

Profiling Applications and Create Accelerators

Introduction

Program hot-spots that are compute-intensive may be good candidates for hardware acceleration, especially when it is possible to stream data between hardware and the CPU and memory and overlap the computation with the communication. This lab guides you through the process of profiling an application, analyzing the results, identifying function(s) for hardware implementation, and then profiling again after targeting function(s) for acceleration.

Objectives

After completing this lab, you will be able to:

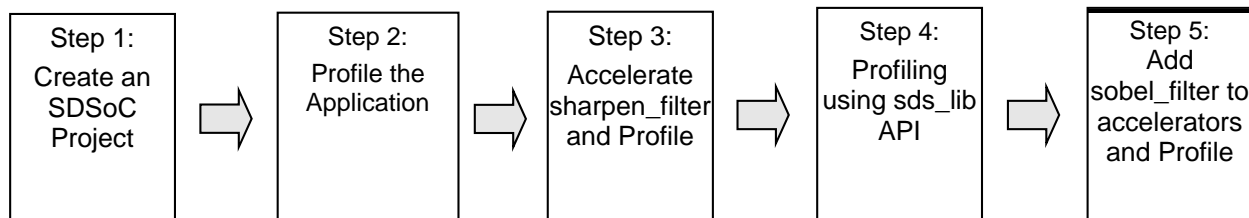
- Use TCF profiler to profile a pure software application
- Use TCF profiler to profile a software application that calls functions ported to hardware
- Use manual profiling method by using sds_lib API and counters

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 five primary steps: You will create an SDSoC project, profile the pure software project, accelerate one function and profile, profile using sds_lib API, and finally add another function to accelerators and profile.

General Flow for this Lab



Create an SDSoC Project

Step 1

1-1. Launch SDSoC and create a project, called *lab3*, using the *Empty Application* template and then using the provided source files, targeting the Zed or Zybo board.

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

1-1-2. Create a new project called **lab3**, and select either *zybo* or *zed*

1-1-3. Select **Standalone** as the target OS, and click **Next**.

1-1-4. Select **Empty Application** and click **Finish**.

Note that the **lab3 > src** folder is empty.

1-2. Import the provided source files from the source\lab3\src folder. Create an SDDebug configuration and build the project.

1-2-1. Right click on *src* under **lab3** in the Project Explorer tab and select **Import...**

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

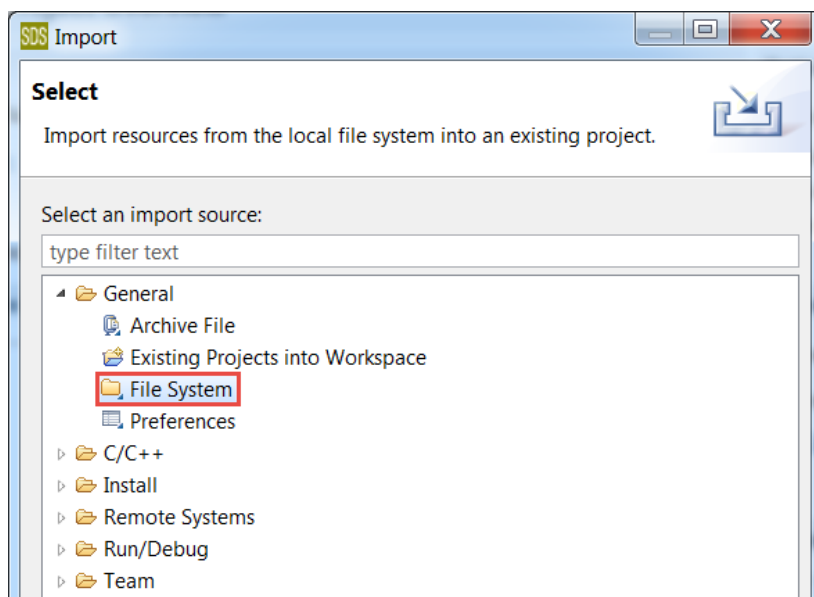


Figure 1. Selecting import source location

1-2-3. For the *From Directory*, click on the **Browse** button and browse to **c:\xup\SDSoC\source\lab3\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.

The files will be copied into the *src* folder under *lab3* folder. This can be verified by expanding the *src* folder in the Project Explorer tab and also by using Windows Explorer.

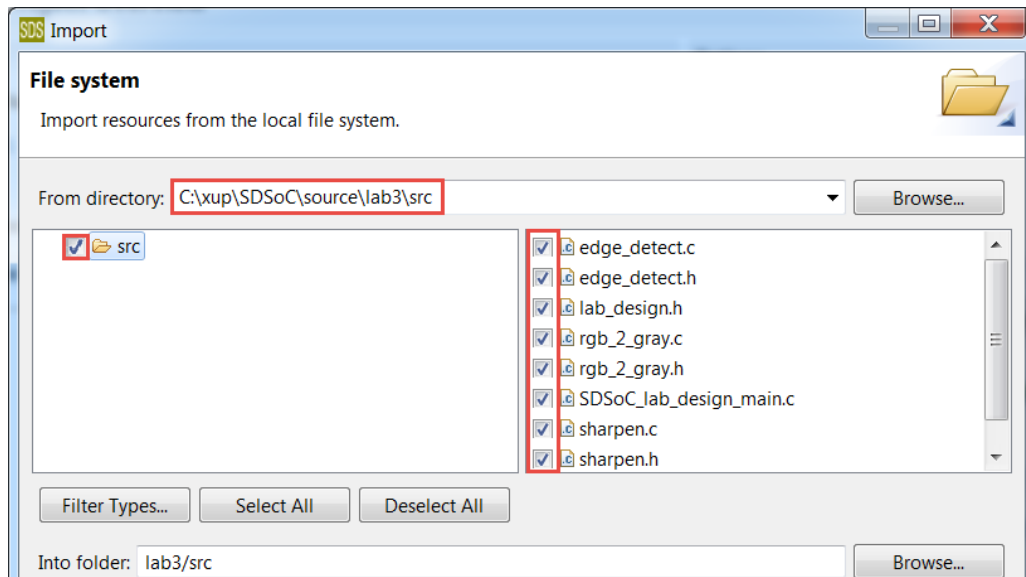


Figure 2. Selecting path and files to be imported

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

1-2-6. Right-click on **lab3** and select **Build Project**

This should only may take about one minute as it is a pure software compilation.

