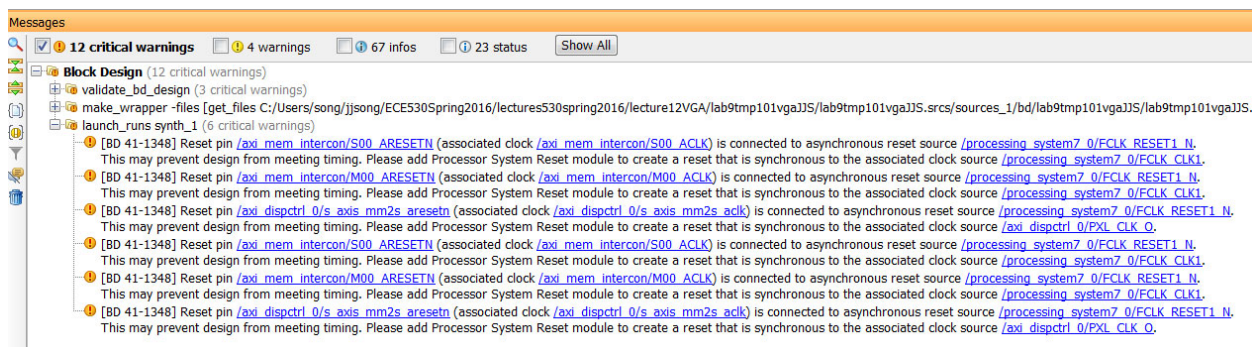
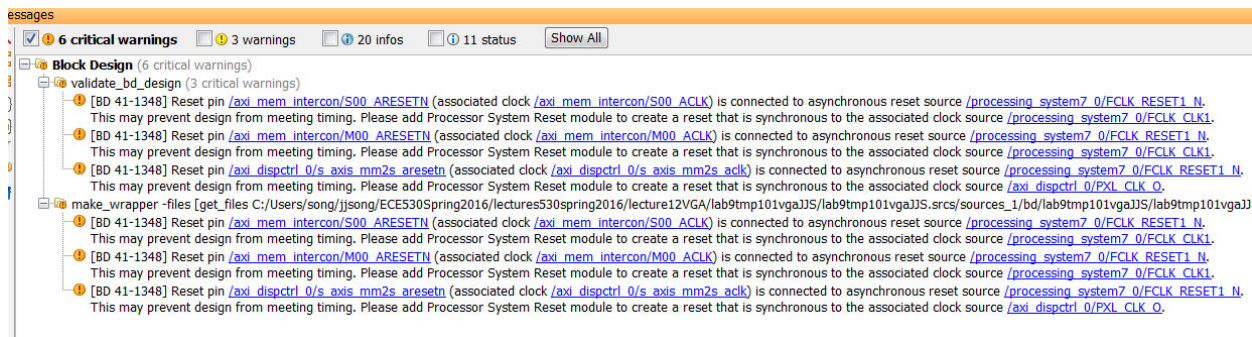
1.6 Three Critical Warnings

These warnings are due to reset signals that are not compatible with clock signals. They can be ignored for now until we can find a better fix.



2 Create a wrapper and synthesize the block design.

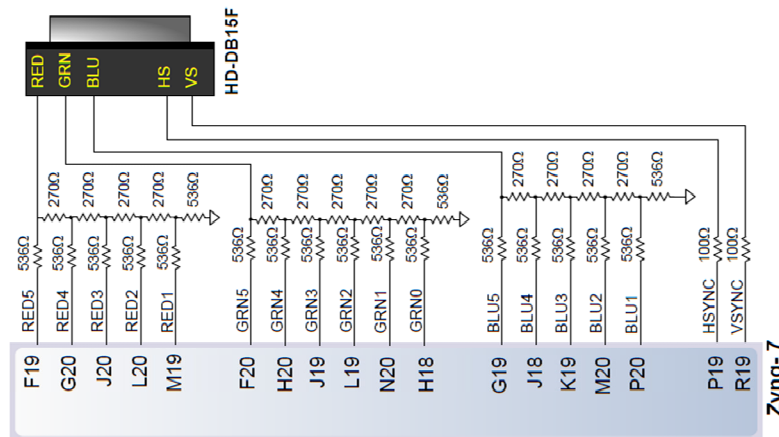
There will be many warnings on reset.



It takes about 8 minutes to synthesize this block design.

