Extending Memory Space with Block RAM

Introduction

The Zynq device supports different types of memory including volatile (e.g. DDR3) and non-volatile (e.g. QSPI Flash). There are volatile and non-volatile hard memory controllers on the Zynq PS. The PL portion of the Zynq device has plenty of Block RAM (BRAM) which can be used by an IP without contending for external resources and creating performance bottleneck. This lab guides you through the process of extending the memory space in Zynq-based platform using available PL based BRAM resource.

Objectives

After completing this lab, you will be able to:

- Add BRAM and connect it to the processing system's AXI master port
- Execute the software application having data section in the BRAM

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Design Description

In this lab, you will add an AXI BRAM memory controller and associated 64 Kb BRAM memory to the system you created in the first lab. The following block diagram represents the completed design (**Figure 1**).

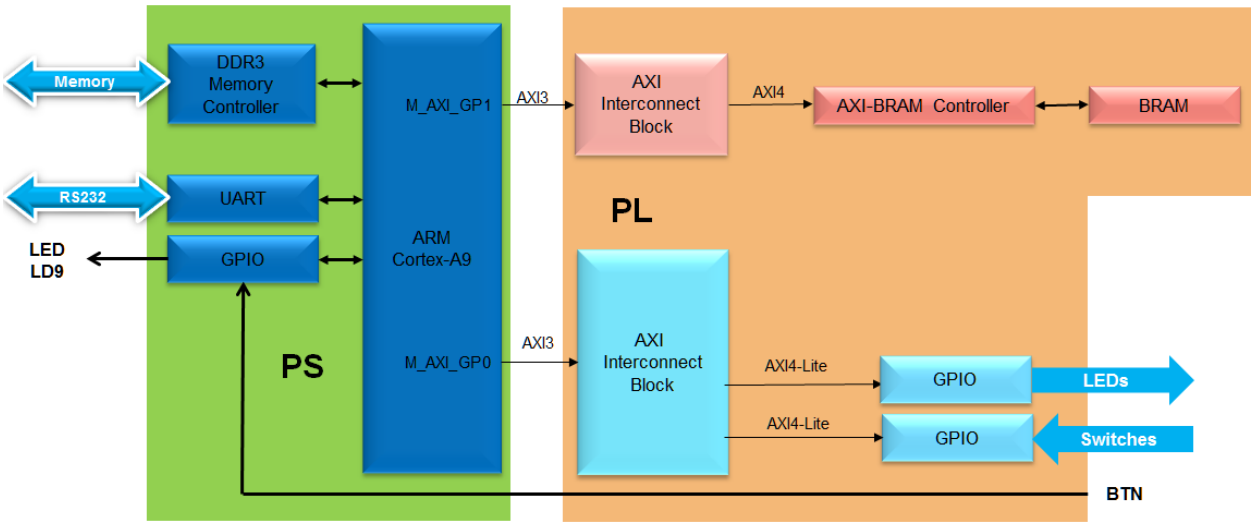
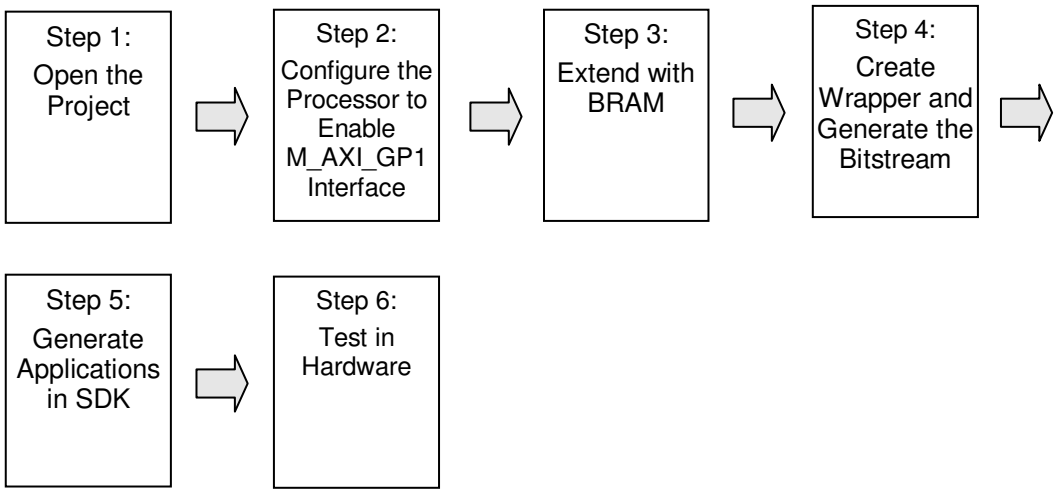


