Profiling Applications and Creating 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:

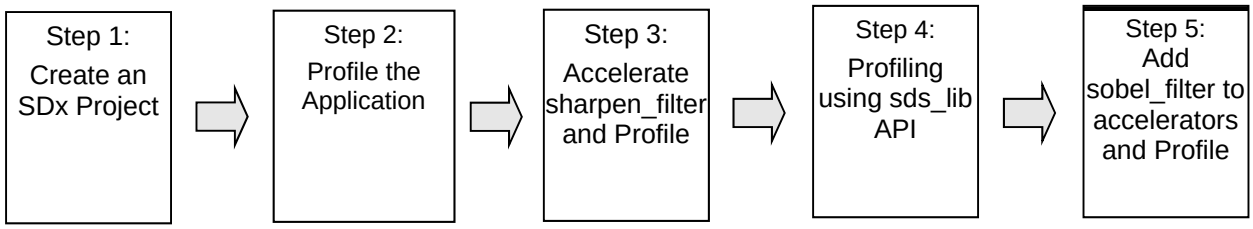
- Use TCF profiler to profile a pure software application
- Use TCF profiler to profile a software application that calls functions ported to hardware
- Use manual profiling method by using sds_lib API and counters

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 SDx 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 SDx Project Step 1

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

1-1-2. Create a new project called **lab3**

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 OS** as the *System Configuration*, and click **Next**.

The Templates page appears, containing source code examples for the selected platform.

1-1-6. 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 Debug 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**.

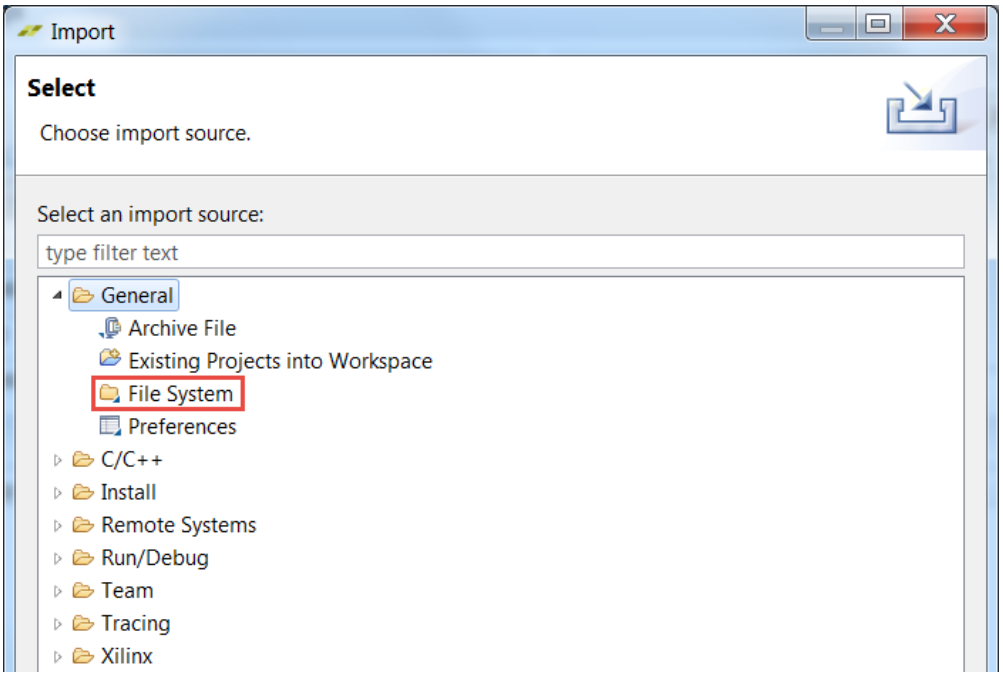


Figure 1. Selecting import source location

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.

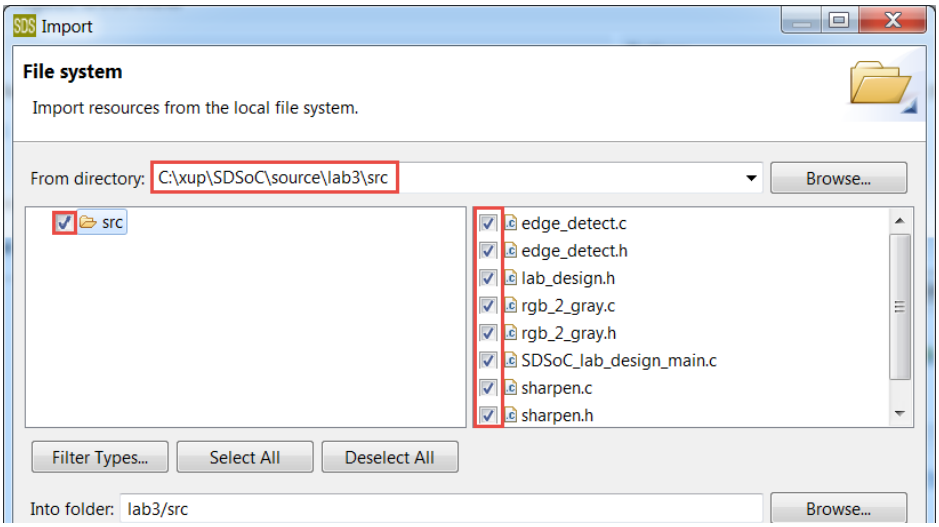


Figure 2. Selecting path and files to be imported

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

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. Right-click on the **lab3** entry in the *Project Explorer* tab and select **Debug As > Launch on Hardware (SDSoC Debugger)**

