

# Using the TFT controller

---

## Acknowledgement

The controller used in this tutorial is provided by the Xilinx IP catalog.

## Goal

- Use IP integrator to connect and configure TFT controller with the MicroBlaze system
- Communicate to video memory using the TFT controller
- Be able to use a C program in SDK to interact with the TFT controller

## Requirements

- Xilinx Vivado software
- Xilinx SDK software
- Xilinx Nexys 4 board and a programming cable
- Enough disk space for the project files

## Background

Before using the AXI TFT controller, it is important to note that this controller has both a master and a slave axi interface. In this tutorial, the TFT controller is connected as a slave of them to on the AXI bus for the purpose of allowing the MicroBlaze processor write to the controller's internal registers. It is also a master as well, because it needs to access video memory through the same bus.

The controller must be connected as a slave to the axi bus because we need to initialize its address register and control registers to tell the controller where to read from the memory. We can do this using the MicroBlaze.

The slave port can be connected to the axi peripheral output of the AXI-interconnect.

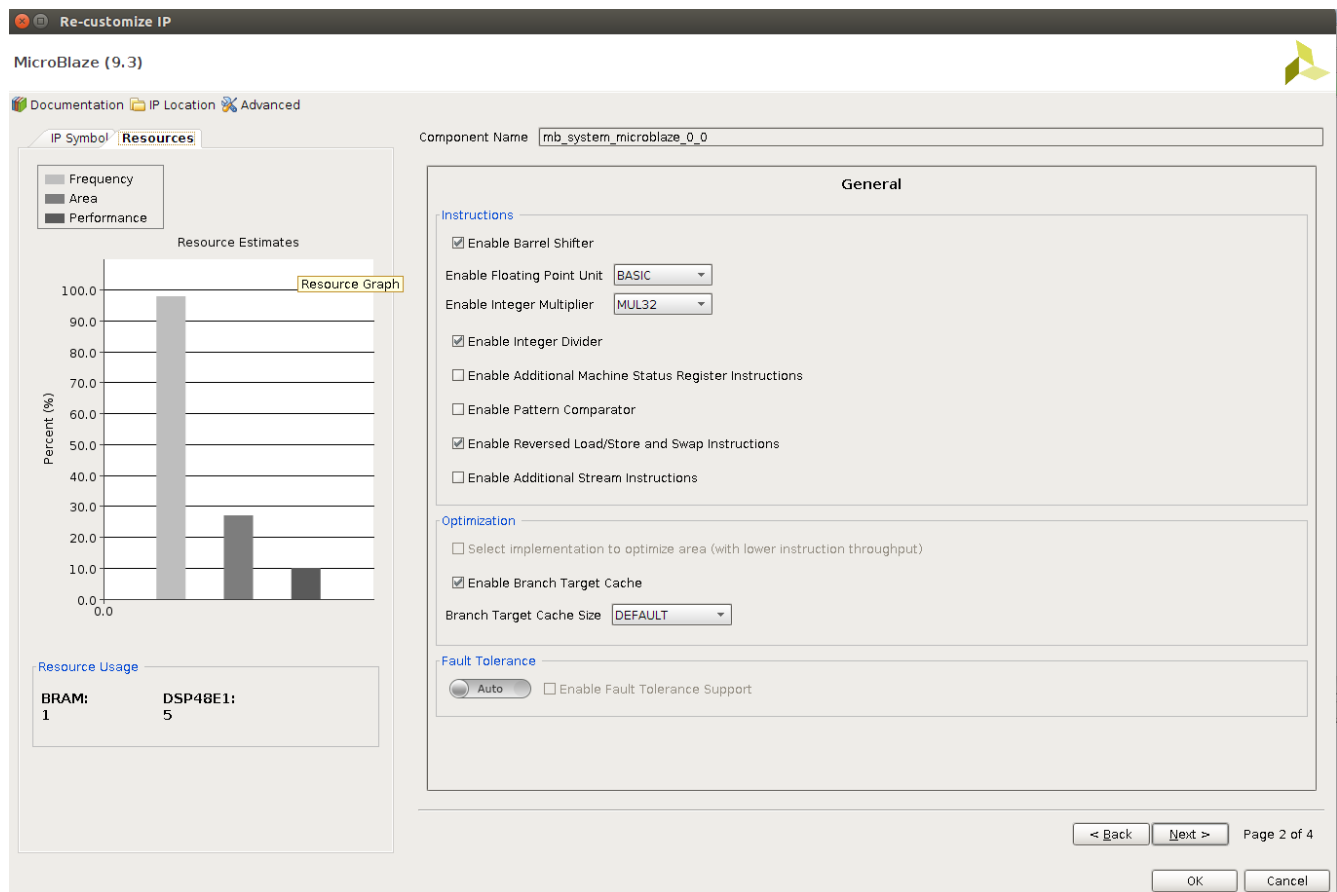
The master port must be connected to the axi peripheral slave input in a similar way to how the MicroBlaze is connected.

## 1. Import the Board Package

1. Invoke the Vivado IDE
2. Bring up the Tcl Console at the bottom of the window and enter:  
**set\_param board.repoPaths <path-to-board-files>/board\_files/**
3. Create a **New Project** specifying the **Nexys 4 DDR** board and leaving the rest of the settings as default

## 2. MicroBlaze Processor

1. From Navigator > IP Integrator, select **Create Block Diagram**
2. Right click anywhere in the Diagram and select Add IP and add a MicroBlaze block to the design
3. Double-click the MicroBlaze Processor Diagram to open the Re-customize IP dialog box.
4. Leave everything in page 1 as default, and click **Next**.
5. On page 2 of the Re-customize IP dialog box:
  - Check the **Enable Barrel Shifter** option.
  - From the pulldown menu, in option **Enable Floating Point Unit** select **BASIC**.
  - From the pulldown menu, in option **Enable Integer Multiplier** select **MUL32** (32-bit).
  - Check the **Enable Integer Divide** option.
  - Check the **Enable Branch Target Cache** option.
  - Click **Next**.



6. On Page 3 of the Re-customize IP dialog box, ensure that the **MicroBlaze Debug Module** is enabled (i.e. BASIC), and click Next.

