Estimating Accelerator Performance and Events Tracing

Introduction

This lab guides you through the steps involved in estimating the expected performance of an application when functions are implemented in hardware, without going through the entire build cycle. You will further analyze how data movement is taking place by inserting an events tracer.

Objectives

After completing this lab, you will be able to:

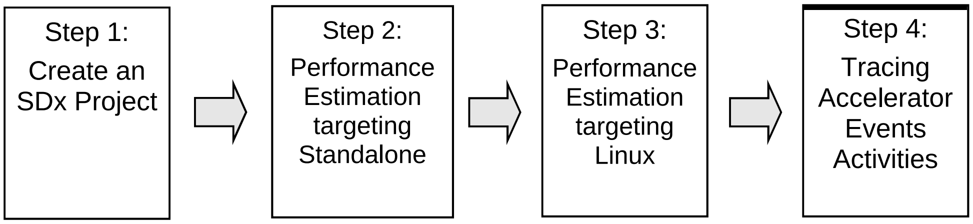
- Use the SDx environment to obtain an estimate of the speedup that you can expect from your selection of functions to accelerate
- Differentiate between the flows targeting Standalone OS and Linux OS
- Track various events taking place with respect to hardware accelerators

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 SDx project, estimate performance speedup targeting the Standalone OS and then estimate performance speedup targeting the Linux OS.

General Flow for this Lab


Step 1:
Create an SDx Project
Step 2:
Performance Estimation targeting Standalone
Step 3:
Performance Estimation targeting Linux
Step 4:
Tracing Accelerator Events Activities

Create an SDx Project Step 1

1-1. Launch SDx and create a project, called *lab4*, using the *Empty Application* template and then using the provided source files, targeting the Zed or Zybo board and Standalone OS.

1-1-1. Open SDx, select **c:\xup\SDSoC\labs** as the workspace and click **OK**.

1-1-2. Create a new project called **lab4**

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

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

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

1-1-6. Select **Empty Application** and click **Finish**.

1-2. Import the provided source files from source\lab4\src folder.

1-2-1. Right click on *src* under **lab4** 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 *c:\xup\SDSoC\source\lab4\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.

Performance Estimation Targeting Standalone Step 2

2-1. Mark sharpen_filter for the hardware acceleration. Run an initial performance estimate of the hardware only.

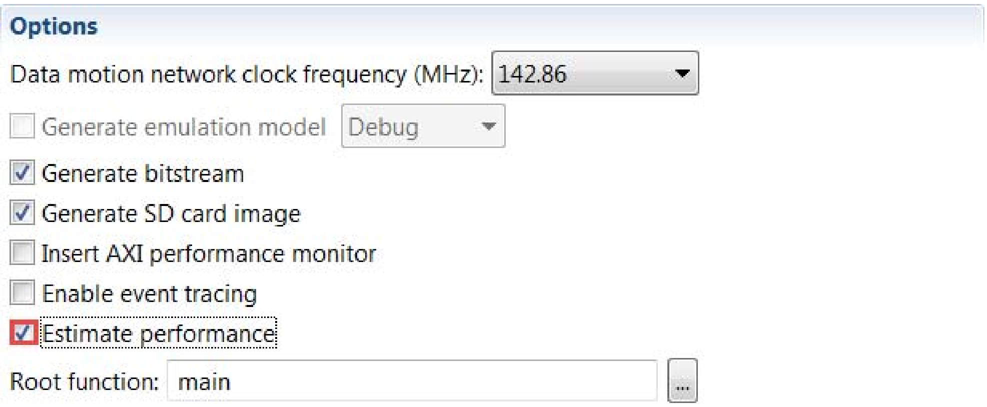
2-1-1. Click on the "+" ( ) sign in the HW Functions area to open up the list of functions which are in the source files.

2-1-2. Select the *sharpen_filter* function and click **OK**.

2-1-3. Set the Clock Frequency to **100** MHz.

2-1-4. In **Options** panel of the *SDx Project Settings* pane, click on *Estimate Performance* checkbox.

This selects the Estimate build configuration and performs the estimation flow.



Options

Data motion network clock frequency (MHz): 142.86

☐ Generate emulation model Debug

☑ Generate bitstream

☑ Generate SD card image

☐ Insert AXI performance monitor

☐ Enable event tracing

☑ Estimate performance

Root function: main

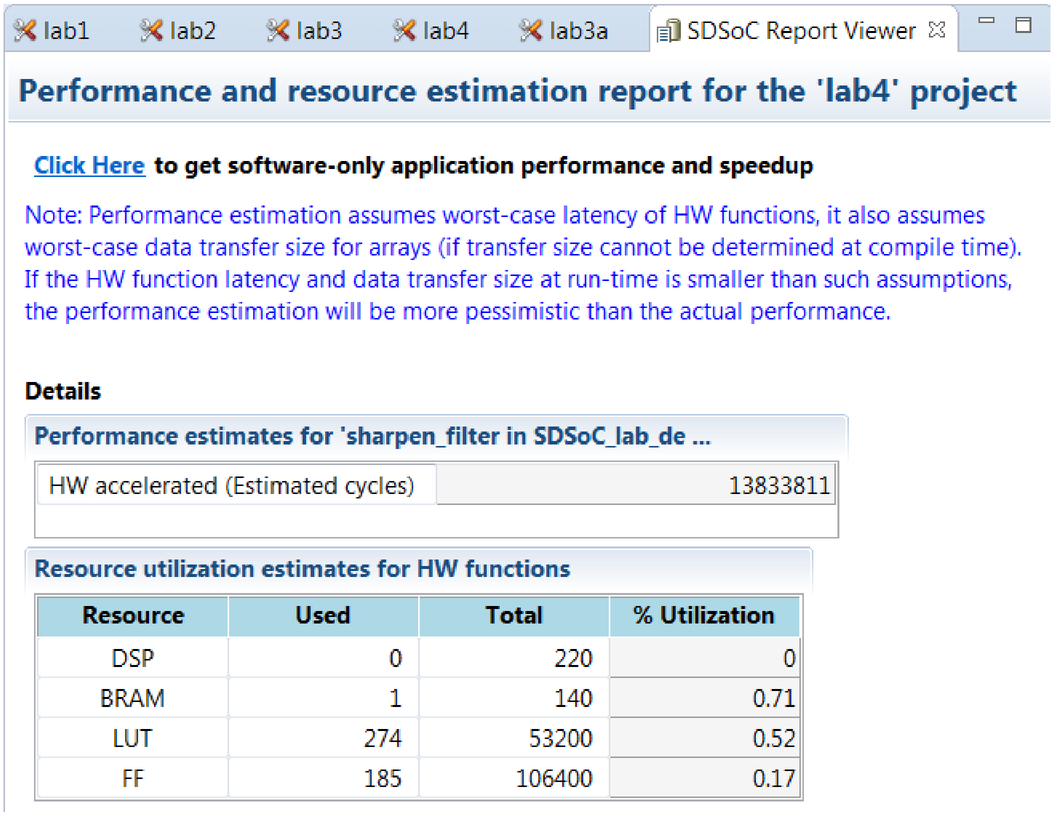
Figure 1. Selecting Estimate performance option (Zedboard)

2-1-5. Select **Build Configurations > Set Active > Debug**

2-1-6. Right-click on **lab4** and select **Build Project**.

The SDx environment builds the project. A dialog box displaying the status of the build process appears.

2-1-7. After the build is over, you can see an initial report. This report contains a hardware-only estimate summary which is calculated from the hardware compilation. There is a link that can be clicked to run the software on the board and obtain the software performance results. Clicking this link will also update the report with an estimated comparison of hardware accelerated implementation versus the software-only information.



Performance and resource estimation report for the 'lab4' project

Click Here to get software-only application performance and speedup

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Details

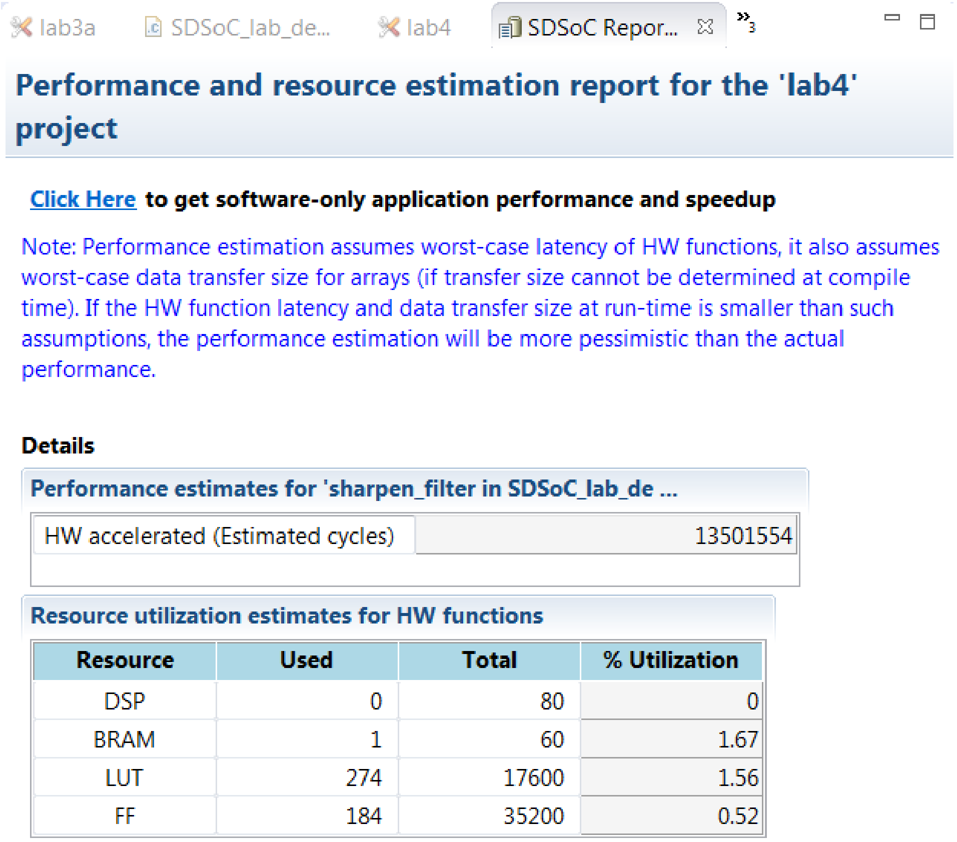
Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