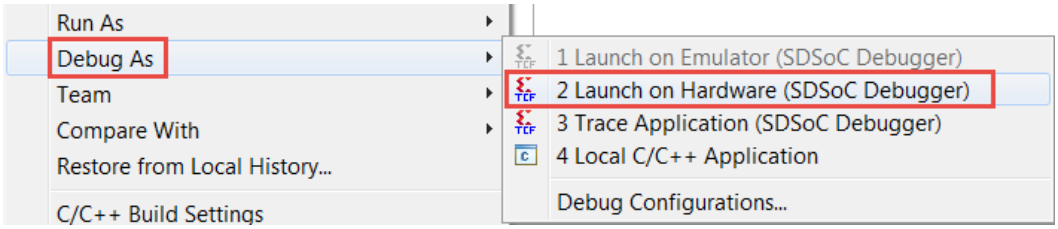


Figure 3. Executing Debug Application action

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

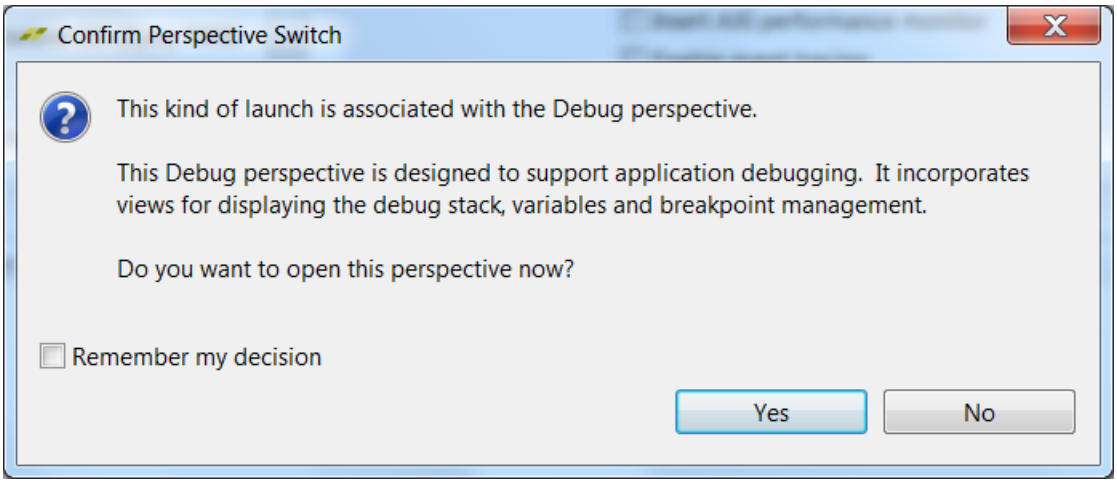


Figure 4. Perspective Switch dialog window

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.

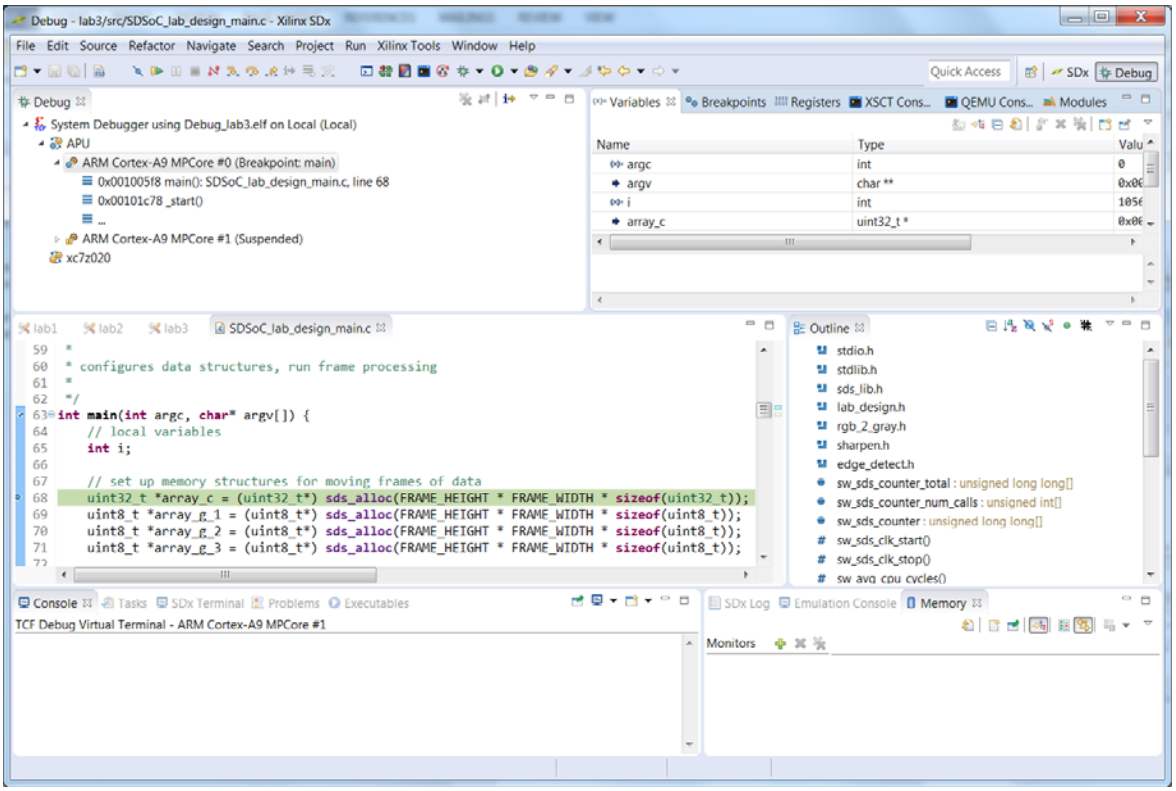


Figure 5. The Debug perspective

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.

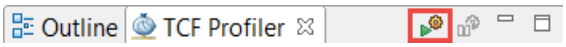


Figure 6. Opening the TCF Profiler configuration

The *Profiler Configuration* window will open.

2-1-7. Leave *Aggregate per function* selected, and select the *Enable stack tracing* option and click **OK**.

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

The *Enable stack tracking* 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 between functions.

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.

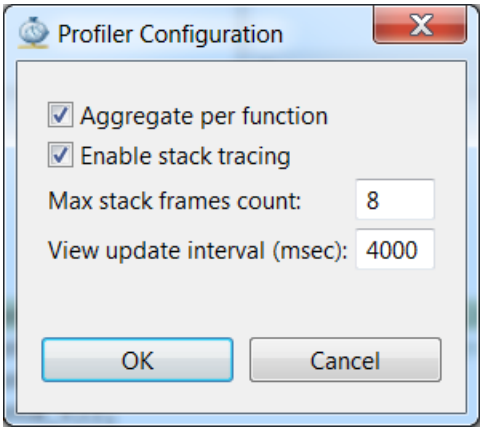


Figure 7. Selecting the options

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

2-2. Run the application 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. Note the number of collected samples, when finished execution (for Zed), may vary depending on your PC's performance and connection speed with the board. For Zybo, press Pause button after collecting about 3200 samples.

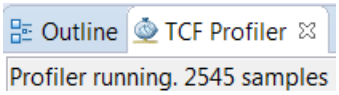


Figure 8. The TCF Profiler view showing the collected number of samples

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*.