Re-customize IP

MicroBlaze (9.3)

Documentation IP Location Advanced

IP Symbol Resources

Component Name mb\_system\_microblaze\_0\_0

Debug

MicroBlaze Debug Module Interface BASIC

Hardware Breakpoints

Number of PC Breakpoints 1 [0..8]

Number of Write Address Watchpoints 0 [0..4]

Number of Read Address Watchpoints 0 [0..4]

Performance Monitoring

Number of Performance Monitor Event Counters 5 [0..48]

Number of Performance Monitor Latency Counters 1 [0..7]

Performance Monitor Counter Width 32

Trace & Profiling

Trace Buffer Size 8kB

Profile Buffer Size NONE

Resource Estimates

Resource Graph

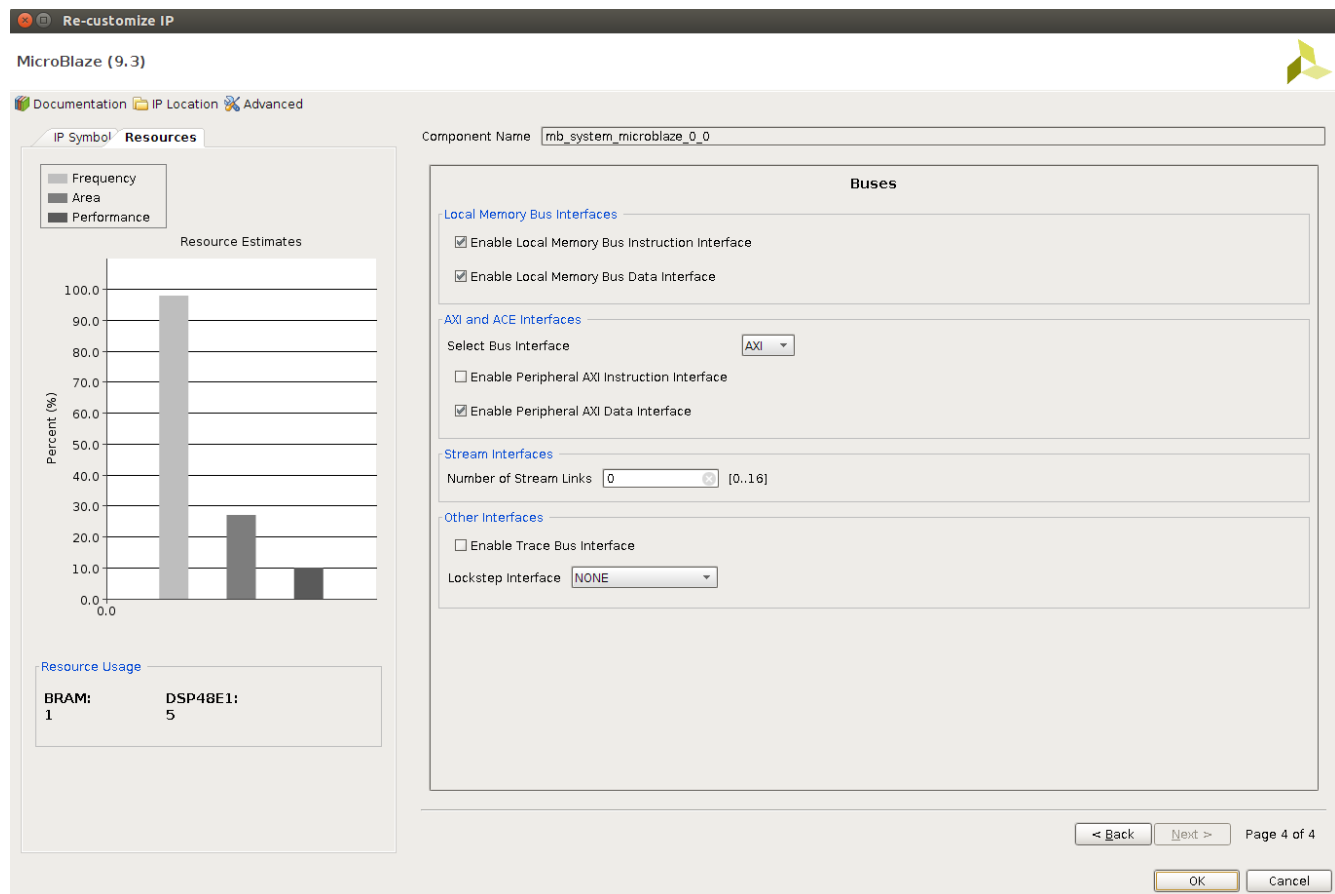
Percent (%)

BRAM: 1 DSP48E1: 5

< Back Next > Page 3 of 4

OK Cancel

7. On Page 4 of the Re-customize IP dialog box, ensure that the **Enable Peripheral AXI Data Interface** option is checked, and click OK to re-configure the MicroBlaze processor



**MicroBlaze (9.3)**

Documentation IP Location Advanced

IP Symbol Resources

Component Name: mb\_system\_microblaze\_0\_0

**Resource Estimates**

Frequency Area Performance

Percent (%)

Resource Usage

Resource	Usage
BRAM	1
DSP48E1	5

**Buses**

**Local Memory Bus Interfaces**

- ☒ Enable Local Memory Bus Instruction Interface
- ☒ Enable Local Memory Bus Data Interface

**AXI and ACE Interfaces**

Select Bus Interface: AXI

- ☐ Enable Peripheral AXI Instruction Interface
- ☒ Enable Peripheral AXI Data Interface

**Stream Interfaces**

Number of Stream Links: 0 [0..16]

**Other Interfaces**

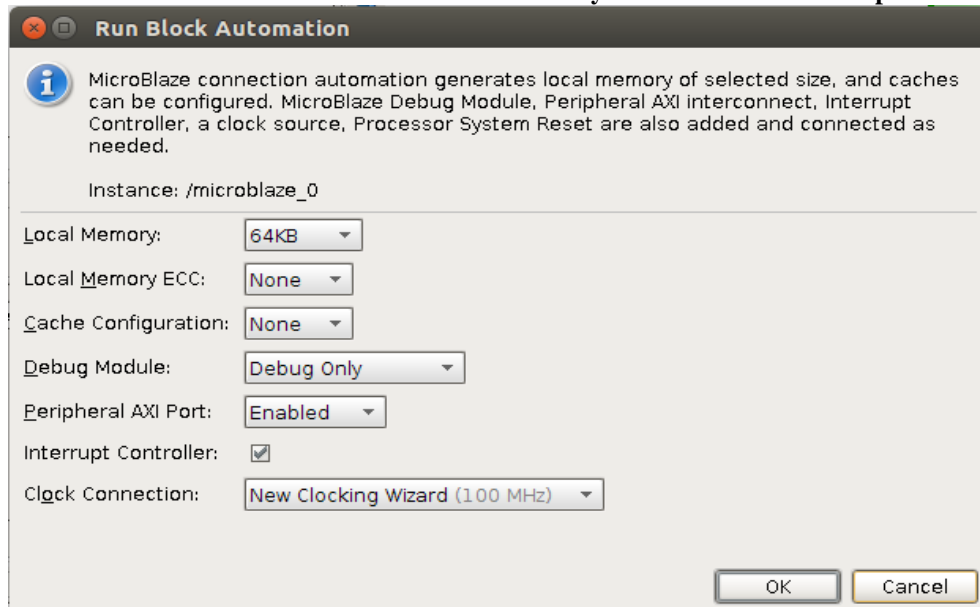
- ☐ Enable Trace Bus Interface

Lockstep Interface: NONE

< Back Next > Page 4 of 4

OK Cancel

8. Run Block Automation for the MicroBlaze with **Local Memory** set to 32KB and **Interrupt Controller** enabled.



**Run Block Automation**

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze\_0

Local Memory: 64KB

Local Memory ECC: None

Cache Configuration: None

Debug Module: Debug Only

Peripheral AXI Port: Enabled

Interrupt Controller: ☒

Clock Connection: New Clocking Wizard (100 MHz)

OK Cancel

### 3. Clock Customization

The processing system will operate at 100 MHz, but the memory controller requires a 200 MHz input clock to generate the appropriate clock for the external DRAM. Additionally, the pixel clock of the TFT controller requires a frequency of 25MHz to operate a 640 by 480 display.

