Debugging Software Application

Introduction

This lab guides you through the steps involved in debugging a software application in SDSoC. SDSoC supports Standalone and Linux application debugging. SDSoC also provides the Dump/Restore Data File feature which can be used to dump a memory snapshot on a disk and restore the memory content from a pre-defined file. SDSoC also provides QEMU emulation capabilities which can be used for hardware/software debugging. In this lab you will go through the QEMU emulation flow.

Objectives

After completing this lab, you will be able to:

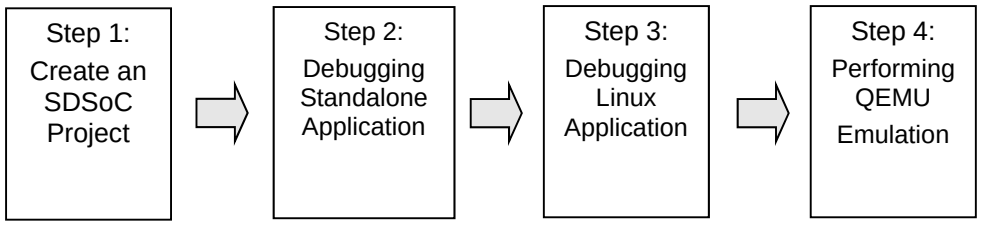
- Use the SDSoC environment to debug Standalone applications
- Use the SDSoC environment to debug Linux application
- Use the SDSoC environment to perform QEMU emulation

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.

This lab comprises three primary steps: You will create an SDSoC project, debug a Standalone application and debug a Linux application.

General Flow for this Lab



Create an SDx Project Step 1

You can execute Step 1 if you want to start from scratch otherwise skip to Step 2.

1-1. Launch SDSoC and create a project, called *lab5*, using Standalone OS and the *Empty Application* template targeting the Zed or Zybo board. Then add the provided source files.

1-1-1. Open SDx, if not already open

The Workspace Launcher window will appear.

1-1-2. Click on the Browse button and browse to **c:\xup\SDSoC\labs,** if necessary, and click **OK**.

1-1-3. Select **File > New > SDx Project** to open the New Project GUI.

1-1-4. Enter **lab5** as the project name.

1-1-5. Click **Next** to see *Choose Hardware Platform* window showing various available platforms.

1-1-6. Select either *zybo* or *zed* (depending on the board you are using) and click **Next.**

1-1-7. Select *Standalone* as the target OS, and click **Next**.

1-1-8. Select **Empty Application** and click **Next**.

1-1-9. Click **Finish**.

1-2. Import the provided source files from the source\lab5\src folder.

1-2-1. Right click on *src* under **lab5** in the Project Explorer tab and select **Import…**

1-2-2. Click on **File System** under *General category* and then click **Next**.

1-2-3. Click on the **Browse** button, browse to the *c:\xup\SDSoC\source\lab5\src* folder, and click **OK**.

1-2-4. Either select all the files in the right-side window or select *src* checkbox in the left-side window and click **Finish** to import the files into the project.

1-3. Mark sharpen_filter for the hardware acceleration. Setup for the Debug configuration.

1-3-1. Add *sharpen_filter* function for acceleration.

1-3-2. Right-click on *lab5* and select **Build Configurations > Set Active > Debug**

1-3-3. Right-click on **lab5** in the **Project Explorer** and select *C/C++ Build Settings*. Select **Miscellaneous** under *SDS++ Linker*, click "+" button of the *Other Options* and enter **–maxjobs <host core count>/2** (substitute <host core count> with the actual number of cores of your machine) and click **OK**

1-3-4. Right-click on *lab5* and select **Build Project**

The project will be built, generating the bit stream, and an SD card image. Since this will take about 40 minutes, we will import the pre-built project.

Debugging Standalone Application Step 2

Skip Step 2-1 if you are continuing from Step 1.

2-1. Import the pre-built lab5 project which has sharpen_filter marked for the hardware acceleration. Uncheck the bitstream generation and SD card image generation.

2-1-1. Select **File > Import**

2-1-2. Click on *Existing Projects into Workspace* under *General* and click **Next**.

2-1-3. Click on the **Browse** button of the *Select archive file* field, browse to *c:\xup\SDSoC\source\lab5*, select *lab5.zip* and click **Open**.

Make sure that *lab5* is checked in the *Projects* window.

2-1-4. Click **Finish**.

The project will be imported and **lab5** folder will be created in the *Project Explorer* tab.

2-1-5. Expand the **lab5** folder and double-click on the *project.sdx* entry.

The project file will be opened and the *sharpen_filter* function entry will be displayed in the *HW Functions* window.

2-1-6. **Uncheck** the *Generate Bit Stream* and *Generate SD Card Image* options as they are already generated.

2-2. Set the board to JTAG boot. Connect and power ON the board. Make terminal connection. Start the debug session. Step through 5 statements. Set a breakpoint on line 16 of the rgb_2_gray.c program.

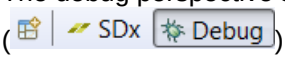
2-2-1. Set the board to JTAG boot. Connect the board and power it ON.

2-2-2. Either use the SDx Terminal tab or use third party terminal emulator program like TeraTerm, Putty, HyperTerminal. Make a connection to an appropriate COM port, setting 115200 baud rate.

2-2-3. Right-click on the **lab5** project in the *Project Explorer* tab, and select **Debug As > Launch on Hardware (SDSoC Debugger)**

The bitstream will be downloaded first to configure the board followed by the application download.

2-2-4. Click **Yes** to switch to the debug perspective, if asked.

The debug perspective should show up. If it doesn't then click on the Debug perspective () button.

Note that the program counter is at the main function entry point- line 75. In the *Debug* view you will see the same information. The *Variables* tab shows various variables visible in the current scope, the type, and their content.