HW accelerated (Estimated cycles)	13833811

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	1	140	0.71
LUT	274	53200	0.52
FF	185	106400	0.17

(a) Zed



Performance and resource estimation report for the 'lab4' project

Click Here **to get software-only application performance and speedup**

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Details

Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

HW accelerated (Estimated cycles)	13501554

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	80	0
BRAM	1	60	1.67
LUT	274	17600	1.56
FF	184	35200	0.52

(b) Zybo

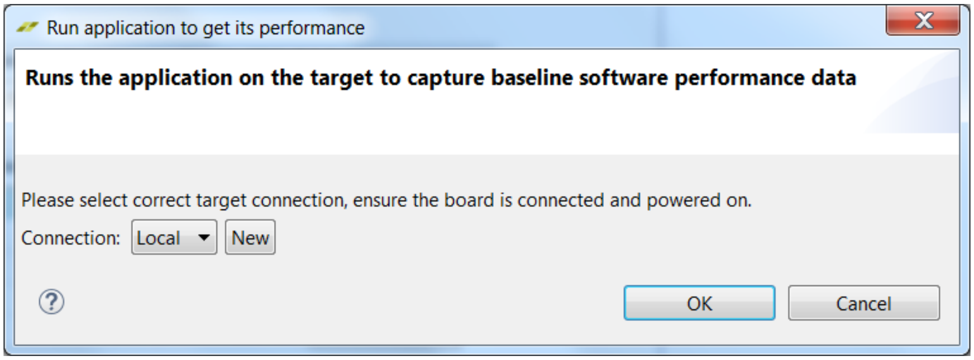
Figure 2. Initial estimate of hardware only performance

2-2. Connect and power up the board. Click on the `Click Here` link of the initial estimation report to run the application and get the entire application speedup.

2-2-1. Connect the board and power it ON.

2-2-2. Click on the **Click Here** link in the *SDSoC Report Viewer* tab to get the software only application performance and speedup.

Since the board is connected using JTAG and the OS is Standalone, the *Local* connection will be used.



Runs the application on the target to capture baseline software performance data

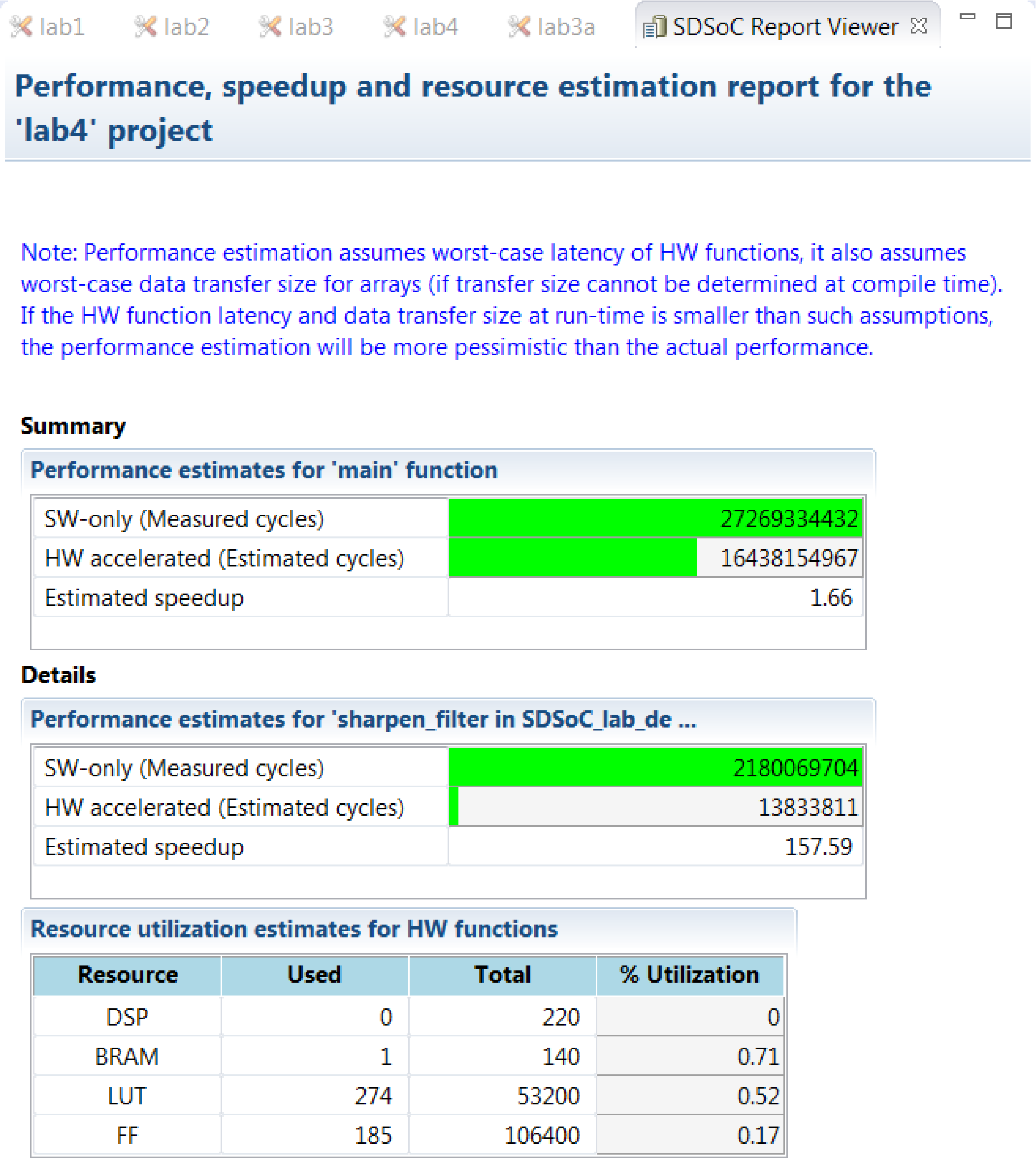
Please select correct target connection, ensure the board is connected and powered on.

Connection: Local

Figure 3. Making connection to download and running application

2-2-3. Click **OK**.

A default bitstream (without the accelerator functionality) and the application will be downloaded and executed.



Performance, speedup and resource estimation report for the 'lab4' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27269334432
HW accelerated (Estimated cycles)	16438154967
Estimated speedup	1.66

Details

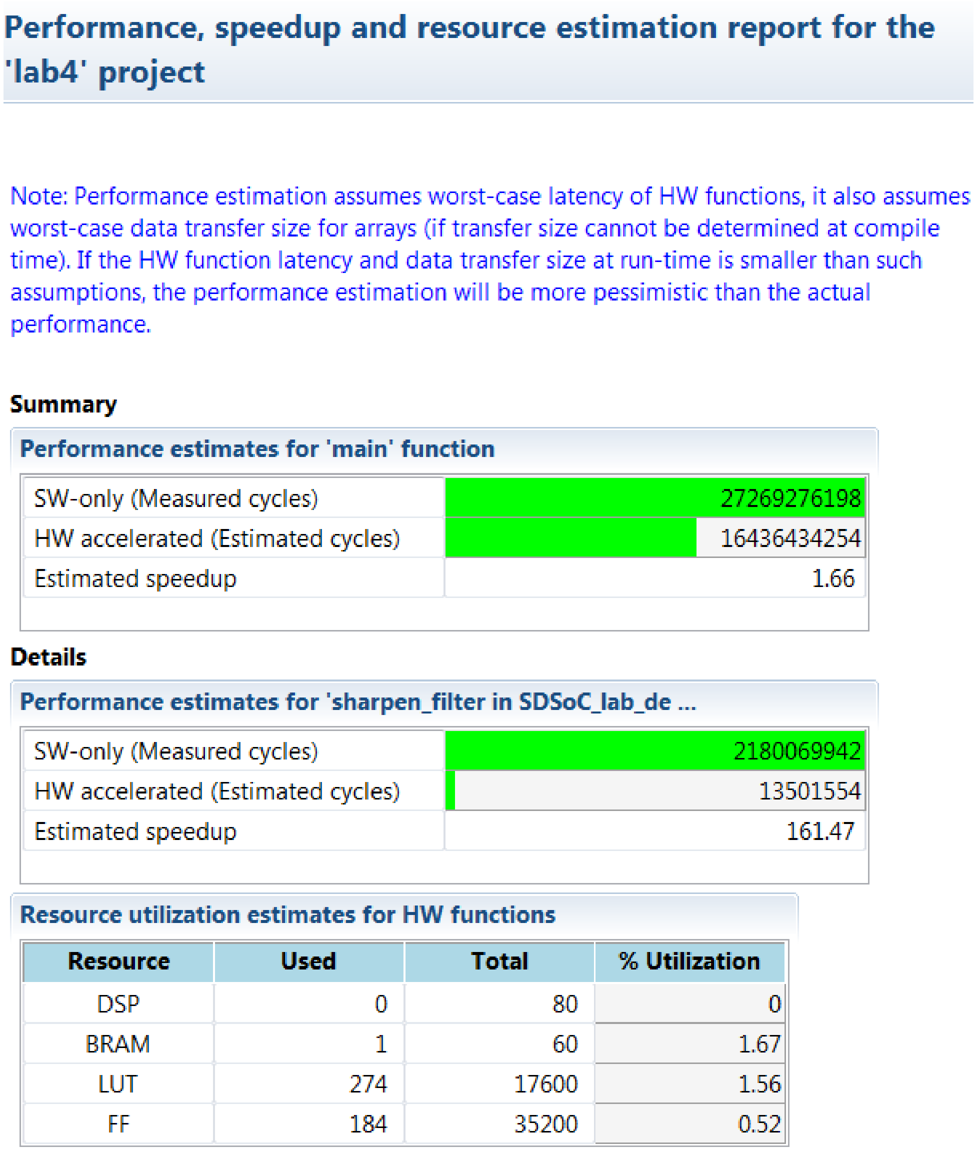
Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

