Laboratory 6 – VGA Applications

I. Colour the World

Objectives

- To build an application with VGA display
- To learn how to perform hardware and software combined design flow with Xilinx MicroBlaze

Activity Summary

1. Open the hardware design and generate bitstream
2. Launch Xilinx SDK after opening and exporting the hardware platform
3. Program the FPGA, setup Run Configuration, and run the application

Introduction

You probably have been wondering all along how the underlying hardware is changed to accommodate new changes in design requirement. For instance, the hardware platform used for the LED applications did not have the hardware driver for VGA display. That explains why you have a different hardware platform for each of the laboratory sessions. The hardware and software design flow for MicroBlaze usually follows the pattern shown in Figure 1. While the flow presents us with different possibilities, the paths we have chosen are enclosed in red lines as shown in Figure 1. The MicroBlaze IP is added to the hardware design. The hardware platform is then exported to Xilinx SDK after synthesis, implementation and bitstream generation. The software is created, downloaded to the FPGA, and run or debugged in the SDK. **Note that in our case, the "Import Hardware Description" and "Import Hardware Implementation" in Figure 1 are actually done in the same Export Hardware step of the Vivado**.

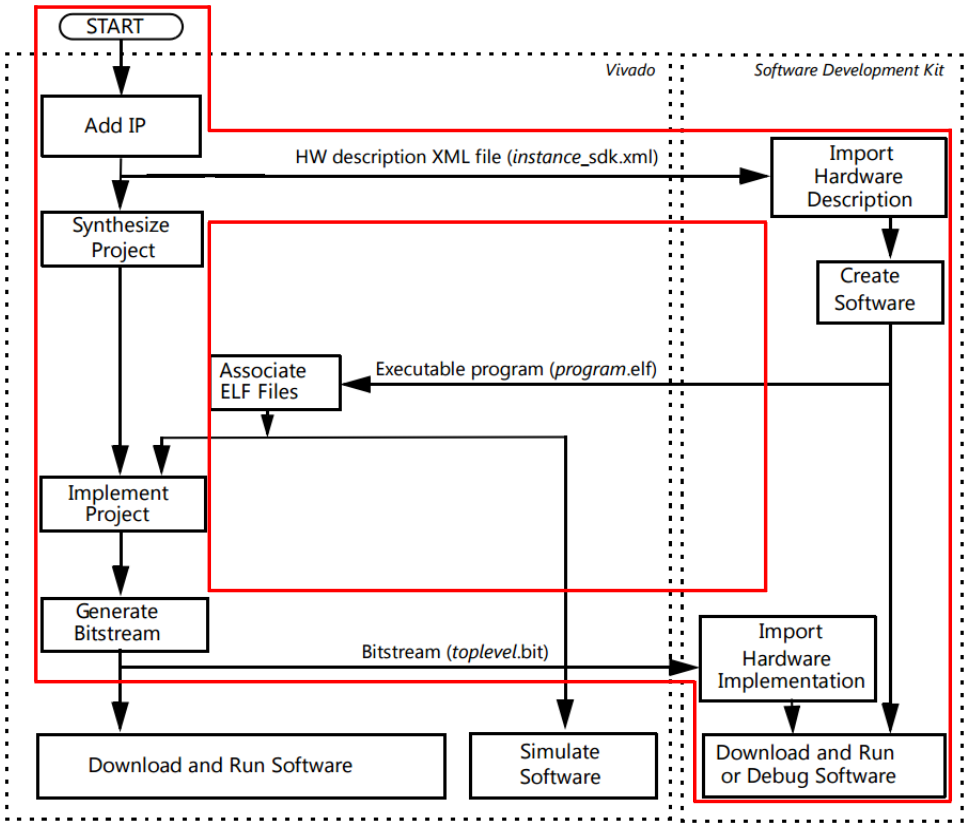


Figure 1: The generic Vivado tool flow for MicroBlaze

Activities

In this session a VGA driver has already been implemented for you in hardware, but the bitstream has not yet been generated. The MicroBlaze IP has been inserted using a block design. To see the block design, click on **Open Block Design** (the one enclosed in a green rectangle in Figure 2) under the **IP Integrator** in the **Flow Navigator** panel. The aim here is to show you how to display colours on the screen in hardware. You will not be writing any HDL codes. Our aim is to display a pattern on the monitor using the VGA input. On the Basys3 board 12 bits are used to drive the VGA colour signals. This gives us 4 bits per colour component (red, green, or blue). These colour components are assigned to the colour port of the VGA using the "colour" register in the Verilog code. The colour_in port in the Verilog code is connected to an XGpio object in software and the bit assignment is shown in Table 1. More on this later!