Outline | TCF Profiler

Profiler running. 2545 samples

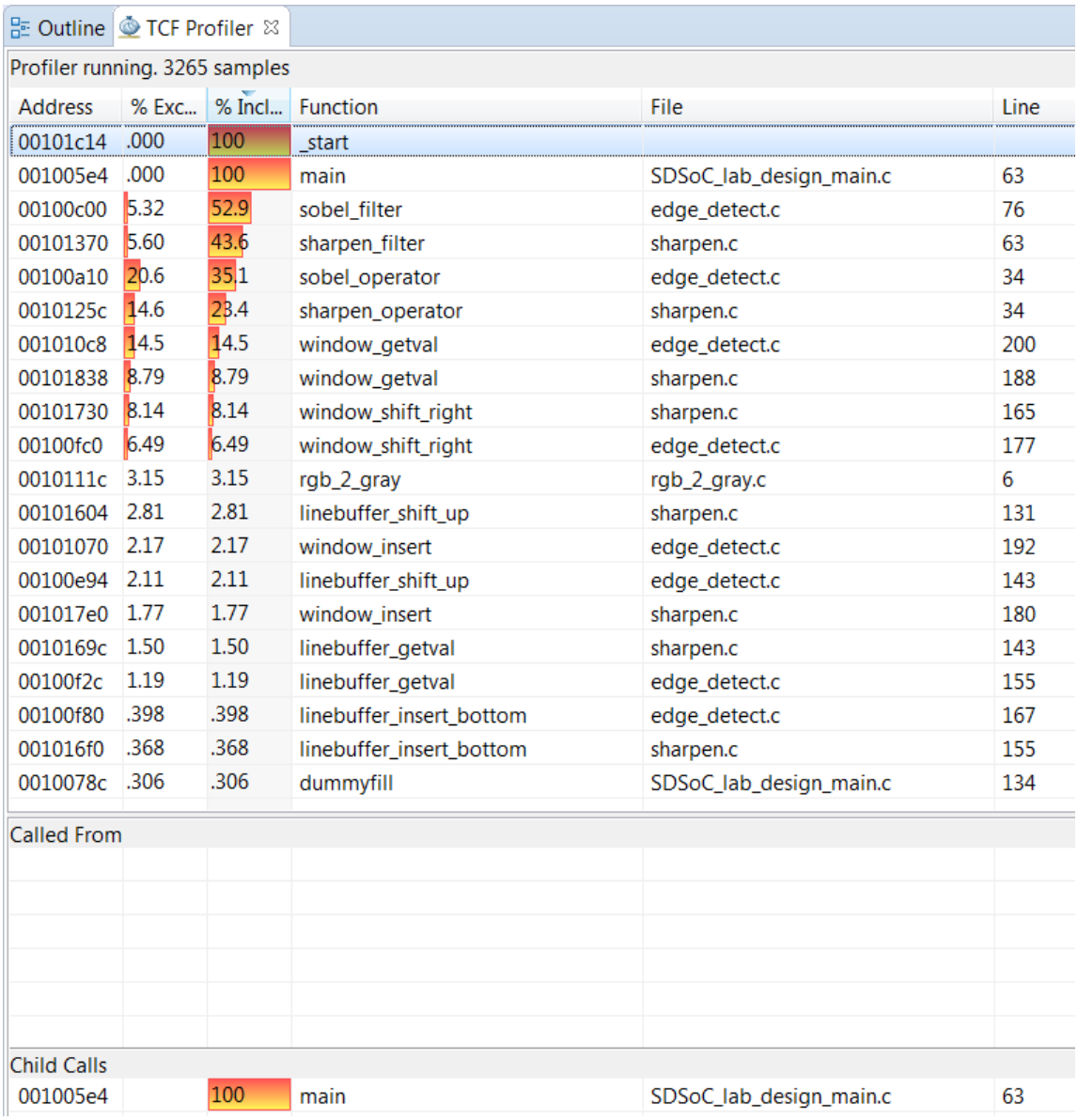
Address	% Exc...	% Incl...	Function	File	Line
00101c14	.000	100	_start		
001005e4	.000	99.9	main	SDSoC_lab_design_main.c	63
00100c00	5.38	59.4	sobel_filter	edge_detect.c	76
00100a10	24.0	40.2	sobel_operator	edge_detect.c	34
00101370	4.87	37.5	sharpen_filter	sharpen.c	63
0010125c	13.5	20.6	sharpen_operator	sharpen.c	34
001010c8	16.2	16.2	window_getval	edge_detect.c	200
00100fc0	7.66	7.66	window_shift_right	edge_detect.c	177
00101838	7.11	7.11	window_getval	sharpen.c	188
00101730	6.60	6.60	window_shift_right	sharpen.c	165
0010111c	2.75	2.75	rgb_2_gray	rgb_2_gray.c	6
00101604	2.27	2.27	linebuffer_shift_up	sharpen.c	131
00101070	2.24	2.24	window_insert	edge_detect.c	192
00100e94	1.92	1.92	linebuffer_shift_up	edge_detect.c	143
001017e0	1.72	1.72	window_insert	sharpen.c	180
00100f2c	1.57	1.57	linebuffer_getval	edge_detect.c	155
0010169c	.904	.904	linebuffer_getval	sharpen.c	143
001016f0	.472	.472	linebuffer_insert_bottom	sharpen.c	155
00100f80	.393	.393	linebuffer_insert_bottom	edge_detect.c	167
0010078c	.275	.275	dummyfill	SDSoC_lab_design_main.c	134
00101d9c	.000	.039	__libc_fini_array	fini.c	25
00115538	.039	.039	_exit		

Called From

Child Calls

001005e4		99.9	main	SDSoC_lab_design_main.c	63
00101d9c		.039	__libc_fini_array	fini.c	25

(a) Zed



(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.

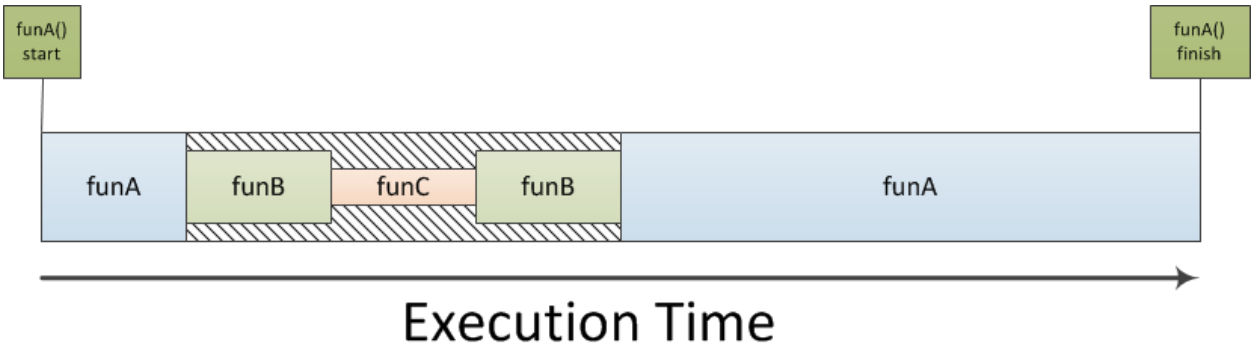


Figure 10. Understanding exclusive vs inclusive execution time

Exclusive: The amount of execution time spent in *funA* alone. Referencing the diagram below, the exclusive time for *funA* is represented by the combined execution time of the funA blocks only.

Inclusive: The amount of execution time spent in *funA* and all of its sub-function calls. From the diagram, this is the exclusive time of funA combined with the hatched area during which time *funB* and *funC* are executing