2-2-5. Click on the *Breakpoints* tab and notice that two breakpoints are defined as default: (i) main and (ii) _exit

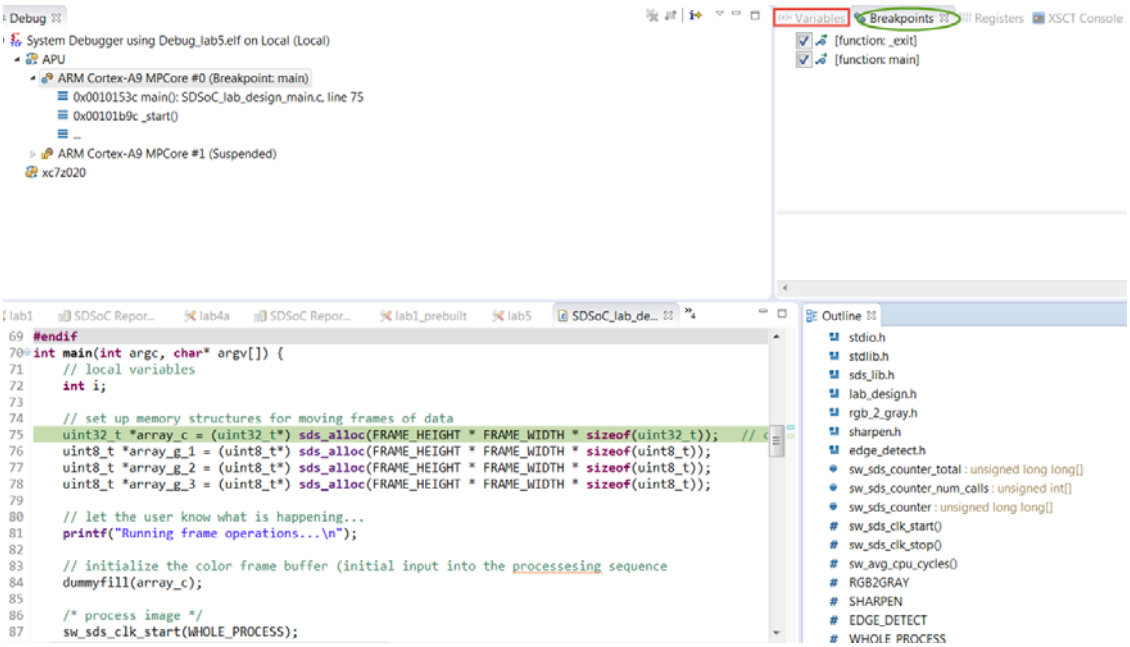
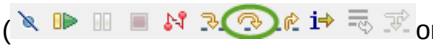


Figure 1. Debug perspective

2-2-6. Click on the Step Over button about five times ( or press F6) to execute the *printf* statement (line 81).

When the statement is executed, you will see a message is being printed in the Terminal tab.

2-2-7. Click on the SDx button on the top-right () to change to the SDx C/C++ perspective.

The Project Explorer will show up.

2-2-8. Expand **lab5 > src** and double-click on the *rgb_2_gray.c* entry to open the file.

2-2-9. Double-click in the left border of the line (line 19(Zed) or 16(Zybo)) to set the breakpoint.

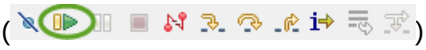
```
16 #ifndef SUPPRESS_RGB_2_GRAY_OPTIMIZATION
17 #pragma AP PIPELINE II = 1
18 #endif
19             index = (i * FRAME_WIDTH + j);
```

Figure 2. Set a breakpoint

2-2-10. Switch back to the Debug perspective by clicking on the **Debug** button.

2-2-11. Click on the **Breakpoints** tab and notice that another entry is added.

2-3. Continue with the execution. Inspect *index* variable. Observe memory content of *gr* variable changing.

2-3-1. Click on the **Resume** button () which will start executing until one of the breakpoints is encountered.

Note that the program stops at line 19(Zed) or 16(Zybo) of *rgb_2_gray*.

2-3-2. Click on the **Variables** tab and note the content of various variables. Select *index* and note the value (30) and its address 0x1016a044.

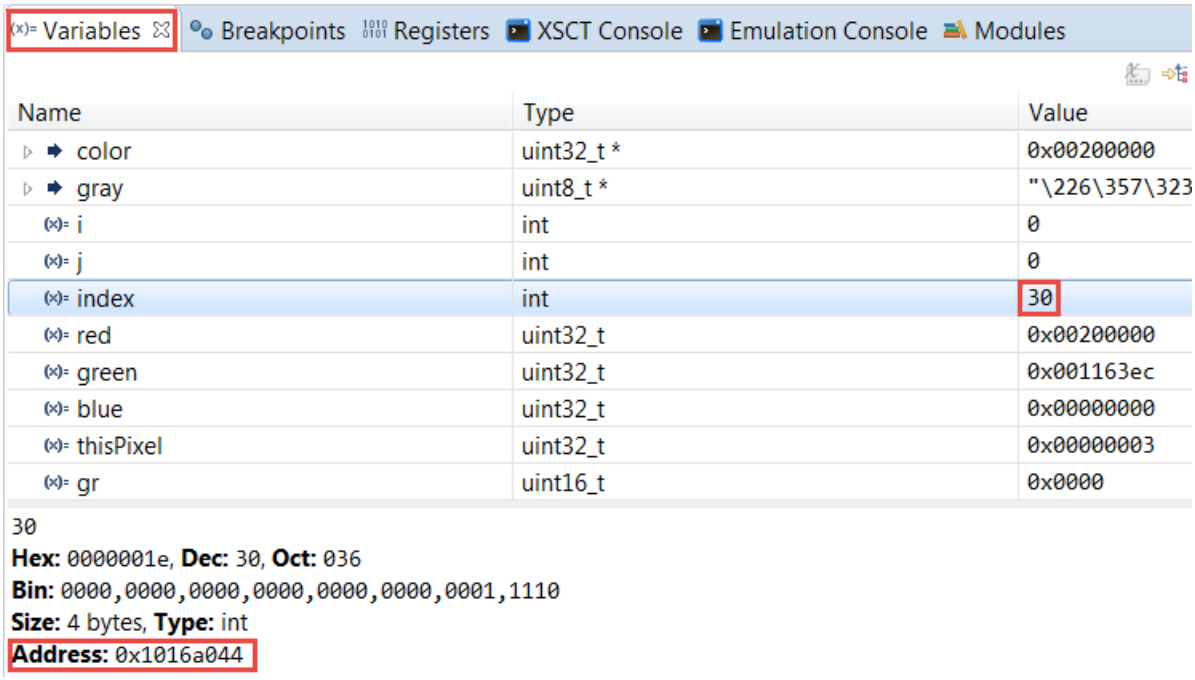


Figure 3. Variables content

2-3-3. Click on the *Step Over* button five times so that line 23 is highlighted. Note the variables content.

