EDK Profiling User Guide

A Guide to Profiling in EDK

UG448 EDK 11.2

XILINX®

About This Guide

This user guide provides information about profiling software running on embedded systems built with Xilinx® EDK. Profiling is software-intrusive, and is based on the GNU `gprof` tool. This document details how profiling works, how to set up the hardware and software systems to perform profiling, and how to view the resulting profile data.

Guide contents include:

- Chapter 1, "Introduction"
- Chapter 2, "Using SDK for Profiling in EDK"
- Chapter 3, "Profiling Flow in EDK"
- Chapter 4, "Navigating the Profile Perspective in SDK"
- Appendix A, "Profiling Applications Using XPS for Software Development"
- Appendix B, "Profiling Restrictions"
- Appendix C, "Glossary"

Additional Resources

The SDK Help contains detailed instructions for using SDK. To access the SDK Help, do one of the following:

- In Windows, select **Start > All Programs > Xilinx ISE 11.1 > EDK > Documentation > SDK Help Contents.**
- In your EDK installation directory, navigate to `\doc\usenglish\SDK_doc` and open `index.html`.

To find additional EDK documentation, see the Xilinx EDK Documentation Web site at:

http://www.xilinx.com/ise/embedded/edk_docs.htm.

To search the Answer Database for silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx Support Web site at:

http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

This document uses the following typographical conventions:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File > Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[`*`option_name`*`]` *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis . . .	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex®-II Handbook.*
Blue, underlined text	Hyperlink to a Website (URL)	Go to http://www.xilinx.com for the latest speed files.

Table of Contents

Chapter 1

Introduction

Welcome

The Xilinx® Embedded Development Kit (EDK) is a suite of tools and IP that enables you to design a complete embedded processor system for implementation in a Xilinx Field Programmable Gate Array (FPGA) device.

This user guide provides information about profiling software running on embedded systems built with EDK. Profiling is software-intrusive, and is based on the GNU `gprof` tool. This document details how profiling works, how to set up the hardware and software systems to perform profiling, and how to view the resulting profile data.

For additional documentation on Profiling, refer to "Additional Resources," page 3.

Profiling with GNU gprof

Profiling a program with GNU `gprof` provides two kinds of information that you can use to optimize the program:

- A histogram with which you can identify the functions in the program that take up the most execution time
- A call graph that shows what functions called which other functions, and how many times

For additional information about GNU `gprof`, refer to http://sourceware.org/binutils/docs-2.18/gprof/index.html.

How Profiling Works

The execution flow of the program is altered to obtain the data needed for `gprof`. Consequently, this method of profiling is considered "software-intrusive."

The program flow is altered in two ways:

- To obtain histogram data, the program is periodically interrupted to obtain a sample of its program counter location. This user-defined interval is usually measured in milliseconds. The program counter location helps identify which function was being executed at that particular sample. Taking multiple samples over a long interval of a few seconds helps identify which functions execute for the longest time in the program.
- To obtain the call graph information, the compiler annotates every function call to store the caller and callee information in a data structure.

The steps involved in profiling are as follows:

1. Compile and link the program for profiling by adding the `-pg` switch to the `gcc` compiler command line.
2. Run the program to generate profile data.
3. Process the profile data obtained from `gprof`.

These steps are explained in more detail in Chapter 2, "Using SDK for Profiling in EDK."