1. Double click the **Clock Wizard** (clk\_wiz\_1) block to re-customize it.
2. Under the **Board** tab, use the Board Interface pull-down menu for IP interface CLK\_IN1 and select sys clock.

Component Name: design\_1\_clk\_wiz\_1\_0

Board | Clocking Options | Output Clocks | MMCM Settings | Summary

Associate IP interface with board interface

IP Interface	Board Interface
CLK_IN1	sys clock
CLK_IN2	Custom
EXT_RESET_IN	Custom

Clear Board Parameters

3. Under the **Output Clocks** tab, check the radio box for Output Clock clk\_out2 and enter a requested clock frequency of 200 MHz.

4. Repeat the process and check the radio box for Output Clock clk\_out3 and enter a requested clock frequency of 25 MHz.

5. While in Output Clocks tab, change the Reset Type to Active Low.

Component Name: design\_1\_clk\_wiz\_1\_0

Board | Clocking Options | Output Clocks | MMCM Settings | Summary

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz) Requested	Actual	Phase (degrees) Requested	Actual	Duty Cycle (%) Requested	Actual	Drives
<input checked="" type="checkbox"/> clk_out1	100.000	100.000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out2	200	200.000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out3	25	25.000	0.000	0.000	50.000	50.0	BUFG
<input type="checkbox"/> clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

☐ USE CLOCK SEQUENCING

Clocking Feedback

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Source: ☒ Automatic Control On-Chip ☐ Automatic Control Off-Chip ☐ User-Controlled On-Chip ☐ User-Controlled Off-Chip

Signaling: ☒ Single-ended ☐ Differential

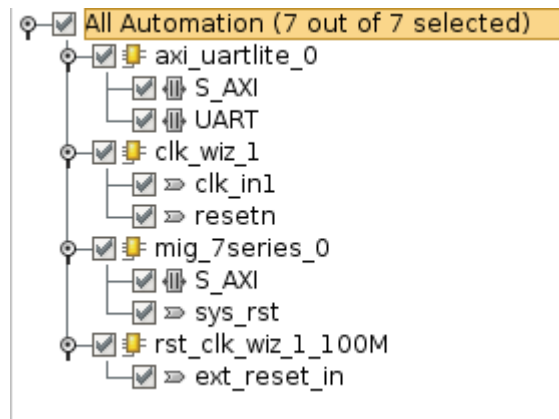
Enable Optional Inputs / Outputs: ☒ reset ☐ power\_down ☐ input\_clk\_stopped ☒ locked ☐ clkfbstopped

Reset Type: ☐ Active High ☒ Active Low

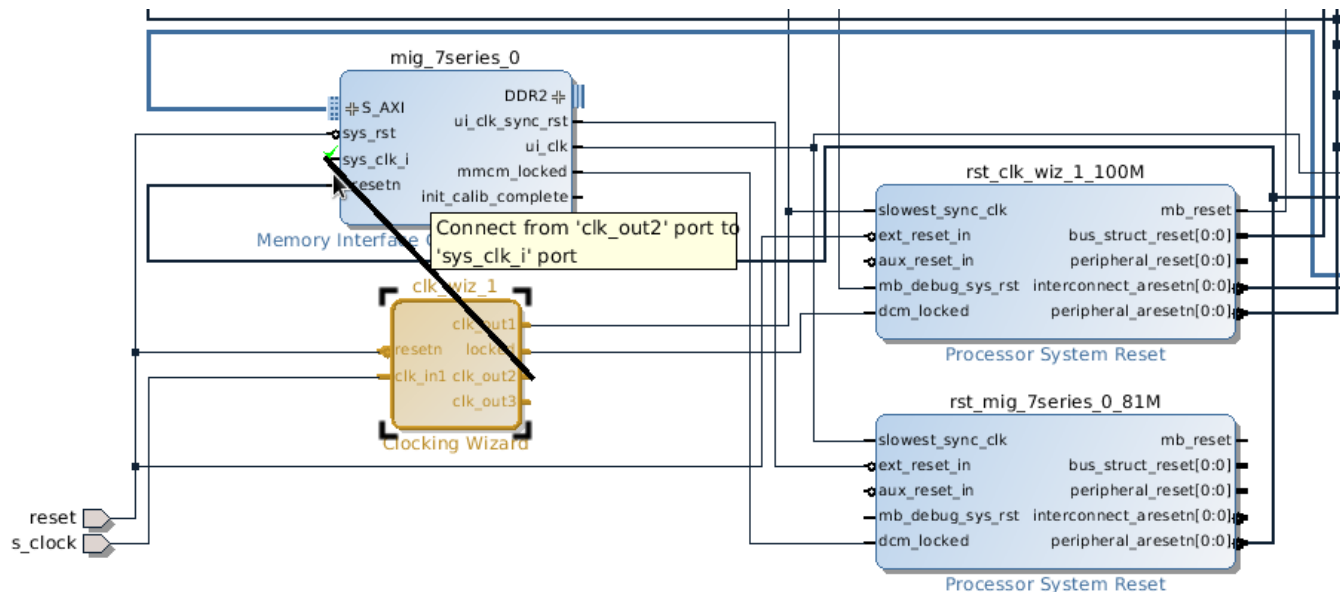
## 4. DDR2 Memory Block

In this section, you will be creating an external memory on the Nexys 4 DDR board that will be used as the video memory for the AXI TFT controller.

1. Right click anywhere in the Diagram and select Add IP and add an AXI Uartlite block to the design.
2. Repeat the process to search and add a Memory Interface Generator (MIG 7) peripheral.
3. Run Block Automation for the Memory Interface Generator. There may be an error during the process, you may ignore that and move on.
4. Run Connection Automation and selection all connections.