Figure 1 Completed Design

General Flow for this Lab



Open the Project Step 1

1-1. Open the Vivado program. Open the *lab1* project you created earlier or use the *lab1* project from the labsolution directory, and save the project as *lab3*.

1-1-1. Start Vivado if necessary and open either the lab1 project (lab1.xpr) you created earlier or the lab1 project in the *labsolutions* directory using the **Open Project** link in the Getting Started page.

1-1-2. Select **File > Save Project As …** to open the *Save Project As* dialog box. Enter **lab3** as the project name. Make sure that the *Create Project Subdirectory* and *Import All Files to the New Project* options are checked, the project directory path is **c:\xup\adv_embedded\labs** and click **OK**.

This will create the *lab3* directory and save the project and associated directory with lab3 name.

Configure the Processor to Enable M_AXI_GP1 Step 2

2-1. Open the Block Design and enable the M_AXI_GP1 interface.

2-1-1. Click **Open Block Design** in the *Flow Navigator* pane

2-1-2. Double-click on the *Zynq processing system* instance to open its configuration form.

2-1-3. Select *PS-PL Configuration* in the Page Navigator window in the left pane, expand *GP Master AXI Interface* on the right, and click on the check-box of the **M_AXI GP1 Interface** to enable it.

2-1-4. Select *Clock Configuration* in the Page Navigator window in the left pane, expand *PL Fabric Clocks* on the right, and click on the check-box of the **FCLK_CLK1** to enable it.

2-1-5. Enter the *Requested Frequency* for the **FCLK_CLK1** as **140.00000** MHz.

2-1-6. Click **OK** to accept the settings and close the configuration form.

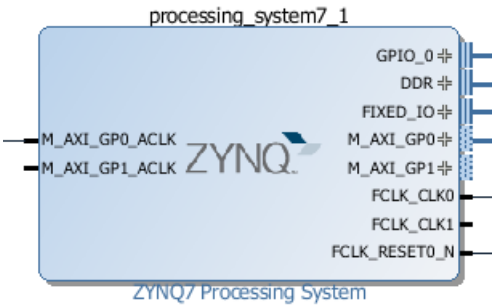


Figure 2 M_AXI_GP1 interface enabled

Extend with BRAM — Step 3

3-1. Add an AXI BRAM Controller instance with BRAM.

3-1-1. Click the Add IP icon and search for **BRAM** in the catalog.

3-1-2. Double-click the **AXI BRAM Controller** to add an instance to the design.

3-1-3. Click on **Run Connection Automation**, and select **axi_bram_ctrl_0**

3-1-4. Click **S_AXI**, and change the *Master* option to /**processing_system7_0**/**M_AXI_GP1**, change the *Clock Connection* to /**processing_system7_0**/**FCLK_CLK1** and click **OK**