Profile the Application

Step 2

2-1. **Connect the board in the JTAG mode and power it ON. Start the Debug session. Add the TCF Profiler view and configure it to include the *Aggregate per Function* option.**

2-1-1. Connect the board in the JTAG mode and power it ON.

2-1-2. Click on the **Debug Application** link under the *Actions* window in the *lab3* tab on the right-side.



Figure 3. Executing Debug Application action

A *Confirm Perspective Switch* window will appear asking you to switch to the Debug perspective.

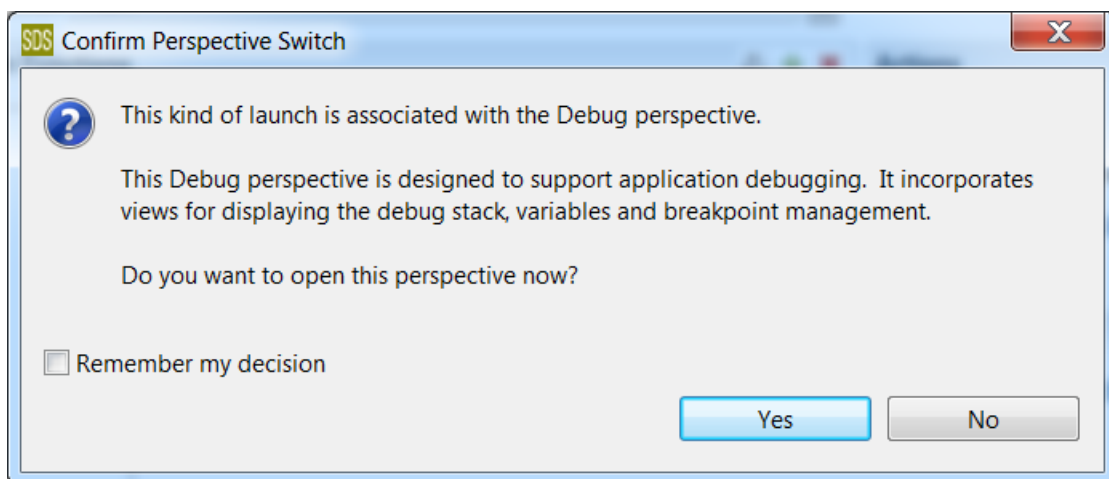


Figure 4. Perspective Switch dialog window

2-1-3. Click **Yes** to open the debug perspective.

The debug perspective will open showing various views: threads, variables, SDSoc_lab_design_main.c source program, Outline tab showing various objects created in the source program, and the console.

Notice that the program is suspended at the main() entry on line 68.

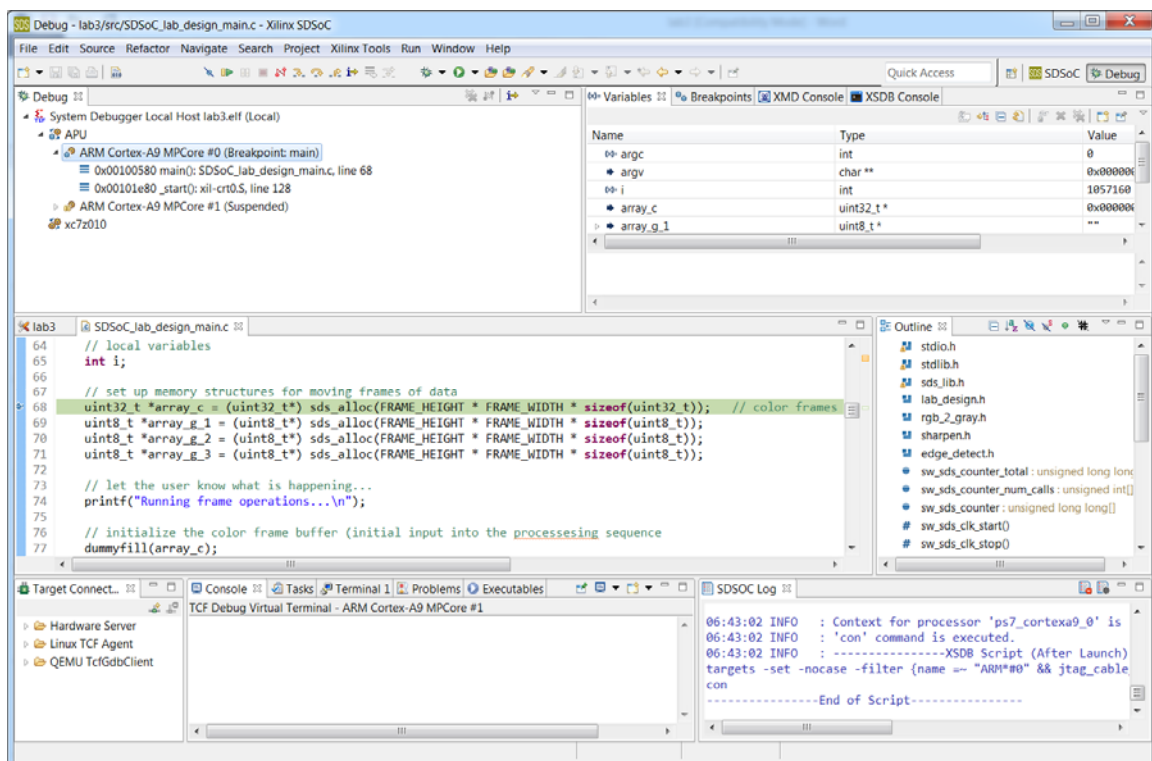


Figure 5. The Debug perspective

2-1-4. Select **Window > Show View > Other** and then expand the *Debug* folder.