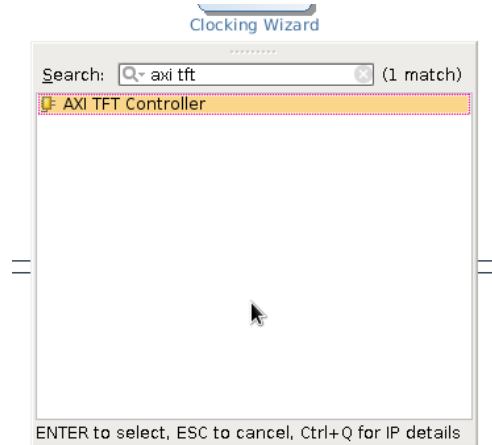
5. Manually connect clk\_out2 from clk\_wiz\_1 to the Memory Interface Generator sys\_clk\_i



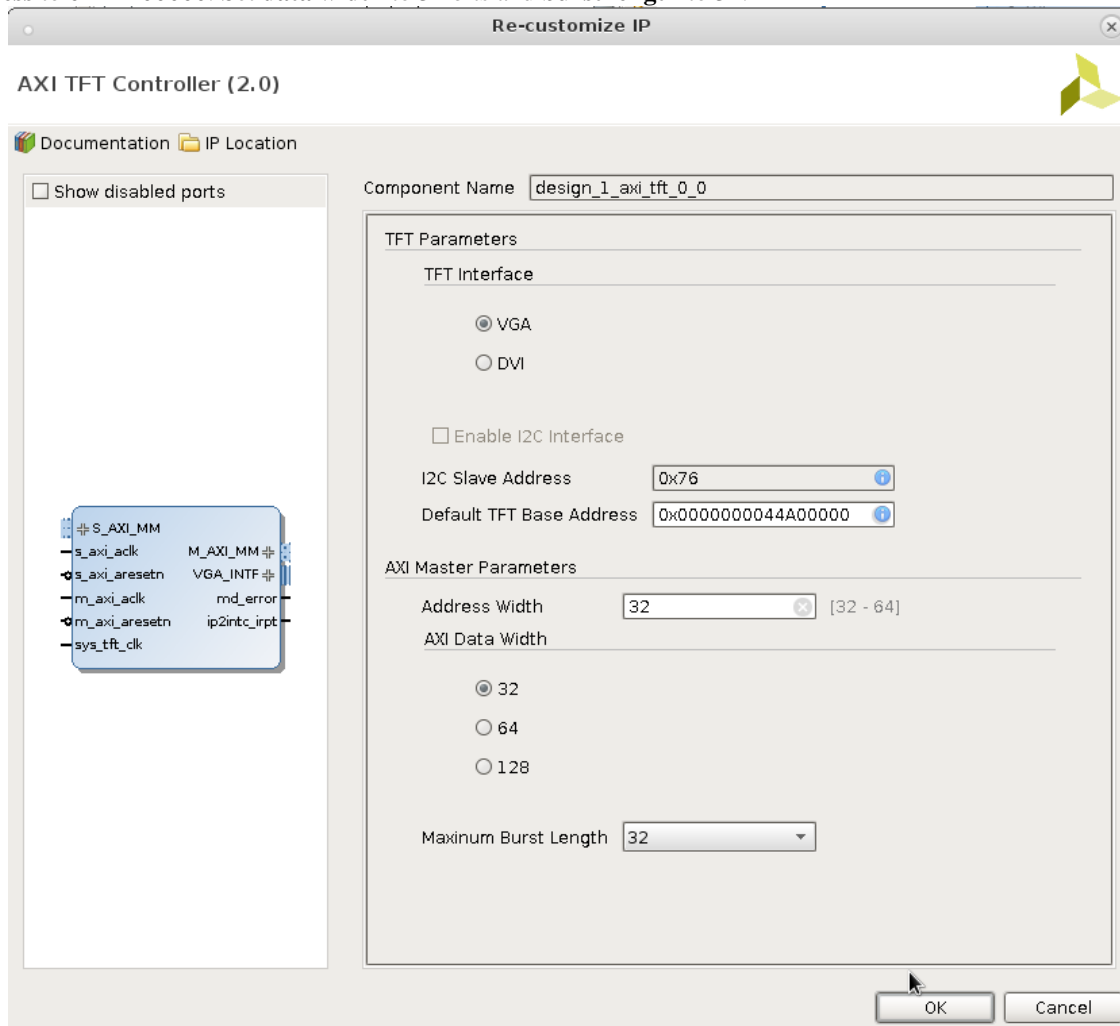
6. Right click **DDR2** in the Memory Interface Generator and make it **external**.

## 5. Connecting the TFT Controller

1. Add IP > AXI TFT Controller



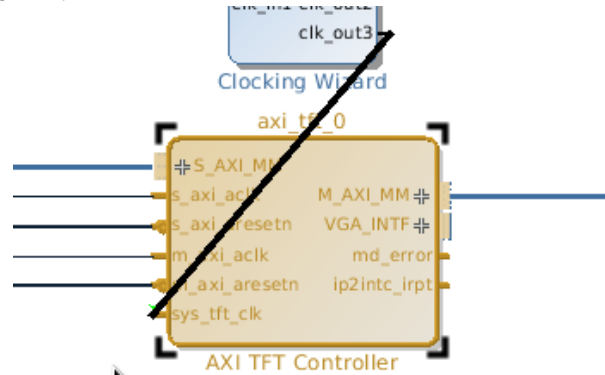
2. Double Click the TFT controller and customize it as in the figure below. Set **TFT Interface** to VGA. Set **base address** to 0x44A00000. Set **data width** to 32 bits and **burst length** to 32.



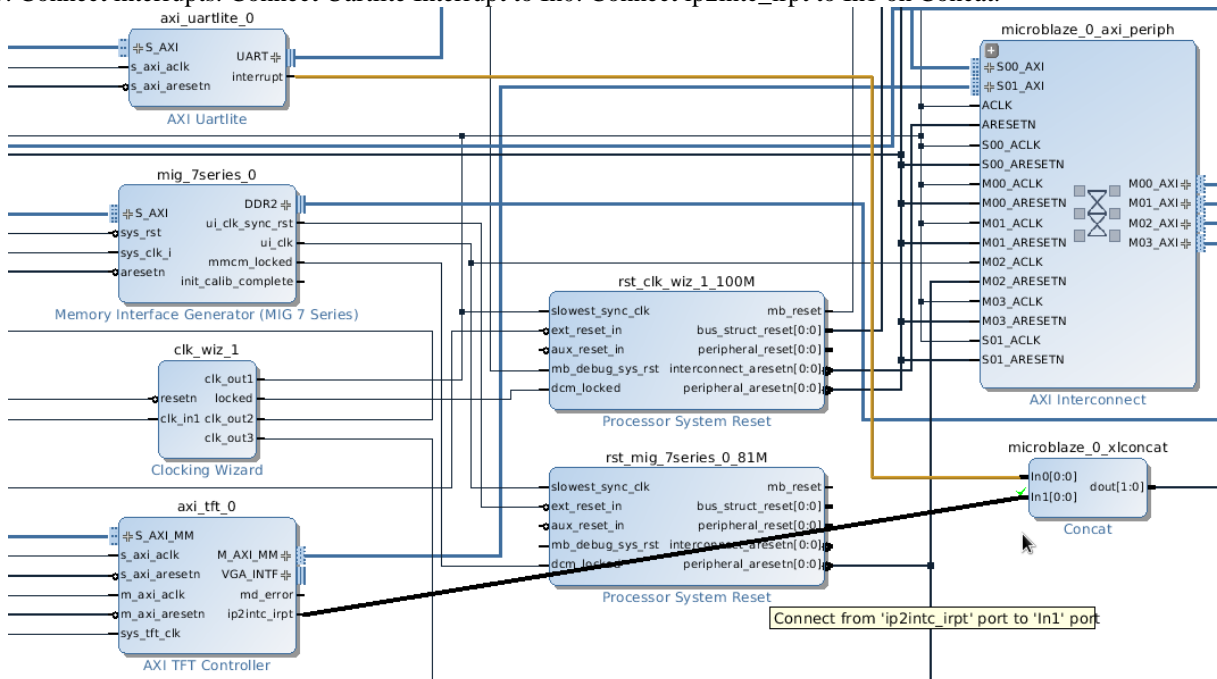
3. Run **Connection Automation** for all connections.