Sample Output from Profiling

This section describes a sample output from a profiling run. Once the application is compiled for profiling and executed, it dumps out the file `gmon.out`, which contains the collected profile data. This file is processed by `gprof` to obtain human-readable profile information. The output of `gprof` contains both the histogram or "flat profile" and the call graph information. For descriptions of these data columns, refer to the GNU gprof document at
http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_chapter/gprof_5.html#SEC10.

```
Flat profile:
Each sample counts as 0.01 seconds.
 %     cumulative self              self    total
 time  seconds     seconds  calls   ns/call ns/call  name
 33.33 0.02        0.02     400000  50.00   50.00    Proc_8
 33.33 0.04        0.02                              main
 16.67 0.05        0.01     400000  25.00   50.00    Proc_1
 16.67 0.06        0.01     400000  25.00   25.00    Proc_6
 0.00  0.06        0.00     1200000 0.00    0.00     Proc_7
 0.00  0.06        0.00     800000  0.00    0.00     Func_1
 0.00  0.06        0.00     400000  0.00    0.00     Func_2
 0.00  0.06        0.00     400000  0.00    0.00     Func_3
 0.00  0.06        0.00     400000  0.00    0.00     Proc_2
 0.00  0.06        0.00     400000  0.00    0.00     Proc_3
 0.00  0.06        0.00     400000  0.00    0.00     Proc_4
 0.00  0.06        0.00     400000  0.00    0.00     Proc_5
```

Figure 1-1: **Histogram (Flat Profile) Information**

From the flat profile, you can determine and optimize the functions that take up the most execution time.

```
Call graph
granularity: each sample hit covers 4 byte(s) for 16.67% of 0.06
seconds.
 index % time   self     children  called          name
                                                   <spontaneous>
 [1]    100.0   0.02     0.04                      main [1]
                0.02     0.00      400000/400000   Proc_8 [3]
                0.01     0.01      400000/400000   Proc_1 [2]
                0.00     0.00      800000/800000   Func_1 [6]
                0.00     0.00      400000/400000   Proc_5 [12]
                0.00     0.00      400000/400000   Proc_4 [11]
                0.00     0.00      400000/400000   Func_2 [7]
                0.00     0.00      400000/1200000  Proc_7 [5]
                0.00     0.00      400000/400000   Proc_2 [9]
 ----------------------------------------------------
                0.01     0.01      400000/400000   main [1]
 [2]    33.3    0.01     0.01      400000          Proc_1 [2]
                0.01     0.00      400000/400000   Proc_6 [4]
                0.00     0.00      400000/400000   Proc_3 [10]
                0.00     0.00      400000/1200000  Proc_7 [5]
 ----------------------------------------------------
                0.02     0.00      400000/400000   main [1]
 [3]    33.3    0.02     0.00      400000          Proc_8 [3]
 .
 .
 .
```

Figure 1-2: **Call Graph Information**

The call graph shows what functions called which other functions and how many times. For descriptions of these data columns, refer to "Call Table View" in Chapter 4.

You can view histogram information graphically if you do your profiling using Platform Studio SDK. Figure 1-3 shows a sample SDK profile view.

