Profiling and Performance Tuning

Introduction

This lab guides you through the process of profiling an application and analyzing the output. The application is then accelerated in hardware and profiled again to analyze the performance improvement.

Objectives

After completing this lab, you will be able to:

- Setup the board support package (BSP) for profiling an application
- Set the necessary compiler directive on an application to enable profiling
- Setup the profiling parameters
- Profile an application and analyze the output

Procedure

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

Design Description

In this lab, you will design an embedded system consists of ARM Cortex-A9 processor SoC and two instances of the provided FIR filter IP. The following diagram represents the completed design (**Figure 1**).

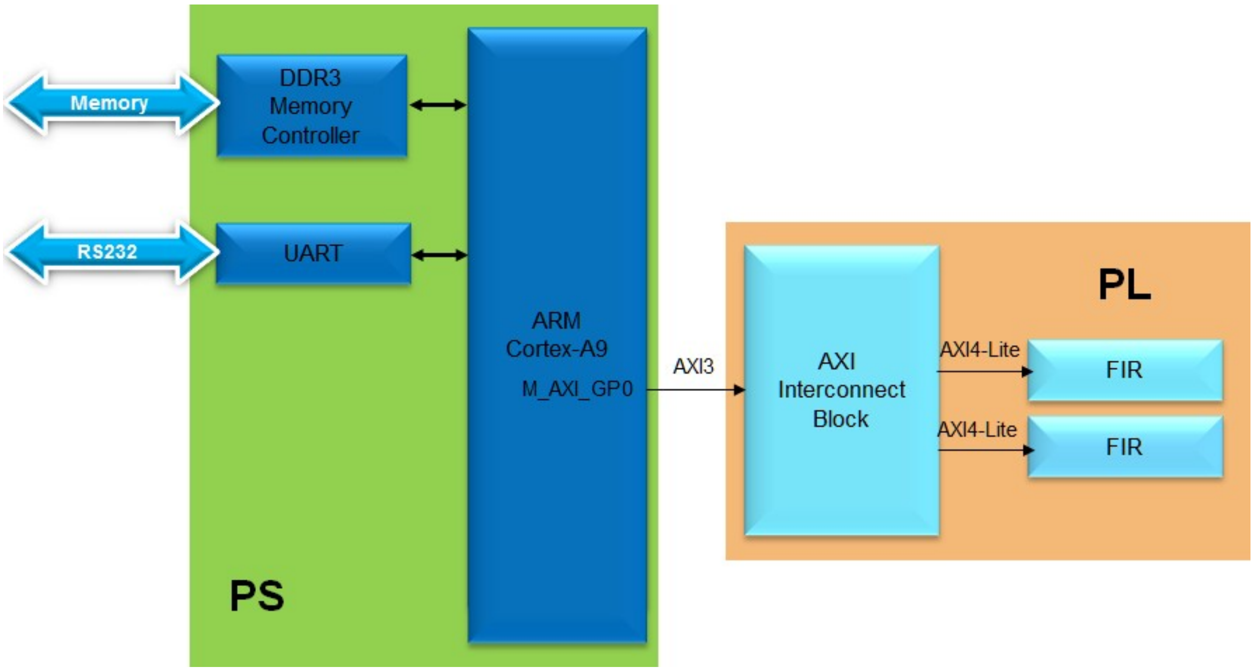
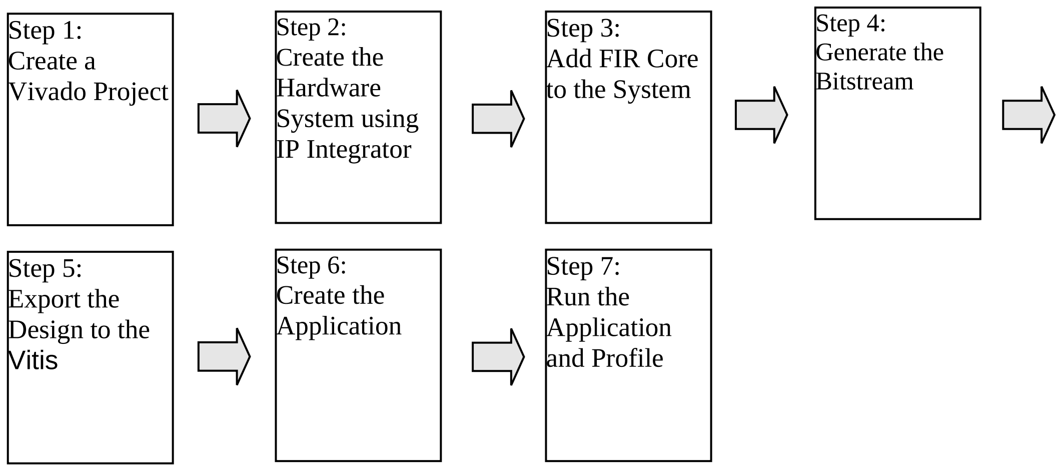


Figure 1. Completed Design

General Flow for this Lab


Step 1: Create a Vivado Project
Step 2: Create the Hardware System using IP Integrator
Step 3: Add FIR Core to the System
Step 4: Generate the Bitstream
Step 5: Export the Design to the Vitis
Step 6: Create the Application
Step 7: Run the Application and Profile

1 Create a Vivado Project Step 1

1.1 Launch Vivado and create an empty project, called lab, targeting the Zybo or ZedBoard Zynq Evaluation and Development Kit and using the VHDL language.

1.1.1 Open Vivado and create a new project new project call *lab* in your hardware directory.

1.1.2 Select the **RTL Project** option in the *Project Type* form, and click **Next**.

1.1.3 Select **VHDL** as the *Target Language* in the *Add Sources* form, and click **Next**.

1.1.4 Click **Next** two times.

1.1.5 In the *Default Part* form, click on *Boards* and select either the Zybo or Zedboard and click **Next**.

1.1.6 Click **Finish** to create an empty Vivado project.

1.2 Set the project settings to include provided fir_top IP

1.2.1 Click **Project Settings** in the *Flow Navigator* pane.

1.2.2 Select **IP** in the left pane of the *Project Settings* form.

1.2.3 Click on the **Green Plus** button, browse to **fir_ip** and click **Select**. fir_ip is provided as a source for this lab.

1.2.4 The *fir_top _v1_0* IP will appear the **IP in the Selected Repository** window.

1.2.5 Click **OK**.

2Creating the Hardware System Using IP Integrator Step 2

2.1 Create a block design in the Vivado project using IP Integrator to generate the Zynq ARM Cortex-A9 processor based hardware system.