The *blue* variable is highlighted as that was the last variable whose content changed while executing line 26 statement.

Note the next statement which will be executed will compute the variable *gr*.

2-3-4. Select *gr* and note the value (0) and the address 0x1016a032.

2-3-5. You can see its content in the Memory tab also. Select the Memory tab and click on "+" to open up the *Monitor Memory* dialog box. Enter **0x1016a032** in the address bar and click **OK**.

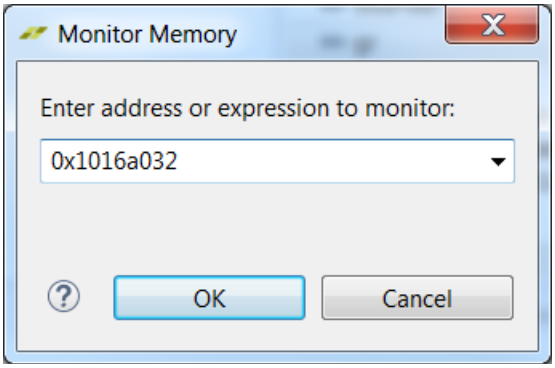


Figure 4. Monitor Memory window

The memory content will be displayed. The upper 16-bits represent the value.

2-3-6. Click on the *Step Over* button one more time and notice that the new value was computed and the memory content change is reflected. The variable tab's content also changed.

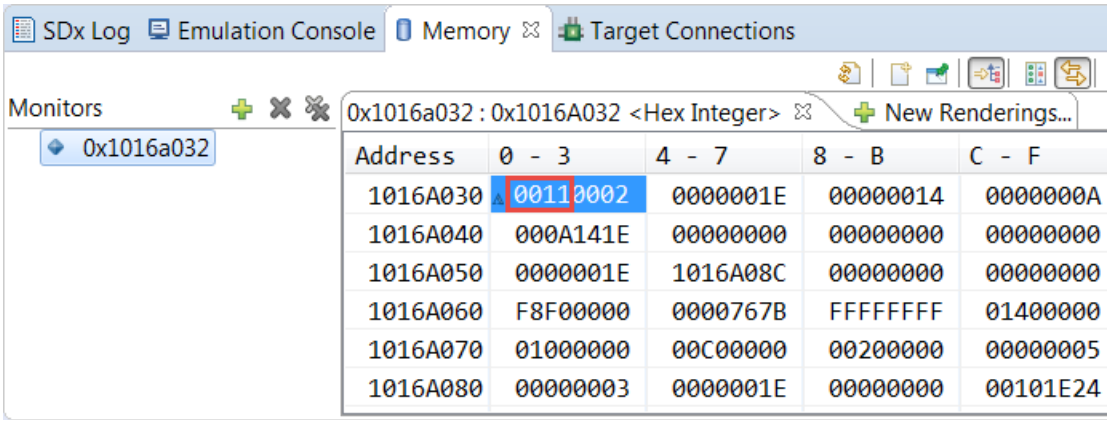


Figure 5. Variable gr's updated value in the Memory tab

2-3-7. Move the mouse close to *gray* array in line 28(Zed) or 25(Zybo) and notice that it is a pointer to an array of type unit8_t. The pointer is stored at 0x1016a028. The pointer value of which is *0x00C00000*.

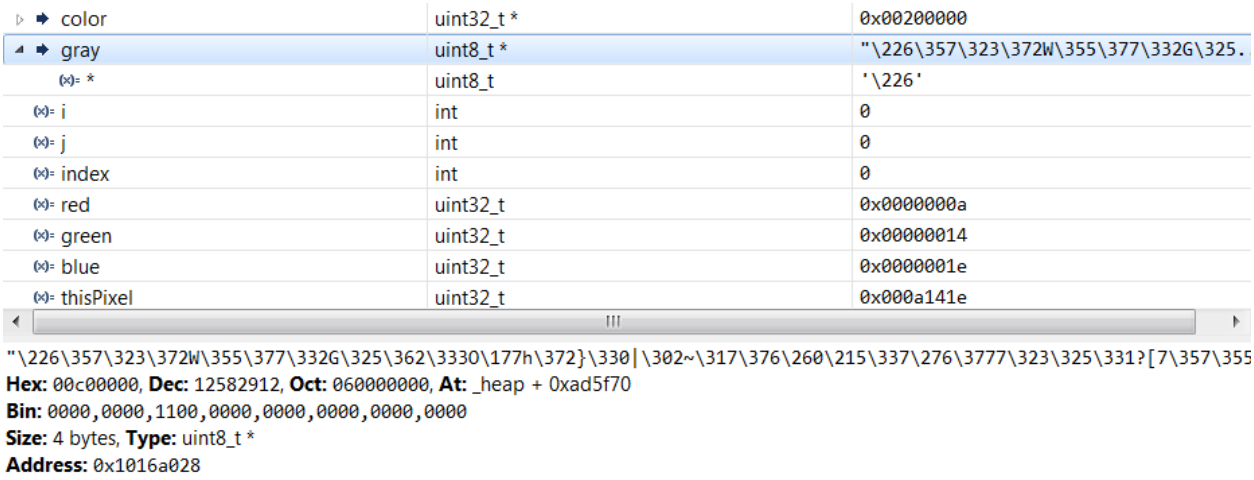


Figure 6. Array gray

2-3-8. Scroll up the Memory tab 1 line to view the contents of location 0x1016e598 and notice that it is pointing to 0x00c00000.

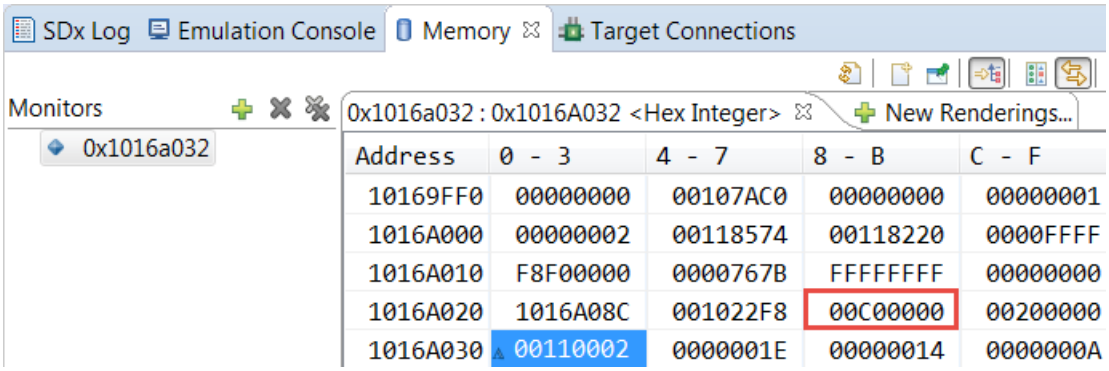


Figure 7. Pointer's content

2-4. Add 0x00C00000 (array_g_1 address) in the Memory tab, click the Resume button four times and observe the changing content. Remove the breakpoint set at line 19 and click the Step Return button to complete the function execution return to the main program.