SW-only (Measured cycles)	2180069704
HW accelerated (Estimated cycles)	13833811
Estimated speedup	157.59

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	1	140	0.71
LUT	274	53200	0.52
FF	185	106400	0.17

(a) Zed



Performance, speedup and resource estimation report for the 'lab4' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27269276198
HW accelerated (Estimated cycles)	16436434254
Estimated speedup	1.66

Details

Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

SW-only (Measured cycles)	2180069942
HW accelerated (Estimated cycles)	13501554
Estimated speedup	161.47

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	80	0
BRAM	1	60	1.67
LUT	274	17600	1.56
FF	184	35200	0.52

(b) Zybo

Figure 4. Comparison between the pure software and hardware accelerated

The Summary section shows that the estimated speedup between the software only and one with the hardware accelerator is 1.24 for Zed and 1.66 for Zybo..

2-3. Add sobel_filter for the hardware acceleration. Run an initial performance estimate of the hardware only.

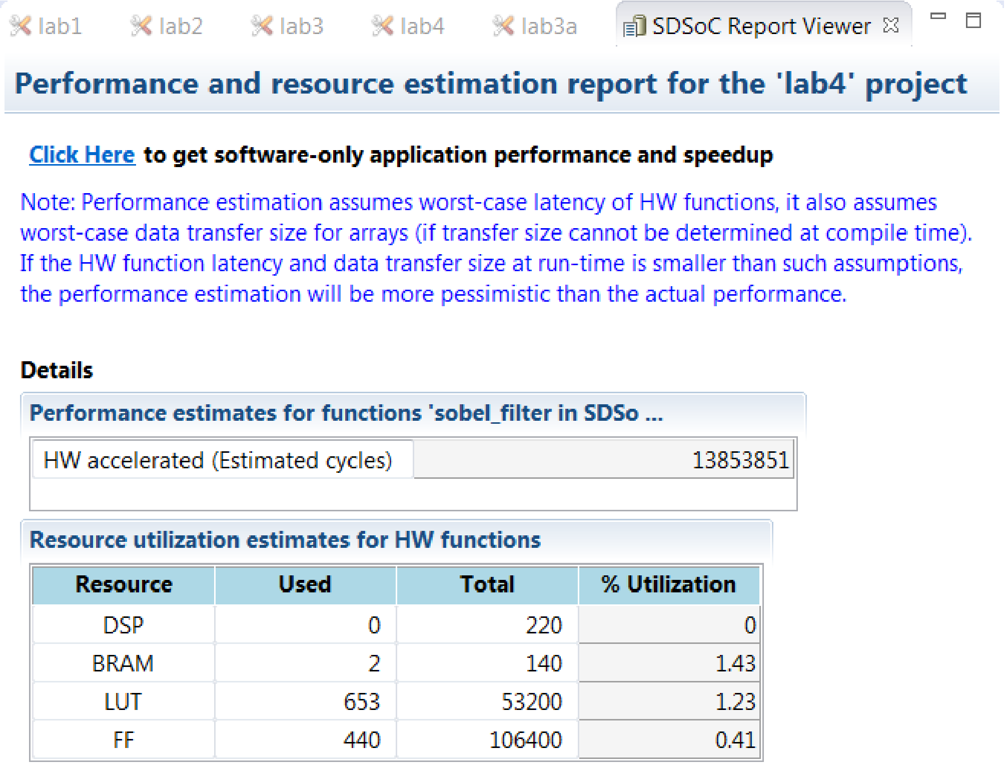
2-3-1. Add the *sobel_filter* function to the accelerator list with 100 MHz Clock Frequency.

2-3-2. Select **lab4 > Clean Project**

2-3-3. Right-click on **lab4** and select **Build Project**.

The SDx environment builds the project. A dialog box displaying the status of the build process appears.

2-3-4. After the build is over, the initial estimate and resources report will be displayed again.



Performance and resource estimation report for the 'lab4' project

Click Here to get software-only application performance and speedup

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Details

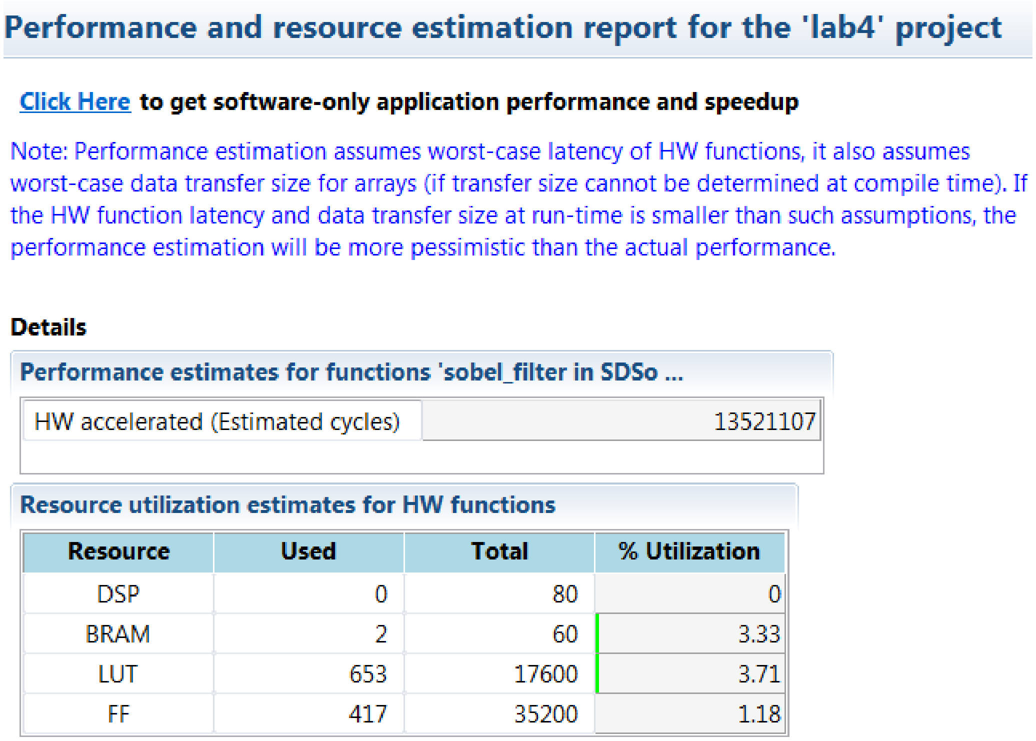
Performance estimates for functions 'sobel_filter in SDSo ...

HW accelerated (Estimated cycles)	13853851

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	2	140	1.43
LUT	653	53200	1.23
FF	440	106400	0.41

(a) Zed



Performance and resource estimation report for the 'lab4' project

Click Here to get software-only application performance and speedup

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Details

Performance estimates for functions 'sobel_filter in SDSo ...