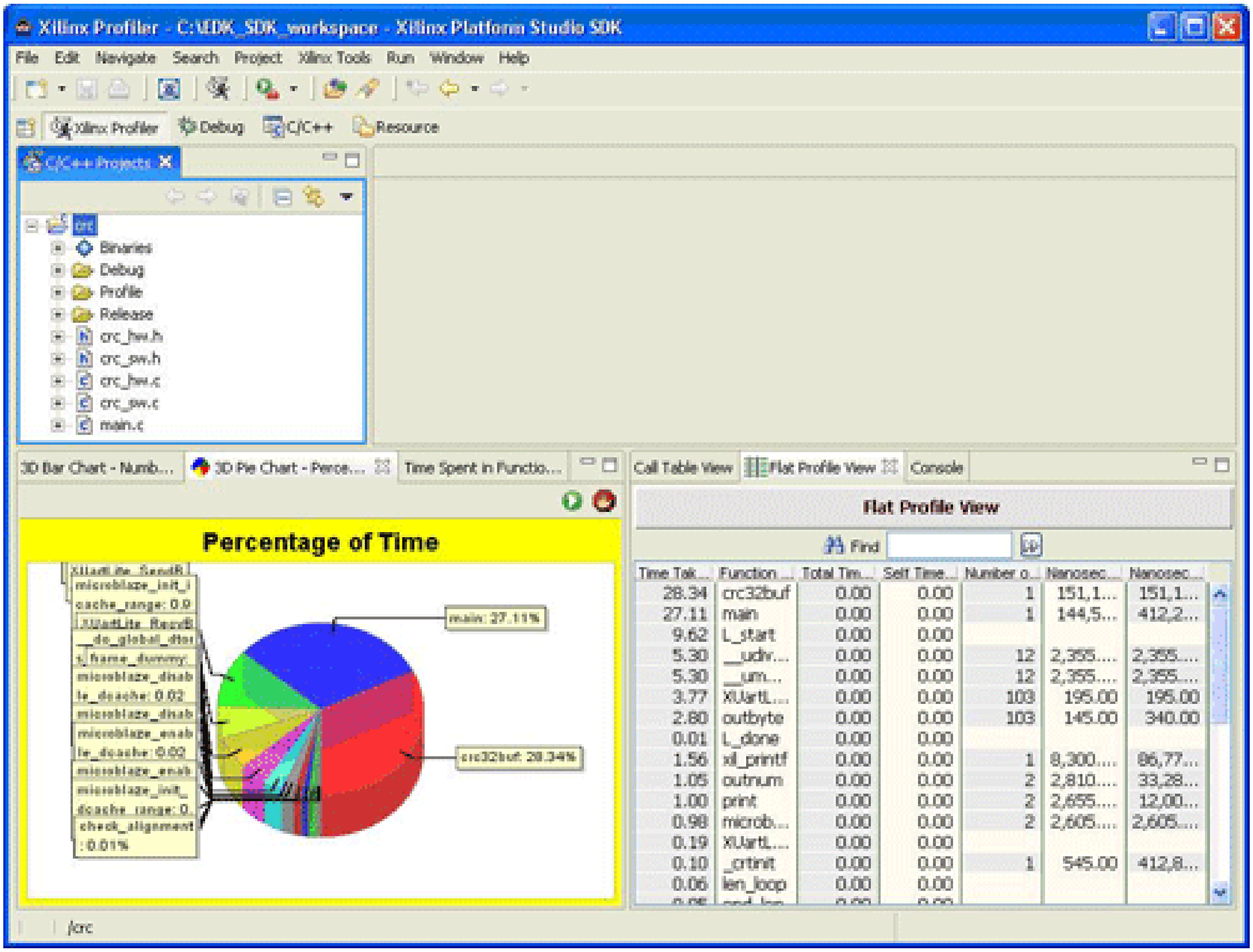

Figure 1-3: **Graphical Histogram Information in SDK**

Chapter 2

Using SDK for Profiling in EDK

This chapter explains the steps involved in profiling an application in EDK. The following sections are included:

- "Setting Up the Hardware for Profiling"
- "Setting Up the Software to Support Profiling"
- "Generating and Viewing Profile Data"

Setting Up the Hardware for Profiling

To profile a software application, you must ensure that interrupts are raised periodically to sample the program counter (PC) value. To do this, you must program a timer and use the timer interrupt handler to collect and store the PC. The profile interrupt handler requires full access to the timer, so a separate timer that is not used by the application itself must be available in the system.

Xilinx® profiling libraries that provide the profile interrupt handler support the `xps_timer` core. When profiling on PowerPC® processors, the internal Programmable Interrupt Timer (PIT) can also be used. Either one of these timers should be available for exclusive use by the profile libraries.

The timer interrupt signal is connected directly to the processor, or it is connected to the processor through an interrupt controller.

Setting Up the Software to Support Profiling

This section assumes that you are using the Xilinx Platform Studio SDK for software development. Alternately, you can perform profiling using Xilinx Platform Studio (XPS) for software development, as described in Appendix A, "Profiling Applications Using XPS for Software Development."