2.1.1 In the Flow Navigator, click **Create Block Design** under IP Integrator.

2.1.2 Name the block **system** and click **OK**.

2.1.3 Click on *Add IP* in the message at the top of the *Diagram* panel.

2.1.4 Once the IP Catalog is open, find **ZYNQ7 Processing System** and add it to the design.

2.1.5 Click *Run Block Automation*, and click **OK** to accept the default settings.

2.1.6 Double click on the Zynq block to open the *Customization* window for the Zynq processing system.

A block diagram of the Zynq should now be open, showing various configurable blocks of the Processing System.

2.2 Configure the I/O Peripherals block to only have UART 1 support.

2.2.1 Click on the *MIO Configuration* panel to open its configuration form.

2.2.2 Expand the *I/O Peripherals* on the right.

2.2.3 Uncheck *ENET 0*, *USB 0*, and *SD 0*, *GPIO (GPIO MIO)*, leaving *UART 1* selected.

2.2.4 In the **MIO Configuration** panel, expand the **Application Processing Unit** and uncheck the **Timer 0**.

2.2.5 Click **OK**.

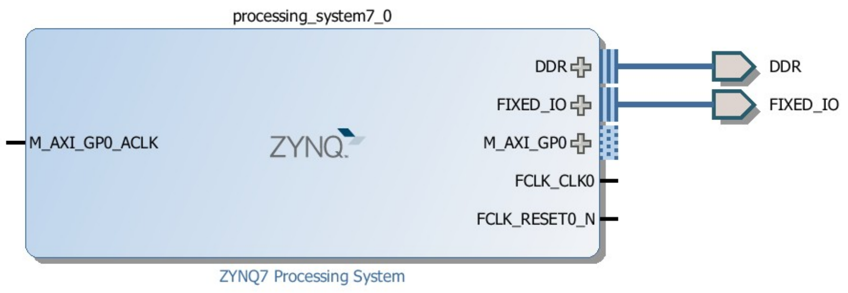


Figure 2. ZYNQ Processing System configured block

3Add FIR Core to the System Step 3

3.1 Instantiate the provided FIR core twice naming the instances as fir_left and fir_right. Validate the design.

3.1.1 Click the Add IP icon and search for **fir** in the catalog.

3.1.2 Double-click on the **fir_top_v1_0** to add the IP instance to the system

3.1.3 Select the *fir_top_1* instance and change its name to **fir_left** in its property form.

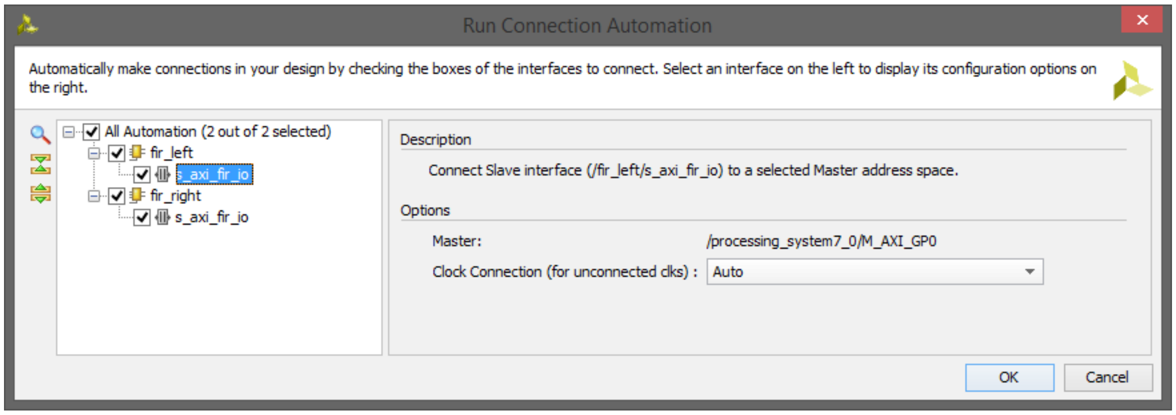
3.1.4 Click the Add IP icon and search for **fir** in the catalog.

3.1.5 Double-click on the **fir_top_v1_0** to add the IP instance to the system

3.1.6 Select the *fir_top_0* instance and change its name to **fir_right** in its property form.

3.1.7 Click on **Run Connection Automation**, and select **All Automation** to select *fir_left* and *fir_right*.

3.1.8 Click on *s_axi_fir_io* for both *fir_left* and *fir_right* and confirm that they will be automatically connected to the Zynq *M_AXI_GP0* port


Run Connection Automation
All Automation (2 out of 2 selected)
fir_left
s_axi_fir_io
fir_right
s_axi_fir_io
OK
Cancel

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

Description

Connect Slave interface (/fir_left/s_axi_fir_io) to a selected Master address space.

Options

Master: /processing_system7_0/M_AXI_GP0

Clock Connection (for unconnected clks) : Auto

Figure 3. Connection automation for the FIR IP blocks

3.1.9 Click **OK** to connect the two blocks to the *M_AXI_GP0* interface.

The design should look similar to shown below:

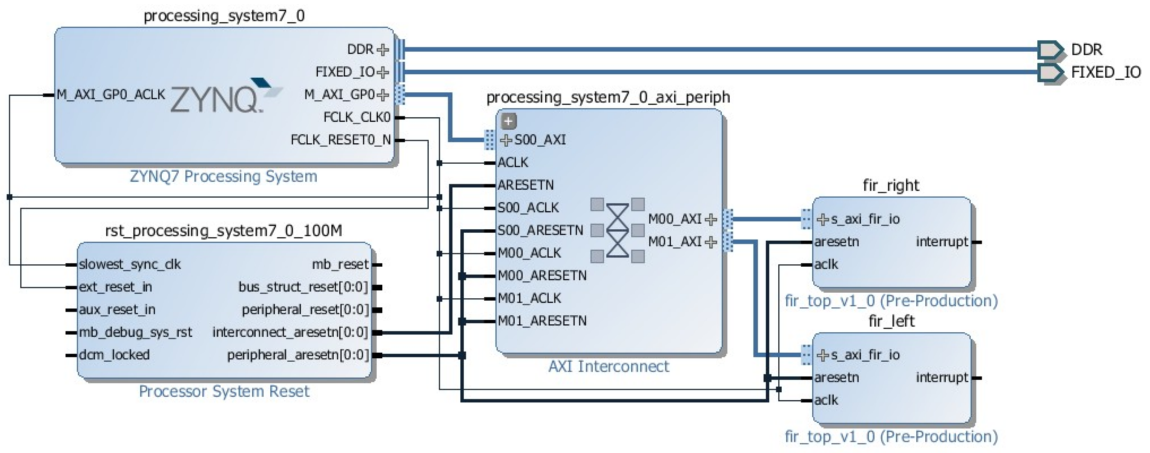


Figure 4. The completed design

It is not necessary to connect the *interrupt* signals of the *fir* blocks.

3.1.10 Select **Tools > Validate Design** to run the design rule checker and to make sure that there are no design errors.