2-1-5. Select *TCF Profiler* and click **OK**.

The *TCF Profiler* tab will open in the same window where *Outline* view was open.

- 2-1-6.** In the *TCF Profiler* view, click the start button.



Figure 6. Opening the TCF Profiler configuration

The *Profiler Configuration* window will open.

- 2-1-7.** Select the *Enable stack tracing* option and click **OK**.

The *Aggregate per function* option will collect the same function calls be collected together.

The *Enable stack tracing* option implements thread stack *back tracing*—essentially a summary of how the program execution gets to where it is when sampled. This allows the determination of parent/child relationships.

The *Max stack frames count* field sets the number of frames to count backwards. This option is useful only if the *Enable stack tracing* is enabled.

The *View update interval (msec)* field indicates at what interval the profile data will be updated in the TCF Profiler window.

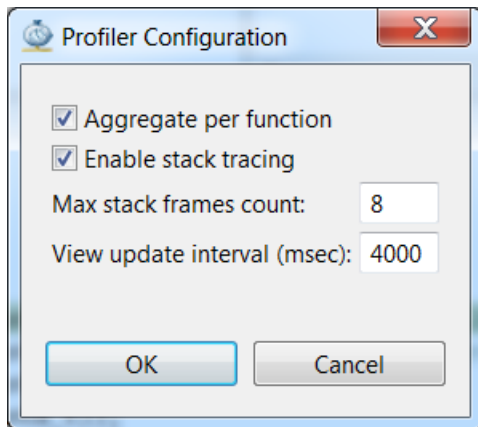




Figure 7. Selecting the options

- 2-1-8.** Click **OK**.

2-2. Run the application for about 2 minutes, then suspend, and analyze the data.

- 2-2-1.** Click on the **Resume** button () on the tool buttons bar or Press F8 to start the execution.

- 2-2-2.** Wait for about two minutes and/or when about 3200 samples are collected as indicated in the *TCF Profiler's* view and click the **Suspend** button ().

Note that the number of collected samples may vary depending on your PC's performance and connection speed with the board.

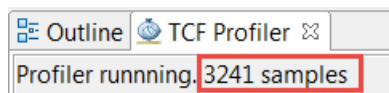



Figure 8. The TCF Profiler view showing the collected number of samples

2-2-3. Click on the **Maximize** view button ().

Note that it shows three sections. The top-section shows various calls made after the execution started. The first function called is `_start`. In the **Called From** sub-window, nothing is listed as it the root function. In the **Child Calls** window, it shows `main` as the function being called from `_start`.

Profiler running. 3239 samples					
Address	% Exclusive	% Inclusive	Function	File	Line
00101e30	.000	100	_start	xil-crt0.S	82
0010056c	.000	100	main	SDSoC_lab_design_main.c	63
00100b94	4.85	50.4	sobel_filter	edge_detect.c	76
0010133c	5.87	45.8	sharpen_filter	sharpen.c	63
001009a4	19.8	33.4	sobel_operator	edge_detect.c	34
00101228	16.1	24.9	sharpen_operator	sharpen.c	34
0010108c	13.6	13.6	window_getval	edge_detect.c	204
00101834	8.83	8.83	window_getval	sharpen.c	191
00101734	7.81	7.81	window_shift_right	sharpen.c	168
00100f8c	6.76	6.76	window_shift_right	edge_detect.c	181
001010e0	3.30	3.30	rgb_2_gray	rgb_2_gray.c	6
00101600	2.32	2.32	linebuffer_shift_up	sharpen.c	134
00100e58	2.25	2.25	linebuffer_shift_up	edge_detect.c	147
001017e0	2.16	2.16	window_insert	sharpen.c	183
0010169c	1.91	1.91	linebuffer_getval	sharpen.c	146
00100ef4	1.39	1.39	linebuffer_getval	edge_detect.c	159
00101038	1.05	1.05	window_insert	edge_detect.c	196
001016f8	.772	.772	linebuffer_insert_bottom	sharpen.c	158
00100f50	.710	.710	linebuffer_insert_bottom	edge_detect.c	171
00100714	.525	.525	dummyfill	SDSoC_lab_design_main.c	134
Called From					
Child Calls					
0010056c		100	main	SDSoC_lab_design_main.c	63

(a) Zed

Profiler running. 3282 samples					
Address	% Exclusive	% Inclusive	Function	File	Line
00101e30	.000	100	_start	xil-crt0.S	82
0010056c	.000	100	main	SDSoC_lab_design_main.c	63
00100b94	6.12	56.5	sobel_filter	edge_detect.c	76
0010133c	4.78	39.7	sharpen_filter	sharpen.c	63
001009a4	22.7	37.4	sobel_operator	edge_detect.c	34
00101228	13.2	21.4	sharpen_operator	sharpen.c	34
0010108c	14.7	14.7	window_getval	edge_detect.c	204
00101834	8.20	8.20	window_getval	sharpen.c	191
00101734	7.07	7.07	window_shift_right	sharpen.c	168
00100f8c	6.92	6.92	window_shift_right	edge_detect.c	181
001010e0	3.23	3.23	rgb_2_gray	rgb_2_gray.c	6
00100e58	2.29	2.29	linebuffer_shift_up	edge_detect.c	147
001017e0	2.10	2.10	window_insert	sharpen.c	183
00101600	2.07	2.07	linebuffer_shift_up	sharpen.c	134
00101038	1.86	1.86	window_insert	edge_detect.c	196
0010169c	1.40	1.40	linebuffer_getval	sharpen.c	146
00100ef4	1.31	1.31	linebuffer_getval	edge_detect.c	159
001016f8	.945	.945	linebuffer_insert_bottom	sharpen.c	158
00100f50	.609	.609	linebuffer_insert_bottom	edge_detect.c	171
00100714	.548	.548	dummyfill	SDSoC_lab_design_main.c	134
Called From					
Child Calls					
0010056c		100	main	SDSoC_lab_design_main.c	63