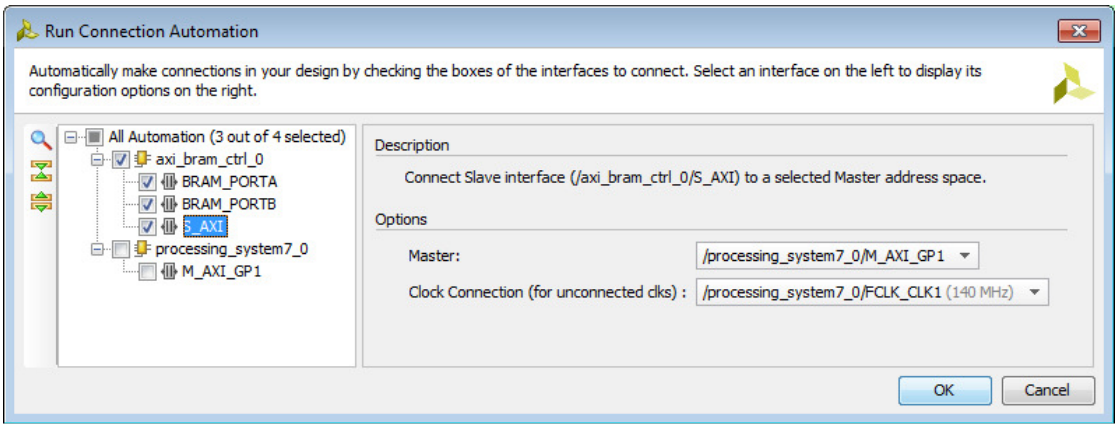


Figure 3 Connecting AXI BRAM Controller to M_AXI_GP1 to run at faster clock speed

Notice that an instance of AXI Interconnect is added, and the M_AXI_GP1_ ACLK is connected to FCLK_CLK1.

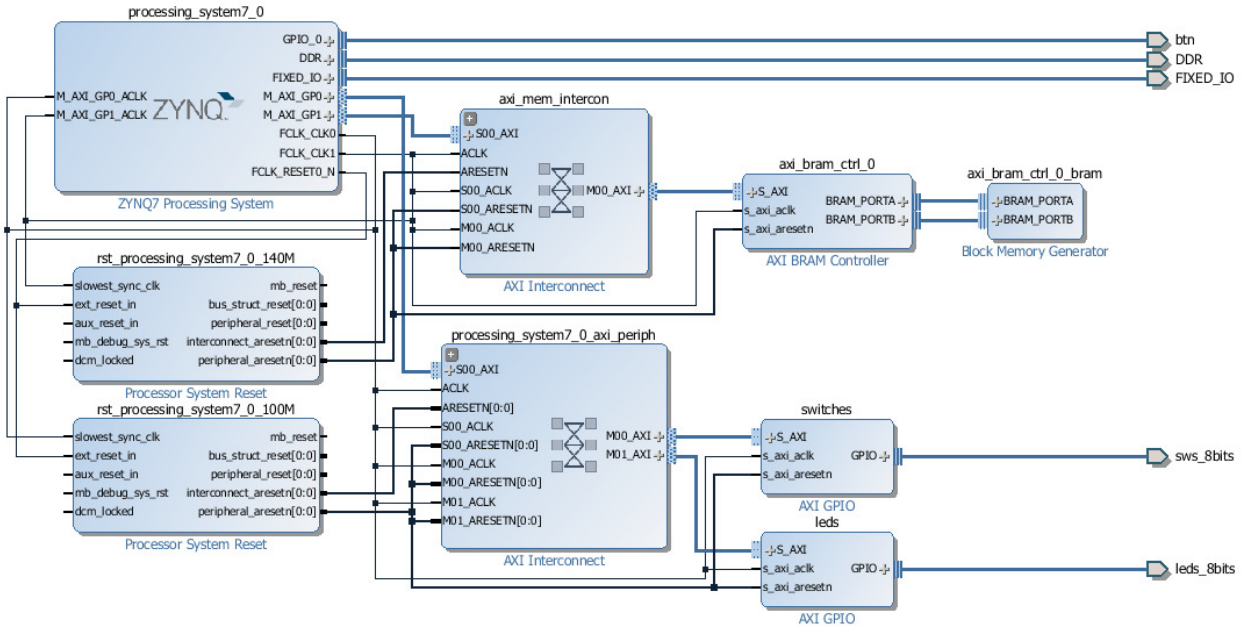


Figure 4 Clocking network connections

3-1-5. Double-click on the **axi_bram_ctrl_0** instance to open the configuration form.

3-1-6. Set the width to **64**.