HW accelerated (Estimated cycles)	13521107

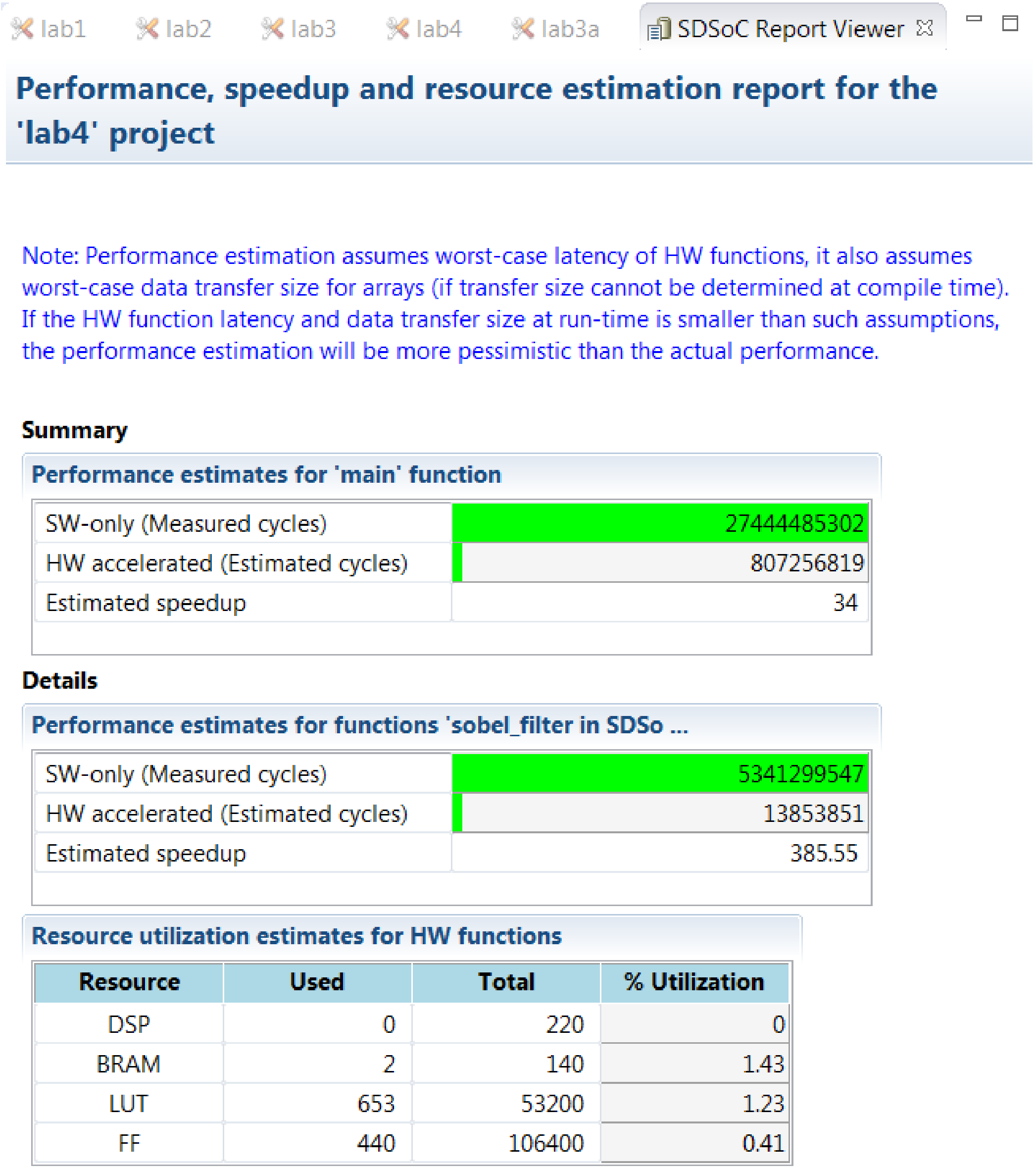
Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	80	0
BRAM	2	60	3.33
LUT	653	17600	3.71
FF	417	35200	1.18

(b) Zybo

Figure 5. Initial hardware acceleration estimate for the two functions

2-3-5. Click on the **Click Here** link in the SDSoC Report Viewer tab and click **OK**.



Performance, speedup and resource estimation report for the 'lab4' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27444485302
HW accelerated (Estimated cycles)	807256819
Estimated speedup	34

Details

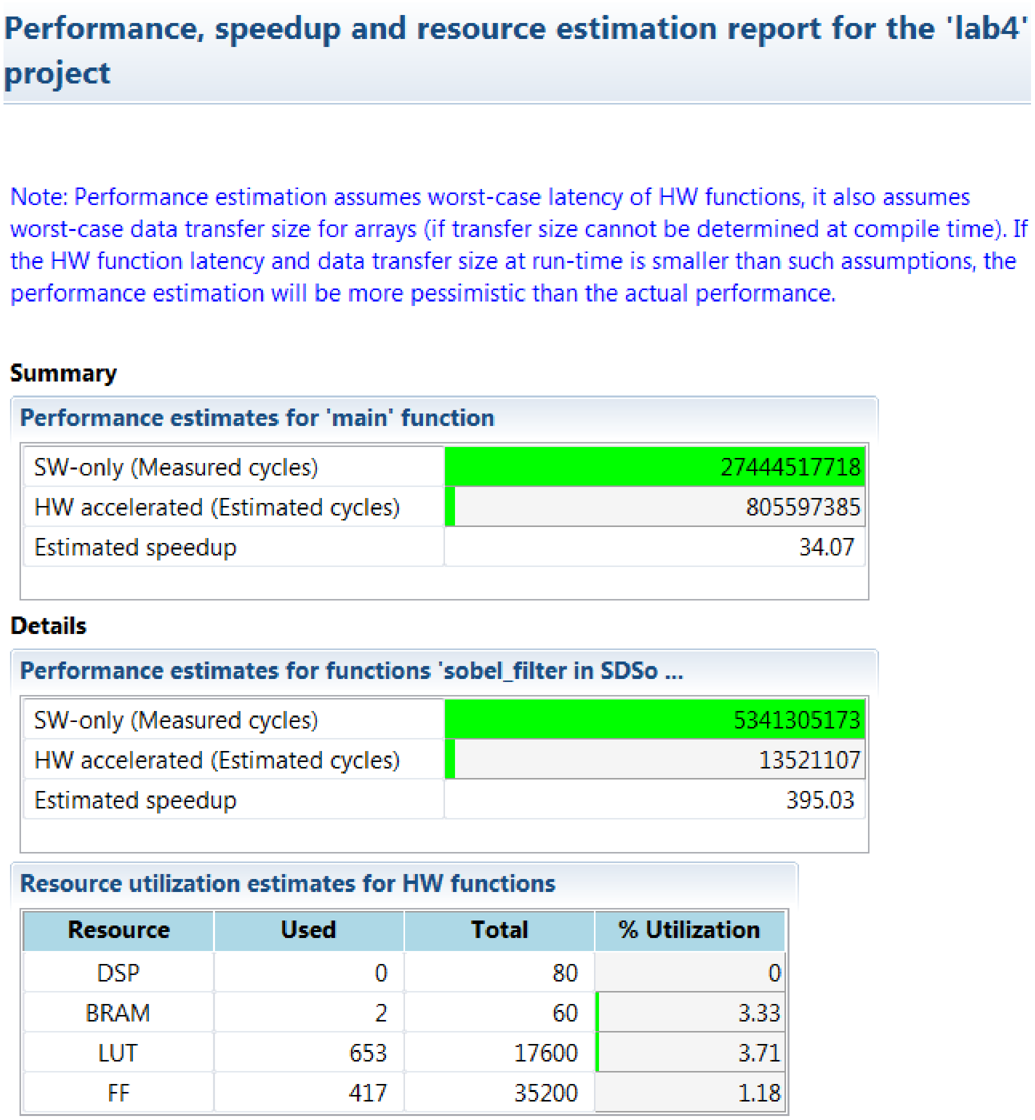
Performance estimates for functions 'sobel_filter in SDSo ...

SW-only (Measured cycles)	5341299547
HW accelerated (Estimated cycles)	13853851
Estimated speedup	385.55

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	2	140	1.43
LUT	653	53200	1.23
FF	440	106400	0.41

(a) Zed



Performance, speedup and resource estimation report for the 'lab4' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27444517718
HW accelerated (Estimated cycles)	805597385
Estimated speedup	34.07

Details

Performance estimates for functions 'sobel_filter in SDSo ...

SW-only (Measured cycles)	5341305173
HW accelerated (Estimated cycles)	13521107
Estimated speedup	395.03

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	80	0
BRAM	2	60	3.33
LUT	653	17600	3.71
FF	417	35200	1.18

(b) Zybo

Figure 6. Actual performance estimation with two functions in hardware

Performance Estimation Targeting Linux Step 3

3-1. Create a new empty application project called lab4a targeting Linux OS. Import the provided source files from source\lab4\src folder

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

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

3-1-2. Enter **lab4a** as the project name.

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

3-1-4. Select either *zybo* or *zed* (depending on the board you are using) and click **Next.**

3-1-5. Select **Linux SMP** as the target OS, and click **Next**.

3-1-6. Select **Empty Application** and click **Finish**.

3-1-7. Right click on *src* under **lab4a** in the Project Explorer tab and select **Import…**

3-1-8. Click on **File System** under *General category* and then click **Next**.

3-1-9. Click on the **Browse** button, browse to *c:\xup\SDSoC\source\lab4\src* folder, and click **OK**.

3-1-10. 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. Run an initial performance estimate of the hardware only.

3-2-1. Click on the “+” ( ) sign in the HW Functions area to open up the list of functions which are in the source files.

3-2-2. Select the *sharpen_filter* function and click **OK**.

3-2-3. Set the Clock Frequency to **100** MHz.

3-2-4. In **Options** panel of the *SDx Project Settings* pane, click on the *Estimate Performance* checkbox.

3-2-5. Right-click on **lab4a** and select **Build Project**

This selects the Debug build configuration and performs the estimation flow.

3-2-6. After the build is over, you will see an initial report.

3-3. Copy the sd_card contents to the SD Card. Configure the board to boot from SD card. Connect and power up the board. Configure the board's Ethernet address to 192.168.0.10 and the PC's to 192.168.0.1

3-3-1. Configure the board to boot from SD card.

3-3-2. Using the Windows Explorer copy the content of the **lab4a > Debug > sd_card** onto the (micro) SD card. Insert the SD card into the board.

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

The board will boot. Make a serial connection using the appropriate COM port.

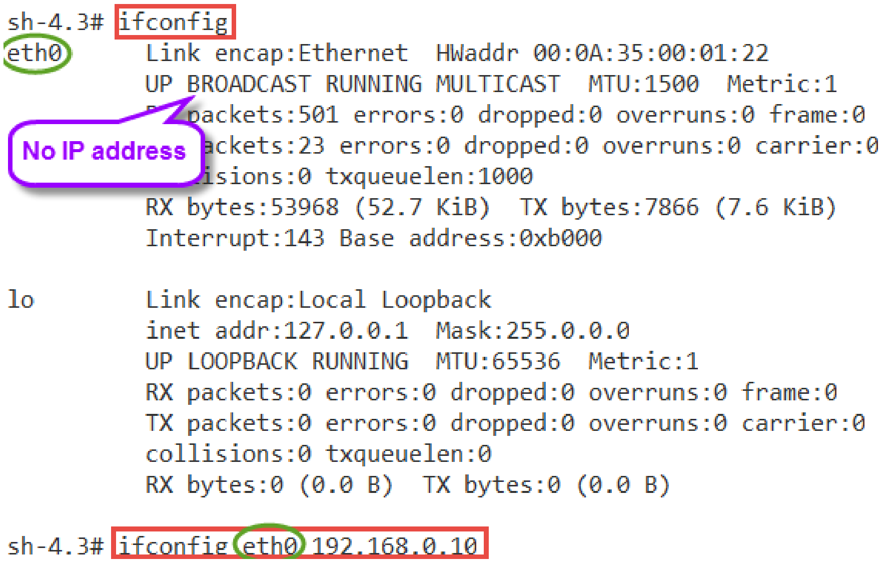
3-3-4. Press the *PS-SRST* button on the board to reboot and notice Linux booting.