FIXED_IO_9964 (59)	INOUT			✓	(Multiple)*
BLUE_O (5)	OUT			✓	(Mult... LVCMOS33*
BLUE_O[4]	OUT	G19	▼	✓	35 LVCMOS33*
BLUE_O[3]	OUT	J18	▼	✓	35 LVCMOS33*
BLUE_O[2]	OUT	K19	▼	✓	35 LVCMOS33*
BLUE_O[1]	OUT	M20	▼	✓	35 LVCMOS33*
BLUE_O[0]	OUT	P20	▼	✓	34 LVCMOS33*
GREEN_O (6)	OUT			✓	(Mult... LVCMOS33*
GREEN_O[5]	OUT	F20	▼	✓	35 LVCMOS33*
GREEN_O[4]	OUT	H20	▼	✓	35 LVCMOS33*
GREEN_O[3]	OUT	J19	▼	✓	35 LVCMOS33*
GREEN_O[2]	OUT	L19	▼	✓	35 LVCMOS33*
GREEN_O[1]	OUT	N20	▼	✓	34 LVCMOS33*
GREEN_O[0]	OUT	H18	▼	✓	35 LVCMOS33*
RED_O (5)	OUT			✓	35 LVCMOS33*
RED_O[4]	OUT	F19	▼	✓	35 LVCMOS33*
RED_O[3]	OUT	G20	▼	✓	35 LVCMOS33*
RED_O[2]	OUT	J20	▼	✓	35 LVCMOS33*
RED_O[1]	OUT	L20	▼	✓	35 LVCMOS33*
RED_O[0]	OUT	M19	▼	✓	35 LVCMOS33*
Scalar ports (2)					
HSYNC_O	OUT	P19	▼	✓	34 LVCMOS33*
VSYNC_O	OUT	R19	▼	✓	34 LVCMOS33*

run implementation for about 5 minutes. Generate the bitstream file. Export hardware and launch SDK.



3 SDK and Demo Program from Digilent

Create an SDL empty project. Import source and header files from base_demo folder of zybo base system demo project: zybo_base_system\source\vivado\SDK\base_demo folder. Select the following files only.

The following terminal display will be present. Choose option 2: VGA output demo. Select Option 1 to change resolution first.

```

display_ctrl 4/10/2016 3:02 PM C File
display_ctrl 4/10/2016 3:02 PM H File
display_demo 4/10/2016 3:02 PM C File
display_demo 4/10/2016 3:02 PM H File
main         4/10/2016 11:18 PM C File
timer_ps     4/10/2016 3:02 PM C File
timer_ps     4/10/2016 3:02 PM H File
vga_modes    4/10/2016 3:02 PM H File
  
```



```

Serial: (COM14, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
*****
*
*      ZYBO Base System User Demo      *
*
*****

1 - Audio Demo
2 - VGA output demo
3 - HDMI output demo
q - Quit

Select a demo to run:

```

```

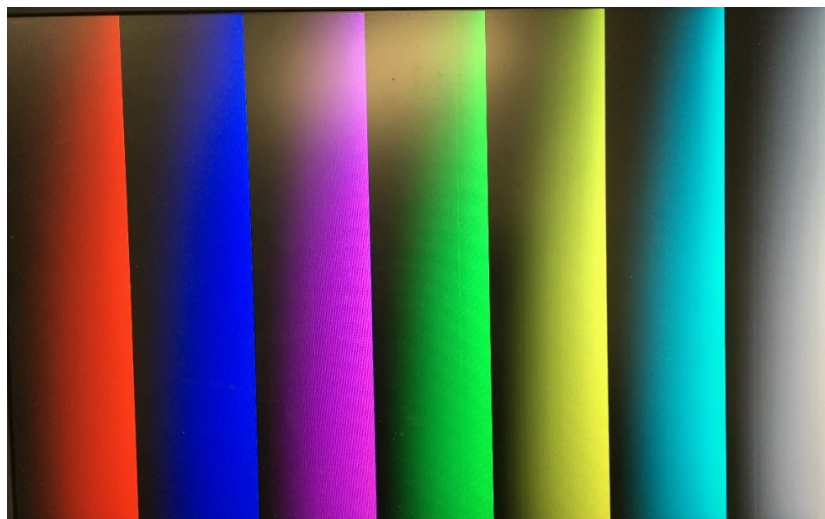
Serial: (COM14, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
*****
1 - Change Resolution
2 - Change Frame
3 - Print Blended Test Pattern to current Frame
4 - Print Color Bar Test Pattern to current Frame
5 - Invert Current Frame colors
6 - Invert Current Frame colors seamlessly*
q - Quit

*Note that option 6 causes the current frame index to be
incremented. This is because the inverted frame is drawn
to an inactive frame. After the drawing is complete, this
frame is then set to be the active frame. This demonstrates
how to properly update what is being displayed without image
tearing. Options 3-5 all draw to the currently active frame,
which is why not all pixels appear to be updated at once.

Enter a selection:

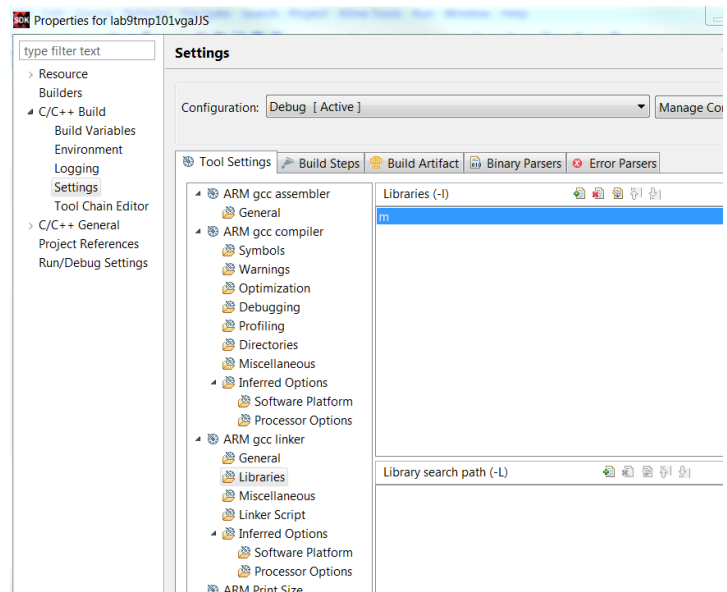
```