2-4-1. Add **0x00C00000** in the Memory tab to view its content.

2-4-2. Click on *Resume* button four times and observe the array content changing.

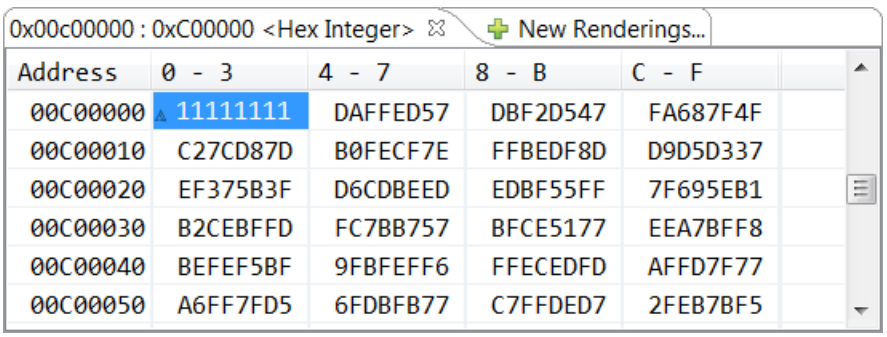
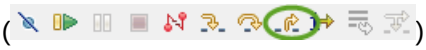


Figure 8. Array content changing

2-4-3. Select the **Breakpoints** tab and uncheck the rgb_2_gray.c – line 19 check box. This will disable the breakpoint.

2-4-4. Click on the **Step Return** button () to execute the function and stop on line 100 of the *SDSoC_lab_design_main.c* program (`_p0_sharpen_filter_1_noasync`).

2-4-5. Select the **Variables** tab and select *array_g_1*. Note its content and the address.

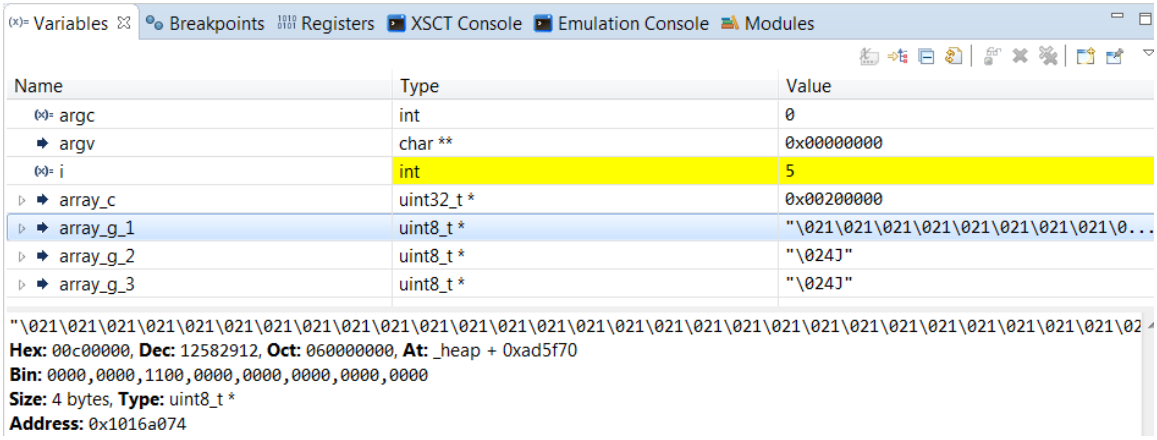


Figure 9. The processed content of array_g_1

2-5. Use Dump/Restore Data File feature of SDK to update the array_g_1's content with the provided binary data file stored in the source/lab5 directory.

After the color buffer has been converted to gray, you will replace the content of array_g_1 with the binary data provided to you in the *lab5_array_g_2.bin* file.

2-5-1. Select **Xilinx > Dump/Restore Data File**

2-5-2. Click the *Select* button, choose **Name=Xilinx Hardware Server** from the *Peers* section.

2-5-3. Expand the **APU** entry in the *Contexts* section and select **ARM Cortex-A9 MPCore #0**.

2-5-4. Click **OK**.

2-5-5. Click the **Browse** button, browse to *C:\xup\SDSoC\source\lab5*, choose **lab5_array_g_1.bin** and click **Save.**

2-5-6. Select the *Restore Memory* option as we want to read the file contents into the memory.

2-5-7. Enter **0x00C00000** in the *Start Address* field and **2073600** in the **Size** field**.**

Where 2073600 is the number of pixels (1920 x 1080).

2-5-8. Click **OK**.