(b) Zybo

Figure 9. The TCF Profiler result

Address is the location of the function in memory that will match what is shown in the Disassembly view.

% Exclusive is the percentage of samples encountered by the profiler for that function only (excluding samples of any child functions). This can also be seen as exclusive percentage for that particular function.

% Inclusive is the percentage of samples of a function, including samples collected during execution of any child functions.

Function is the name of the function being sampled.

File is the name of the file containing the function.

Line indicates the line number where the function is found in the source file.

- 2-2-4.** Note that `_start` and `main` functions are 100% under the **%inclusive** column as all other functions are called from `main`. They are essentially 0% under the **%exclusive** column as a negligible time spent in those functions. Also note that `_exit` function is not listed since we did not profile to the completion (it would have taken about 10 minutes).
- 2-2-5.** Looking under the **%inclusive** column, notice that the CPU spent about 50% of its time executing the `sharpen_filter` function and its sub-functions.
- 2-2-6.** Click on the `sharpen_filter` entry to see that the source code window shows up.
- You can view the source code and see that it processes some data and calls several functions.
- 2-2-7.** Switch back to the *TCF Profile* result window and observe that the `sharpen_filter` function calls `sharpen_operator`, `window_shift_right`, `linebuffer_shift_up`, `linebuffer_getval`, and `window_insert` functions.

The same **Child Calls** window shows how much time the CPU spent in each of those functions.

00100b94	4.85	50.4	sobel_filter
0010133c	5.87	45.8	sharpen_filter
001009a4	19.8	33.4	sobel_operator
00101228	16.1	24.9	sharpen_operator
00101734	7.81	7.81	window_shift_right
00101600	2.32	2.32	linebuffer_shift_up
001017e0	2.16	2.16	window_insert
0010169c	1.91	1.91	linebuffer_getval
001016f8	.772	.772	linebuffer_insert_bottom

Child Calls			
00101228		24.9	sharpen_operator
00101734		7.81	window_shift_right
00101600		2.32	linebuffer_shift_up
001017e0		2.16	window_insert
0010169c		1.91	linebuffer_getval
001016f8		.772	linebuffer_insert_bottom

(a) Zed

00100b94	6.12	56.5	sobel_filter
0010133c	4.78	39.7	sharpen_filter
001009a4	22.7	37.4	sobel_operator
00101228	13.2	21.4	sharpen_operator
0010108c	14.7	14.5	window_getval

Child Calls			
00101228		21.4	sharpen_operator
00101734		7.07	window_shift_right
001017e0		2.10	window_insert
00101600		2.07	linebuffer_shift_up
0010169c		1.40	linebuffer_getval
001016f8		.945	linebuffer_insert_bottom

(b) Zybo

Figure 10. Child Calls from sharpen_filter function

2-2-8. Also note that *window_shift_right*, *linebuffer_shift_up*, *linebuffer_getval*, and *window_insert* appear multiple times. This is because they are called from different places. Click on any of these entries and notice that it only has a **Called From** entry and not a **Child Calls** indicating that these functions do not have any sub-functions, or they are leaf functions.

2-2-9. Looking at the results sorted in the **%inclusive** column, we can see that *sharpen_filter* may be a good candidate for the hardware acceleration. The function and sub-functions should be carefully considered to determine suitability for acceleration. Typical candidates for acceleration are functions that can process a stream of data, or can be implemented in parallel, without excessive resource utilization.

2-2-10. Click on the **%Exclusive** column to sort the results.

You can see that the CPU spends a large proportion of the total time in the *sharpen_operator* function. This may be a good candidate for acceleration.

2-2-11. Click on the Disconnect button (🔌) to terminate the session.

Accelerate sharpen_filter and Profile

Step 3

3-1. Add sharpen_filter function for hardware acceleration. Change SDSCC compiler setting to define TIME_SHARPEN symbol. Build the project and analyze the data motion network.

3-1-1. Switch back to the SDSoc perspective.

- 3-1-2. Click on the “+” sign in the Hardware Functions area to open up the list of functions which are in the source files.
- 3-1-3. Select *sharpen_filter* function and click **OK**.
- 3-1-4. Double-click the **SDSoC_lab_design_main.c** under *lab3 > src*.
- 3-1-5. Note several conditional compilation statements around lines 83 to 103.
- 3-1-6. Right click on *lab3* in the Project Explorer window and select *C/C++ Build Settings*.
- 3-1-7. Select *Symbols* under SDSCC Compiler and click “+” button to define a symbol.
- 3-1-8. Enter **TIME_SHARPEN** in the field and click **OK**.
- 3-1-9. Click **OK** again.