3-3-5. Once the board boot is complete, set the IP address of the board to 192.168.0.10 by typing the following command at the Linux prompt:

```
ifconfig
```

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
            packets:501 errors:0 dropped:0 overruns:0 frame:0
             ackets:23 errors:0 dropped:0 overruns:0 carrier:0
              isions: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
```

Figure 7. Assigning an IP address

3-3-6. Configure the Ethernet adaptor IP address on the Zynq board to 192.168.0.10

```
ifconfig eth0 192.168.0.10
```

3-3-7. Configure the PC Ethernet adaptor IP address to 192.168.0.1

3-3-8. Expand Linux TCF Agent in the Target Connection tab.

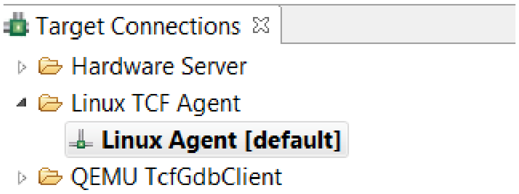
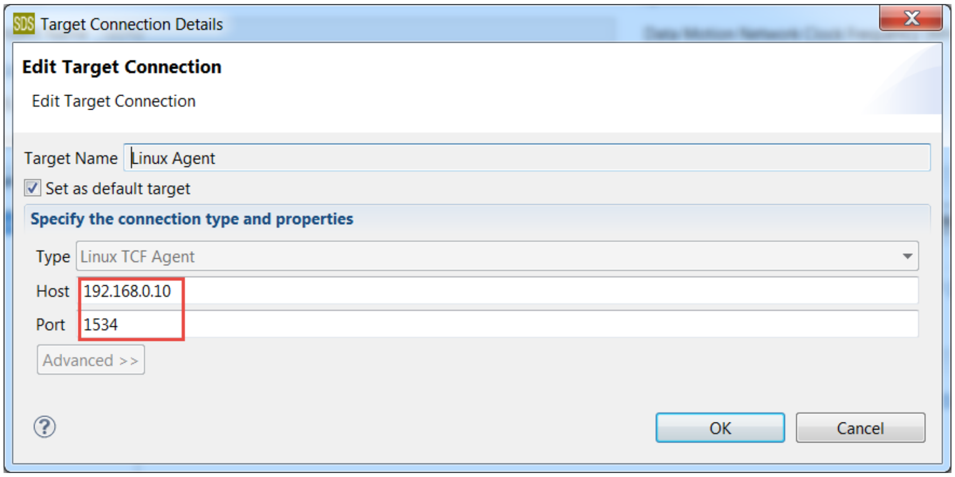


Figure 8. Configuring the Linux TCF Agent

3-3-9. Double-click on the **Linux Agent [default]** entry to open the connection form.

3-3-10. Set the *Host IP* address to **192.168.0.10**, the *Port* field to **1534,** and then click ***OK***.



Target Connection Details

Edit Target Connection

Edit Target Connection

Target Name: Linux Agent

☑ Set as default target

Specify the connection type and properties

Type: Linux TCF Agent

Host: 192.168.0.10

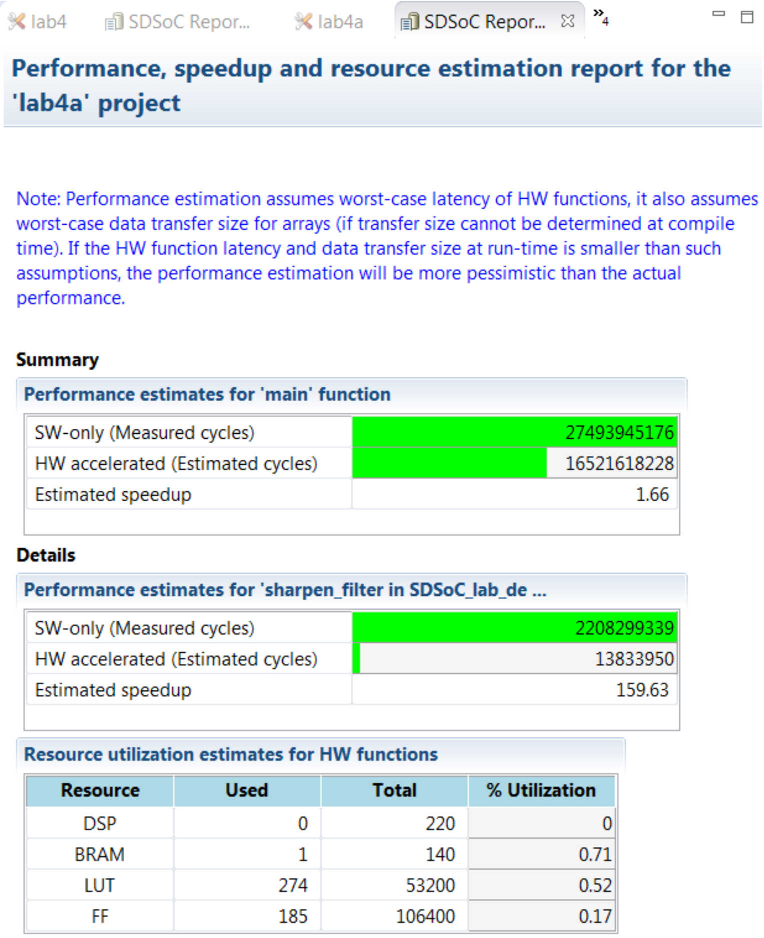
Port: 1534

Figure 9. Making connection for Linux target

3-4. Estimate the accelerator speedup.

3-4-1. In the performance and resource estimate report, click on the **Click Here** link. Click **OK** to launch the Linux TCF agent.

3-4-2. When the execution completes the performance estimate report will be displayed in the SDSoC report viewer.



Performance, speedup and resource estimation report for the 'lab4a' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27493945176
HW accelerated (Estimated cycles)	16521618228
Estimated speedup	1.66

Details

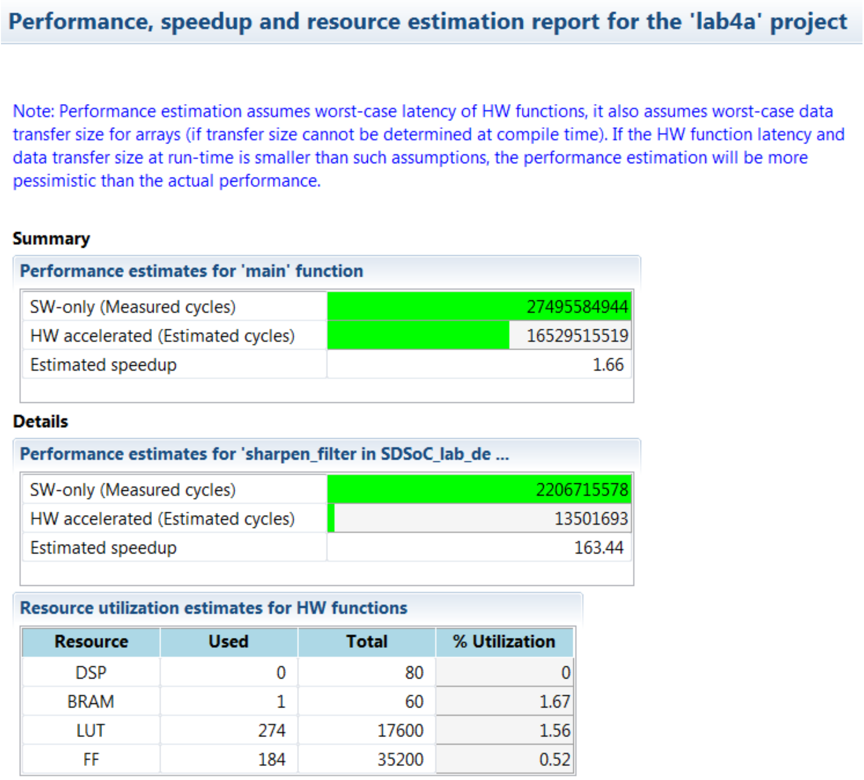
Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

SW-only (Measured cycles)	2208299339
HW accelerated (Estimated cycles)	13833950
Estimated speedup	159.63

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	220	0
BRAM	1	140	0.71
LUT	274	53200	0.52
FF	185	106400	0.17

(a) Zed



Performance, speedup and resource estimation report for the 'lab4a' project

Note: Performance estimation assumes worst-case latency of HW functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the HW function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	27495584944
HW accelerated (Estimated cycles)	16529515519
Estimated speedup	1.66

Details

Performance estimates for 'sharpen_filter in SDSoC_lab_de ...

SW-only (Measured cycles)	2206715578
HW accelerated (Estimated cycles)	13501693
Estimated speedup	163.44

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	0	80	0
BRAM	1	60	1.67
LUT	274	17600	1.56
FF	184	35200	0.52

(b) Zybo

Figure 10. Performance estimation targeting Linux OS

Note that the number of SW-only cycles have increased compared to Figure 4. This is due to the overhead running in Linux.

Tracing Accelerator Events Activities Step 4

4-1. Import the provided prebuilt lab4b project from c:\xup\SDSoC\source\lab4. Analyze the SDx Project Settings content.

4-1-1. Select **File > Import** and then select **General > Existing Projects into Workspace** and click **Next**.

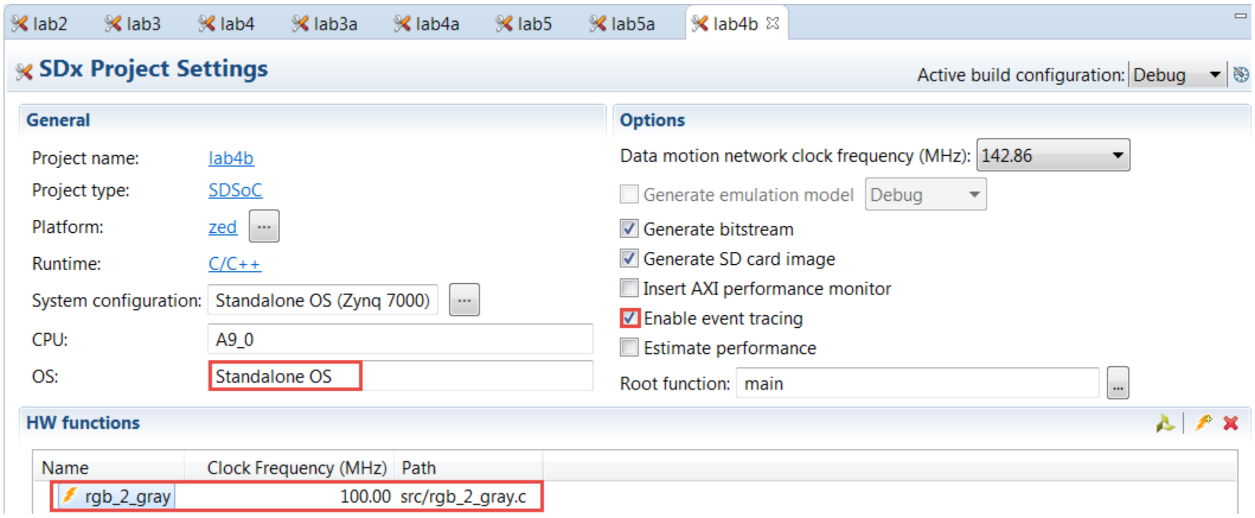
4-1-2. Select **Select archive file** and click **Browse** to navigate to *c:\xup\SDSoC\source\lab4*

4-1-3. Select *lab4b.zip*, and click **Open**.

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

4-1-5. Double-click **project.sdx** under the *lab4b* folder to view the **SDx Project Settings** pane.

Note that the *Enable event tracing* option is checked and the *rgb_2_gray* function operating at 100 MHz is included in the *HW functions* pane. The project was created targeting Standalone OS.



SDx Project Settings

Active build configuration: Debug

General

Project name: lab4b
Project type: SDSoC
Platform: zed
Runtime: C/C++
System configuration: Standalone OS (Zynq 7000)
CPU: A9_0
OS: Standalone OS

Options

Data motion network clock frequency (MHz): 142.86
☐ Generate emulation model Debug
☑ Generate bitstream
☑ Generate SD card image
☐ Insert AXI performance monitor
☑ Enable event tracing
☐ Estimate performance
Root function: main

HW functions

Name	Clock Frequency (MHz)	Path
rgb_2_gray	100.00	src/rgb_2_gray.c

Figure 11. Pre-built project with event tracing feature enabled

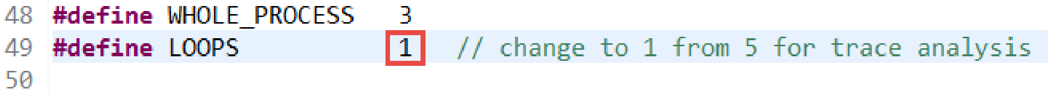
4-1-6. **Uncheck** the *Generate bitstream* and *Generate SD card image* check boxes as they are already generated in the imported project.

4-2. Analyze the source code with the tracing code included in the sharpen_filter stub.

4-2-1. Open **SDSoC_lab_design_main.c** from the *c:\xup\SDSoC\labs\lab4b\Debug_sds\swstubs* directory.

4-2-2. Change the number of times the algorithm loops over is changed from 5 to 1.

This is to reduce the amount of trace data collected and to give a better view of state analysis.

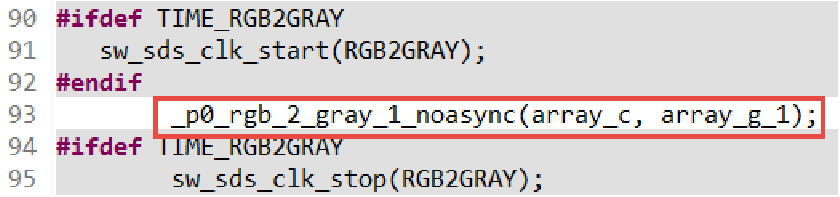