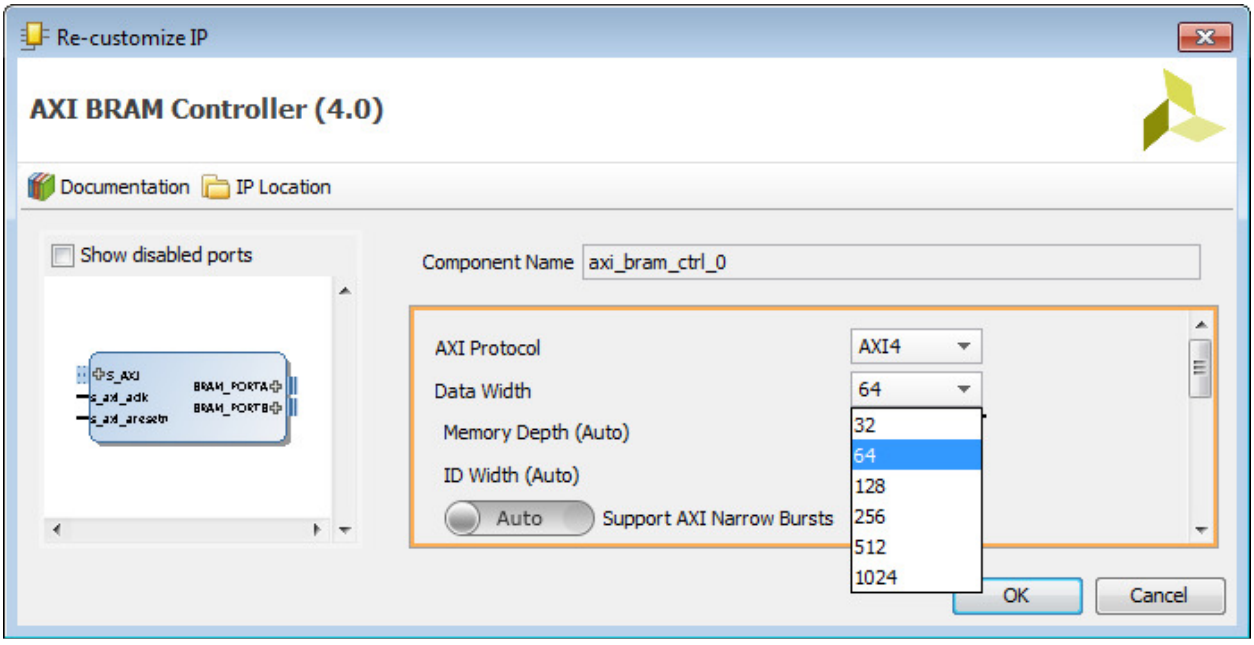


Figure 5 Setting the BRAM controller size to support 64KB

3-1-7. Click **OK**.

3-2. Using the Address Editor tab, set the BRAM controller size to 64KB. Validate the design.

3-2-1. Select the **Address Editor** tab and notice that the BRAM controller memory space is **4K**.

3-2-2. Click in the *Range* column of the *axi_bram_ctrl_1* instance and set the size as **64K**.

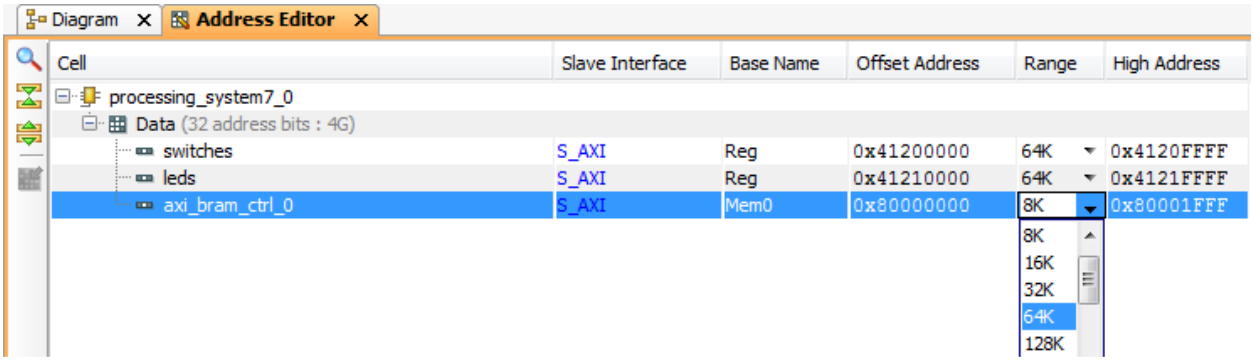


Figure 6 AXI BRAM space assignment

Notice that the address range changed to 0x80000000-0x8000FFFF. This is in the M_GP1 addressing space.