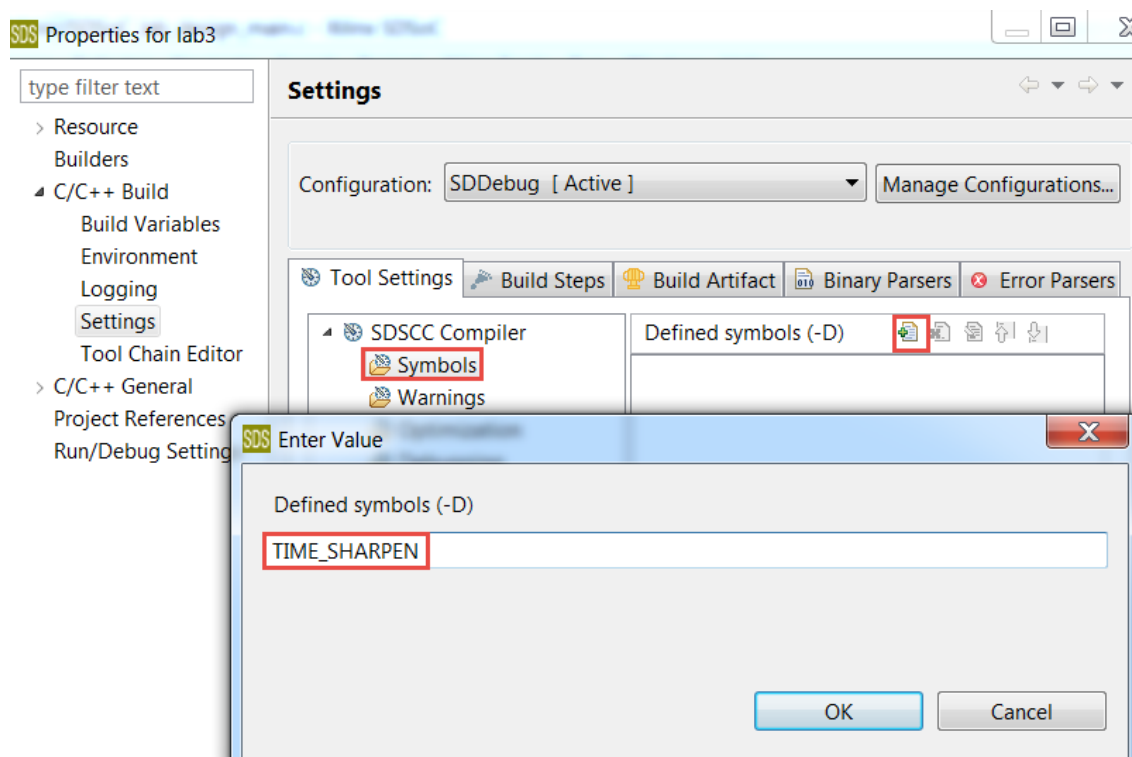


Figure 11. Defining symbol for conditional compilation

3-2. Build the project and analyze the data motion network.

- 3-2-1. Right-click the top-level folder for the project and click on **Clean Project** in the menu.
- 3-2-2. Right-click the top-level folder for the project and click on **Build Project** in the menu.

This may take about 20 minutes.

3-2-3. When build process is done, select the **lab3** tab so you can access Data Motion link.

3-2-4. Click on the **Data Motion report** link and analyze the result.

Data Motion Network						
Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
sharpen_filter_0	input	input_r	IN	2073600*1	• buffer_depth:2073600	S_AXI_HP0:AXIDMA_SIMPLE
	output	output_r	OUT	2073600*1	• buffer_depth:2073600	S_AXI_HP0:AXIDMA_SIMPLE
	return	ap_return	OUT	4		M_AXI_GP0:AXILITE:0xC0

Accelerator Callsites						
Accelerator	Callsite		IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
sharpen_filter_0	SDSoC_lab_design_main.c:93:3		input_r	2073600 * 1	contiguous	cacheable
			output_r	2073600 * 1	contiguous	cacheable
			ap_return	4	paged	cacheable

Figure 12. Data Motion network showing `buffer_depth` pragmas

3-3. Open Vivado IPI design.

3-3-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDSoC 2015.4 > Vivado Design Suite > Vivado 2015.4**

3-3-2. Open the design by browsing to `c:\xup\SDSoC\labs\lab3\SDDebug_sds\p0\ipi` and selecting either the **zybo.xpr** or **zed.xpr**.

3-3-3. 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).

3-3-4. Click on the **show interface connections only** () button followed by click on the **regenerate layout** () button.

3-3-5. Follow through both input and output data paths of the *sharpen_filter_0* instance and observe that they are connected to the **S_AXI_HP0** port of PS7.

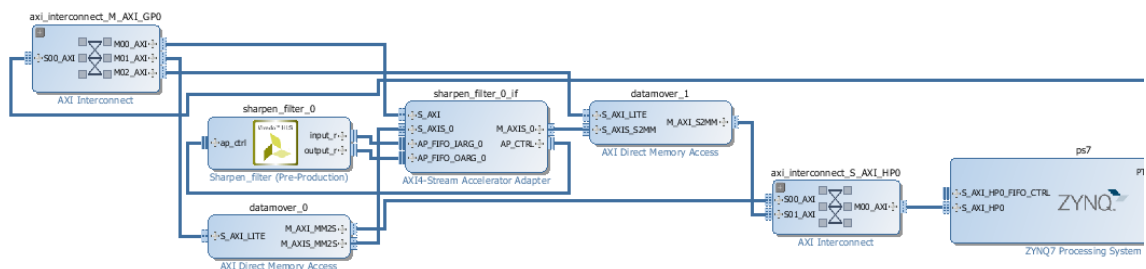


Figure 13. Built design

Notice that two data movers are used; one for input and another for output data. They both connect to S_AXI_HP0 of PS7 through the axi_interconnect_S_AXI_HP0 instance. The two data movers and the sharpen_filter_0_if instance can be configured by their S_AXI_LITE interfaces which are connected to the ps7 via the axi_interconnect_M_AXI_GP0 instance.