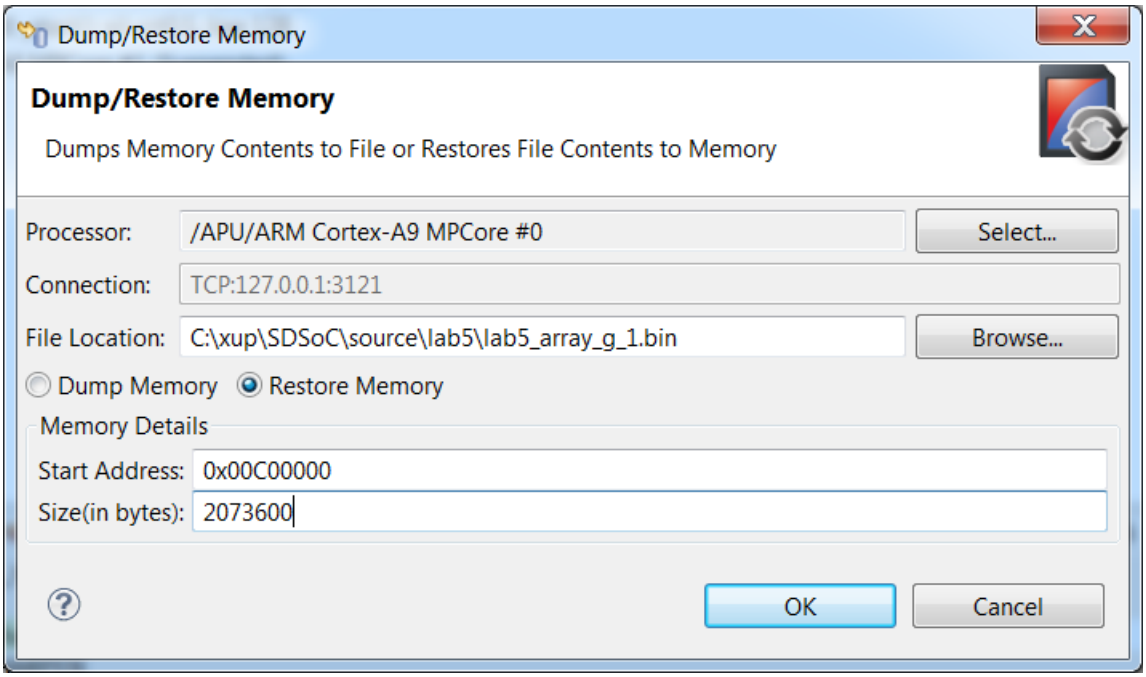


Figure 10. Updating memory content with a pre-created binary content

This will load the content into the array (you can see the progress in the SDK log window). You can see the updated content in the Memory tab.

Note that the next statement which will be executed will be using the hardware accelerator (line 100).

2-5-9. Click the **Step Over** button.

The *array_g_2* content will be updated due to the execution of the statement.

2-5-10. Click the **Disconnect** ( ) button to terminate the session.

Debugging Linux Application Step 3

For this portion of the lab, you will need an Ethernet port on the PC configured to 192.168.0.1 as an IP address and an Ethernet cable connected between the PC and the board.

You can execute Step 3-1 and Step 3-2 if you want to start from scratch otherwise skip to Step 3-3.

3-1. Create a new empty application project called lab5a targeting Linux OS. Import the provided source files from source\lab5\src folder

3-1-1. Select **File > New > Xilinx SDx Project** to open the New Project GUI.

3-1-2. Enter **lab5a** as the project name, select either *zybo* or *zed* (depending on the board you are using), select *Linux* as the target OS, select Empty Application and click **Finish**.

3-1-3. Right click on *src* under **lab5a** in the Project Explorer tab and select **Import…**

3-1-4. Click on **File System** under *General category* and then click **Next**.

3-1-5. Click on the **Browse** button, browse to *c:\xup\SDSoC\source\lab5\src* folder, and click **OK**.

3-1-6. Either select all the files in the right-side window or select *src* checkbox in the left-side window and click **Finish** to import the files into the project.

3-2. Mark sharpen_filter for the hardware acceleration. Build the Debug project.

3-2-1. Add the *sharpen_filter* in HW Function pane.

3-2-2. Right-click on *lab5a* and select **Build Configurations > Set Active > Debug**

3-2-3. Right-click on **lab5** in the **Project Explorer** and select *C/C++ Build Settings*. Select **Miscellaneous** under *SDS++ Linker*, click "+" button of the *Other Options* and enter **–maxjobs <host core count>/2** (substitute <host core count> with the actual number of cores of your machine) and click **OK**

3-2-4. Right-click on *lab5a* and select **Build Project**

The project will be build, generating bit stream, and the SD card image.

Since this will take about 35 minutes, we will import the pre-build project.

If you are continuing from Step 3-2, then skip Step 3-3.

3-3. Import the pre-built lab5a project which has sharpen_filter marked for the hardware acceleration. Uncheck the bitstream generation option.

3-3-1. Select **File > Import**

3-3-2. Click on *Existing Projects into Workspace* under *General* and click **Next**.

3-3-3. Click on the **Browse** button of the *Select archive file* field, browse to *c:\xup\SDSoC\source\lab5*, select *lab5a.zip* and click **Open**.

Make sure that lab5a is checked in the Projects window.

3-3-4. Click **Finish**.

The project will be imported and **lab5a** folder will be created in the *Project Explorer* tab.

3-3-5. Expand the **lab5a** folder and double-click on the *project.sdx* entry.

The project file will be opened and the *sharpen_filter* function entry will be displayed in the *HW Functions* window.

3-3-6. **Uncheck** the *Generate Bit Stream* option making sure that the *Generate SD Card Image* option is still checked.

3-4. Copy the sd_card content on the SD Card. Configure the board to boot from the SD card. Connect and power up the board. Setup the ip addresses both on the board and the PC Ethernet adaptor.

3-4-1. Using the Windows Explorer copy the content of the *lab5a* > *Debug* > *sd_card* onto the SD card. Place the SD card into the board.

3-4-2. Configure the board to boot from the SD card.

3-4-3. Connect the board, including network cable, and power it ON.

The board will boot.

3-4-4. Make the serial connection using the appropriate COM port.

3-4-5. Press the *PS-SRST* button on the board to reboot and notice Linux booting.

3-4-6. Once the board boot is complete, set the ip address of the board to 192.168.0.10 typing the following command at the Linux prompt:

ifconfig and note if any address is being assigned. If not assigned then execute the following command to assign to the correct Ethernet adaptor.

```
sh-4.3# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:01:22
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:501 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:53968 (52.7 KiB)  TX bytes:7866 (7.6 KiB)
          Interrupt:143 Base address:0xb000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

sh-4.3# ifconfig eth0 192.168.0.10
```

No IP address

Figure 11. Assigning an IP address

3-4-7. Using the control panel on the PC, configure the PC Ethernet adaptor with the static IP address to 192.168.0.1.

You can verify the connectivity by using *ping 192.168.0.1* command from the board's prompt.

3-5. Make target connection and start debugging the application.