3-2-3. Select **Tools > Validate Design** and fix any errors if necessary.

Generate the Bitstream Step 4

4-1-1. Click on the **Generate Bitstream** to run the synthesis, implementation, and bit generation processes.

4-1-2. Click **Save** if prompted to save the project, and **Yes** to run the processes.

4-1-3. When the bitstream generation process has completed successfully, click **Cancel**.

Generate Applications in the SDK Step 5

5-1. Export the implemented design, and start SDK

5-1-1. Export the hardware configuration by clicking **File > Export > Export Hardware…**

5-1-2. Click the box to *Include Bitstream* and click **OK** (Click *Yes* if prompted to overwrite the previous module)

5-1-3. Launch SDK by clicking **File > Launch SDK** and click **OK**

5-1-4. Right-click on the **lab1** and **standalone_bsp_0** and **system_wrapper_hw_platfrom_0** projects in the Project Explorer view and select close project.

5-2. Create a hello_world application project using the standard template.

5-2-1. Select **File > New** > **Application Project.**

5-2-2. In the *Project Name* field, enter **hello_world** as the project name.

5-2-3. Use the default settings to create a new BSP and click **Next.**

5-2-4. Select the **Hello World** template and click **Finish.**

The hello_world and hellow_world_bsp projects will be created in the Project Explorer window of SDK

5-3. Create an empty application project, named lab3, and import the provided lab3.c or lab3.c file.

5-3-1. Select **File > New** > **Application Project.**

5-3-2. In the *Project Name* field, enter **lab3** as the project name.

5-3-3. Use the default settings to create a new BSP and click **Next.**

5-3-4. Select the **Empty Application** template and click **Finish.**

The lab3 and lab3_bsp projects will be created in the Project Explorer window of SDK.

5-3-5. Select **lab3>src** directory in the project view, right-click, and select **Import.**

5-3-6. Expand the **General** category and double-click on **File System.**

5-3-7. Browse to **c:\xup\adv_embedded\sources\lab3** folder.

5-3-8. Select **lab3.c** and click **Finish.**

A snippet of the source code is shown in the following figure. It shows that we write a pattern to the LED port, execute a software delay loop, and repeat for 256 times and repeat the process again. It also shows the code (greyed) which will be used in Lab5 which will execute the loop only sixteen times.

```
#include "xparameters.h"
#include "xgpio.h"
#ifdef MULTIBOOT
#include "xdevcfg.h"
#endif
//----------------------------------------------------

int main (void)
{

    XGpio led;
    int j=0;
    int i;

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&led, XPAR_LEDS_DEVICE_ID);

#ifndef MULTIBOOT
    while (1)
    {
      j=0;

      for(j=0; j<16; j++) {
          XGpio_DiscreteWrite(&led, 1, j);
          for (i=0; i<99999999; i++);
      }
    }

#else
      for(j=0; j<16; j++) {
          XGpio_DiscreteWrite(&led, 1, j);
          for (i=0; i<99999999; i++);
      }
      xil_printf("End of the program\r\n");
      print("Loading master image\r\n");
      // Driver Instantiations
      XDcfg XDcfg_0;
      u32 MultiBootReg = 0;
      #define PS_RST_CTRL_REG   (XPS_SYS_CTRL_BASEADDR + 0x200)
      #define PS_RST_MASK   0x1 /* PS software reset */
      #define SLCR_UNLOCK_OFFSET 0x08

      // Initialize Device Configuration Interface
      XDcfg_Config *Config = XDcfg_LookupConfig(XPAR_XDCFG_0_DEVICE_ID);
      XDcfg_CfgInitialize(&XDcfg_0, Config, Config->BaseAddr);

      MultiBootReg = 0; // Once done, boot the master image stored at 0xfc00_0000
      Xil_Out32(0xF8000000 + SLCR_UNLOCK_OFFSET, 0xDF0DDF0D); // unlock SLCR
      XDcfg_WriteReg(XDcfg_0.Config.BaseAddr, XDCFG_MULTIBOOT_ADDR_OFFSET, MultiBootReg); // write to multiboot
      // synchronize
        __asm__(
            "dsb\n\t"
            "isb"
        );
        // Generate soft reset
        Xil_Out32(PS_RST_CTRL_REG, PS_RST_MASK);
#endif
        return 0;
}
```

Figure 7 Source Code

Test in Hardware Step 6

6-1. Connect and power up the board. Establish the serial communication using the SDK Terminal tab. Program the FPGA. Run the hello_world.elf application.