Option 4 is to print color bar test patterns as shown below.



4 How to add math library Compile Option

If you still cannot use math library after you add `#include "math.h"`, you can try to add a link option in Project Properties->C/C++ Build->Settings->ARM gcc linker->Libraries. Click + sign to type lowercase m to add math library in the libraries as shown on the right hand side.



Part 2: Create One New Image and Boot Load Image on QSPI

Follow Lab 5 Configuration and Booting, Advanced Embedded System Design on Zynq using Vivado from Xilinx University Program to create a boot image and store it on QSPI.

1 Create a New Project, entitled lab9imagesJJS

Save your project as a new project called lab9imagesJJS, where JJS is your name initials. Create at least two images similar to those from Tweetable Mathematical Art Competition (<http://codegolf.stackexchange.com/questions/35569/tweetable-mathematical-art>). One of the two images could be the same as one of the example images from the website. One of them must be your own creation.

2 How to Create a Bootable System from the QSPI Flash

This section describes how to create boot Image and store it on the QSPI memory. It is adapted from Xilinx University Program's Lab 5 Configuration and Booting from Lab 5 Configuration and Booting, Advanced Embedded System Design on Zynq using Vivado to create a boot image and store it on QSPI. Follow this section or the above mentioned Lab 5 to create your own bootable system and demonstrate you can start your lab9imagesJJS from the QSPI flash drive of your Zybo board.

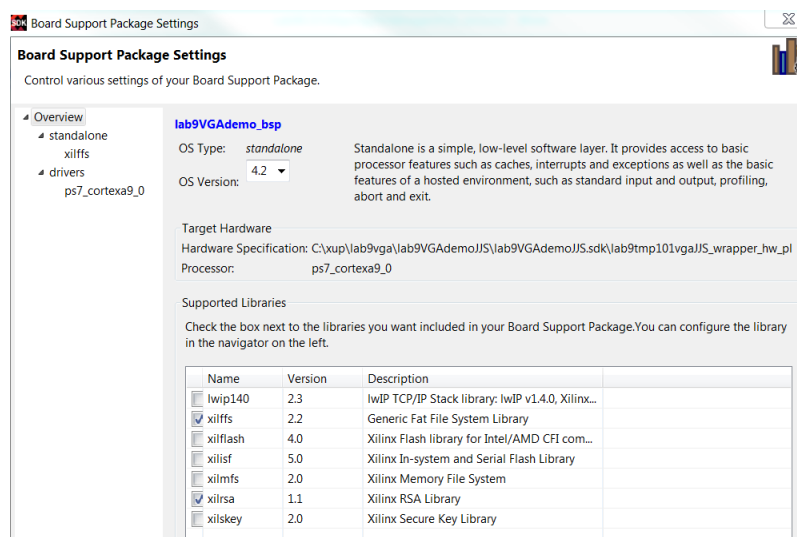
After you have debugged and tested your project, you can create a bootable system to be stored on a SD card or flash memory so that your project can be started from the bootable system. In this tutorial, a project call lab9VGAdemo is used as an example.

2.1 Create and debug your project first

Before creating a bootable system, you should create and debug your project and run it under interactive mode to make sure your project works. Close other projects if any under this SDK.

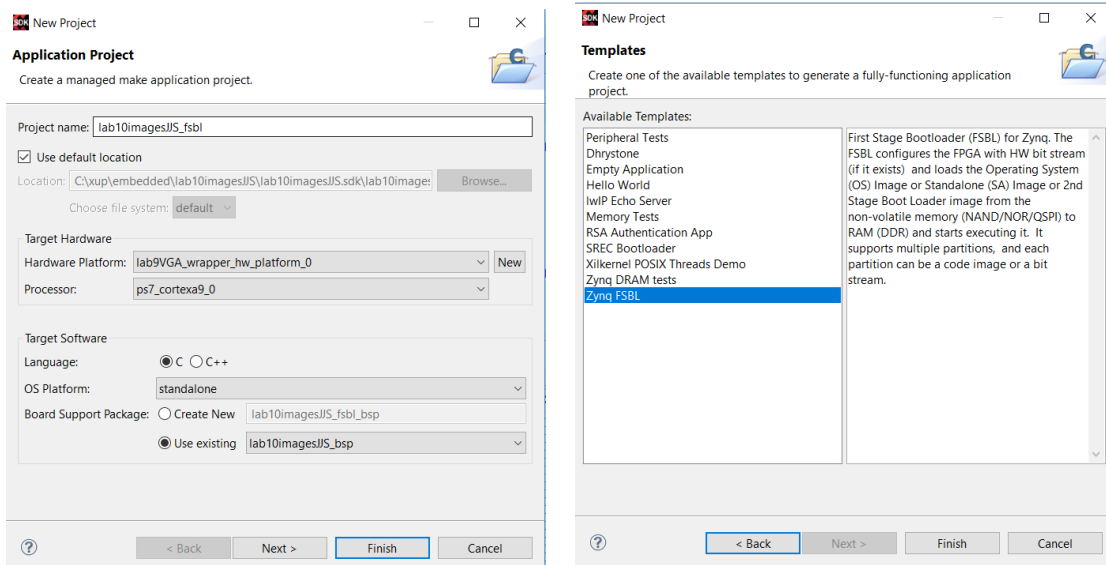
2.2 Add options to Board Support Package Settings