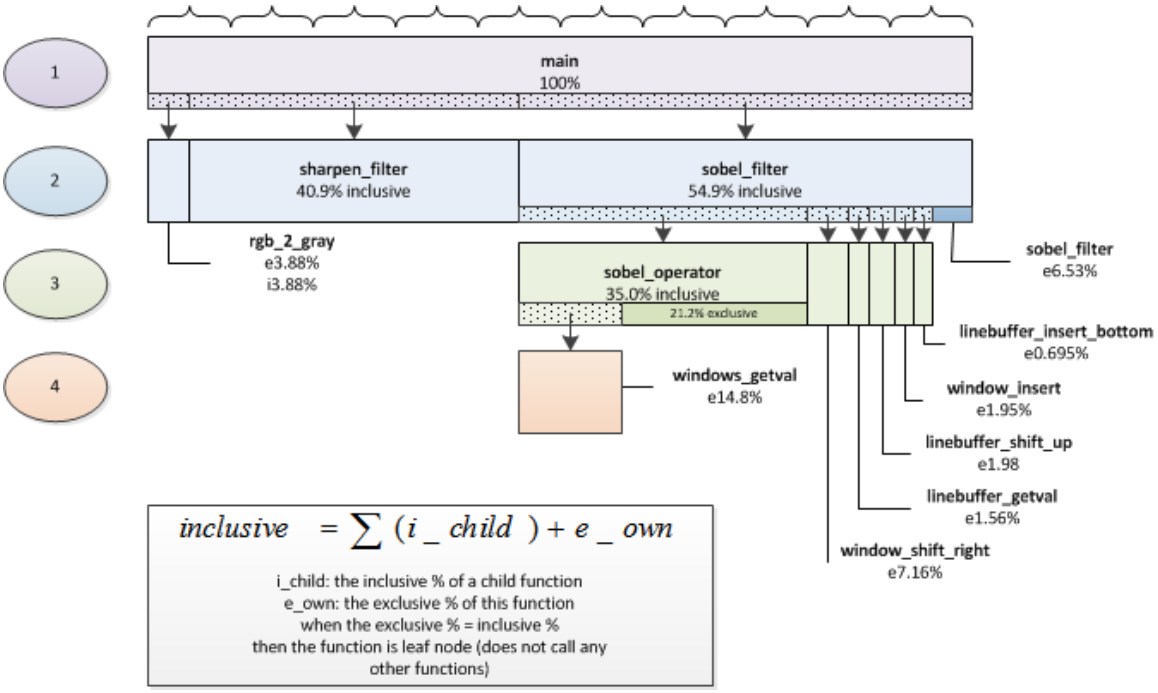


Figure 11. Various functions execution times (numbers may vary)

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.

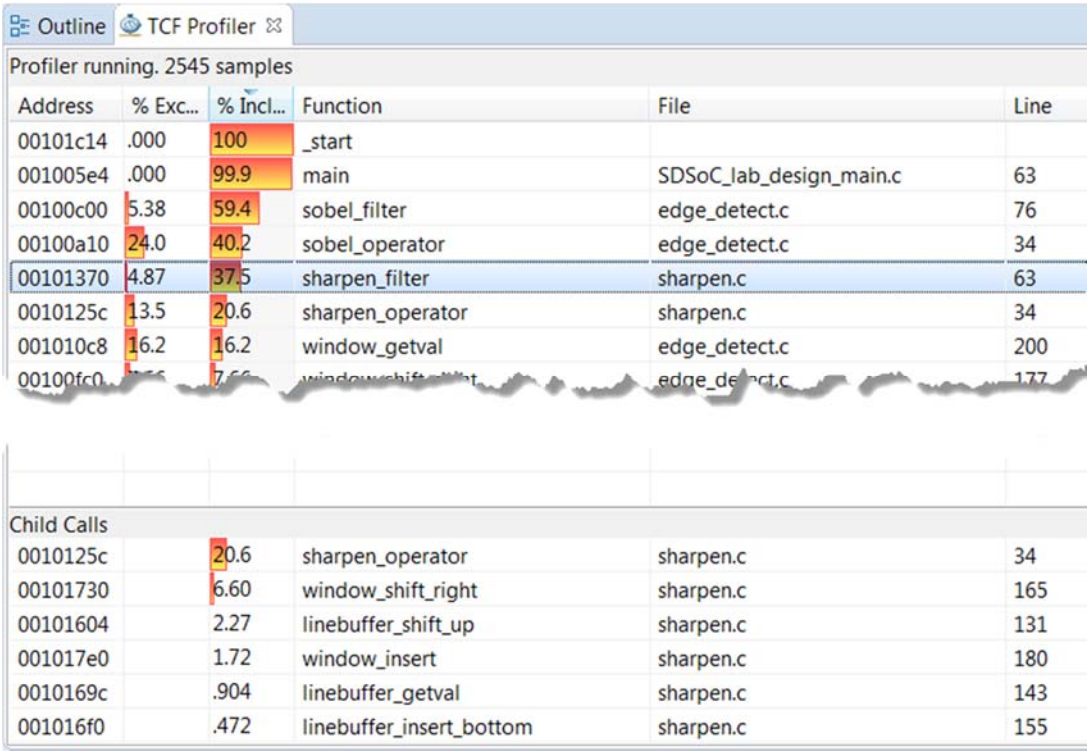
2-2-5. Looking under the **%inclusive** column, notice that the CPU spent about 37.5% (Zed) or 43.6% (Zybo) of its time executing the *sharpen_filter* function and its sub-functions.

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, window_insert , linebuffer_getval,* and *linebuffer_insert_bottom* functions.

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



Outline | TCF Profiler

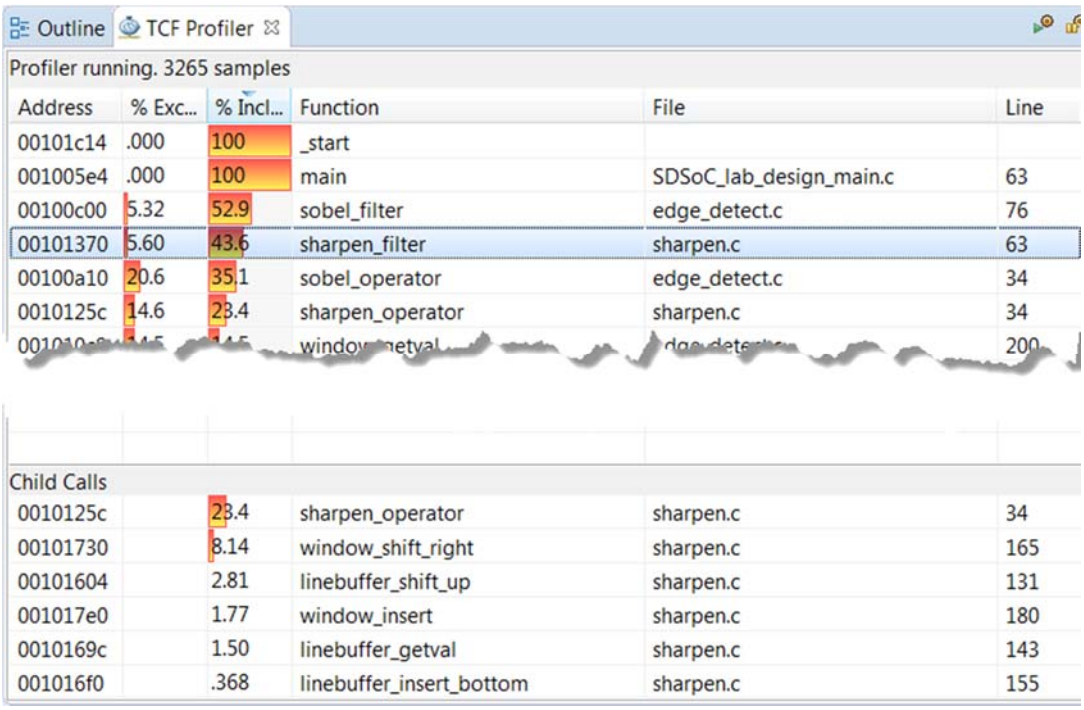
Profiler running. 2545 samples

Address	% Exc...	% Incl...	Function	File	Line
00101c14	.000	100	_start		
001005e4	.000	99.9	main	SDSoC_lab_design_main.c	63
00100c00	5.38	59.4	sobel_filter	edge_detect.c	76
00100a10	24.0	40.2	sobel_operator	edge_detect.c	34
00101370	4.87	37.5	sharpen_filter	sharpen.c	63
0010125c	13.5	20.6	sharpen_operator	sharpen.c	34
001010c8	16.2	16.2	window_getval	edge_detect.c	200