4 Generate the Bitstream Step 4

4.1 Create the top-level HDL of the embedded system, and generate the bitstream..

4.1.1 In Vivado, select the *Sources* tab, expand the *Design Sources,* right-click the *system.bd* and select **Create HDL Wrapper** and click **OK**.

4.1.2 Click on the **Generate Bitstream** in the *Flow Navigator* pane to synthesize and implement the design, and generate the bitstream.

4.1.3 Click **Save** to save the design and **Yes** to run the necessary processes.

4.1.4 During the bitstream generation process click “Open Implemented Design” if asked.

5 Export the Design to the Vitis Step 5

5.1 Export the design to the Vitis, create the software XSA using the standalone operating system and enable the profiling options.

5.1.1 Export the hardware configuration by clicking **File > Export > Export Hardware…**

5.1.2 Tick the box to *Include Bitstream*, and click **OK**

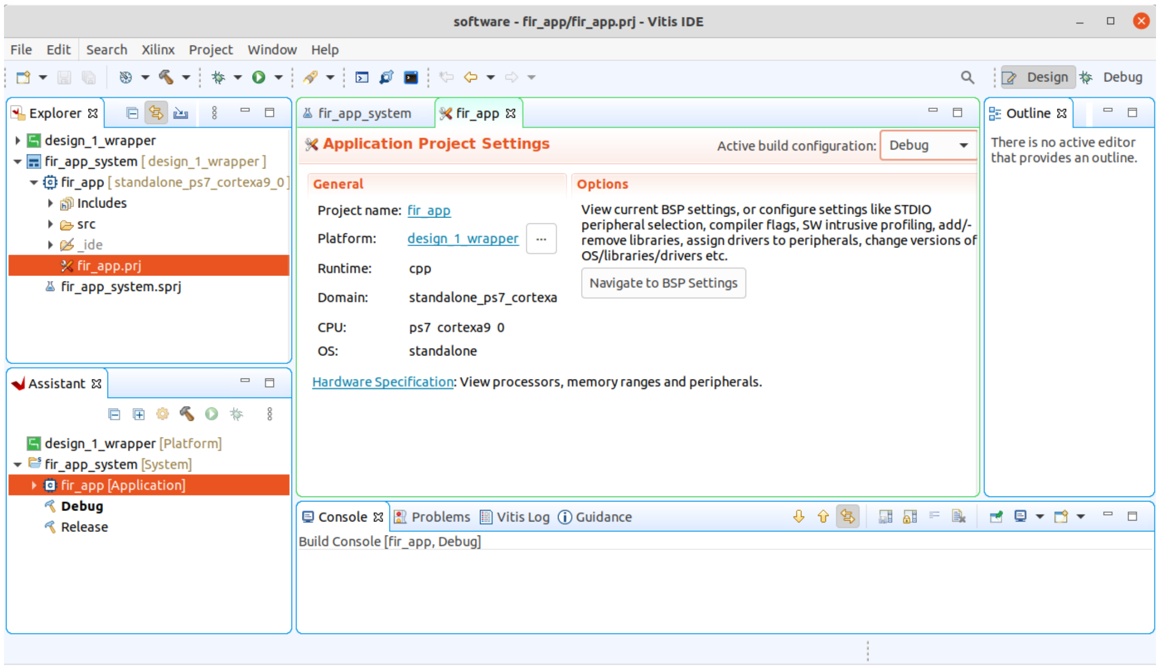
5.1.3 Launch Vitis by clicking **File > Launch Vitis IDE** and click **OK**

5.1.4 In Vitis, select **File** > **New** > **Application Project.**

5.1.5 Select you exported XSA file from the hardware

5.1.6 Notice **Standalone_bsp_0** in the **Project name** field and click **Finish** with default settings.

A Board Support Package Settings window will appear.


software - fir_app/fir_app.prj - Vitis IDE
File Edit Search Xilinx Project Window Help
Design Debug
Explorer
design_1_wrapper
fir_app_system [design_1_wrapper]
fir_app [standalone_ps7_cortexa9_0]
Includes
src
_ide
fir_app.prj
fir_app_system.sprj
Assistant
design_1_wrapper [Platform]
fir_app_system [System]
fir_app [Application]
Debug
Release
fir_app_system
fir_app
Outline
Console Problems Vitis Log Guidance

Application Project Settings

Active build configuration: Debug

General

Project name: fir_app

Platform: design_1_wrapper

Runtime: cpp

Domain: standalone_ps7_cortexa

CPU: ps7_cortexa9_0

OS: standalone

Hardware Specification: View processors, memory ranges and peripherals.

Options

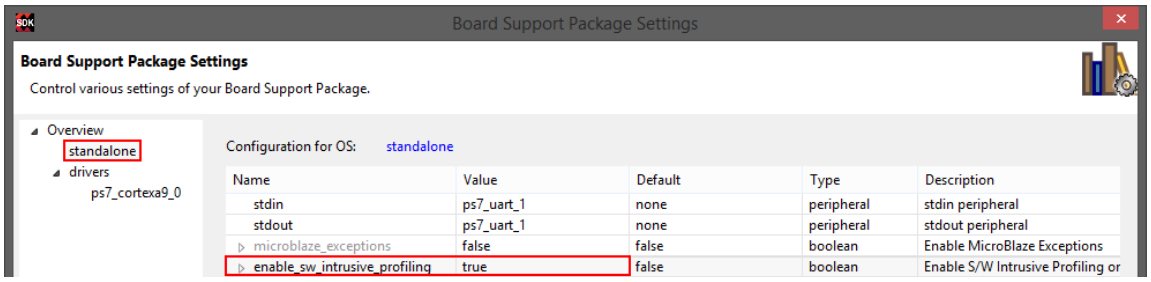
View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/-remove libraries, assign drivers to peripherals, change versions of OS/libraries/drivers etc.

Navigate to BSP Settings

There is no active editor that provides an outline.

Build Console [fir_app, Debug]

5.1.7 Click on Navigate to BSP setting. Then click modify BSP setting. Select the **Overview > standalone** entry in the left pane, click on the drop-down arrow of the *enable_sw_intrusive_profiling Value* field and select **true**.