In SDK, after a project has been created and tested, right click on *.bsp and click Board Support Package Settings. Choose xilffs and xilrsa and click OK. (This is required for the next step to create the FSBL.)



2.3 Create first stage bootloader (FSBL)

To create a first stage bootloader (FSBL) for your project, create a new application project and call it your project name_fsbl, e.g., lab9imagesJJS_fsbl and choose “use existing Board Support Package”. Click “Next”. In Template menu, select Zynq FSBL in the Available Templates pane and click “Finish”.

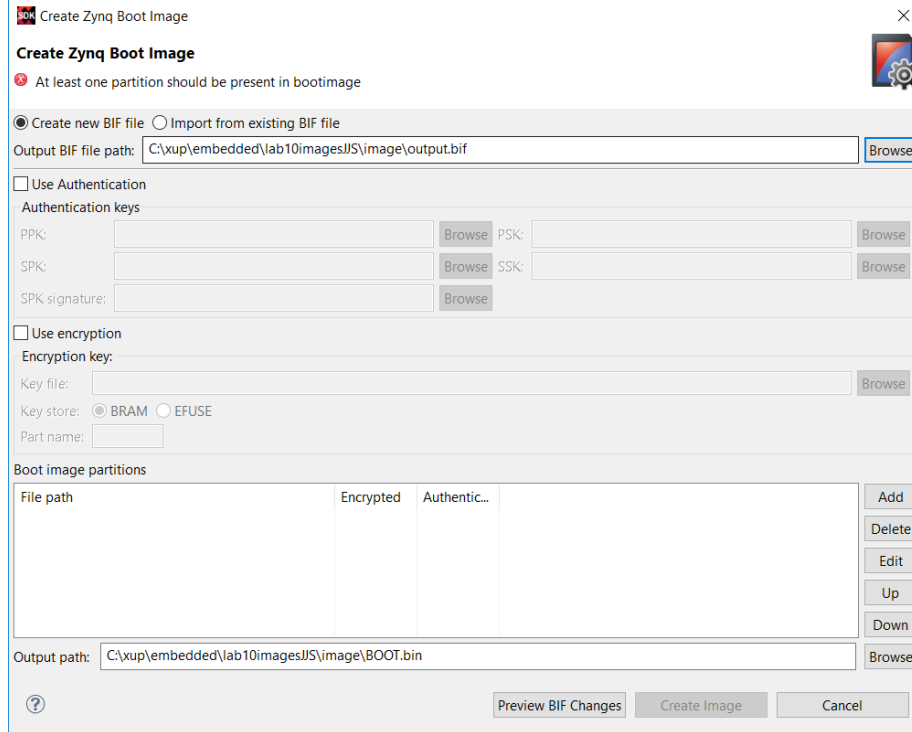


2.4 Create directory called image and BOOT.bin file

Create a directory under your project directory call it image with Windows Explorer to store bootable system files as shown below.

This PC > Windows (C:) > xup > embedded > lab10imagesJJS				
<input type="checkbox"/>	Name	Date modified	Type	Size
	image	4/4/2017 9:10 PM	File folder	
	lab10imagesJJS.cache	4/4/2017 8:45 PM	File folder	
	lab10imagesJJS.runs	4/4/2017 8:45 PM	File folder	
	lab10imagesJJS.sdk	4/4/2017 9:09 PM	File folder	
	lab10imagesJJS.srds	4/4/2017 8:45 PM	File folder	
	sources	4/4/2017 8:38 PM	File folder	
	lab10imagesJJS.xpr	4/4/2017 8:48 PM	Vivado Project File	7 KB

In SDK, select Xilinx Tools->Create Boot Image. Click on the Browse button of Output BIF field to browse to your image directory */image. Leave the default name of output.bif.