VGA Colour	R	R	R	R	G	G	G	G	B	B	B	B
`XGpio VGA_COLOUR`	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0

Table 1: VGA colour assignment

In order to modify the underlying hardware if necessary, you would have to open it and write some HDL codes (we have used the Verilog HDL) to describe the hardware. In Lab 1, you were shown the hardware design file tree structure in Vivado. Open the Vivado project for Lab 6. We will not be modifying any of the files, you will only be introduced to how the colours are printed to the screen. A Verilog code has already been written to display square regions on the VGA display. To see the code, in the **Sources** window under the **Hierarchy tab**, expand the folder structure and open VGA_Hardware (the one enclosed in a red rectangle in Figure 2) by double-clicking it.

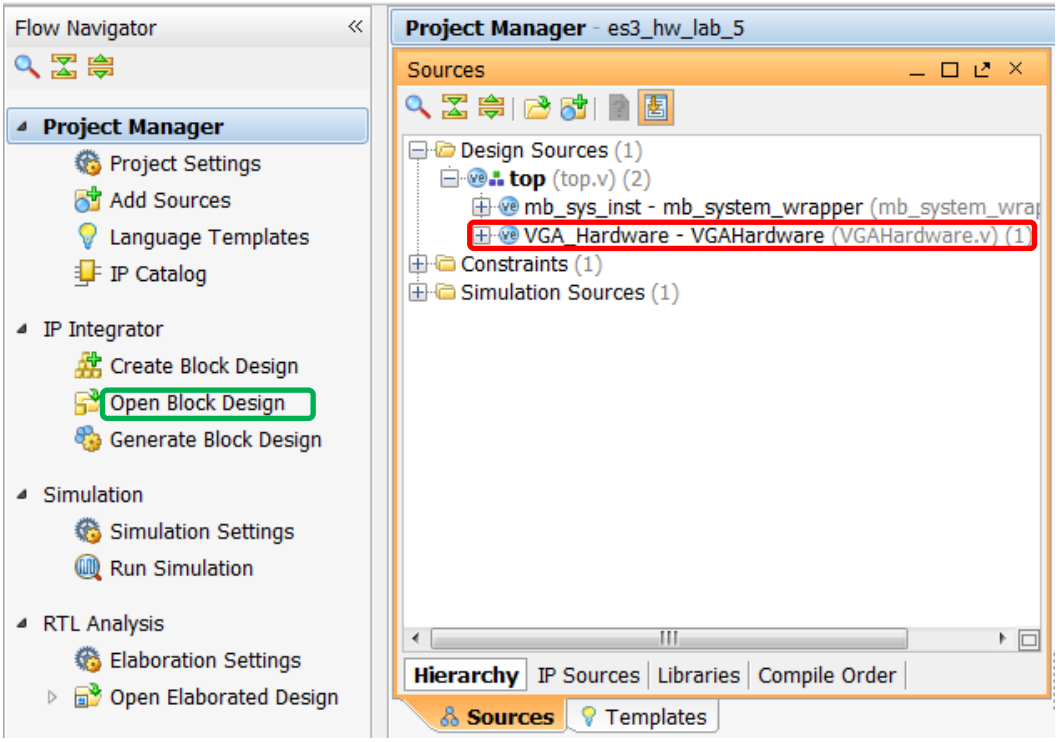


Figure 2

In the opened file, you should find the code shown in Figure 3. This hardware code helps to display different colours in different regions marked out on the screen as shown in Figure 4. Note the following:

- Three key hardware variables (signals) of note are **9-bit region**, **12-bit colour**, and **12-bit colour_in**.
- The region signal represents the VGA regions.
- The bit positions (0 to 8 from LSB to MSB) are mapped to the regions of the VGA with bit 0 selecting region 0, 1 selecting region 1 and up to region 8.
- This region signal can be written to from software (by using the `XGpio` object `VGA_REGION`) in order to select which of the regions to colour.
- The region mapping is shown in Table 2.
- A "1" written to a bit position selects the corresponding region. For example, in the line 5 of code in Figure 3, `region[0] == 1` means that region 0 is selected. Then line 6 contains `colour <= colour_in`, which passes the colour written from software to the VGA interface.
- The 12-bit **colour_in** signal determines the colour of a selected region and this can be controlled from software by using the the `XGpio` object `VGA_COLOUR`.
- The 12-bit colour is 4 bits each for red, green, and blue as shown in Table 3.
- The **colour_in** signal should not be confused with **colour** signal in Figure 3. The **colour** signal is used to interface to the colour pins of the VGA monitor while **colour_in** allows us to control the colour itself from software.
- As a final example, writing 0b100000001 to the `XGpio` object `VGA_REGION` and writing 0b000000001111 to the `XGpio` object `VGA_COLOUR` in software will select regions 0 and 8 and display the colour blue in them. An exercise on this will be done in the next section.

```
process(clk)
begin
    if rising_edge(clk) then
       if ((addrh = "0010100000") or (addrh = "0101000000") or (addrh = "0111100000") or (((addrv = "0010100000") or (addrv = "0101000000"))
       and (addrh <= "0111100000"))) then
            colour <= "111011101110";
       elsif ((addrh <= "0010100000") and (addrv <= "0010100000") and (region(0) = '1'))then
            colour <= colour_in;
       elsif ((addrh >= "0010100000") and (addrv <= "0010100000") and (addrh <= "0101000000") and (region(1) = '1')) then
            colour <= colour_in;
       elsif ((addrh >= "0101000000") and (addrv <= "0010100000") and (addrh <= "0111100000") and (region(2) = '1')) then
            colour <= colour_in;
       elsif ((addrh >= "0101000000") and (addrh <= "0111100000") and (addrv >= "0010100000") and (addrv <= "101000000") and (region(3) = '1')) then
            colour <= colour_in;
       elsif ((addrh >= "0101000000") and (addrh <= "0111100000") and (addrv >= "0101000000") and (addrv < "0111100000") and (region(4) = '1')) then
            colour <= colour_in;
       elsif ((addrh >= "0010100000") and (addrh <= "0101000000") and (addrv >= "0101000000") and (addrv < "0111100000") and (region(5) = '1')) then
            colour <= colour_in;
       elsif ((addrh <= "0010100000") and (addrv >= "0101000000") and (addrv < "0111100000") and (region(6) = '1')) then
            colour <= colour_in;
       elsif ((addrh <= "0010100000") and (addrv >= "0010100000") and (addrv <= "0101000000") and (region(7) = '1')) then
            colour <= colour_in;
       elsif ((addrh >= "0010100000") and (addrh <= "0101000000") and (addrv >= "0010100000") and (addrv <= "0101000000") and (region(8) = '1')) then
            colour <= colour_in;
       else
            colour <= "111111111111";
       end if;
    end if;
end process;
```

Figure 3

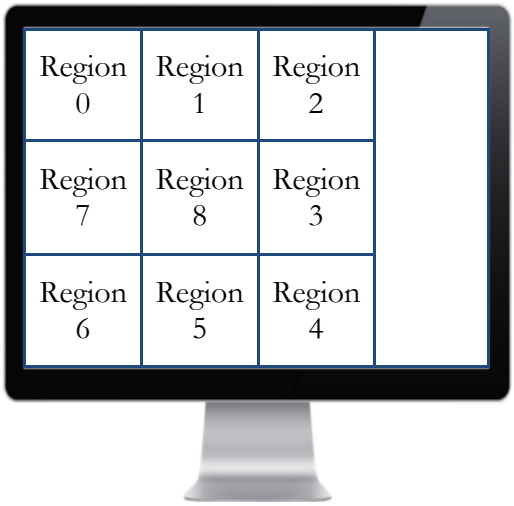


Figure 4

Region	8	7	6	5	4	3	2	1	0
`XGpio VGA_REGION`	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Table 2: VGA region selection mapping to `XGpio VGA_REGION` object

`XGpio VGA_COLOUR`	**Colour**	**Colour Bit Positions**
Bit 11	Red	3
Bit 10		2
Bit 9		1
Bit 8		0
Bit 7	Green	3
Bit 6		2
Bit 5		1
Bit 4		0
Bit 3	Blue	3
Bit 2		2
Bit 1		1
Bit 0		0