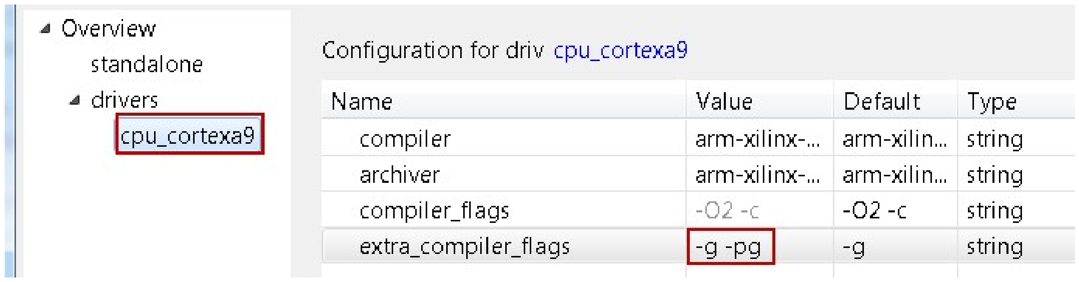

Board Support Package Settings
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
drivers
ps7_cortexa9_0

Configuration for OS: standalone

Name	Value	Default	Type	Description
stdin	ps7_uart_1	none	peripheral	stdin peripheral
stdout	ps7_uart_1	none	peripheral	stdout peripheral
microblaze_exceptions	false	false	boolean	Enable MicroBlaze Exceptions
enable_sw_intrusive_profiling	true	false	boolean	Enable S/W Intrusive Profiling or

Figure 5. Enable profiling in the board support package

5.1.8 Select the **Overview > drivers > ps7_cortexa9_0** and add **–pg** in addition to the –g in the *extra_compiler_flags Value* field.



Configuration for driv cpu_cortexa9

Name	Value	Default	Type
compiler	arm-xilinx-...	arm-xilin...	string
archiver	arm-xilinx-...	arm-xilin...	string
compiler_flags	-O2 -c	-O2 -c	string
extra_compiler_flags	-g -pg	-g	string

Figure 6 Adding profiling switch

5.1.9 Click **OK** to accept the settings and create the BSP.

6 Create the Application Step 6

6.1 Create the *lab* application using the provided lab7.c, fir.c, fir.h, fir_coef.dat, and xfir_fir_io.h files.

6.1.1 Select **File** > **New** > **Application Project.**

6.1.2 Enter a name for the project name, select the **Use existing** *standalone_bsp_0* option, and click **Next**.

6.1.3 Select **Empty Application** in the *Available Templates* pane and click **Finish**.

6.1.4 In the *lab7* project, right click on the *src* directory and select **Import.**

6.1.5 Expand the General folder and double-click on **File system,** and browse to the lab_sources directory.

6.1.6 Select **fir_coef.dat, fir.c, fir.h, lab.c, lab.h** and **xfir_fir_io.h,** and click **Finish.**

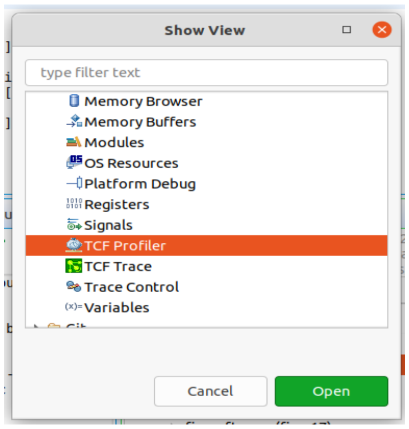
The program should compile successfully and generate the .elf file.

6.1.7 Open the *lab.c* file and scroll to the main function at the bottom. Notice the following code:

```
#ifdef SW_PROFILE
    fir_software(&output,signal);
#else
    filter_hw_accel_input(&output,signal);
#endif
```

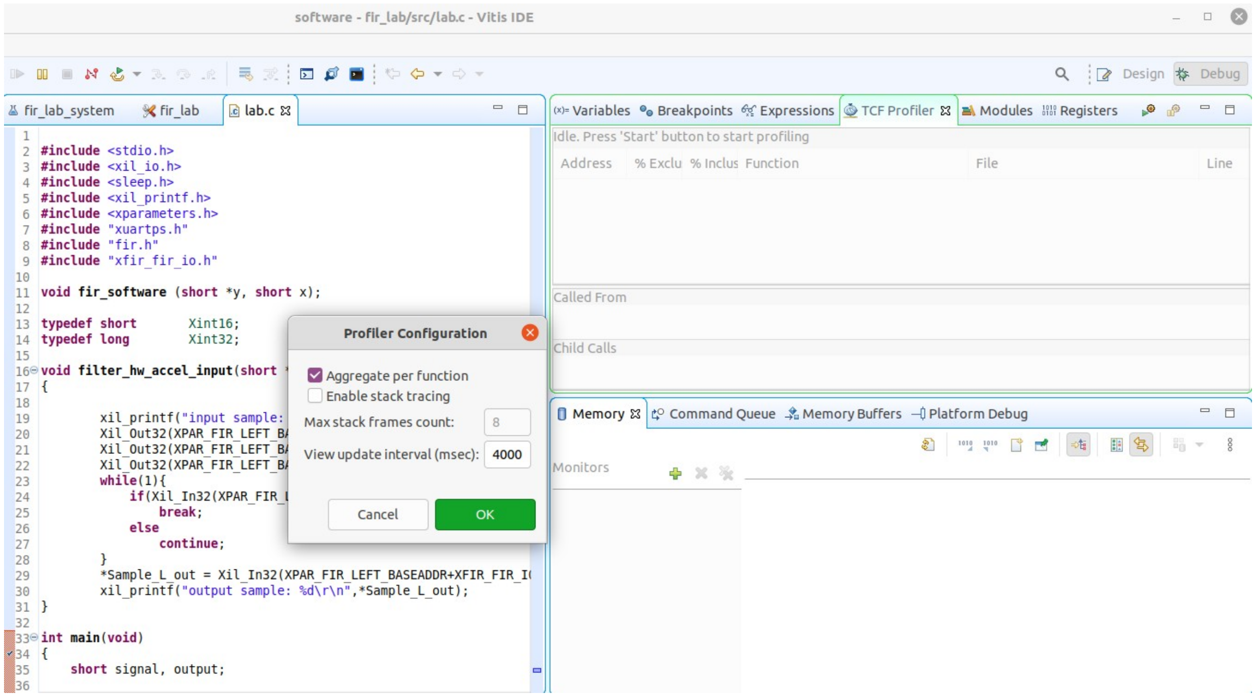
The function *fir_software*( ) is a software implementation of the FIR function. The *filter_hw_accel_input*( ) function offloads the FIR function to the two FIR blocks that have been implemented in the PL. In this case, you are using the implementation of the FIR filter on hardware (PL) as this gray code disabled.