Create Zynq Boot Image

At least one partition should be present in bootimage

☒ Create new BIF file ☐ Import from existing BIF file

Output BIF file path:

☐ Use Authentication

Authentication keys

PPK: PSK:
 SPK: SSK:
 SPK signature:

☐ Use encryption

Encryption key:

Key file:

Key store: ☒ BRAM ☐ EFUSE

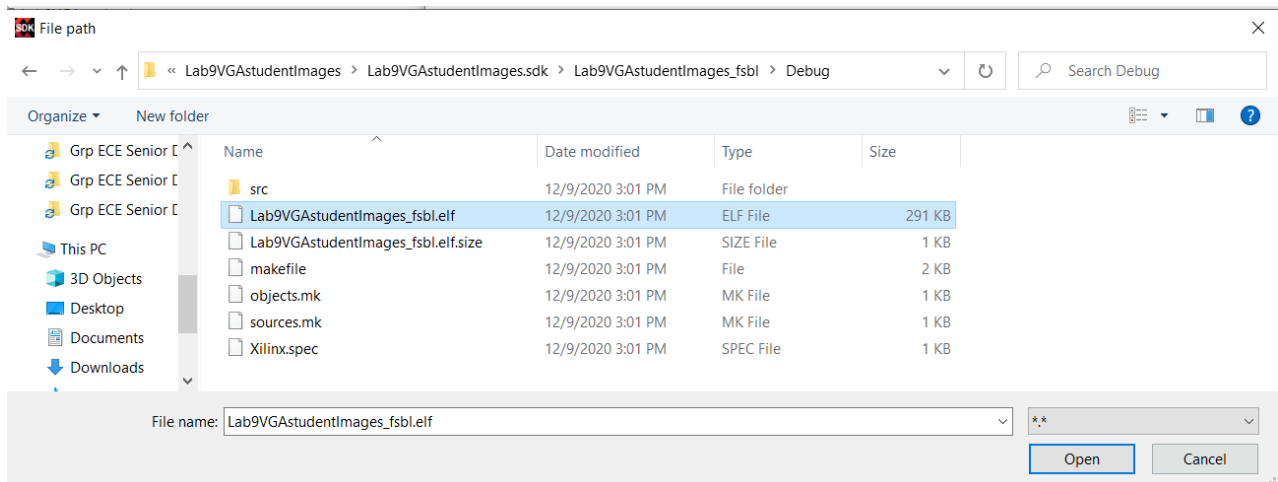
Part name:

Boot image partitions

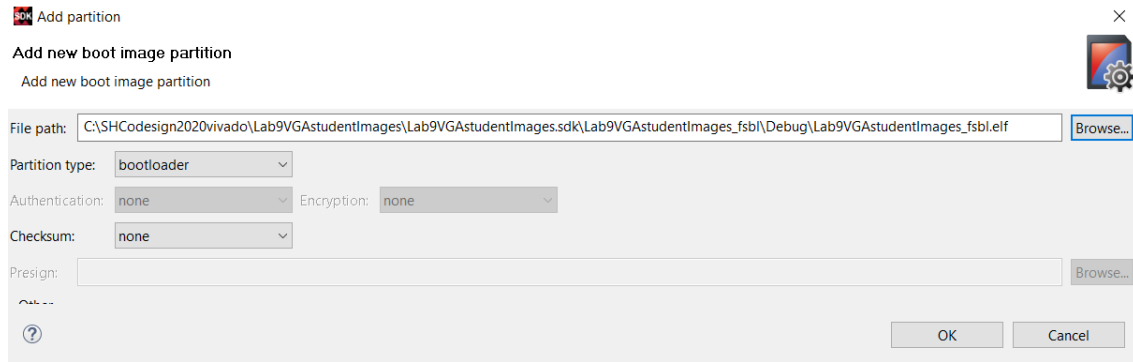
File path	Encrypted	Authentic...

Output path:

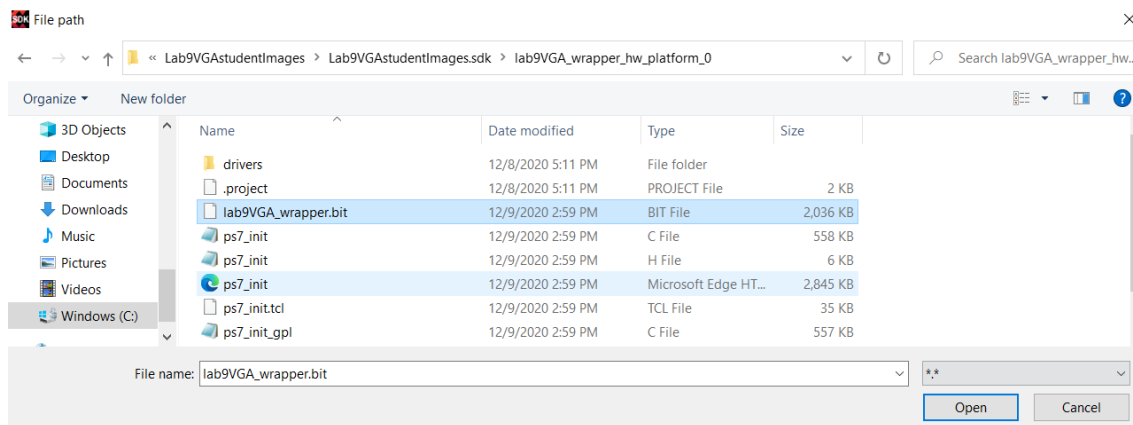
Click on Add button of the Boot image partitions, click Browse button Add partition form. Browse to \Debug of the sdk subdirectory of your project directory. Open *_fsbl.elf file as shown below for project lab9imagesJJS_fsbl. Here is a possible example path: C:\xup\embedded\lab9imagesJJS\lab9imagesJJS.sdk\lab9imagesJJS_fsbl\Debug.