4. Connect **sys\_tft\_clk** to **clk\_out3**, which is 25MHz.

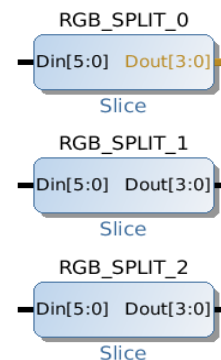
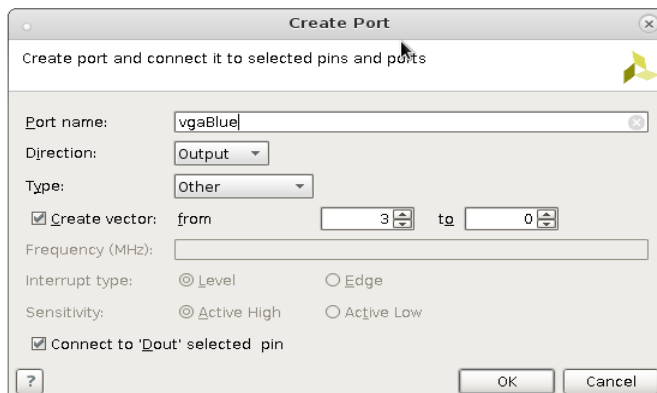
**SYS\_TFT\_CLK** – this is essentially the pixel clock of the controller. For operating a 640 by 480 display the clock should have a frequency of 25Mhz.



5. Connect interrupts. Connect Uartlite Interrupt to In0. Connect ip2intc\_irpt to In1 on Concat.

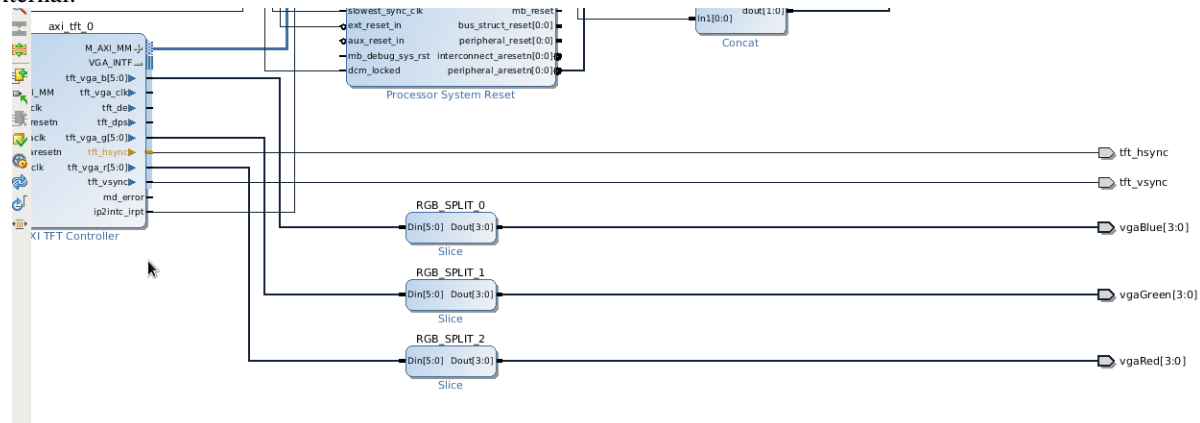


6. You will need to truncate the 2 least significant bits of each of the pixel colour output ports because the board only supports up to 12 bit RGB (4 bits for R, 4 bits for B, 4 bits for G). Search for **Slice** in the **Add IP** pop-up. Add three to the block diagram. Use it to truncate the 2 least significant bits of each pixel colour output port. Right click on the slice output and select make port – name them accordingly. Do the same for all R, G, and B.