3-5-1. In the *Target Connections* tab, expand *Linux TCF Agent* and double-click on **Linux Agent [default]**

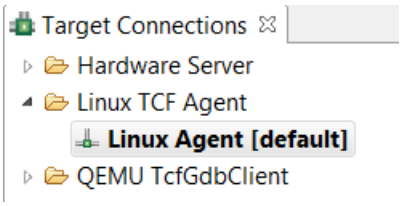


Figure 12. Accessing Linux Agent

Alternately, in the *Actions* panel, for the connection, click on the **New** button.

3-5-2. Enter **192.168.0.10** in the *Host* field and then click **OK** making sure that the Port field is set to **1534**.

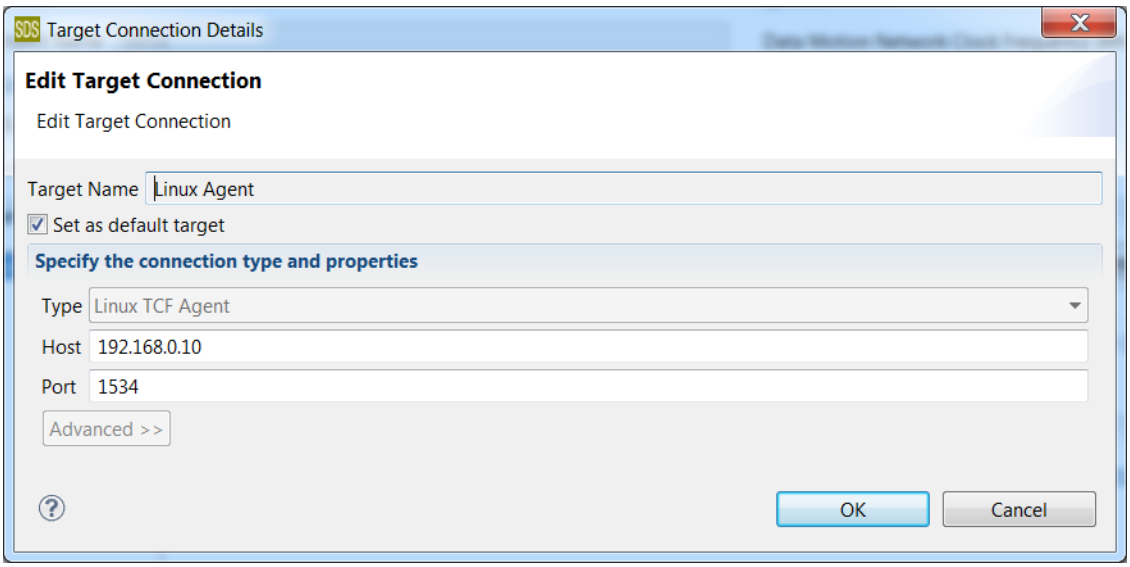


Figure 13. Making connection for Linux target

3-5-3. Right-click on *lab5a* project in the *Project Explorer* and select **Debug As > Launch on Hardware (SDSoC Debugger)**.

The connection will be made.

The debug perspective should show up. If it doesn't then click on the *Debug* perspective button.

Note that the program counter is at the main function - line 75. The *Variables* tab shows various variables visible in the current scope, the type, and their content.

3-5-4. Click on the **Step Over** button five times to execute the *printf* statement. When executed, you will see the message in the **Console** tab.

The variables tab will show various variables and arrays. Note that the value may be same as in the Standalone application but the addresses where they are defined will be different as the application is running under Linux.

3-5-5. Click on the SDx button on the top-right to change to the *SDx C/C++* perspective.

The Project Explorer will show up.

3-5-6. Expand **lab5a > src** and double-click on the *rgb_2_gray.c* entry to open the file.

3-5-7. Double-click in the left border of the line (line 19(Zed) or 16(Zybo)) to set the breakpoint.

3-5-8. Switch back to the *Debug* perspective by clicking on the **Debug** button.

3-5-9. Click on the **Resume** button which will start executing until one of the breakpoints is encountered.

3-5-10. Note that the program stops at line 19(Zed) or 16(Zybo) of *rgb_2_gray*.

3-5-11. Click Step Over button to execute the statement.

3-5-12. Select *index* and note the value (0) and its address `0xbe972c74`. Note the address may be different as MMU is used to translate virtual address into a physical address.

3-5-13. Click on the *Step Over* button four times such that line 26(Zed) or 23(Zybo) is highlighted. Note the variables content.

Note the next statement which will be executed will compute the variable *gr*.

3-5-14. Click Step Over button to execute the statement.

3-5-15. Select *gr* and note the value (0x0011) and the address `0xbe972c62`. Note the address may be different as MMU is used to translate virtual address into a physical address.

3-5-16. You can see its content in the Memory tab also. Select the Memory tab and click on "+" to open up the *Monitor Memory* dialog box. Enter the address and click **OK**.

The memory content will be displayed. The upper 16-bits represent the value.

Since MMU is used in Linux, you won't be able to see the content of the arrays and you won't be able to use the Dump/Restore Data File feature of SDx.

3-5-17. Remove the breakpoint and click Resume to execute the program to the completion.

This may take about 30 seconds.

3-5-18. Click on the Terminate button followed by click on the Disconnect button.

3-5-19. Turn OFF the board.

Performing QEMU Emulation Step 2

4-1. Create a project, called *lab5_qemu*, using Standalone OS and the *Matrix Multiplication and Addition* template targeting the Zed or Zybo board.

4-1-1. Select **File > New > SDx Project** to open the New Project GUI.

4-1-2. Enter **lab5_qemu** as the project name.

4-1-3. Click **Next** to see *Choose Hardware Platform* window showing various available platforms.

4-1-4. Select either *zybo* or *zed* (depending on the board you are using) and click **Next.**

4-1-5. Select *Standalone* as the target OS, and click **Next**.

4-1-6. Select **Matrix Multiplication and Addition (area reduced)** in case of *zybo* or **Matrix Multiplication and Addition** in case of *zed* as the source.

4-1-7. Click **Finish**.

4-2. Uncheck the Generate Bitstream and Generate SD Card image options. Check Generate emulation model option. Setup for the Debug configuration.

4-2-1. Notice that the mmult and madd functions are already targeted for hardware**.**

4-2-2. **Uncheck** the *Generate Bitstream* and *Generate SD Card image* options.

4-2-3. **Check** the *Generate emulation model* option.