6.1.8 Run TCP Profiler. Select Window > Show View > Debug > TCP Profiler



6.1.9 Start the application

6.1.10 Start the TCF profiler



software - fir_lab/src/lab.c - Vitis IDE

```
#include <stdio.h>
#include <xil_io.h>
#include <sleep.h>
#include <xil_printf.h>
#include <xparameters.h>
#include "xuartps.h"
#include "fir.h"
#include "xfir_fir_io.h"

void fir_software (short *y, short x);

typedef short       Xint16;
typedef long        Xint32;

void filter_hw_accel_input(short *
{

        xil_printf("input sample:
        Xil_Out32(XPAR_FIR_LEFT_BA
        Xil_Out32(XPAR_FIR_LEFT_BA
        Xil_Out32(XPAR_FIR_LEFT_BA
        while(1){
            if(Xil_In32(XPAR_FIR_L
                break;
            else
                continue;
        }
        *Sample_L_out = Xil_In32(XPAR_FIR_LEFT_BASEADDR+XFIR_FIR_I(
        xil_printf("output sample: %d\r\n",*Sample_L_out);
}

int main(void)
{
    short signal, output;

```

Profiler Configuration

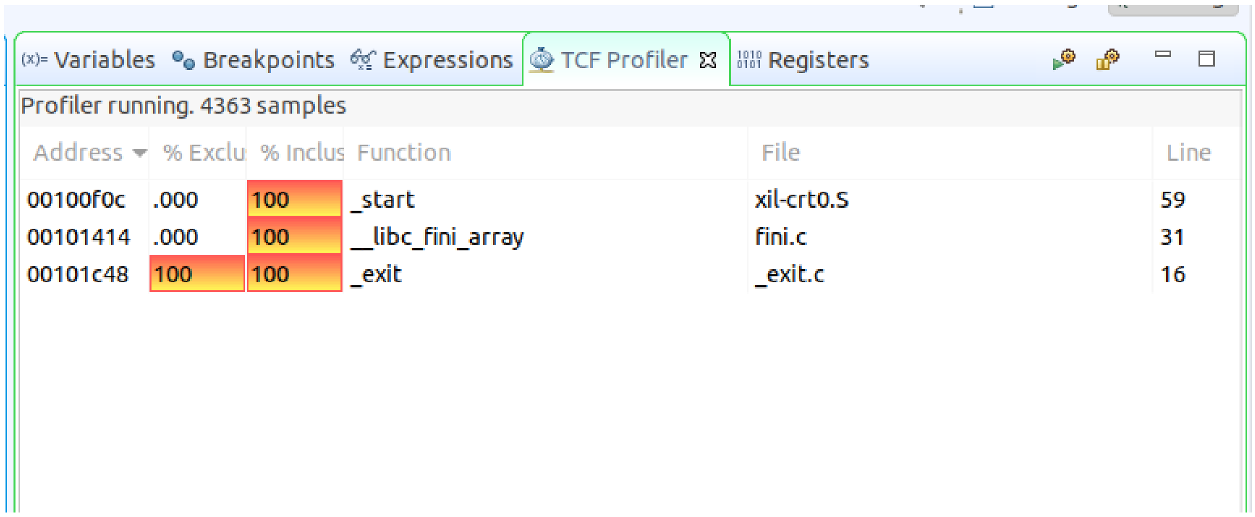
- [x] Aggregate per function
- [ ] Enable stack tracing

Max stack frames count: 8

View update interval (msec): 4000

Idle. Press 'Start' button to start profiling

6.1.11 Observe the output of the TCF profiler



Profiler running. 4363 samples

Address	% Exclu	% Inclus	Function	File	Line
00100f0c	.000	100	_start	xil-crt0.S	59
00101414	.000	100	__libc_fini_array	fini.c	31
00101c48	100	100	_exit	_exit.c	16

7 Run the Application and Profile Step 7

7.1 Place the board into the JTAG boot up mode. Program the PL section and run the application using the user defined SW_PROFILE symbol.

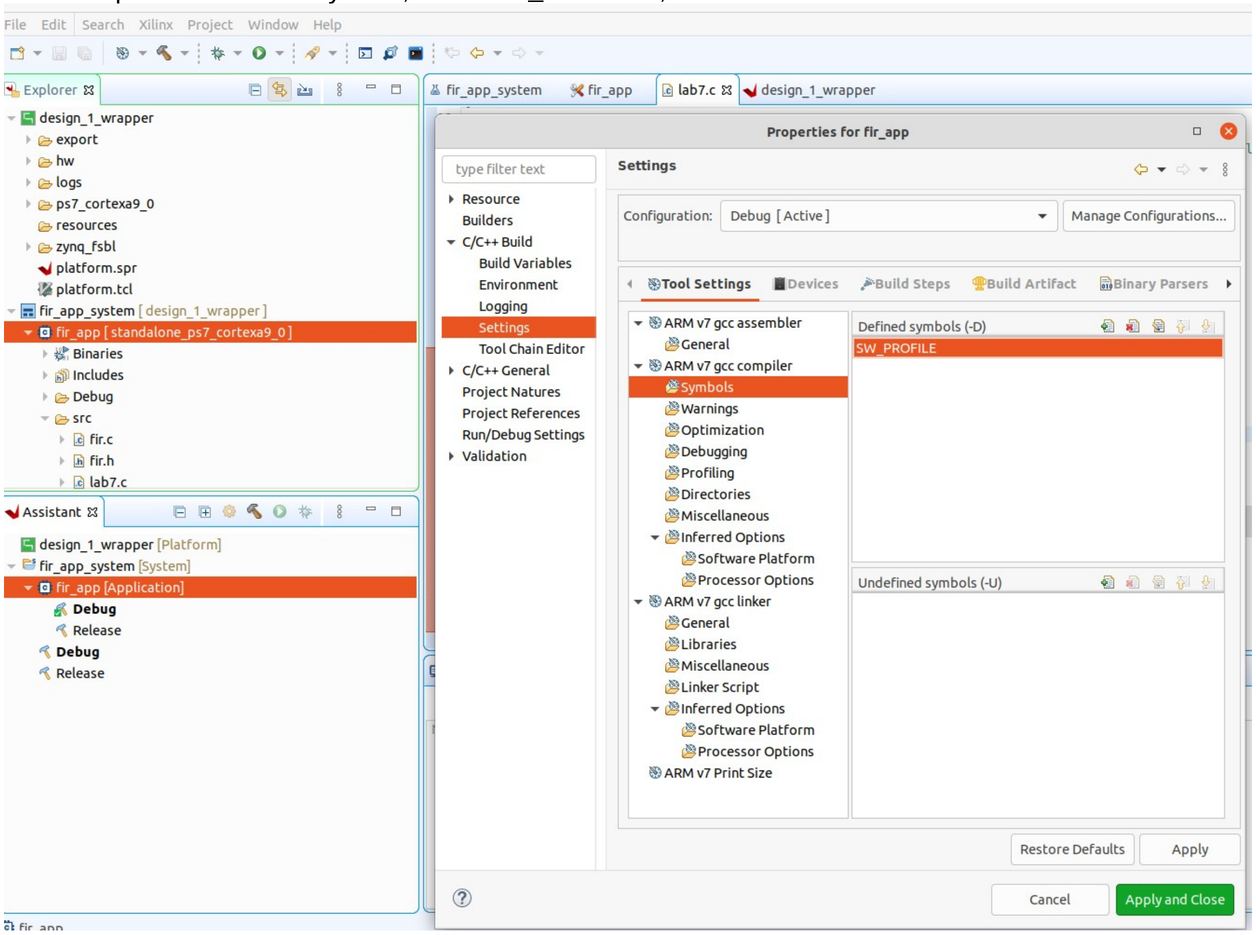
7.1.1 Place the board in the JTAG boot up mode.