Child Calls

0010125c	20.6	sharpen_operator	sharpen.c	34
00101730	6.60	window_shift_right	sharpen.c	165
00101604	2.27	linebuffer_shift_up	sharpen.c	131
001017e0	1.72	window_insert	sharpen.c	180
0010169c	.904	linebuffer_getval	sharpen.c	143
001016f0	.472	linebuffer_insert_bottom	sharpen.c	155

(a) Zed



Outline | TCF Profiler

Profiler running. 3265 samples

Address	% Exc...	% Incl...	Function	File	Line
00101c14	.000	100	_start		
001005e4	.000	100	main	SDSoC_lab_design_main.c	63
00100c00	5.32	52.9	sobel_filter	edge_detect.c	76
00101370	5.60	43.6	sharpen_filter	sharpen.c	63
00100a10	20.6	35.1	sobel_operator	edge_detect.c	34
0010125c	14.6	23.4	sharpen_operator	sharpen.c	34

Child Calls

0010125c	23.4	sharpen_operator	sharpen.c	34
00101730	8.14	window_shift_right	sharpen.c	165
00101604	2.81	linebuffer_shift_up	sharpen.c	131
001017e0	1.77	window_insert	sharpen.c	180
0010169c	1.50	linebuffer_getval	sharpen.c	143
001016f0	.368	linebuffer_insert_bottom	sharpen.c	155

(b) Zybo

Figure 12. Child Calls from sharpen_filter function

2-2-8. 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.

2-2-9. 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.

2-2-10. 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 SDx perspective.

Make sure that the Project Overview tab for the lab is displayed.

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. When a symbol is defined, and the condition is true, these statements will allow the corresponding function(s) to be timed.

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.

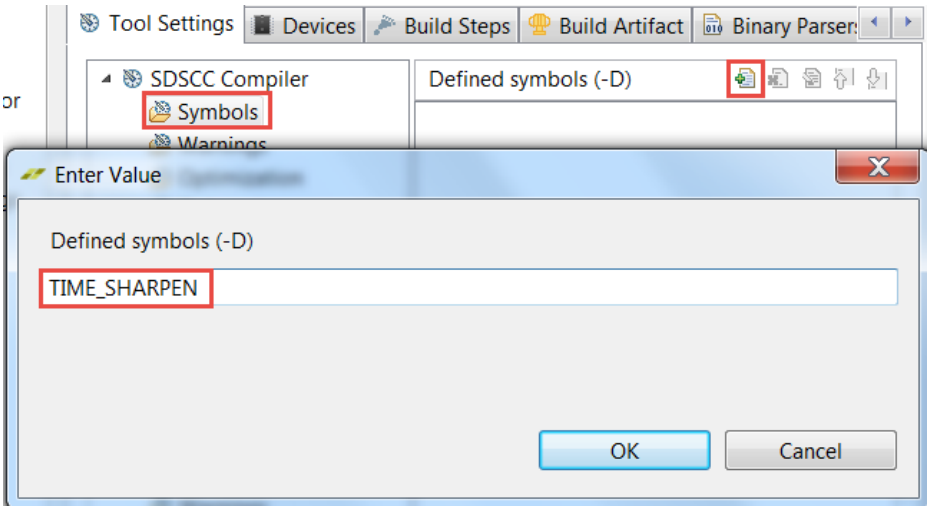


Figure 13. Defining symbol for conditional compilation

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_1	input	input_r	IN	2073600*1		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	output	output_r	OUT	2073600*1		ps7_S_AXI_ACP:AXIDMA_SIMPLE

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time (CPU cycles)
sharpen_filter_1	SDSoC_lab_design_main.c:93:3	input_r	2073600	contiguous	1015	2434592
		output_r	2073600	contiguous	1015	2434592

(a) Zed

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
sharpen_filter_1	input	input_r	IN	2073600*1		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	output	output_r	OUT	2073600*1		ps7_S_AXI_ACP:AXIDMA_SIMPLE

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time (CPU cycles)
sharpen_filter_1	SDSoC_lab_design_main.c:93:3	input_r	2073600	contiguous	1015	3457121
		output_r	2073600	contiguous	1015	3457121

(b) Zybo

Figure 14. Data Motion network

3-3. Open Vivado IPI design.

3-3-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > SDx 2016.3 > Vivado Design Suite > Vivado 2016.3**

3-3-2. Open the design by browsing to *c:\xup\SDSoC\labs\lab3\Debug_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_1* instance and observe that they are connected to the **S_AXI_ACP** port of *PS7*.

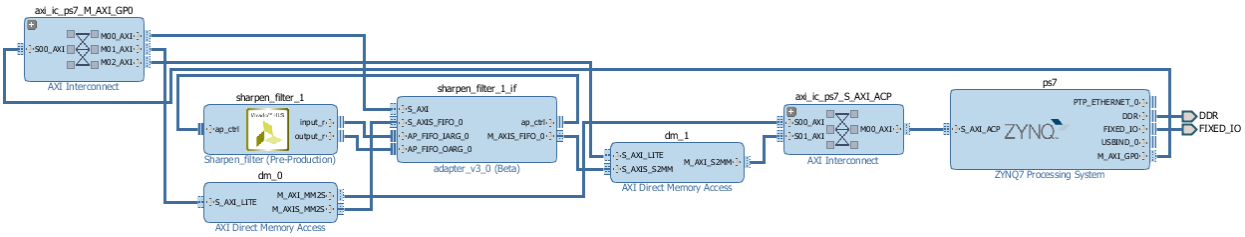


Figure 15. Built design

Notice that two data movers are used; one for input and another for output data. They both connect to S_AXI_ACP of PS7 through the axi_ic_ps7_S_AXI_ACP instance. The two data movers and the sharpen_filter_1_if instance can be configured by their S_AXI_LITE interfaces which are connected to the ps7 via the axi_ic_ps7_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. Right-click on the **lab3** entry in the *Project Explorer* tab and select **Debug As > Launch on Hardware (SDSoC Debugger)**

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_1_noasync* function prototype.

```
63 #ifdef __cplusplus
64 extern "C" {
65 #endif
66 void _p0_sharpen_filter_1_noasync(uint8_t input[2073600], uint8_t output[2073600]);
67 #ifdef __cplusplus
68 }
69 #endif
```

Figure 16. 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 and analyze the data.

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

3-5-2. Change the update interval setting to **1000** (for Zed only) since we want to collect samples at finer resolution.