4-2-4. Right-click on *lab5* and select **Build Configurations > Set Active > Debug**

4-2-5. Right-click on **lab5_qemu** in the **Project Explorer** and select *C/C++ Build Settings*. Select **Miscellaneous** under *SDS++ Linker*, click "+" button of the *Other Options* and enter **–maxjobs <host core count>/2** (substitute <host core count> with the actual number of cores of your machine) and click **OK**

4-2-6. Change *NUM_TESTS* value from *1024* to **1** in *main.cpp* file under the *lab5_qemu > src* folder.

This is to cut down the simulation time. Typically, this is how one would debug by reducing number of iterations and duration of execution to detect errors.

4-2-7. Right-click on *lab5_qemu* and select **Build Project**

The project will be built, generating the bit stream, and an SD card image. This will take about 15 minutes.

Skip to 4-4 if you are continuing from Step 4-2 above.

4-3. Import the pre-built lab5_qemu project which has mmult and madd functions marked for the hardware acceleration.

4-3-1. Select **File > Import**

4-3-2. Click on *Existing Projects into Workspace* under *General* and click **Next**.

4-3-3. Click on the **Browse** button of the *Select archive file* field, browse to *c:\xup\SDSoC\source\lab5*, select *lab5_qemu.zip* and click **Open**.

Make sure that *lab5_qemu* is checked in the *Projects* window.

4-3-4. Click **Finish**.

The project will be imported and **lab5_qemu** folder will be created in the *Project Explorer* tab.

4-3-5. Expand the **lab5_qemu** folder and double-click on the *project.sdx* entry.

The project file will be opened and the *mmult and madd* functions entries will be displayed in the *HW Functions* window.

4-3-6. Notice that the *Generate emulation mo*del option is checked.

4-4. Launch the QEMU emulation.

4-4-1. Right-click the project name and select **Debug As** > **Launch on Emulator (SDSoC Debugger)**

4-4-2. Click **Yes** in the Emulation dialog box to start the emulator.

4-4-3. Ensure that the **Show Waveform (Programmable Logic Only)** option is enabled.

4-4-4. Click **Start** to start the emulator in the Emulation dialog box.

4-4-5. Click **Yes i**n the Confirm Perspective Switch dialog box.

This will launch the Vivado simulator and simultaneously the program stops at the first executable code of the `main()` function in SDx IDE.

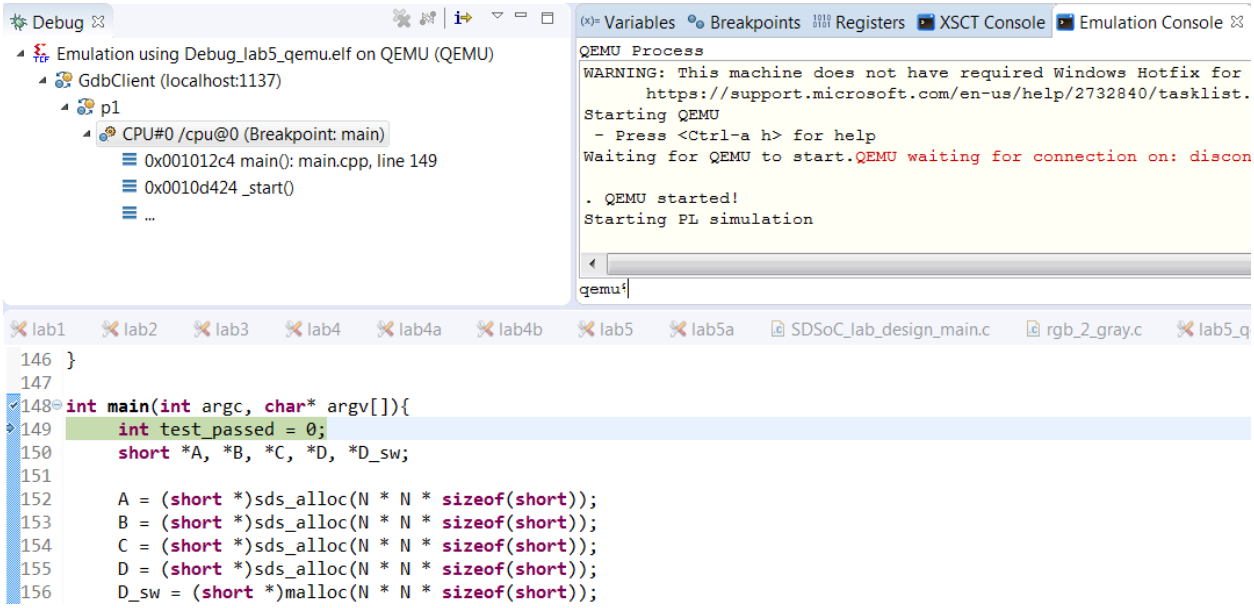


Figure 14. Debug perspective in SDx

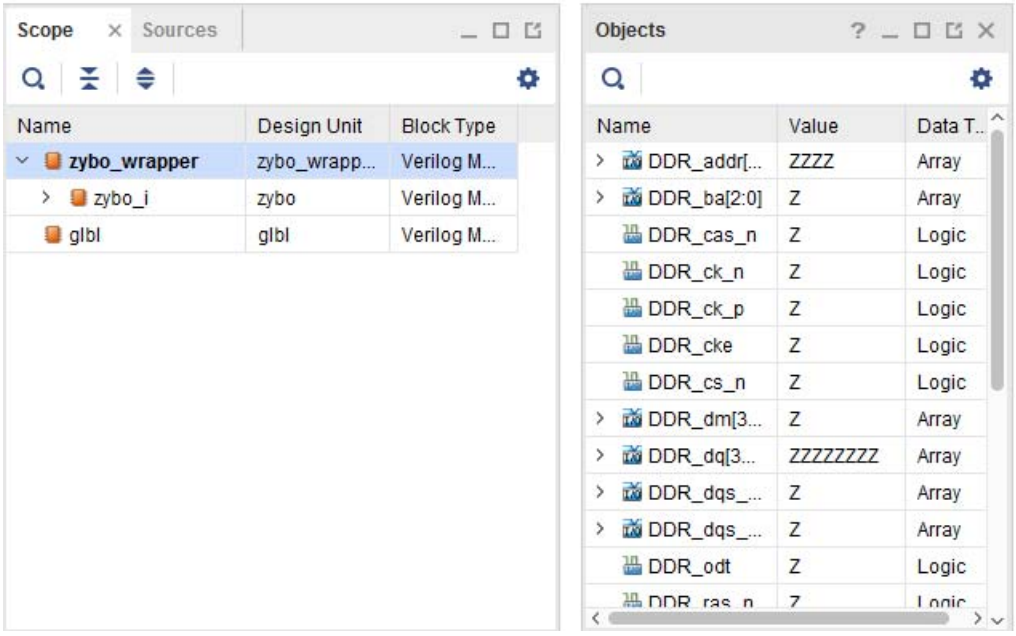


Figure 15. Vivado simulator view

4-4-6. Set the breakpoint at the code `std::cout << "TEST " << (test_passed ? "FAILED" : "PASSED") << std::endl;` (near line 167).