```
#define WHOLE_PROCESS   3
#define LOOPS           1   // change to 1 from 5 for trace analysis

```

Figure 12. Loop iteration changed to 1

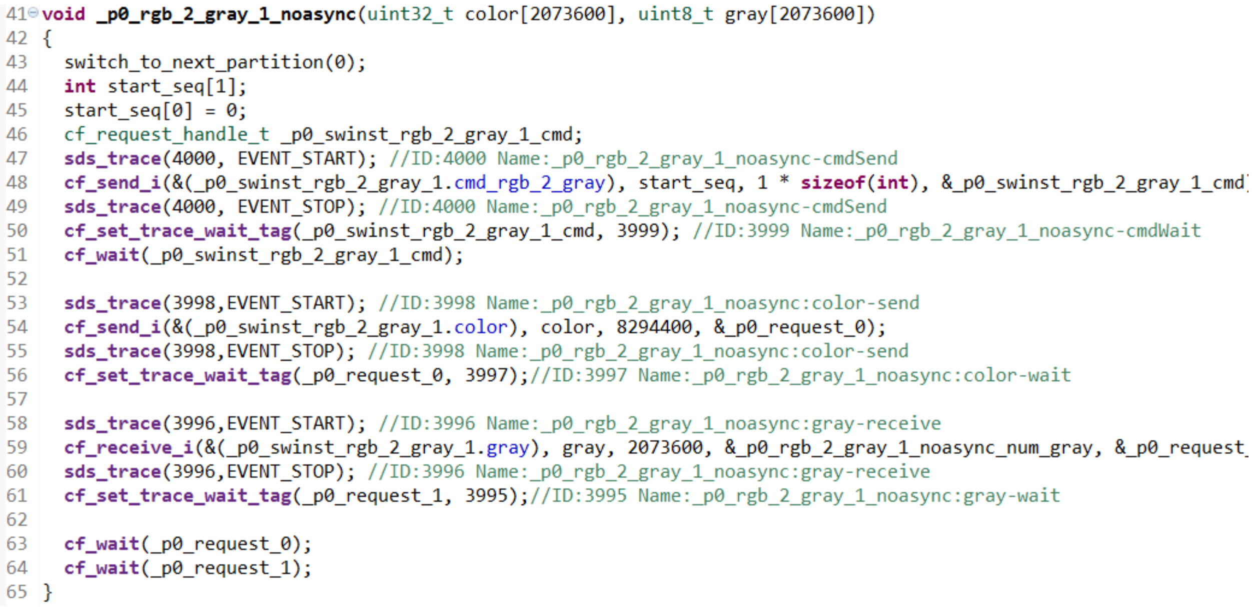
4-2-3. Notice that the call to the *rgb_2_gray* is replaced by the call to the stub.



```
#ifdef TIME_RGB2GRAY
   sw_sds_clk_start(RGB2GRAY);
#endif
        _p0_rgb_2_gray_1_noasync(array_c, array_g_1);
#ifdef TIME_RGB2GRAY
        sw_sds_clk_stop(RGB2GRAY);
```

Figure 13. Hardware function call

4-2-4. Double-click on the **rgb_2_gray.c** entry under the *Debug/_sds/swstubs* directory and notice the actual call is updated.