6-1-1. Connect and power up the board.

6-1-2. In SDK, select **Xilinx Tools > Program FPGA** and **c**lick the **Program** button to program the FPGA.

6-1-3. Select the Terminal tab. If it is not visible then select **Window > Show view > Terminal**.

6-1-4. Click on to initiated the serial connection and select the appropriate COM port (depending on your computer). Configure it with 115200 baud rate.

6-1-5. Select **hello_world** in *Project Explorer*, right-click and select **Run As > Launch on Hardware** to download the application, execute ps7_init, and execute hello_world.elf

You should see "Hello World" displayed in the Terminal window.

6-2. Modify the linker script to use the BRAM for the data section and run it.

6-2-1. Select the **hello_world** application in the *Project Explorer* view.

6-2-2. Right-click and select **Generate Linker Script**.

6-2-3. Change the *data* segment memory to **axi_bram_ctrl_0.**

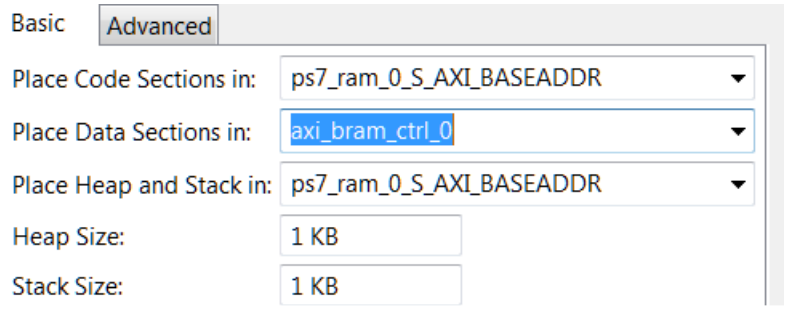


Figure 8 Assigning Data Segments to AXI BRAM

6-2-4. Click the **Generate** button.

6-2-5. Click the **Yes** button to overwrite.

6-2-6. Select the **hello_world** application, right-click, and run it.

You should see "Hello World" displayed in the Terminal window again, this time the data section is running from BRAM.

6-3. Run the lab3 application from the RAM memory.

6-3-1. Right-click *lab3,* select **Generate Linker Script**, verify the program is set to execute from *ram* and click **Cancel**.

6-3-2. Select the **lab3** project in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB).**

The application (lab3.elf) will be downloaded into the target device, execute ps7_init, and execute.

6-3-3. You should see the on-board LEDs changing patterns at roughly a one second delay rate.

6-3-4. Click the Terminate button ( )on the Console ribbon bar to terminate the execution.

6-4. Modify the linker scipt to use the BRAM for the data section and execute.

6-4-1. Select the **lab3** application in the *Project Explorer* view.

6-4-2. Right-click and select **Generate Linker Script**.

6-4-3. Change the *data* segment memory to **axi_bram_ctrl_0.**

6-4-4. Click the **Generate** button.

6-4-5. Click the **Yes** button to overwrite.

6-4-6. Select the **lab3** project in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB).**

The application (lab3.elf) will be downloaded into the target device, execute ps7_init, and will be executed.

6-4-7. Click the Terminate button ( )on the Console ribbon bar to terminate the execution.

6-4-8. Close the SDK program by selecting **File > Exit**.

6-4-9. Close the Vivado program by selecting **File > Exit.**

6-4-10. Turn OFF the power on the board.

Conclusion

This lab led you through adding BRAM memory in the PL section thereby extending the total memory space available to the PS. You have verified the functionality by creating an application, targeting the data section to the added BRAM, and executing the application.