4-4-7. Select **File** > **Open Waveform Configuration** in the Vivado simulator.

4-4-8. Browse to *C:\xup\SDSoC\source\lab5* and select **zybo_wrapper_behav.wcfg**

4-4-9. Click **OK**.

4-5. Run the application and simulation.

4-5-1. Click **Resume** ( ) in the SDx IDE to run the application

4-5-2. Set the simulation time to **100 us** in the Vivado simulator

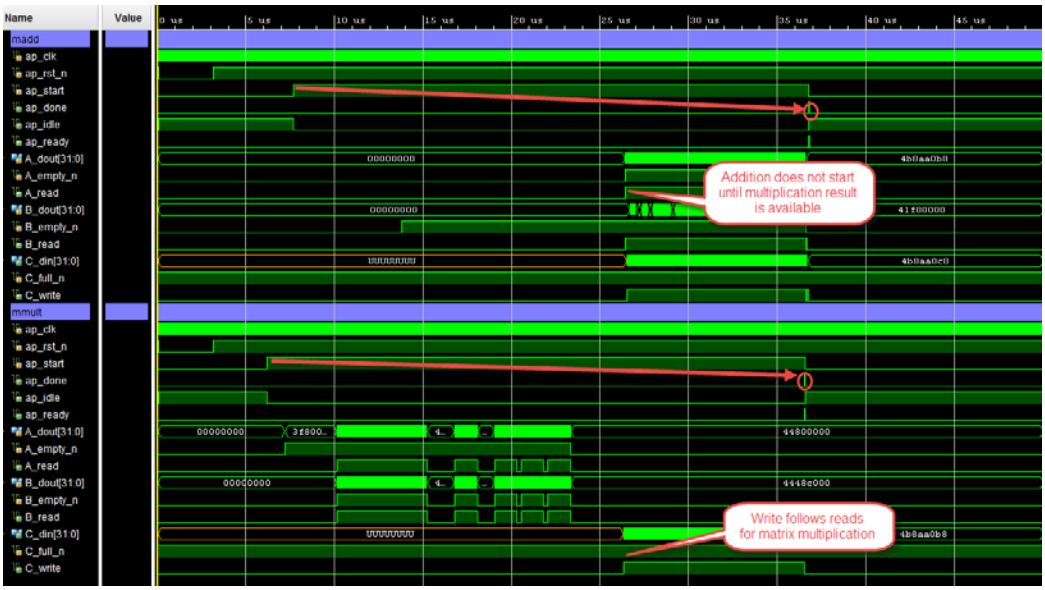
4-5-3. Click the **Run for** icon.

Wait for few minutes. Observe simulation progress.

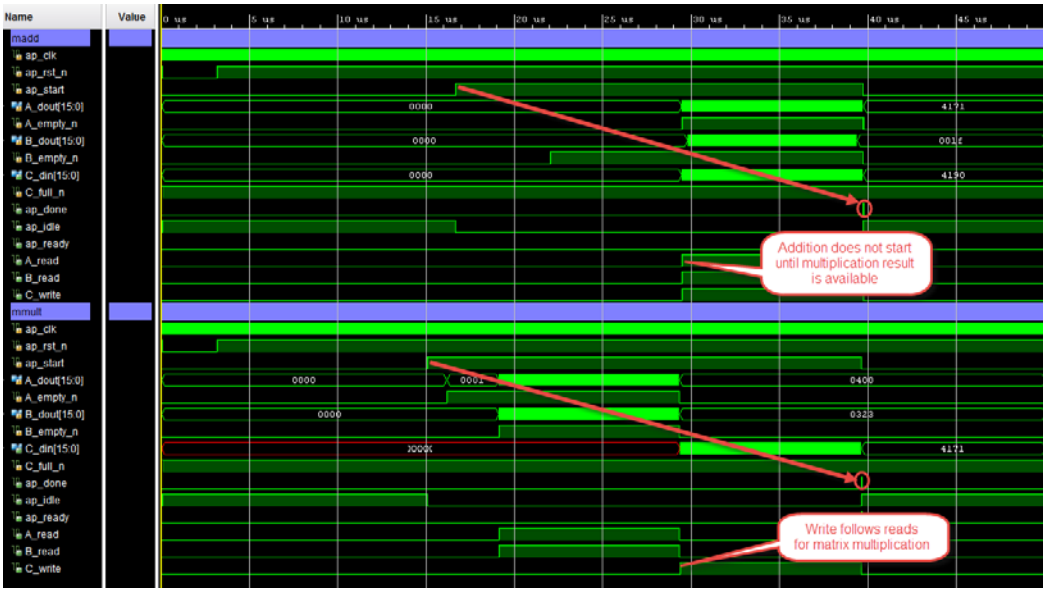
4-5-4. When RTL simulation is done, switch to SDSoC window and click Pause button followed by clicking Disconnect button.

4-5-5. Click the Zoom Fit ( ) icon in the waveform toolbar to view the full simulation waveform.

4-5-6. Zoom in using the icon to analyze the required signals.



(a) Zed



(b) Zybo

Figure 16. Vivado simulator output

4-5-7. Close the Vivado simulator and SDx IDE

Conclusion

In this lab, you debugged Standalone and Linux applications using SDx software debugger, and learned to how to perform QEMU emulation to perform hardware/software debugging. The Standalone application was debugged using JTAG connection whereas the Linux application was debugged over Ethernet. In Standalone application you were able to look into various arrays using the addresses and able to use the Dump/Restore Data File feature of SDSoC. In Linux application this was not possible as MMU translates the virtual addresses of arrays and pointers into physical addresses. The QEMU emulation debugging provides visibility to accelerator's internal activities which can be used for hardware/software debugging.