3-3-6. Close Vivado by selecting **File > Exit**. Do not save the block design.

3-4. Connect the board and power it ON. Start the Debug session. Add the TCF Profiler view and configure it to include the *Aggregate per Function* option.

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

3-4-2. Click on the **Debug Application** link under the *Actions* window in the *lab3* tab on the right-side.

Notice that the Done LED on the board goes OFF and then goes back ON this time as the FPGA fabric gets configured using the built hardware design.

3-4-3. Click **Yes** to open the debug perspective, if prompted.

Notice that the program is suspended at the `main()` entry on line 75 (instead of 68 in Figure 5).

If you scroll up into the `main()` function window, you will notice code is added on lines 63 to 69 which declares `_p0_sharpen_filter_0` function prototype.

```
63 #ifdef __cplusplus
64 extern "C" {
65 #endif
66 int _p0_sharpen_filter_0(uint8_t * input, uint8_t * output);
67 #ifdef __cplusplus
68 }
69 #endif
```

Figure 14. Function prototype for the accelerated function

3-4-4. Add *TCF Profiler* view as before, and configure the TCF Profiler view to include the *Aggregate per function* option.

3-5. Run the application for about 2 minutes, then suspend, and analyze the data.

3-5-1. Press the **Start** button of the TCF Profiler.

3-5-2. Click on the **Resume** button (Green box) on the tool buttons bar to start the execution.

3-5-3. Wait for about two minutes and/or when about 3200 samples are collected as indicated in the *TCF Profiler's* view and click the **Suspend** button (green oval).

Note that the number of collected samples may vary depending on your PC's performance and connection speed with the board.

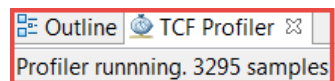


Figure 15. The TCF Profiler view showing the collected number of samples

3-5-4. Click on the **Maximize** view button.

Address	% Exclusive	% Inclusive	Function	File	Line
00101e80	.000	100	_start	xil-crt0.S	82
00100df0	.000	100	main	SDSoC_lab_design_main.c	70
0010075c	10.3	96.1	sobel_filter	edge_detect.c	76
0010056c	36.6	62.2	sobel_operator	edge_detect.c	34
00100c54	25.6	25.6	window_getval	edge_detect.c	204
00100b54	12.6	12.6	window_shift_right	edge_detect.c	181
00100a20	4.28	4.28	linebuffer_shift_up	edge_detect.c	147
00100ca8	3.30	3.30	rgb_2_gray	rgb_2_gray.c	6
00100c00	3.18	3.18	window_insert	edge_detect.c	196
00100abc	2.23	2.23	linebuffer_getval	edge_detect.c	159
00100b18	1.28	1.28	linebuffer_insert_bottom	edge_detect.c	171
00101074	.595	.595	dummyfill	SDSoC_lab_design_main.c	141
001016dc	.000	.060	_p0_sharpen_filter_0	sharpen.c	141
0010b1ec	.000	.059	cf_wait		
00112038	.059	.059	axi_dma_simple_wait		

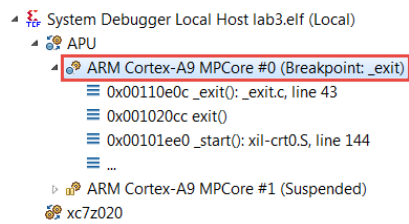
Figure 16. The TCF Profiler result (Zed)

Address	% Exclusive	% Inclusive	Function	File	Line
00101e80	.000	100	_start	xil-crt0.S	82
00100df0	.000	100	main	SDSoC_lab_design_main.c	70
0010075c	9.80	96.7	sobel_filter	edge_detect.c	76
0010056c	39.2	64.7	sobel_operator	edge_detect.c	34
00100c54	25.5	25.5	window_getval	edge_detect.c	204
00100b54	11.8	11.8	window_shift_right	edge_detect.c	181
00100a20	3.90	3.90	linebuffer_shift_up	edge_detect.c	147
00100c00	3.15	3.15	window_insert	edge_detect.c	196
00100ca8	3.06	3.06	rgb_2_gray	rgb_2_gray.c	6
00100abc	2.19	2.19	linebuffer_getval	edge_detect.c	159
00100b18	1.08	1.08	linebuffer_insert_bottom	edge_detect.c	171
00101074	.240	.240	dummyfill	SDSoC_lab_design_main.c	141
001016dc	.000	.060	_p0_sharpen_filter_0	sharpen.c	141
0010b1ec	.000	.060	cf_wait		
00112038	.060	.060	axi_dma_simple_wait		

Figure 16. The TCF Profiler result (Zybo)

Note that *_start* and *main* functions are 100% under the **%inclusive** column as all other functions are called from *main*. Now the CPU spent most of its time executing the *sobel_filter* function and its sub-functions and very small amount of time is spent on the *_p0_sharpen_filter_0* call (the hardware accelerator).

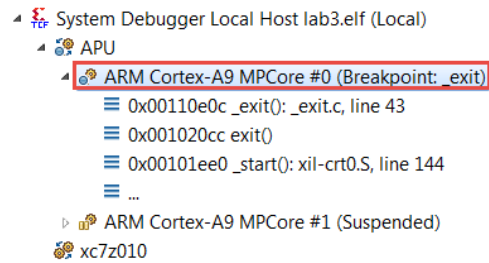
- 3-5-5.** Click on the *Resume* button and wait for little longer (3 more minutes) and see that the application runs to the end and the profiling stops at around 8600 samples in case of Zed or 5600 samples in case of Zybo.