7.1.2 Power ON the board.

7.1.3 Select **Xilinx Tools > Program FPGA** and click on **Program**.

7.1.4 Right click on the *lab directory*, and select **C/C++ Build Settings**.

7.1.5 Under the **ARM gcc compiler** group, select the **Symbols** sub-group**,** click on the button to open the value entry form, enter **SW_PROFILE**, and click **OK**.


Properties for fir_app

Settings

Configuration: Debug [Active]

Manage Configurations...

Tool Settings | Devices | Build Steps | Build Artifact | Binary Parsers

ARM v7 gcc compiler > Symbols

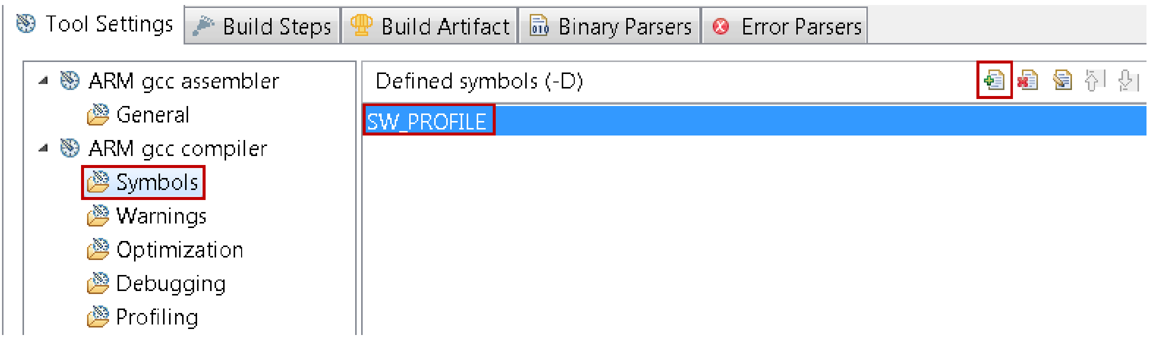
Defined symbols (-D)

SW_PROFILE

Undefined symbols (-U)

Restore Defaults | Apply

Cancel | Apply and Close

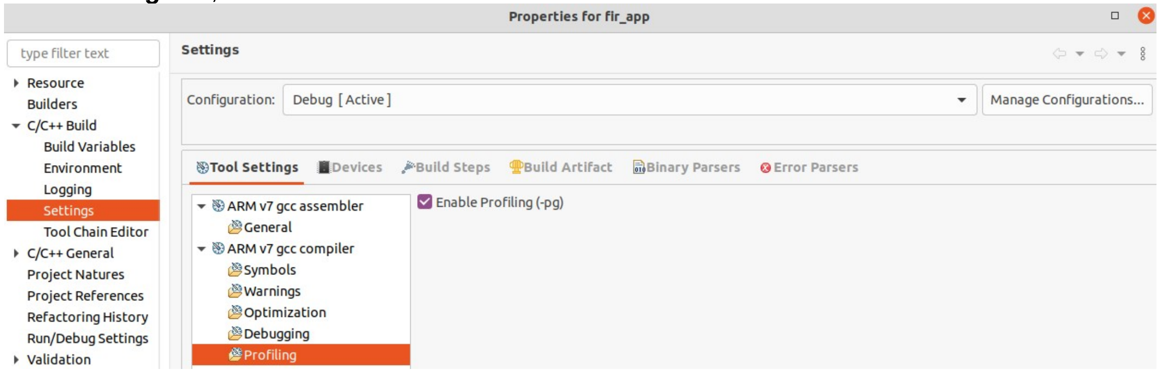


This will allow us to profile the software loop of the FIR application. You can verify this by looking at lab7.c source code to verify that the proper part of the code is inactive.

Q2. What change do you see in lab7.c source file as a result of this step?

Figure 7. Add user-defined symbol

7.1.6 Under the **ARM gcc compiler** group, select the **Profiling** sub-group, then check the **Enable Profiling** box, and click **OK**.



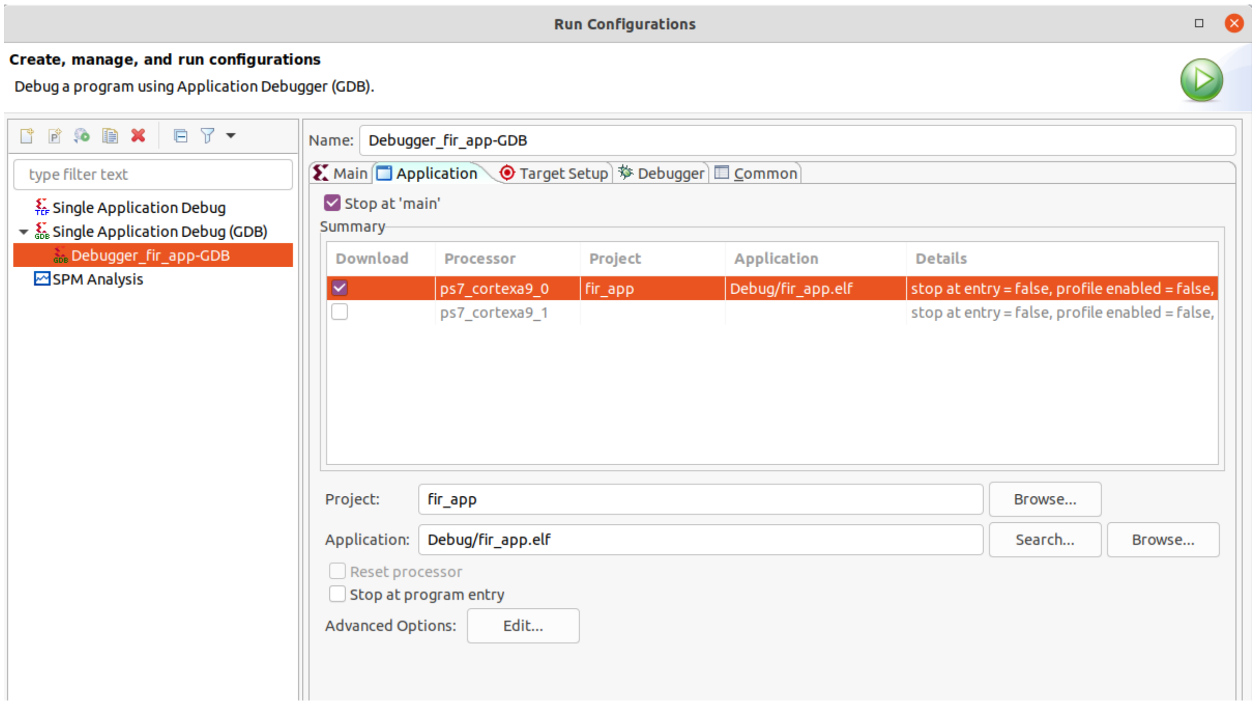
Configuration: Debug [Active]

Enable Profiling (-pg) ✓

Figure 8. Compiler setting for enabling profiling

7.1.7 From the menu bar, Select **Run > Run Configurations…** and double click on *Xilinx C/C++ application* to create a new configuration.