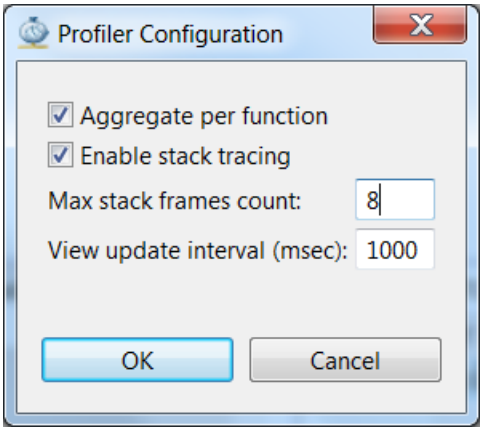
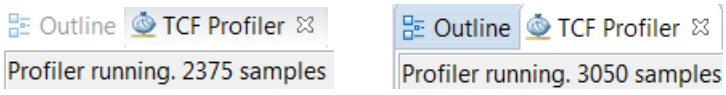


Figure 17. Setting update interval to 1 second (1000 msec)

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

3-5-4. Wait for the execution to complete.

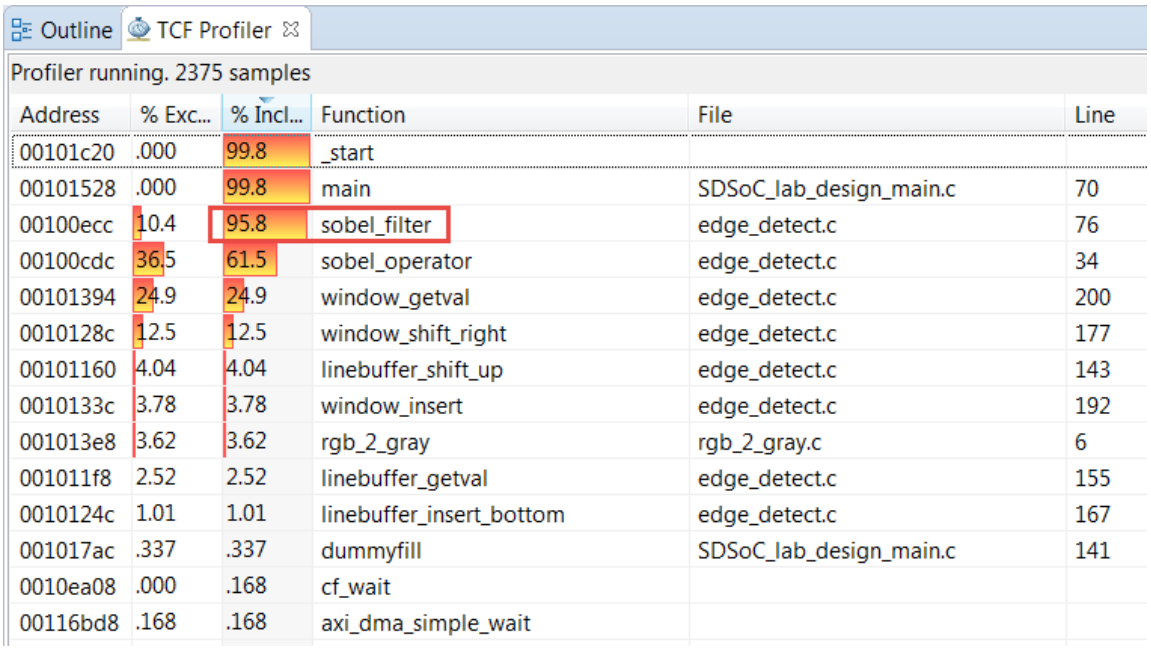
Note that the number of collected samples may vary depending on your PC's performance and connection speed with the board.



(a) Zed **(b) Zybo**

Figure 18. The TCF Profiler view showing the collected number of samples

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

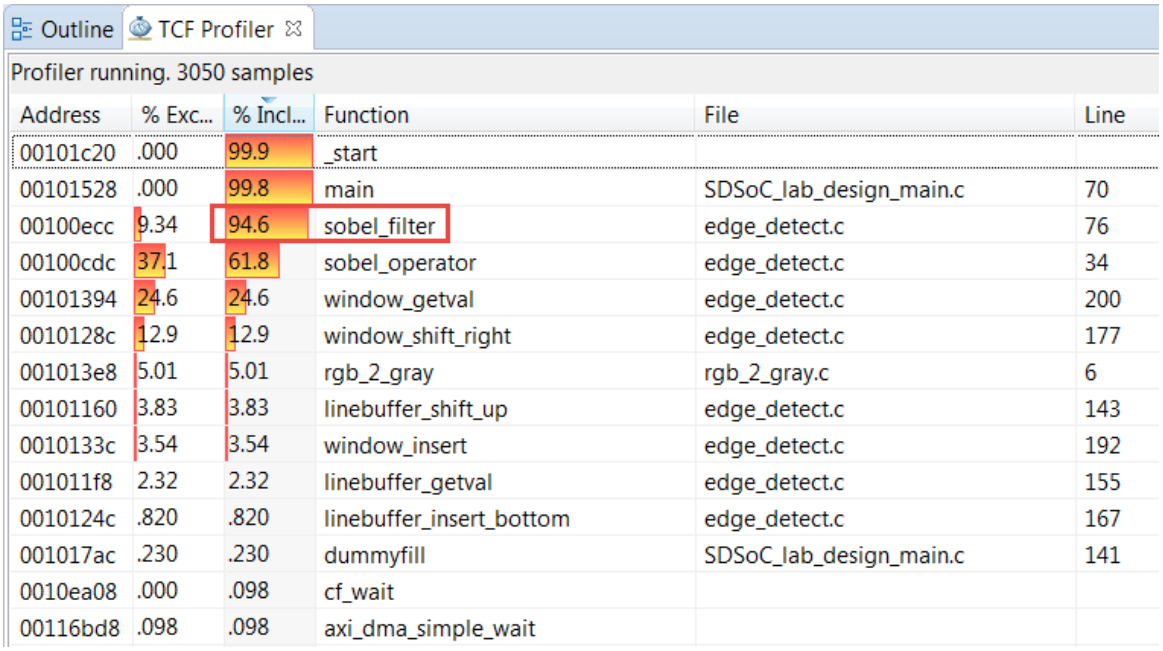


Outline | TCF Profiler

Profiler running. 2375 samples

Address	% Exc...	% Incl...	Function	File	Line
00101c20	.000	99.8	_start		
00101528	.000	99.8	main	SDSoC_lab_design_main.c	70
00100ecc	10.4	95.8	sobel_filter	edge_detect.c	76
00100cdc	36.5	61.5	sobel_operator	edge_detect.c	34
00101394	24.9	24.9	window_getval	edge_detect.c	200
0010128c	12.5	12.5	window_shift_right	edge_detect.c	177
00101160	4.04	4.04	linebuffer_shift_up	edge_detect.c	143
0010133c	3.78	3.78	window_insert	edge_detect.c	192
001013e8	3.62	3.62	rgb_2_gray	rgb_2_gray.c	6
001011f8	2.52	2.52	linebuffer_getval	edge_detect.c	155
0010124c	1.01	1.01	linebuffer_insert_bottom	edge_detect.c	167
001017ac	.337	.337	dummyfill	SDSoC_lab_design_main.c	141
0010ea08	.000	.168	cf_wait		
00116bd8	.168	.168	axi_dma_simple_wait		