Table 3: VGA 12-bit colour mapping to `XGpio VGA_COLOUR` object

Now, take the following steps:

1. Click on **Flow** in the menu bar and select **Generate Bitstream**. You can as well click on **Generate Bitstream** under the **Program and Debug** section of the **Flow Navigator**. Note that for any modification done to any of the hardware codes, the bitstream has to be generated again
2. If the window in Figure 5 shows up select **Yes**. Wait for bitstream generation to be complete. You can monitor the progress by looking at the top-right corner of the Vivado window (see Figure 6). The window in Figure 7 may appear at the end of bitstream generation. If it does, simply click **Cancel**
3. **Export Hardware** and **Launch SDK** as usual
4. When the SDK opens, do not create any application yet; simply wait for the workspace to be built and then program the FPGA
5. Connect your board to the monitor with the VGA cable provided and select the alternate monitor display source using the "Source" button at the front of the monitor. The pattern in Figure 4 (without the text "Region" and the associated bit numbers) should be displayed on the monitor

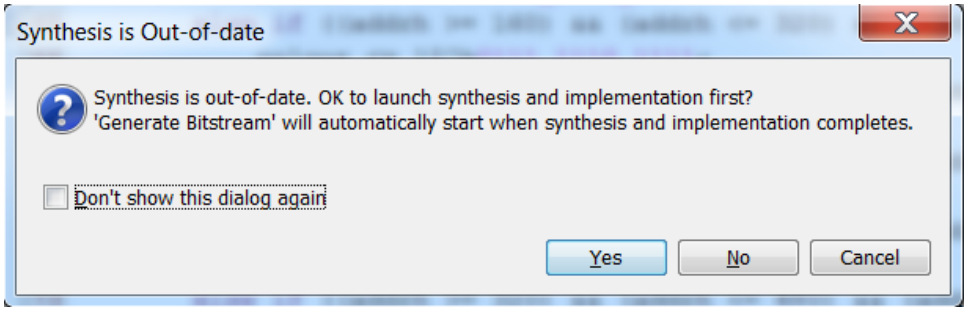


Figure 5

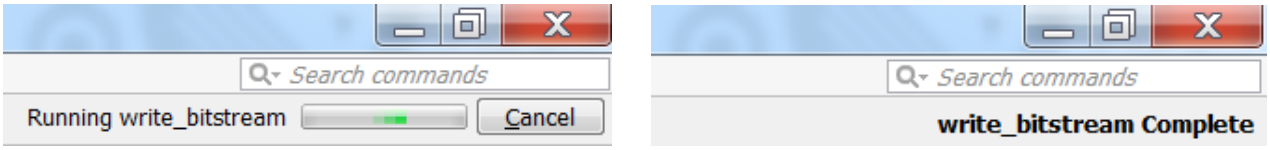


Figure 6

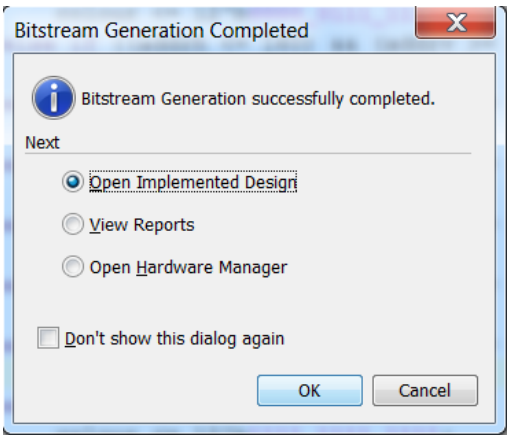


Figure 7

II. Colour the World in Software

Objectives

- To use the slide switch input to control VGA display

Activity Summary

1. Create a new application named ColourTheWorld
2. Write the application code
3. Program the FPGA (if necessary), setup Run Configuration, and run the application

Activities

The activities here involve reading from the slide switch inputs and using the value read to set the colour and the region values to be written to the VGA hardware driver. Note that regions are selected by the bit positions of the `VGA_REGION` variable according to the mapping in Table 2 in section I. `XGpio` objects called `VGA_REGION` and `VGA_COLOUR` should be declared and initialized. Look inside the `xparameters.h` file for the device IDs of the objects. The following steps are suggested:

1. Following from the previous section, create a new application called ColourTheWorld. However, if you have closed the SDK, simply go back to Vivado and relaunch it.
2. Create the necessary source and header files (main.c, gpio_init.c, and gpio_init.h).
3. Create the XGpio objects for slide switches, region and colour in the gpio_init.h and initialize them in the gpio_init.c file.
4. In the main(),
   - Call the initGpio() function
   - Declare two u16 variables `slideSwitchIn` and `region` and set them to 0.
   - In a while(1) loop, use 12 LSBs from the slide switches to set the `VGA_COLOUR`, and use the remaining 4 bits to set the `VGA_REGION`

     Note that because you have 4 slide switches for the region selection, you cannot directly map these switches to the 9-bit region selection. You need a code that maps the 4-bit switches to the 9-bit region selection object. You can use the code provided in Figure 8
5. Program the FPGA if necessary, setup Run Configuration and run the application
6. You should be able to colour different regions on the VGA monitor by setting region values from the slide switches

```
slideSwitchIn = XGpio_DiscreteRead(&SLIDE_SWITCHES, 1);

/* Write slideSwitch to VGA_COLOUR. Note that even though VGA_COLOUR
 * expects 12 bits, and slideSwitchIin is 16 bits, the processor
 * automatically picks the 12 LSBs and passes them to VGA_COLOUR. */
XGpio_DiscreteWrite(&VGA_COLOUR, 1, slideSwitchIn);

/* Shift slideSwitch to the right by 12 bits to extract the 4 MSBs
 * from the slide switches. This is used for region selection.*/
u16 region_selector = slideSwitchIn >> 12;

// Select region
switch (region_selector) {
    case 0:
        region = 0b000000000; // Select none of the regions
        break;
    case 1:
        region = 0b000000001; // Select region 0
        break;
    case 2:
        region = 0b000000010; // Select region 1
        break;
    case 3:
        region = 0b000000100; // Select region 2
        break;
    case 4:
        region = 0b000001000; // Select region 3
        break;
    case 5:
        region = 0b000010000; // Select region 4
        break;
    case 6:
        region = 0b000100000; // Select region 5
        break;
    case 7:
        region = 0b001000000; // Select region 6
        break;
    case 8:
        region = 0b010000000; // Select region 7
        break;
    case 9:
        region = 0b100000000; // Select region 8
        break;
    default:
        break;
}
XGpio_DiscreteWrite(&VGA_REGION, 1, region);
```

Figure 8

III. Advanced Colour the World

Objectives

- To display different patterns on the VGA screen

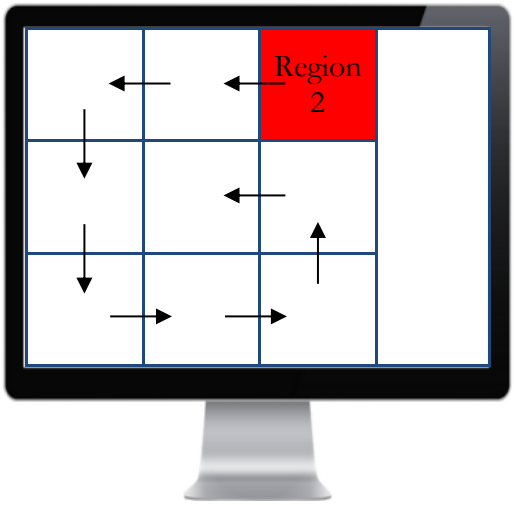
Activity Summary

1. Create a new application named ColourTheWorldAdv
2. Write the application code
3. Program the FPGA (if necessary), setup Run Configuration, and run the application

Activities

The activities here involve displaying different coloured patterns on the screen. Note that the hardware timer in this lab has a duration of 0.2 s. Create a new application named ColourTheWorldAdv and do the following:

1. Write codes to trace the letter "G" with moving coloured squares on the VGA display as depicted below. The colour of the square should also be changing as it moves from region to region. You are free to change the colours as you like. A simple addition of a constant number to the colour variable can be used. Moreover, remember that the coloured square has to move every 0.2 s. So, you need the interrupt for this part. Consult the previous labs on how to use the interrupt. You can reuse the **xinterruptES3.c** file you used in earlier labs but remember to add a declaration and definition for the **`hwTimerISR`** callback function.



2. Repeat (1) for an "O"