7.1.8 Click on the newly created **lab Debug** configuration, and select the **Application** tab and press **Advanced Options**: Edit


Run Configurations

Create, manage, and run configurations

Debug a program using Application Debugger (GDB).

- Single Application Debug
- Single Application Debug (GDB)
  - Debugger_fir_app-GDB
- SPM Analysis

Name: Debugger_fir_app-GDB

Tabs: Main | Application | Target Setup | Debugger | Common

☑ Stop at 'main'

Summary

Download	Processor	Project	Application	Details
✓	ps7_cortexa9_0	fir_app	Debug/fir_app.elf	stop at entry = false, profile enabled = false,
☐	ps7_cortexa9_1			stop at entry = false, profile enabled = false,

Project: fir_app

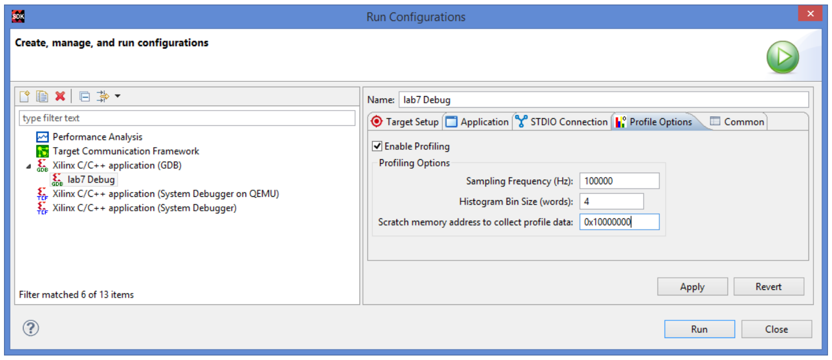
Application: Debug/fir_app.elf

☐ Reset processor

☐ Stop at program entry

Advanced Options: Edit...

7.1.9 Click on the *Enable Profiling* check box, enter **100000** (100 kHz) in the Sampling Frequency field, enter **0x10000000** in the scratch memory address field as shown in Figure 9, and click **Apply**.


Run Configurations

Create, manage, and run configurations

- Performance Analysis
- Target Communication Framework
- Xilinx C/C++ application (GDB)
  - lab7 Debug
- Xilinx C/C++ application (System Debugger on QEMU)
- Xilinx C/C++ application (System Debugger)

Filter matched 6 of 13 items

Name: lab7 Debug

Tabs: Target Setup | Application | STDIO Connection | Profile Options | Common

☑ Enable Profiling

Profiling Options

Sampling Frequency (Hz): 100000

Histogram Bin Size (words): 4

Scratch memory address to collect profile data: 0x10000000

Apply | Revert | Run | Close

Figure 9. Profiling options

7.1.10 Click the **Run** button to download the application and execute it.

The program will run, and when execution has completed, a message will be displayed indicating that the profiling results are being saved in gmon.out file at the lab\Debug directory.

7.1.11 Click **OK**.

7.2 Invoke gprof and analyze the results.

7.2.1 Expand the *Debug* folder under the **lab7** project in the Project Explorer view, and double click on the **gmon.out** entry.

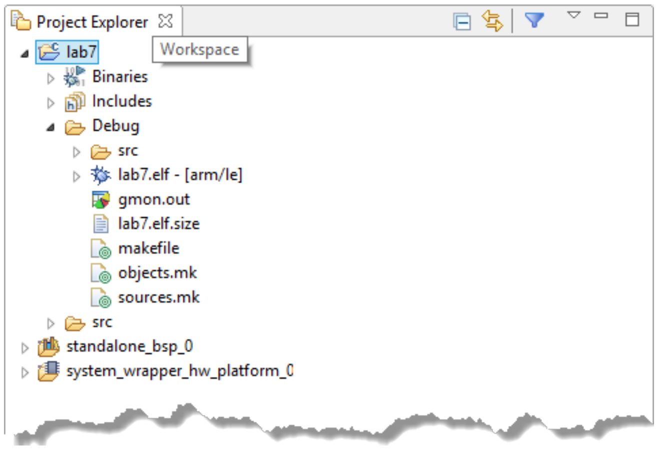
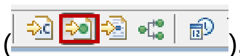


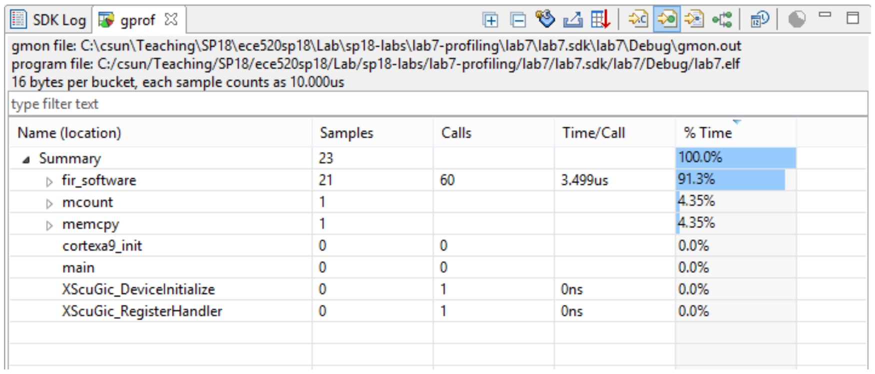
Figure 10. Invoking gprof on gmon.out

7.2.2 The Gmon File Viewer dialog box will appear showing *lab7.elf* as the corresponding binary file. Click **OK**. This will open gprof window on the bottom right corner of the screen.