```
void _p0_rgb_2_gray_1_noasync(uint32_t color[2073600], uint8_t gray[2073600])
{
  switch_to_next_partition(0);
  int start_seq[1];
  start_seq[0] = 0;
  cf_request_handle_t _p0_swinst_rgb_2_gray_1_cmd;
  sds_trace(4000, EVENT_START); //ID:4000 Name:_p0_rgb_2_gray_1_noasync-cmdSend
  cf_send_i(&(_p0_swinst_rgb_2_gray_1.cmd_rgb_2_gray), start_seq, 1 * sizeof(int), &_p0_swinst_rgb_2_gray_1_cmd
  sds_trace(4000, EVENT_STOP); //ID:4000 Name:_p0_rgb_2_gray_1_noasync-cmdSend
  cf_set_trace_wait_tag(_p0_swinst_rgb_2_gray_1_cmd, 3999); //ID:3999 Name:_p0_rgb_2_gray_1_noasync-cmdWait
  cf_wait(_p0_swinst_rgb_2_gray_1_cmd);

  sds_trace(3998,EVENT_START); //ID:3998 Name:_p0_rgb_2_gray_1_noasync:color-send
  cf_send_i(&(_p0_swinst_rgb_2_gray_1.color), color, 8294400, &_p0_request_0);
  sds_trace(3998,EVENT_STOP); //ID:3998 Name:_p0_rgb_2_gray_1_noasync:color-send
  cf_set_trace_wait_tag(_p0_request_0, 3997);//ID:3997 Name:_p0_rgb_2_gray_1_noasync:color-wait

  sds_trace(3996,EVENT_START); //ID:3996 Name:_p0_rgb_2_gray_1_noasync:gray-receive
  cf_receive_i(&(_p0_swinst_rgb_2_gray_1.gray), gray, 2073600, &_p0_rgb_2_gray_1_noasync_num_gray, &_p0_request_
  sds_trace(3996,EVENT_STOP); //ID:3996 Name:_p0_rgb_2_gray_1_noasync:gray-receive
  cf_set_trace_wait_tag(_p0_request_1, 3995);//ID:3995 Name:_p0_rgb_2_gray_1_noasync:gray-wait

  cf_wait(_p0_request_0);
  cf_wait(_p0_request_1);
}
```

Figure 14. The rgb_2_gray function having sds_trace function calls

The stub function initializes the hardware accelerator, initiates any required data transfers for the function arguments, and then synchronizes hardware and software by waiting at an appropriate point in the program for the accelerator and all associated data transfers to complete.

Event tracing provides visibility into each phase of the hardware function execution, including the software setup for the accelerators and data transfers, as well as the hardware execution of the accelerators and data transfers.

The above code is instrumented for trace. Each command that starts the accelerator, starts a transfer, or waits for a transfer to complete is instrumented (sds_trace(xxxx, EVENT_START | EVENT_STOP);).

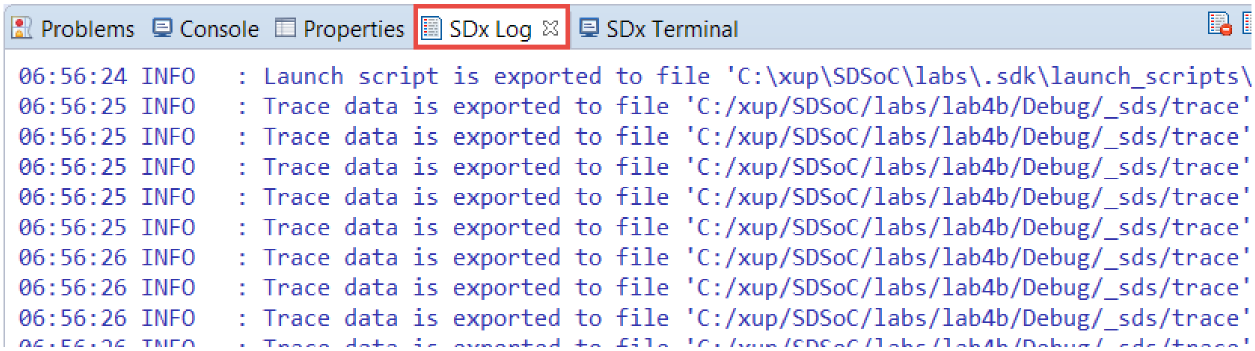
4-3. Run the application and collect the trace data.

4-3-1. Connect and power ON the board.

4-3-2. Right-click on the *lab4b* project and select **Run As > Trace Application (SDSoC Debugger).**

This will download the bitstream, then the application and finally run the application.

Notice in the *SDx Log* tab that the trace data is exported to the c:\xup\SDSoC\labs\lab4b\Debug\*_sds\trace* directory.