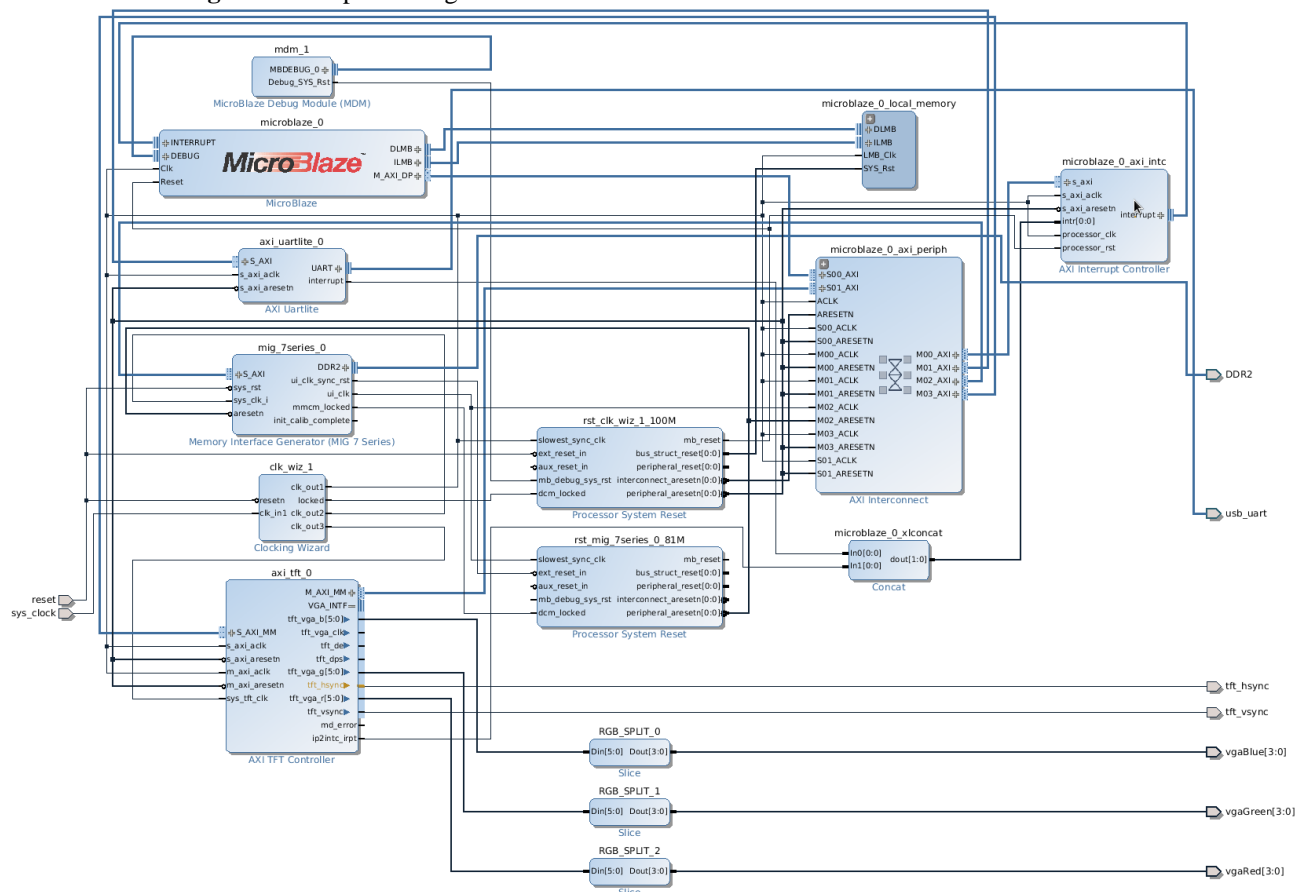




7. Expand VGA\_INTF and connect the wires accordingly. Right click tft\_vsync and tft\_hsync and select make external.



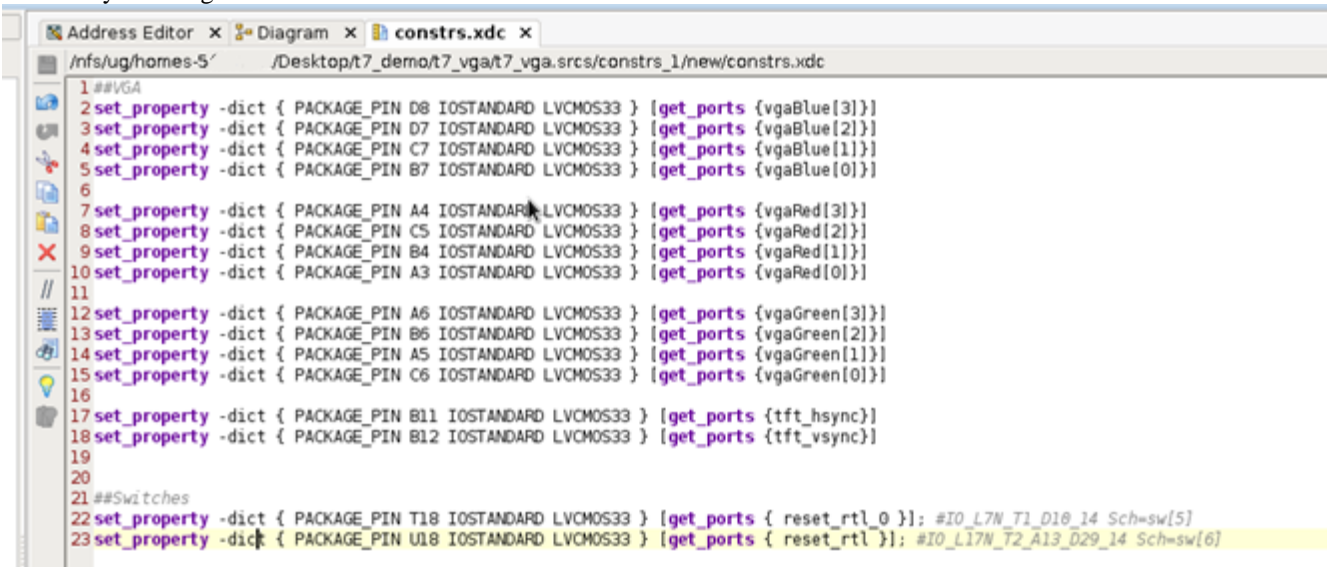
8. Click **Validate Design**. The complete design should look like:



The Address Map should look like:

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
microblaze_0_axi_intc	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_tft_0	S_AXI_MM	Reg	0x44A0_0000	64K	0x44A0_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
axi_tft_0					
Video_data (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
Excluded Address Segments (3)					

9. Create a new constraint file and enter the following port assignments. Also note to modify the external port names based on your design.



```
Address Editor x Diagram x constrs.xdc x
/Infs/ug/homes-5' /Desktop/t7_demo/t7_vga/t7_vga.srscs/constrs_1/new/constrs.xdc

1 ##VGA
2 set_property -dict { PACKAGE_PIN D8 IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[3]}]
3 set_property -dict { PACKAGE_PIN D7 IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[2]}]
4 set_property -dict { PACKAGE_PIN C7 IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[1]}]
5 set_property -dict { PACKAGE_PIN B7 IOSTANDARD LVCMOS33 } [get_ports {vgaBlue[0]}]
6
7 set_property -dict { PACKAGE_PIN A4 IOSTANDARD LVCMOS33 } [get_ports {vgaRed[3]}]
8 set_property -dict { PACKAGE_PIN C5 IOSTANDARD LVCMOS33 } [get_ports {vgaRed[2]}]
9 set_property -dict { PACKAGE_PIN B4 IOSTANDARD LVCMOS33 } [get_ports {vgaRed[1]}]
10 set_property -dict { PACKAGE_PIN A3 IOSTANDARD LVCMOS33 } [get_ports {vgaRed[0]}]
11
12 set_property -dict { PACKAGE_PIN A6 IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[3]}]
13 set_property -dict { PACKAGE_PIN B6 IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[2]}]
14 set_property -dict { PACKAGE_PIN A5 IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[1]}]
15 set_property -dict { PACKAGE_PIN C6 IOSTANDARD LVCMOS33 } [get_ports {vgaGreen[0]}]
16
17 set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports {tft_hsync}]
18 set_property -dict { PACKAGE_PIN B12 IOSTANDARD LVCMOS33 } [get_ports {tft_vsync}]
19
20
21 ##Switches
22 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { reset_rtl_0 }]; #IO_L7N_T1_D10_14 Sch=sw[5]
23 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { reset_rtl }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
```

10. Create **HDL wrapper**.

11. **Generate bitstream**.

12. Export hardware – **include bitstream**. And then, launch SDK. Now, we finally finished the hardware setup and move to the software design, which is also very interesting.