7.2.3 Click on the **Sort samples per function** button ().

7.2.4 Click in the **%Time** column to sort in the descending order.

Note that the fir_software routine is called 60 times, 21 samples were taken during the profiling, and on an average of 3.499 microseconds were spent per call.



gmon file: C:\csun\Teaching\SP18\ece520sp18\Lab\sp18-labs\lab7-profiling\lab7\lab7.sdk\lab7\Debug\gmon.out
program file: C:/csun/Teaching/SP18/ece520sp18/Lab/sp18-labs/lab7-profiling/lab7/lab7.sdk/lab7/Debug/lab7.elf
16 bytes per bucket, each sample counts as 10.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
Summary	23			100.0%
fir_software	21	60	3.499us	91.3%
mcount	1			4.35%
memcpy	1			4.35%
cortexa9_init	0	0		0.0%
main	0	0		0.0%
XScuGic_DeviceInitialize	0	1	0ns	0.0%
XScuGic_RegisterHandler	0	1	0ns	0.0%

Figure 11. Sorting results

7.2.5 Go back to the *Run Configuration*, and change the sampling frequency to **10000** (10 KHz), **1000000** (1 MHz), and **5000000** (5 MHz) and profile the application again.

7.2.6 For each case invoke **gprof**, select the **Sorts samples per function** output, and sort the **%Time** column.

Q3. Present a table/graph and compare your results. Write your conclusion from applying different sampling frequency to the profiling process. Is there any relationship between the number of samples taken and number of function calls? How about other factors? Explain exactly how profiling is affected by sampling frequency? What is more accurate; lower sampling rate or higher sampling rate? What happens if sampling rate is decreases to its minimum possible? What happens if sampling rate is decreases to its maximum possible?

At this stage, the designer of the system would decide if the FIR function should be ported to hardware.

7.3 Profile the application using the hardware FIR filter IP by removing the user defined SW_PROFILE symbol.

7.3.1 Select the *lab7* application, right-click, and select **C/C++ Build Settings**.

7.3.2 Under the **ARM gcc compiler** group, select the **Symbols** sub-group**,** select **SW_PROFILE**, and delete it by clicking on the delete button.

This will allow us to profile the hardware IP of the FIR application. You can verify this by looking at lab7.c source code to verify that the proper part of the code is inactive.

Q4. What change do you see in lab7.c source file as a result of this step?

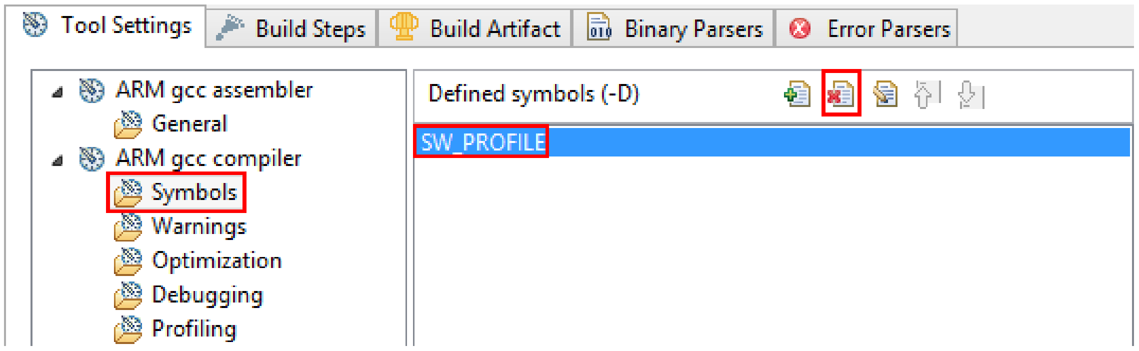


Figure 12. Deleting the user-defined symbol

7.3.3 Click **Apply**, and then click **OK**

7.3.4 Select **Run > Run Configurations** and change the sampling frequency to **10000** (10 KHz), **100000** (100 KHz), **1000000** (1 MHz), and **5000000** (5 MHz) and profile the application again.

7.3.5 Open **gmon.out**, select the **Sorts samples per function** output, and sort the %Time column.

Q5. Write your conclusion from applying different sampling frequency to the profiling process. Is there any relationship between the number of samples taken and number of function calls? How about other factors? Explain exactly how profiling is affected by sampling frequency? What is more accurate; lower sampling rate or higher sampling rate? What happens if sampling rate is decreases to its minimum possible? What happens if sampling rate is decreases to its maximum possible?

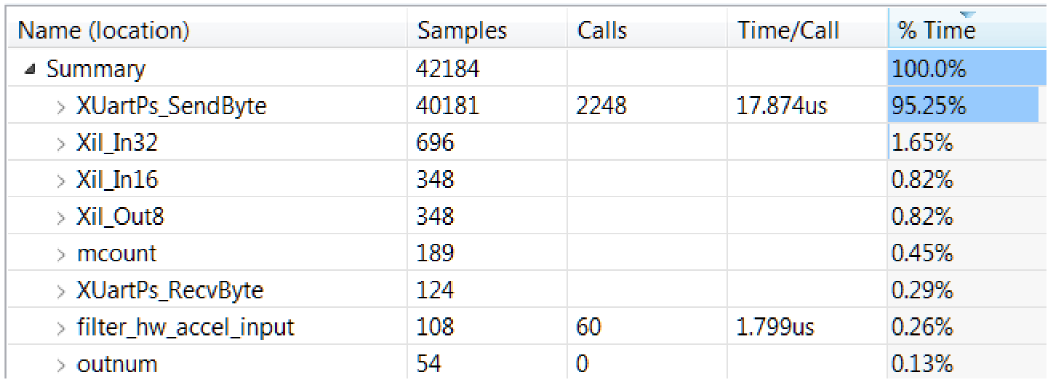
Q6. How does software profiling compare to hardware profiling? Elaborate your answer. Has the number of calls changed? Why or why not? Has the time changed? Why or why not?

Q6. What is the most noticeable change in the hardware profile compared to software profile?

Notice that the output now shows filter_hw_accel_input function call instead of the fir_software function call. Note that the number of calls to the filter function has not changed but the average time spent per call is 1.799 us as the filtering is done in the hardware instead of the software.

Also notice that the amount of time spent in the filtering function reduced from about 98% to 0.26%.

Q7. What is the reason for this significant reduction in average time spent per call as well as time spent for filtering function? Explain your answer. Find the ratio of the speed up for each sampling frequency while profiling the application. Explain your result using a table or graph.



Name (location)	Samples	Calls	Time/Call	% Time
Summary	42184			100.0%
XUartPs_SendByte	40181	2248	17.874us	95.25%
Xil_In32	696			1.65%
Xil_In16	348			0.82%
Xil_Out8	348			0.82%
mcount	189			0.45%
XUartPs_RecvByte	124			0.29%
filter_hw_accel_input	108	60	1.799us	0.26%
outnum	54	0		0.13%

Figure 13. Profiling the application with the hardware IP

7.3.6 Close the SDK and Vivado programs by selecting **File > Exit** in each program.

7.3.7 Turn OFF the power on the board.

Q8. Explain how FIR filter is integrated to the SDK project as a hardware accelerated HLS core? How does software communicate with the FIR core on FPGA fabric? What is being sent to the FIR core as input data and what do you expect to see as output?

Q9. Write down your conclusion from this lab.