(a) Zed



Outline | TCF Profiler

Profiler running. 3050 samples

Address	% Exc...	% Incl...	Function	File	Line
00101c20	.000	99.9	_start		
00101528	.000	99.8	main	SDSoC_lab_design_main.c	70
00100ecc	9.34	94.6	sobel_filter	edge_detect.c	76
00100cdc	37.1	61.8	sobel_operator	edge_detect.c	34
00101394	24.6	24.6	window_getval	edge_detect.c	200
0010128c	12.9	12.9	window_shift_right	edge_detect.c	177
001013e8	5.01	5.01	rgb_2_gray	rgb_2_gray.c	6
00101160	3.83	3.83	linebuffer_shift_up	edge_detect.c	143
0010133c	3.54	3.54	window_insert	edge_detect.c	192
001011f8	2.32	2.32	linebuffer_getval	edge_detect.c	155
0010124c	.820	.820	linebuffer_insert_bottom	edge_detect.c	167
001017ac	.230	.230	dummyfill	SDSoC_lab_design_main.c	141
0010ea08	.000	.098	cf_wait		
00116bd8	.098	.098	axi_dma_simple_wait		

(b) Zybo

Figure 19. The TCF Profiler result

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. You don't see *_p0_sharpen_filter_1* call (the hardware accelerator) since very little time is spent in that function.

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**.

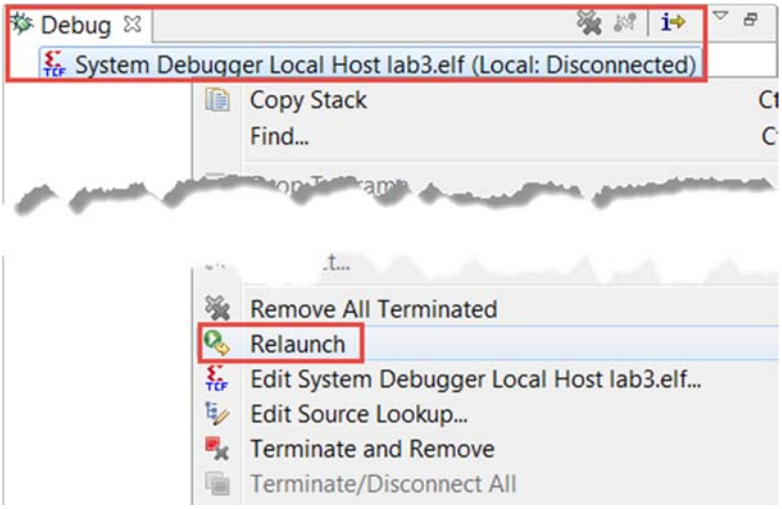


Figure 20. Re-launching the debugger

4-1-2. Click on the SDK *Terminal* window and make a connection with an appropriate COM port OR use any other terminal emulator program like TeraTerm, Putty, HyperTerminal. Choose **115200** as the baud rate.

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 one minute 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:    16257563474
Average SW cycles for sharpen:     13849149
```

(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions:    16255349408
Average SW cycles for sharpen:     13503816
```

(b) Zybo

Figure 21. 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 both the sharpen_filter and sobel_filter already added for hardware with the compiler setting added.

5-1-1. Switch back to the SDx 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 *Select archive file* option, then click the **Browse** button and then browse to *c:\xup\SDSoC\source\lab3*, select *lab3a.zip* and click **Open**.

Make sure that lab3a is checked in the Projects window.

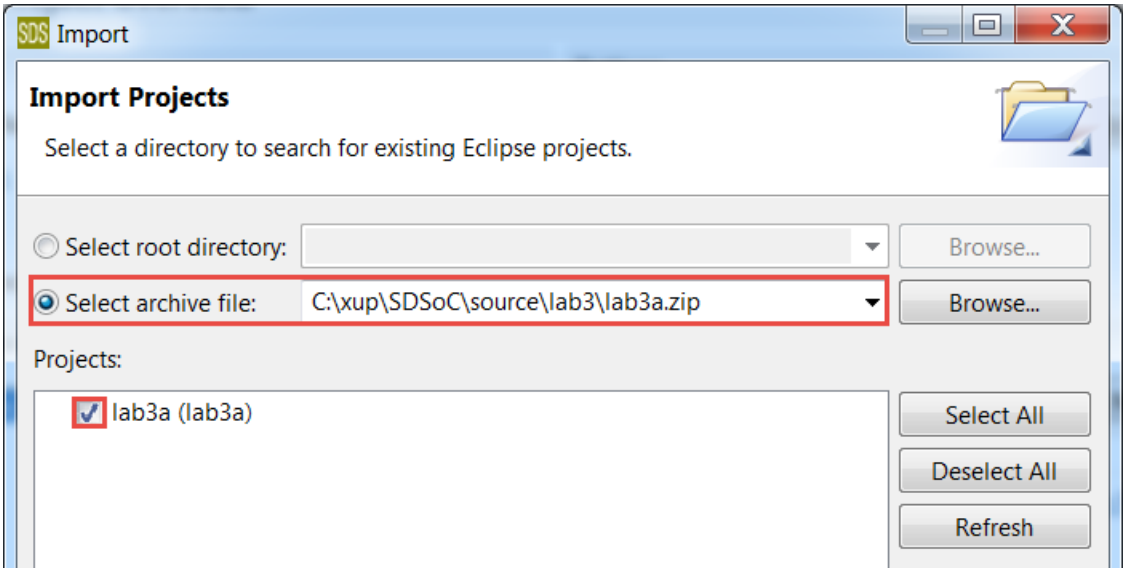


Figure 22. Importing an existing project in the workspace

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

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

5-1-6. Double-click on the **project.sdx** under *lab3a* to access the *SDx Project Settings*.

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

5-1-8. Right Click on the **lab3a** project folder, select **Debug As**, and **Launch on Hardware**

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, enable the *Aggregate per function* option and *Enable stack tracing*. Click **OK**.

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

5-2-1. Connect a terminal as before. (Terminal Tab, TeraTerm, PuTTY, HyperTerminal etc.)