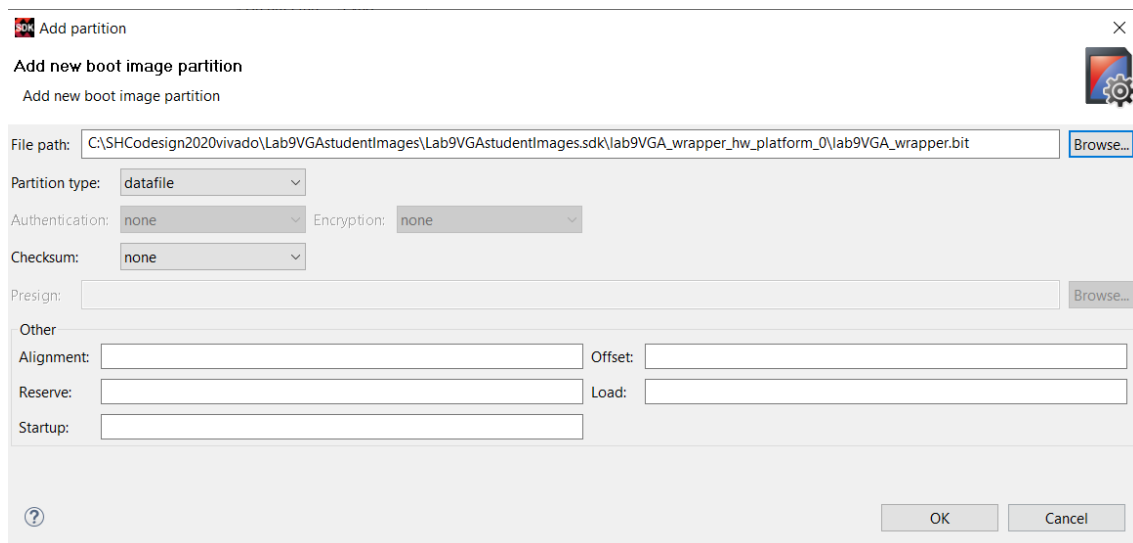
Notice the partition type for *.elf file is bootloader. Click OK.



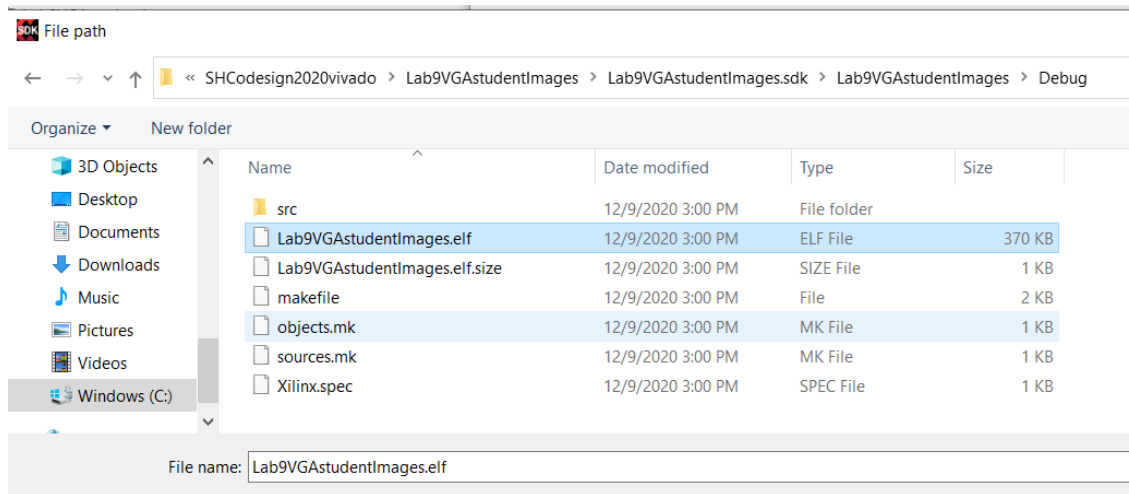
Add bit stream file from project hw_platform directory of your project under SDK folder. The bit stream file is often called project_wrapper.bit. The file for this example is lab9VGA_wrapper.bit as shown below under directory:
 C:\SHCodesign2020vivado\Lab9VGAstudentImages\Lab9VGAstudentImages.sdk\lab9VGA_wrapper_hw_platform_0. Click Open to add it.