Profiler running. 8663 samples

Address	% Exclusive	% Inclusive	Function
00101e80	.000	100	_start
00100df0	.000	100	main
0010075c	10.1	96.3	sobel_filter
0010056c	38.5	63.4	sobel_operator
00100c54	24.9	24.9	window_getval
00100b54	11.9	11.9	window_shift_right
00100a20	4.29	4.29	linebuffer_shift_up
00100ca8	3.38	3.38	rgb_2_gray
00100c00	3.29	3.29	window_insert
00100abc	2.32	2.32	linebuffer_getval

(a) Zed



Profiler running. 4744 samples

Address	% Exclusive	% Inclusive	Function
00101e80	.000	100	_start
00100df0	.000	100	main
0010075c	9.89	96.5	sobel_filter
0010056c	39.4	64.8	sobel_operator
00100c54	25.5	25.5	window_getval
00100b54	11.8	11.8	window_shift_right
00100a20	3.67	3.67	linebuffer_shift_up

(b) Zybo

Figure 17. Execution suspended at _exit

3-5-6. Click on the **Disconnect** button (🔌) to terminate the execution.

Profiling Using sds_lib API

Step 4

4-1. Re-launch the application in the Debug perspective. Start the terminal session and run the application to the end.

4-1-1. In the *Debug* view, right-click on the disconnected entry and select **Relaunch**.

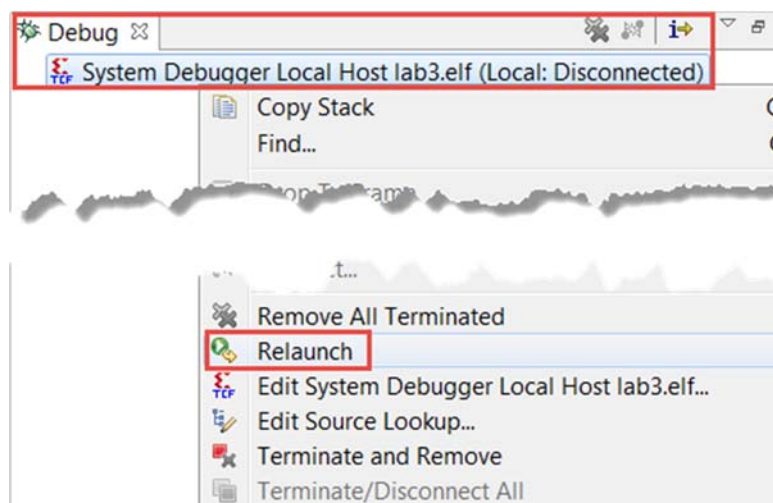


Figure 18. Re-launching the debugger

4-1-2. Click on the SDK *Terminal* window and make a connection.

4-1-3. Click on the **Resume** button.

4-1-4. You will see dots being displayed as the execution is continuing. You will also see progress is made in the TCF Profiler view.

Wait for about five minutes to complete the execution and the result is displayed in the Terminal window.

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions:    16344568146
Average SW cycles for sharpen:                      9696221
```

(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions:    16363538526
Average SW cycles for sharpen:                      13504170
```

(b) Zybo

Figure 19. The sharpen function profiling

4-1-5. Click on the **Disconnect** button (🔌).

Add sobel_filter to Accelerators and Profile

Step 5

5-1. **Add sobel_filter function for hardware acceleration. Change SDSCC compiler setting to define TIME_EDGE_DETECT symbol. Build the project.**

Since this will take time to build, you will import lab3a project from the source\lab3 folder and then profile the application. The precompiled project has the sobel_filter already added for hardware with the compiler setting added.

5-1-1. Switch back to the SDSoc perspective.

5-1-2. Select **File > Import**

5-1-3. Double-click on *Import Existing Projects into Workspace*.

5-1-4. In the *Import Projects* window, click on the **Browse** button of the *Select archive file* option, browse to c:\xup\SDSoC\source\lab3, select *lab3a.zip* and click **Open**.

Make sure that lab3a is checked in the Projects window.

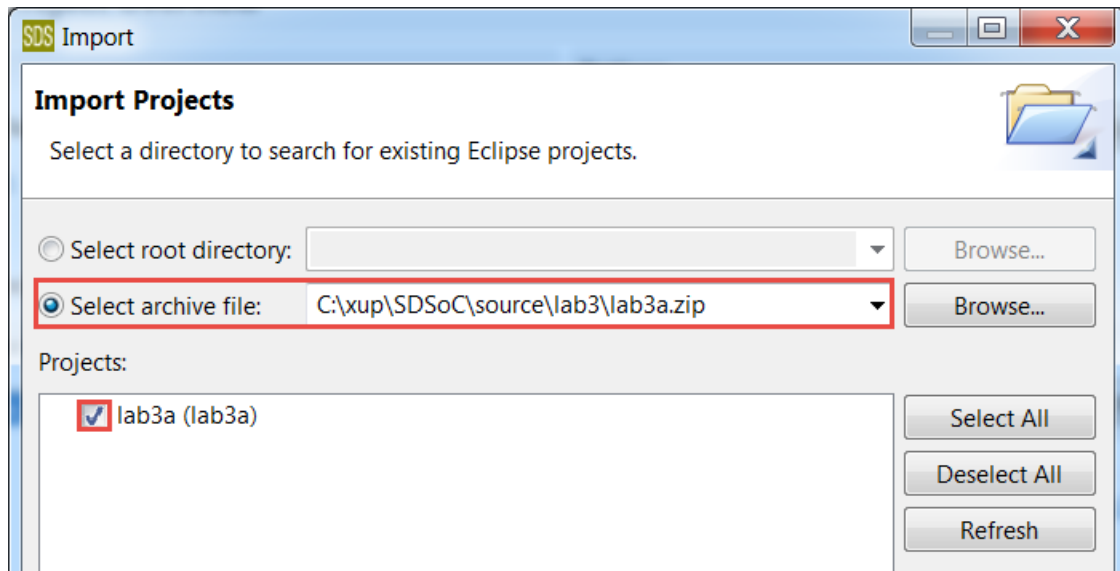


Figure 20. Importing an existing project in the workspace

5-1-5. Click **Finish**.