```
06:56:24 INFO   : Launch script is exported to file 'C:\xup\SDSoC\labs\.sdk\launch_scripts\
06:56:25 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:25 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:25 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:25 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:25 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:26 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:26 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
06:56:26 INFO   : Trace data is exported to file 'C:/xup/SDSoC/labs/lab4b/Debug/_sds/trace'
```

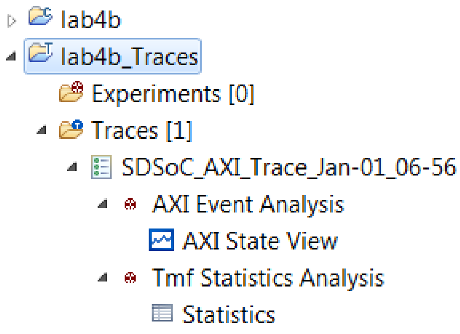
Figure 15. Exporting trace data

When the trace data export is completed, the tool will create a trace folder named *lab4b_Traces* in the *Project Explorer* tab.

4-4. View the AXI State to analyze the application flow.

4-4-1. Expand the **lab4b_Traces** project folder in the *Project Explorer* tab**.**

4-4-2. Expand all the folders under the *Traces[1]* folder.



- lab4b
- lab4b_Traces
  - Experiments [0]
  - Traces [1]
    - SDSoC_AXI_Trace_Jan-01_06-56
      - AXI Event Analysis
        - AXI State View
      - Tmf Statistics Analysis
        - Statistics

Figure 16. Trace Project Folder

4-4-3. Click on the Home button in the **AXI State View** tab.

This will show the entire trace history. You will notice

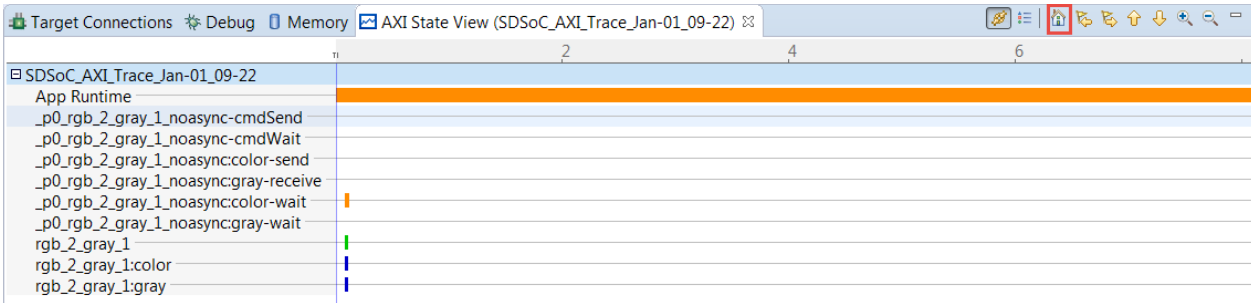
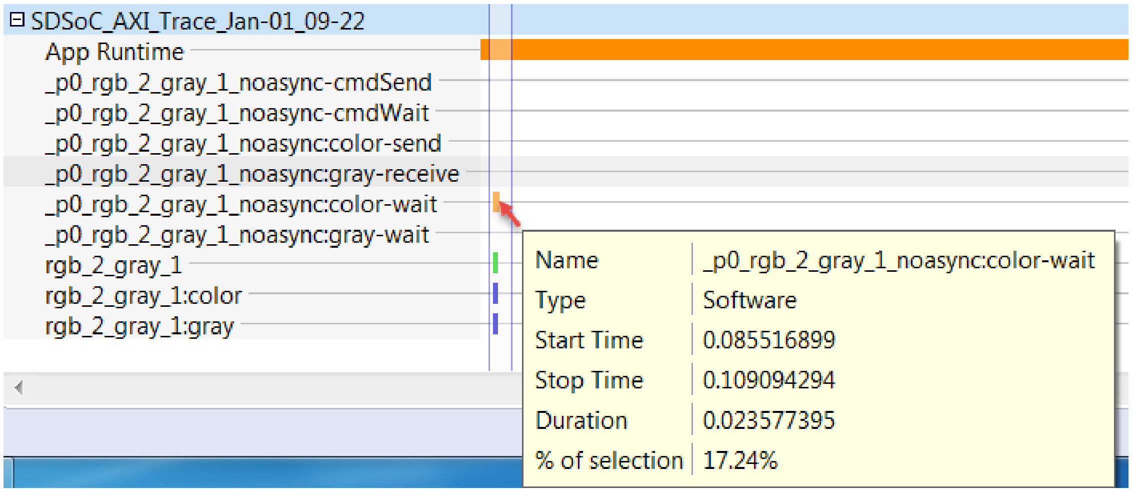

Target Connections
Debug
Memory
AXI State View (SDSoC_AXI_Trace_Jan-01_09-22)
2
4
6
SDSoC_AXI_Trace_Jan-01_09-22
App Runtime
_p0_rgb_2_gray_1_noasync-cmdSend
_p0_rgb_2_gray_1_noasync-cmdWait
_p0_rgb_2_gray_1_noasync:color-send
_p0_rgb_2_gray_1_noasync:gray-receive
_p0_rgb_2_gray_1_noasync:color-wait
_p0_rgb_2_gray_1_noasync:gray-wait
rgb_2_gray_1
rgb_2_gray_1:color
rgb_2_gray_1:gray

Figure 17. Trace Visualization Highlighting the Different Types of Events – Stub Code Comparison

4-4-4. Hover the cursor above one of the events.

Each trace point in the user application is given a unique name and axis, or swimlane, on the timeline.

Each trace event has a few different attributes, such as name, type, start time, stop time, and duration.


SDSoC_AXI_Trace_Jan-01_09-22
App Runtime
_p0_rgb_2_gray_1_noasync-cmdSend
_p0_rgb_2_gray_1_noasync-cmdWait
_p0_rgb_2_gray_1_noasync:color-send
_p0_rgb_2_gray_1_noasync:gray-receive
_p0_rgb_2_gray_1_noasync:color-wait
_p0_rgb_2_gray_1_noasync:gray-wait
rgb_2_gray_1
rgb_2_gray_1:color
rgb_2_gray_1:gray

Name	_p0_rgb_2_gray_1_noasync:color-wait
Type	Software
Start Time	0.085516899
Stop Time	0.109094294
Duration	0.023577395
% of selection	17.24%

Figure 18. Detailed Information Available for Each Event

4-4-5. Click on the previous/next event button ( ) to see the start. Zoom out appropriately to see initial events.

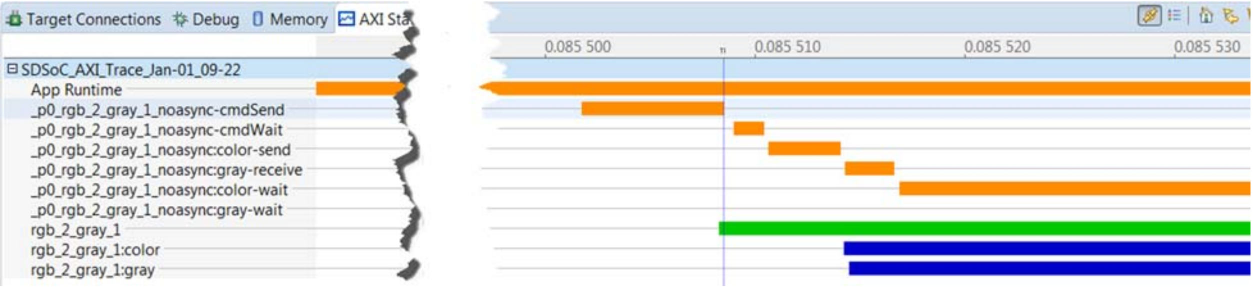

Target Connections
Debug
Memory
AXI Sta
SDSoC_AXI_Trace_Jan-01_09-22
App Runtime
_p0_rgb_2_gray_1_noasync-cmdSend
_p0_rgb_2_gray_1_noasync-cmdWait
_p0_rgb_2_gray_1_noasync:color-send
_p0_rgb_2_gray_1_noasync:gray-receive
_p0_rgb_2_gray_1_noasync:color-wait
_p0_rgb_2_gray_1_noasync:gray-wait
rgb_2_gray_1
rgb_2_gray_1:color
rgb_2_gray_1:gray
0.085 500
0.085 510
0.085 520
0.085 530

Figure 19. Various events which setup and start accelerator

Note the time axis is in seconds. The first orange event (software) is the command being sent to the accelerator. The green bar indicates the accelerator being used.

The second event is the wait for the dma to respond. The third, fourth and the fifth (software) events deal with the dma associated with input and output.

The first blue event (rgb_2_gray_1:color) indicates the actual data being transferred whereas the second blue event is when the output (rgb_2_gray_1:gray) has started. The time difference between the start of the input and start of the output would be the latency.

4-4-6. Click somewhere on the second blue event and then click on the next event button ( ). Zoom out to see the desired view.

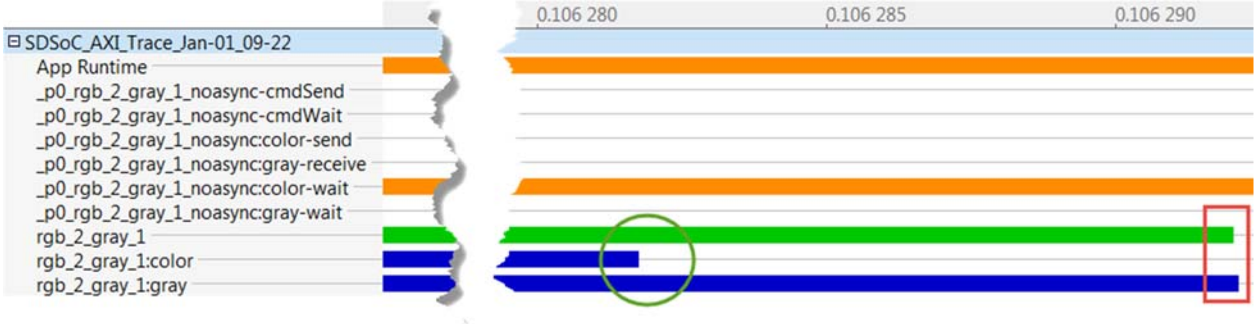

SDSoC_AXI_Trace_Jan-01_09-22
App Runtime
_p0_rgb_2_gray_1_noasync-cmdSend
_p0_rgb_2_gray_1_noasync-cmdWait
_p0_rgb_2_gray_1_noasync:color-send
_p0_rgb_2_gray_1_noasync:gray-receive
_p0_rgb_2_gray_1_noasync:color-wait
_p0_rgb_2_gray_1_noasync:gray-wait
rgb_2_gray_1
rgb_2_gray_1:color
rgb_2_gray_1:gray
0.106 280
0.106 285
0.106 290

Figure 20. Tail end of the hardware accelerator events

As can be seen, the blue input (rgb_2_gray_1:color) finishes first.

The second blue (rgb_2_gray_1:gray) finishes next just after the green accelerator.

4-4-7. Click on the *:color-wait orange bar and then click the next event button to see the tail end activities. Zoom as necessary.

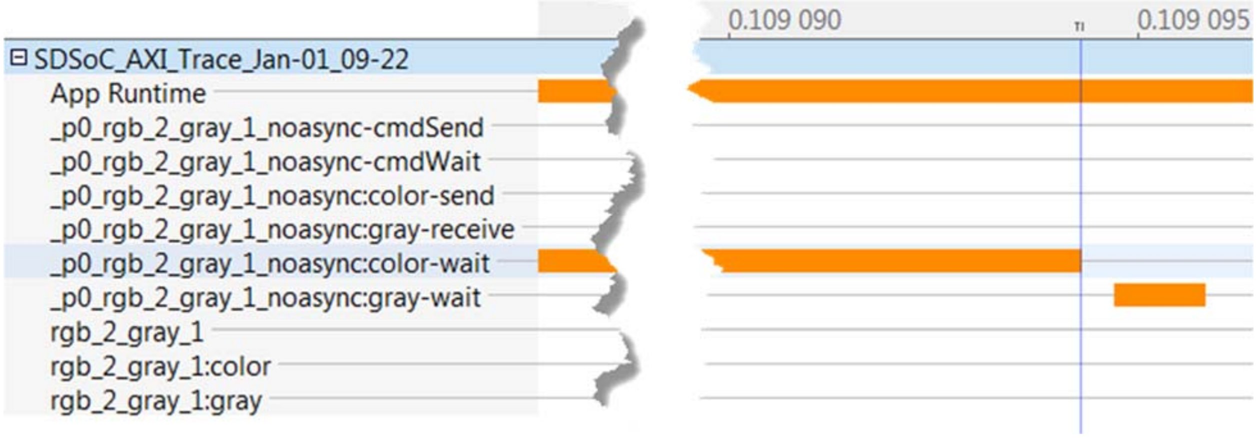

SDSoC_AXI_Trace_Jan-01_09-22
App Runtime
_p0_rgb_2_gray_1_noasync-cmdSend
_p0_rgb_2_gray_1_noasync-cmdWait
_p0_rgb_2_gray_1_noasync:color-send
_p0_rgb_2_gray_1_noasync:gray-receive
_p0_rgb_2_gray_1_noasync:color-wait
_p0_rgb_2_gray_1_noasync:gray-wait
rgb_2_gray_1
rgb_2_gray_1:color
rgb_2_gray_1:gray
0.109 090
T1
0.109 095

Figure 21: Tail end of the transaction

The orange output (_p0_rgb_2-gray_1_noasync:color-wait) finishes followed by (_p0_rgb_2-gray_1_noasync:gray-wait) indicating the completion of the execution.

4-5. Analyze the built hardware using Vivado.

4-5-1. Start Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDx 2016.3 > Vivado Design Suite > Vivado 2016.3**

4-5-2. Click the **Open Project** link, open the design by browsing to *c:\xup\SDSoC\labs\lab4b\Debug_sds\p0\ipi* and selecting either the **zybo.xpr** or **zed.xpr**.

4-5-3. Click on **Open Block Design** in the *Flow Navigator* pane. The block design will open. Note various system blocks which connect to the Cortex-A9 processor (identified by ZYNQ in the diagram).

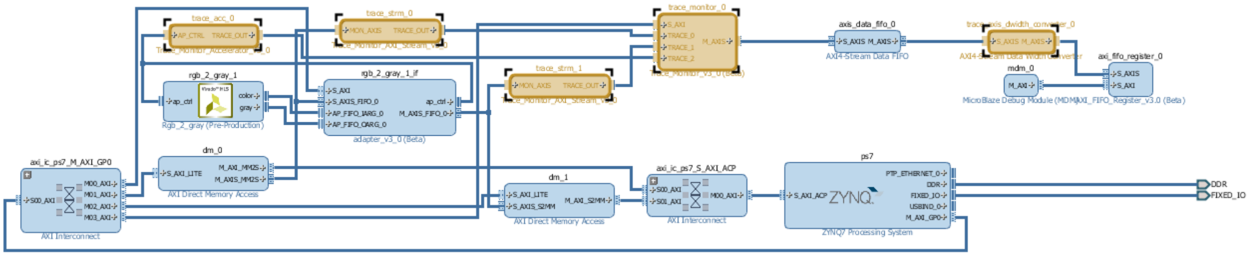


Figure 22. Block diagram with tracing hardware

4-5-4. Close Vivado without saving the block diagram.

4-5-5. Close SDx by selecting **File > Exit**

4-5-6. Turn OFF the power to the board.

Conclusion

In this lab, you performed speedup estimation of an application running under Standalone OS and Linux OS, after targeting desired function for acceleration. Performance estimation does not require the full bitstream generation and it gives speedup estimate by looking at the performance report generated by HLS for each function targeted in hardware. Event tracing provides insight into how various events are taking place and the relative time spent in data movement and data processing.