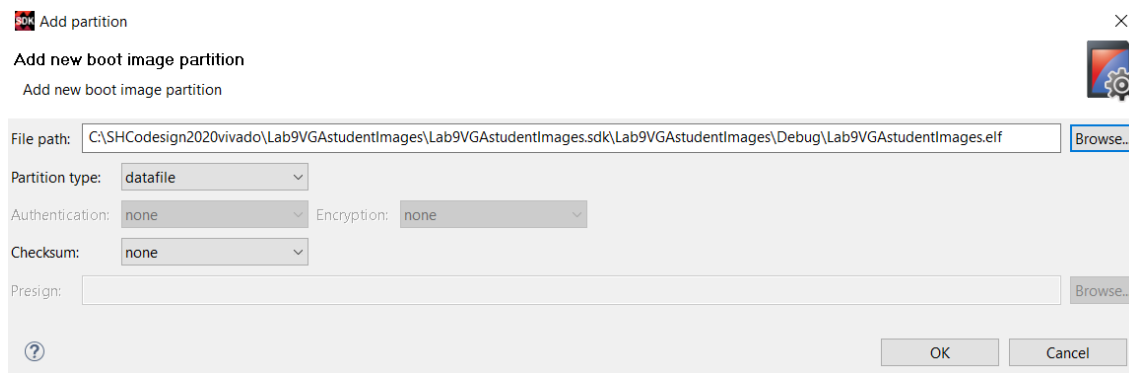
Notice the file type for *.bit file is datafile. Click OK on Add new boot image partition menu to add this bit stream file.



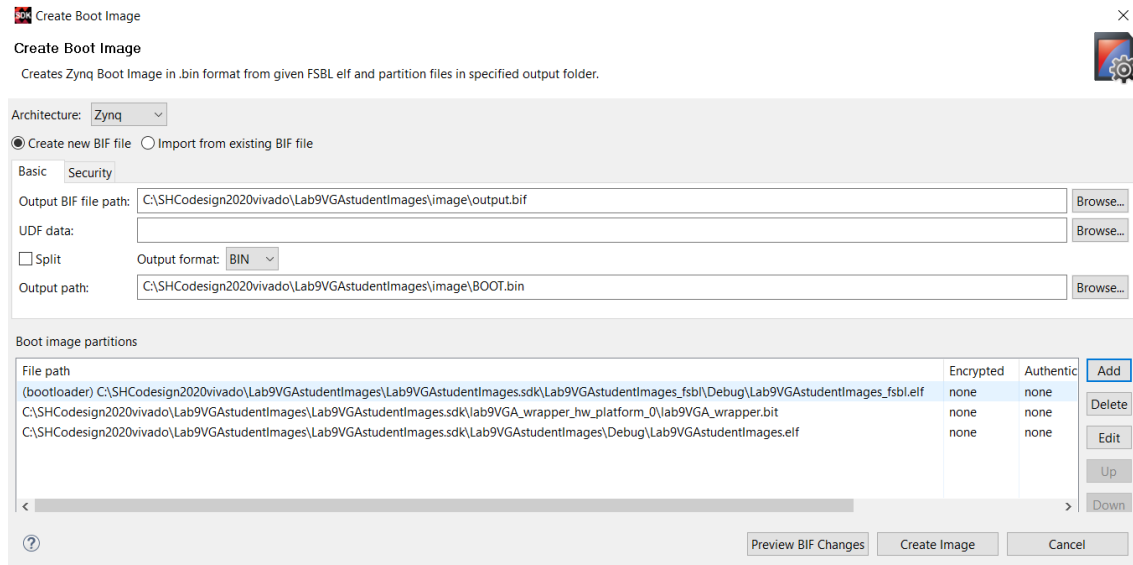
Click Add button of the Boot image partitions and add software application lab9VGAdemo from lab9imagesJJS\Debug directory under the SDK directory of your project. Click Open to add this file.



This type of this application elf file is also datafile.



Notice there should be three files in your Boot image partitions folder. The file order must be first stage boot load file _fsbl.elf, hardware bit stream file _wrapper.bit, and the application *.elf file. Otherwise, there could be a boot loading error.