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:    3687726480
Average SW cycles for sharpen:     5176
Average SW cycles for edge_detect: 25108562
```

With TCF Profiler running

(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions:    3143725812
Average SW cycles for sharpen:     5442
Average SW cycles for edge_detect: 18280618
```

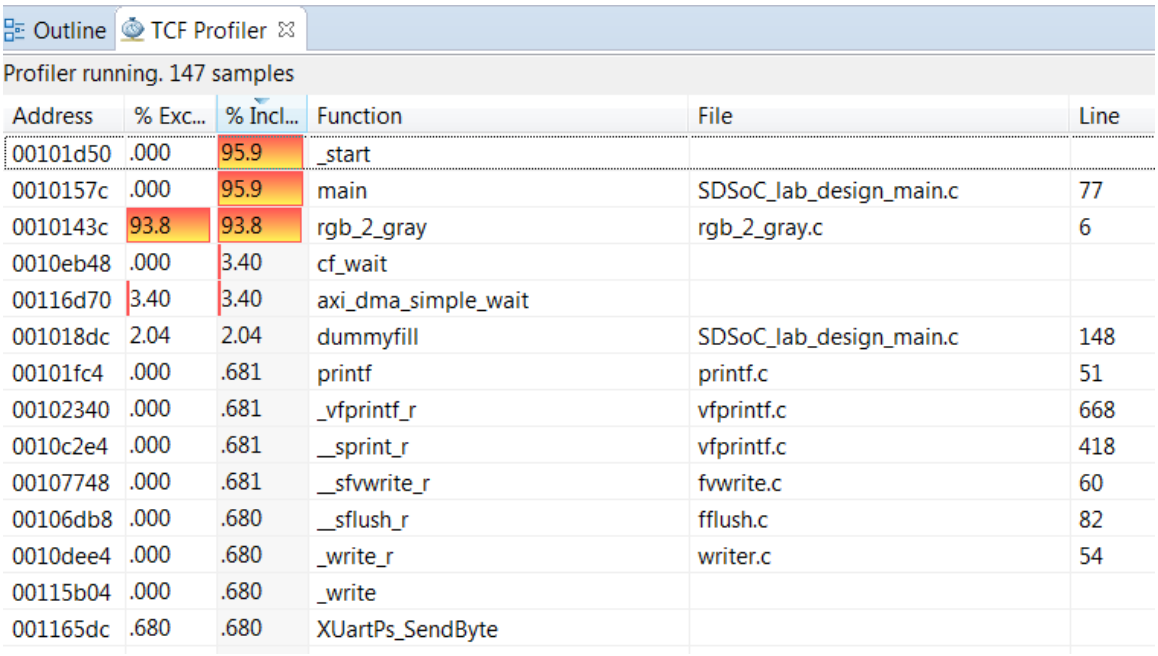
With TCF Profiler Running

(b) Zybo

Figure 23. The sharpen and sobel filter functions profiling

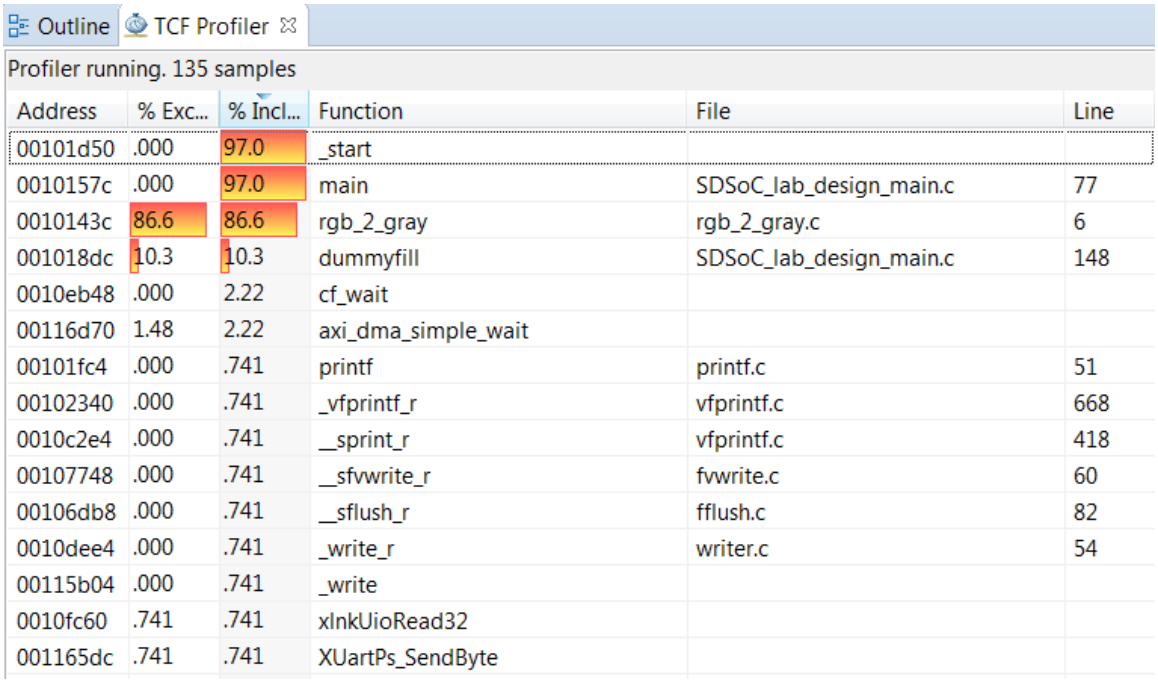
5-2-4. Switch to the TCF Profiler tab and see the results.

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 it's execution time is so short that the profiler does not see it.



Address	% Exc...	% Incl...	Function	File	Line
00101d50	.000	95.9	_start		
0010157c	.000	95.9	main	SDSoC_lab_design_main.c	77
0010143c	93.8	93.8	rgb_2_gray	rgb_2_gray.c	6
0010eb48	.000	3.40	cf_wait		
00116d70	3.40	3.40	axi_dma_simple_wait		
001018dc	2.04	2.04	dummyfill	SDSoC_lab_design_main.c	148
00101fc4	.000	.681	printf	printf.c	51
00102340	.000	.681	_vfprintf_r	vfprintf.c	668
0010c2e4	.000	.681	__sprint_r	vfprintf.c	418
00107748	.000	.681	__sfvwrite_r	fvwrite.c	60
00106db8	.000	.680	__sflush_r	fflush.c	82
0010dee4	.000	.680	_write_r	writer.c	54
00115b04	.000	.680	_write		
001165dc	.680	.680	XUartPs_SendByte		

(a) Zed



Address	% Exc...	% Incl...	Function	File	Line
00101d50	.000	97.0	_start		
0010157c	.000	97.0	main	SDSoC_lab_design_main.c	77
0010143c	86.6	86.6	rgb_2_gray	rgb_2_gray.c	6
001018dc	10.3	10.3	dummyfill	SDSoC_lab_design_main.c	148
0010eb48	.000	2.22	cf_wait		
00116d70	1.48	2.22	axi_dma_simple_wait		
00101fc4	.000	.741	printf	printf.c	51
00102340	.000	.741	_vfprintf_r	vfprintf.c	668
0010c2e4	.000	.741	__sprint_r	vfprintf.c	418
00107748	.000	.741	__sfvwrite_r	fvwrite.c	60
00106db8	.000	.741	__sflush_r	fflush.c	82
0010dee4	.000	.741	_write_r	writer.c	54
00115b04	.000	.741	_write		
0010fc60	.741	.741	xlnkUioRead32		
001165dc	.741	.741	XUartPs_SendByte		

(b) Zybo

Figure 24. Profiled data

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:    753397704
Average SW cycles for sharpen:     3347
Average SW cycles for edge_detect: 13858586
```

Without TCF Profiler running

(a) Zed

```
Running frame operations...
.
.
.
.
.
Average SW cycles for all of the image functions:    751709714
Average SW cycles for sharpen:     3537
Average SW cycles for edge_detect: 13512787
```

Without TCF Profiler Running

(b) Zybo

Figure 25. 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 SDx 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.