## 6. SDK Configuration

1. Open SDK and create a new project (Xilinx Application). Make sure it is an **empty** application.
2. Right click the new project to create a new C source file and name it **main.c**.
3. Copy and paste the provided code into **main.c**. Try to understand the code. Make sure that the **TFT base address** and the **DDR2 base address** are matching to your hardware address.

```
#define X_MAX 640
#define Y_MAX 480

typedef struct color_struct {
    char R;
    char G;
    char B;
} color;

void drawFilledRect(int* DDR_addr, int top_left_x, int top_left_y, int length,
    int width, color c);
void drawFilledCircle(int* DDR_addr, int cent_x, int cent_y, int radius,
    color c);
void clearScreen(int* DDR_addr);
void writeRGB(int* DDR_addr, int x, int y, color c);
void setVideoMemAddr(volatile int* TFT_addr, int* DDR_addr);
void disableVGA(volatile int* TFT_addr);
void enableVGA(volatile int* TFT_addr);

int main() {
    int* DDR_addr = (int*)0x80000000;
    volatile int* TFT_addr = (int *) 0x44a00000;

    int x_pos, y_pos, delay_counter, old_x_pos = 0, old_y_pos = 0;
    int radius = 20;
    int speed = 8;

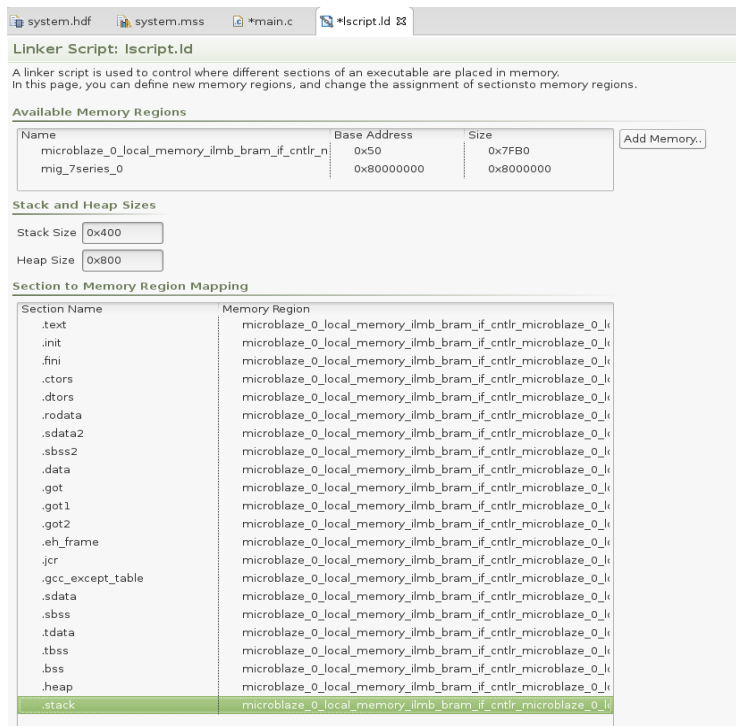
    color black = { .R = 0x0, .G = 0x0, .B = 0x0 };
    color lettercolor = { .R = 0x0, .G = 0xa, .B = 0xa };
    color circlecolor = { .R = 0xd, .G = 0x4, .B = 0x9 };

    // screen setups
    disableVGA(TFT_addr);
    setVideoMemAddr(TFT_addr, DDR_addr);
    clearScreen(DDR_addr);
    enableVGA(TFT_addr);

    // "E"
    drawFilledRect(DDR_addr, 50, 95, 160, 60, lettercolor);
    drawFilledRect(DDR_addr, 50, 215, 160, 60, lettercolor);
    drawFilledRect(DDR_addr, 50, 335, 160, 60, lettercolor);
    drawFilledRect(DDR_addr, 50, 95, 45, 300, lettercolor);

    // "C"
    drawFilledRect(DDR_addr, 240, 95, 60, 300, lettercolor);
    drawFilledRect(DDR_addr, 240, 95, 160, 80, lettercolor);
    drawFilledRect(DDR_addr, 240, 315, 160, 80, lettercolor);
```

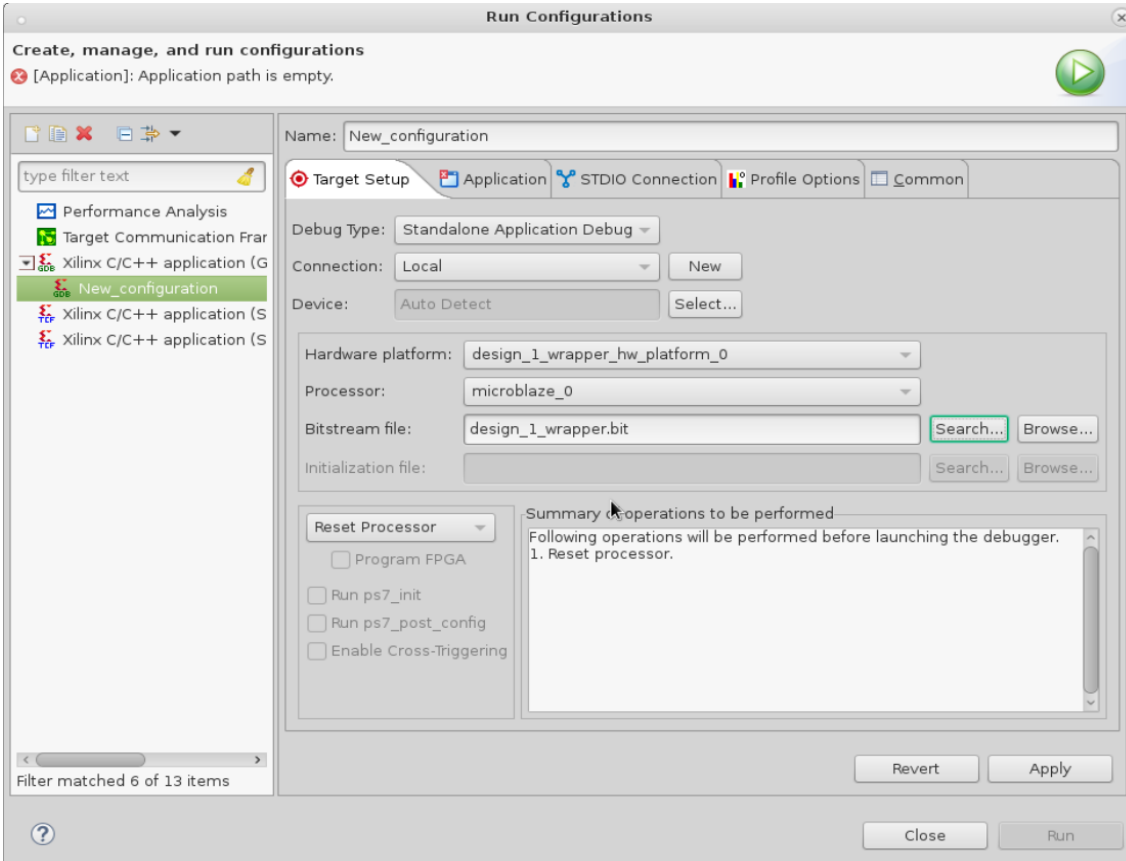
4. Open the **lscript.ld** (linker script) under src – change all Memory Region Mapping to local memory as shown.
5. Save the main.c and the linker file and build the program.