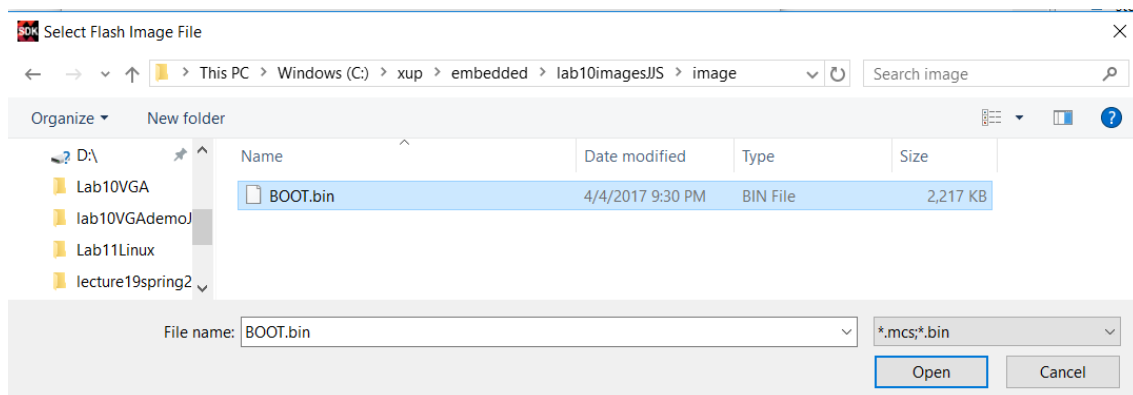
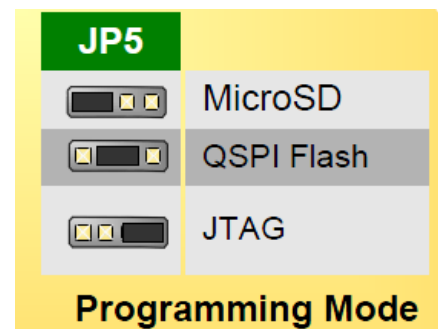


Click “Create Image” button. The BOOT.bin and the output.bif files will be created in the \image directory.


2.5 Program the QSPI flash memory with the bootable system files.

Set your Zybo board to be in JTAG boot mode by setting jumper JP5 to JTAG position.

In SDK, select Xilinx Tools->Program Flash. Click Browse button for Image File and go to \image directory, select BOOT.bin file and click Open.





In the Offset field, enter 0 as the offset, click Program button. A number of messages will appear on the Console and the end message should be “flash Operation Successful”.



Program Flash Memory

Program Flash Memory via In-system Programmer.





Hardware Platform: lab9VGA_wrapper_hw_platform_0

Connection: Local New

Device: Auto Detect Select...

Image File: C:\SHCodesign2020vivado\Lab9VGAstudentImages\image\BOOT.bin Browse

Offset: 0

Flash Type: qspi_single

FSBL File: Browse

☐ Convert ELF to bootloadable SREC format and program
☐ Blank check after erase
☐ Verify after flash

?
Program
Cancel

```

Problems Tasks Console Properties SDK Terminal
Program Flash
Device 1: jsn-Zybo-210279759007A-13722093-0

Retrieving Flash info...

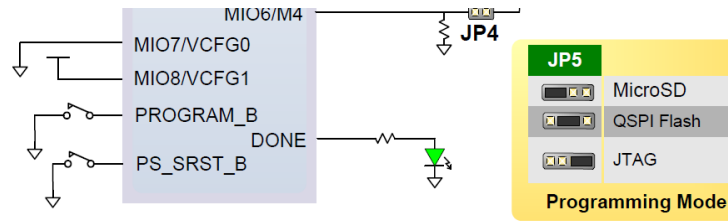
JTAG chain configuration
-----
Device  ID Code      IR Length  Part Name
1       4ba00477      4          arm_dap
2       13722093      6          xc7z010

CortexA9 Processor Configuration
-----
Version.....0x00000003
User ID.....0x00000000
No of PC Breakpoints.....6
No of Addr/Data Watchpoints.....4
Waiting for Bootrom to re-enable DAP after reset
Processor Reset .... DONE
SF: Detected S25FL128S 64K with page size 256 Bytes, erase size 64 K
Performing Erase Operation...
Erase Operation successful.
INFO: [Xicom 50-44] Elapsed time = 7 sec.
Performing Program Operation...
0%.....
.....Program Operation successful.
INFO: [Xicom 50-44] Elapsed time = 12 sec.

Flash Operation Successful
  
```

2.6 Test your QSPI bootable system

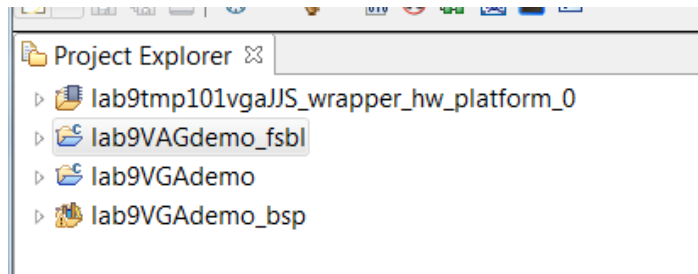
Set your Zybo board to be in QSPI mode by inserting the jumper in the middle. Power the board off and on. Start a serial terminal if needed. Press PS_RST (Red button) button to see if your program will start.



2.7 An error

The following error occurred because the order of three files was wrong. The bit stream file was placed after the application *.elf file. In the Boot image partitions pane, the bit stream file *.bit must be before the application file *.elf.

You can right click on _fsbl folder under your Project Explorer menu to open Create Boot Image. You can then edit this configuration and create another boot loader image. You may notice that *.bit file is gone. Add it again and make sure *.bit is before *.elf.



3 Acknowledgment

I am grateful to my students David Robinson and Joe Milittello for their research and solutions to some issues and problems with this lab.