The project will be imported and the *sobel_filter* and *sharpen_filter* function entries will be displayed in the *Hardware Functions* window.

5-1-6. Double-click on the **project_sdsoc** under *lab3a* to access the *SDSoC Project Overview*.

5-1-7. Uncheck the *Generate Bit Stream* and *Generate SD Card Image* options.

5-1-8. Click on the **Debug Application** link under the *Actions* section.

5-1-9. Click **Yes** to switch to the debug perspective if prompted.

5-1-10. Select **Window > Show View > Other** and then expand the *Debug* folder. Select *TCF Profiler* and click **OK**.

5-1-11. In the *TCF Profiler* view, click the start button, select the *Aggregate per function* option and click **OK**.

5-2. Start serial communication. Profile the complete application and observe the improvements.


5-2-1. Select the Terminal tab and make serial communication.

5-2-2. Click on the Resume button.

5-2-3. You will see dots being displayed as the execution progresses. You will also see progress is made in the TCF Profiler view.


The execution should complete in under a minute and the result is displayed in the Terminal window.

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions: 6001461064
Average SW cycles for sharpen: 21728298
Average SW cycles for edge_detect: 22476573
```



(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions: 4692222236
Average SW cycles for sharpen: 22760834
Average SW cycles for edge_detect: 21106570
```



(b) Zybo

Figure 21. The sharpen and sobel filter functions profiling

5-2-4. Switch to the TCF Profiler tab and see the results.

Note that now CPU spends time in `rgb_2_grap` function. The `_p0_sobel_filter_0` takes very little time and you don't see the `_p0_sharpen_filter_0` entry does not appear at all since its execution time is so short that the profiler does not see it.

Address	% Exclusive	% Inclusive	Function	File	Line
00102080	.000	99.6	_start	xil-crt0.S	82
001006b4	.000	99.6	main	SDSoC_lab_design_main.c	77
0010056c	87.4	87.4	rgb_2_gray	rgb_2_gray.c	6
00100a14	8.79	8.79	dummyfill	SDSoC_lab_design_main.c	148
0010b3ec	.000	3.35	cf_wait		
00112338	3.35	3.35	axi_dma_simple_wait		
00101158	.000	1.67	_p0_sobel_filter_0	edge_detect.c	154
00101904	.000	1.67	_p0_sharpen_filter_0	sharpen.c	141
001023b8	.000	.419	printf		
001025c0	.000	.419	_vfprintf_r		
00109500	.000	.419	__sprintf_r.part.0		
001065d4	.000	.419	__sfvwrite_r		
00105c58	.000	.419	__sflush_r		
0010a998	.000	.418	_write_r		
00111388	.000	.418	_write	write.c	84
00111ed4	.418	.418	XUartPs_SendByte	xuartps_hw.c	81

Figure 22. Profiled data (Zed)

Address	% Exclusive	% Inclusive	Function	File	Line
00102080	.000	99.4	_start	xil-crt0.S	82
001006b4	.000	99.4	main	SDSoC_lab_design_main.c	77
0010056c	90.4	90.4	rgb_2_gray	rgb_2_gray.c	6
0010b3ec	.000	4.79	cf_wait		
00112338	4.79	4.79	axi_dma_simple_wait		
00100a14	4.19	4.19	dummyfill	SDSoC_lab_design_main.c	148
00101158	.000	2.40	_p0_sobel_filter_0	edge_detect.c	154
00101904	.000	2.40	_p0_sharpen_filter_0	sharpen.c	141
001023b8	.000	.599	printf		
001025c0	.000	.599	_vfprintf_r		
00109500	.000	.599	__sprintf_r.part.0		
001065d4	.000	.599	__sfvwrite_r		
00105c58	.000	.599	__sflush_r		
0010a998	.000	.599	_write_r		
00111388	.000	.599	_write	write.c	84
00111ed4	.599	.599	XUartPs_SendByte	xuartps_hw.c	81

Figure 22. Profiled data (Zybo)

5-2-5. Click on the **Disconnect** button (🔌) to terminate the execution.

5-3. Profile the application without running the profiler and compare the result.

5-3-1. In the *Debug* view, right-click on the disconnected entry and select **Relaunch**

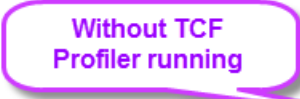
5-3-2. This time do not click on the start button of the TCF Profiler.

5-3-3. Click on the Resume button.

5-3-4. You will see dots being displayed quickly as the execution is continuing.


5-3-5. Notice the terminal output.

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions: 731594608
Average SW cycles for sharpen: 9695774
Average SW cycles for edge_detect: 9694562
```



(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions: 769662714
Average SW cycles for sharpen: 13503676
Average SW cycles for edge_detect: 13502574
```



(b) Zybo

Figure 23. The terminal window output

Compared to output with the profiler running, the execution takes significantly fewer cycles.

5-3-6. Click on the **Disconnect** button (🔌) to terminate the execution.

5-3-7. Close SDSoc by selecting **File > Exit**

5-3-8. Turn OFF the power to the board.

Conclusion

In this lab, you profiled a pure software application which consist of three major functions. You saw the amount of time those three functions took to execute. Then you ported one of the most time-consuming function into hardware and profiled again. You then ported second most time-consuming function into hardware and profiled again and observed the performance improvement. You used the TCF profiler and sds_lib API to collect the data.