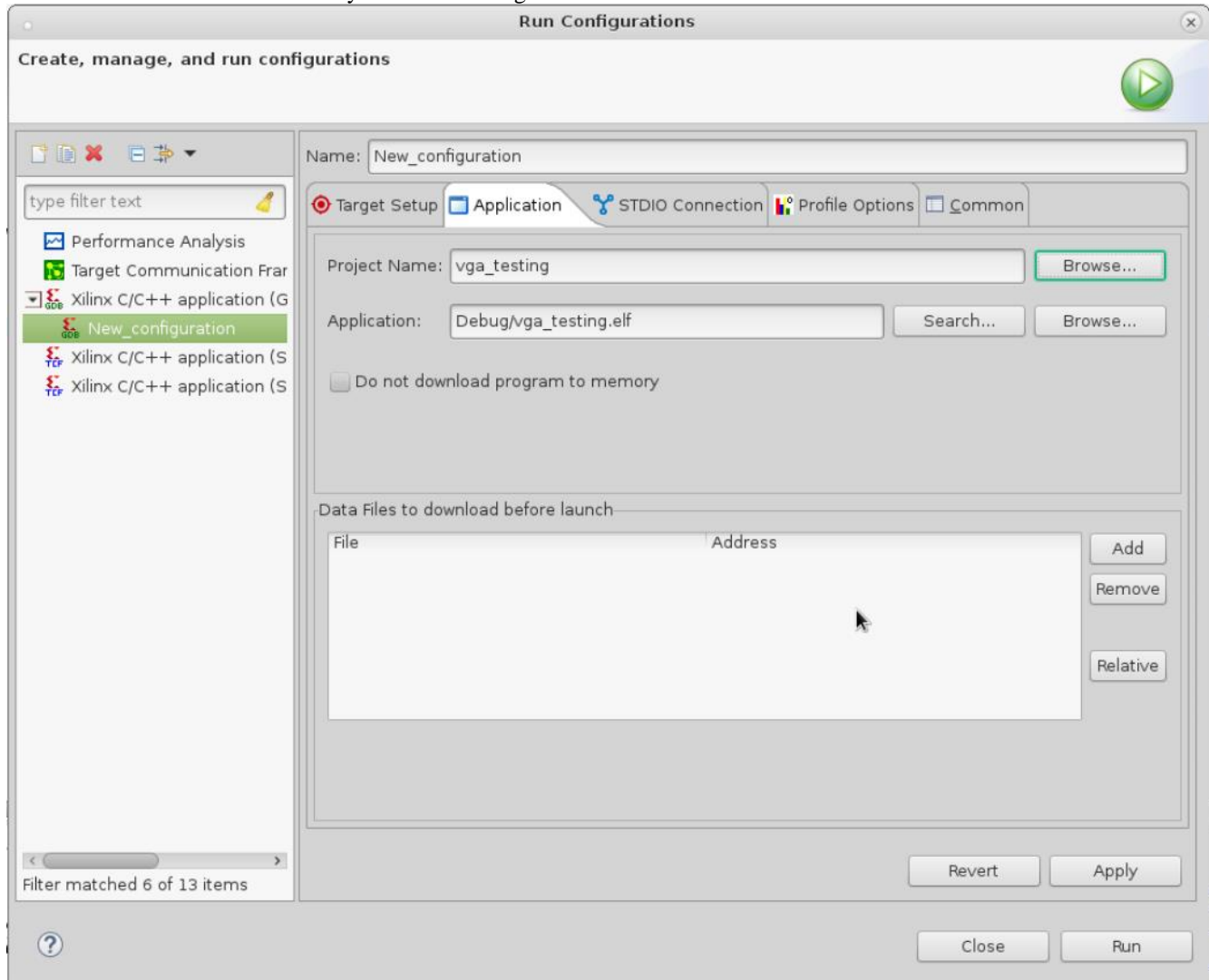
6. Program FPGA to load the hardware into the board.

7. Under Run, click Run configuration. Make SW5 active high or else the CPU will be in reset mode.

8. In **Target Setup**, search for your bitstream file. Choose **wrapper.bit** and click Ok.



8. In **Application**, browse the Project Name and choose your project. In this example it is vga\_testing. The Application file should be selected automatically under the Debug folder.



9. Go through the given source code (**main.c**) and try to draw some other shapes on the screen.

## Further notes on the AXI TFT CONTROLLER

There are three registers that you must write to in order to configure the AXI TFT controller. The first is the address register, which is offset 0x00 from the address of the controller. Here you write the address of the memory space that the video memory is located. This is to allow the controller to access the video memory. Only the eleven most significant bits matter (but the memory space must be aligned to a 2M boundary)



Figure 2-1: Address Register

The other register is the TFT control register. Only the least two significant bits of this register are used to set the controller. Bit 0 is called the TDE bit. Writing '1' to this location will enable the TFT display, while writing '0' will disable it. Bit 1 is called the DPS bit. Writing '1' to this location will cause the scan direction of the screen to be reversed (rotates the screen by 180 degrees). Writing '0' in this location will cause the normal display to be shown.

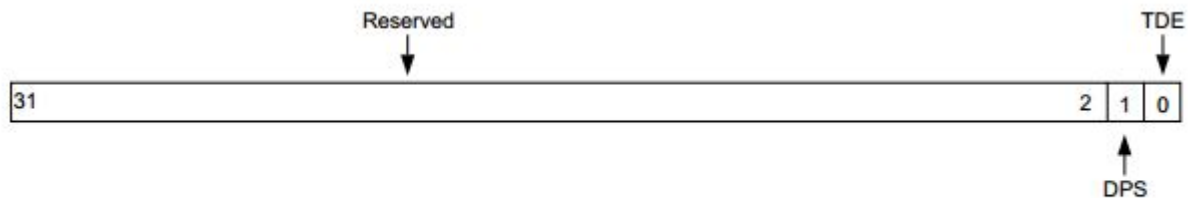


Figure 2-2: Control Register

The last register that can be configured for use is the interrupt enable and status register (IESR). This register contains the VSYNC interrupt enable bit as well as the status bit. These two bits work together to switch frames after the current frame is already displayed. If the VSYNC interrupt is enabled, the core generates an interrupt for the VSYNC pulse of every frame. For every rising edge of the VSYNC pulse, the core sets status bit to indicate that the core has displayed the current frame completely and accepted the new address from the AR.

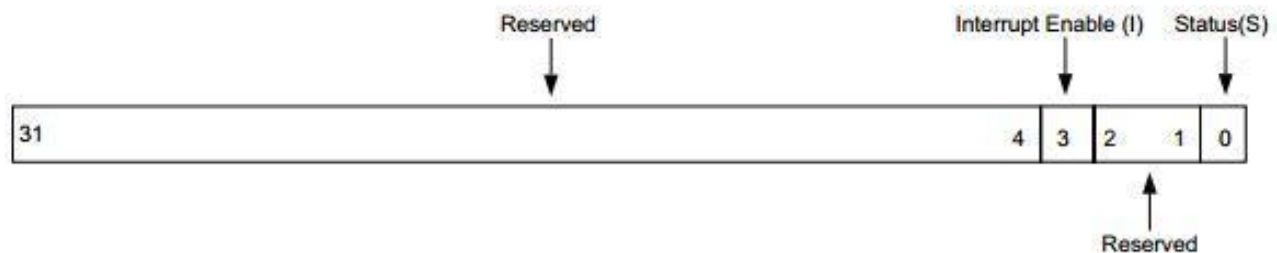


Figure 2-3: Interrupt Enable and Status Register