There are three important steps involved in setting up the software application for profiling:

1. Specify the Board Support Package (BSP) Settings:
   a. In SDK, open the **Software Platform Settings** dialog box.
   b. Set the **enable_sw_intrusive_profiling** field to **true** and select the timer for use by the profile libraries.
   c. Generate libraries and the BSP. This step configures the profiling libraries to be part of the standalone BSP (`libxil.a`).

2. Modify the software application code to enable interrupts. If the profile timer is directly connected to the processor without an interrupt controller, you must enable interrupts in the processor. If there is an interrupt controller present in the system, then in addition to enabling interrupts in the processor, the interrupt controller should also be enabled and allowed to pass interrupts from the profile timer to the processor.

   Example code is shown below:

```
/* enable interrupt controller */
XIntc_mMasterEnable(SYSINTC_BASEADDR);

/* service all interrupts */
XIntc_SetIntrSvcOption(SYSINTC_BASEADDR, XIN_SVC_ALL_ISRS_OPTION);

/* enable the profile timer interrupt */
XIntc_mEnableIntr(SYSINTC_BASEADDR, PROFILE_TIMER_INTR_MASK);

/* enable interrupts in the processor */
microblaze_enable_interrupts();
```

If the profiling timer is the only entity that connects to the input of interrupt controler, the tool will set up the interrupt for you automatically, and no change is required in the application code.

3. Build the application with the Profile build configuration using the **Profile Configuration** setting in the C/C++ Build configuration options tab. This step appends a `-pg` option to the compiler flags.

Note: To obtain the call graph for functions inside libxil (BSP and drivers), specify `-pg` as extra compiler flags in the Software Platform Settings dialog box.

When libraries are generated, code required for profiling is automatically configured by the standalone BSP and becomes a part of the `libxil.a` library. The compiler inserts a call to the `_mcount` function after every software application function call. The `_mcount` function then gathers data on how often each of these software application functions are called. This function is also provided in the profiling library, and it handles collection of call graph data. The profiling timers initialize during software initialization, and the timer interrupt handlers collect information to provide the histogram data.

Generating and Viewing Profile Data

After compiling the application for profiling, you must run it once to obtain profile data. Start by creating a new run configuration in SDK:

1. In SDK, select **Run > Run** to open the **Run** dialog box.
2. In the Profiler tab:
    a. Select the **Enable Profiling** checkbox.
    b. Enter values for the following three profiling parameters. These are described in more detail in the following sections.
3. Save this configuration and run the application using the profile you created.

SDK runs the program and waits for it to complete (by reaching `exit`). Once the program execution is complete, SDK downloads the generated profile data, processes it using `gprof`, and displays it in the profile view. You can use the XMD console to stop the program at any point during its execution. SDK then downloads the profile information that has been generated up to that point.

For a step-by-step guide to profiling in SDK, refer to Chapter 3, "Profiling Flow in EDK."

Sampling Frequency

The sampling frequency determines the frequency at which timer interrupts are generated. When you set a higher frequency, more samples are obtained. This provides more accuracy but is highly software-intrusive because of the number of interrupts. More calls are inserted to collect data.

Bin Size

The program text region is divided into multiple bins. When a program is interrupted because of the sampling frequency, the bin size determines how accurate the PC location is in the sample.

When you set a smaller bin size, the program text region is divided into a large number of small bins. This allows a more accurate sample because profile data can be attributed to a specific area of the text region. For example, if you set the bin size to 4 bytes, you can narrow down the specific instruction at which the program execution occurred to four bytes of the text region. The disadvantage to using a smaller bin size is that it requires a large number of bins to cover the entire text region, so a large amount of memory space is required for storing profile data.

When you set a larger bin size, the program text region is divided into a small number of large bins. This requires less memory space for storing profile data. However, it is much more difficult to identify specific text regions for the sample because of the larger bin size. For example, if you set the bin size to 40 bytes, you can only determine that the program was executing instructions between x and x+40 on each profile interrupt.

Profile Memory

The profile memory parameter indicates where in memory the profile data must be stored. This memory needs to lie outside the program memory area (including the text, data, heap and stack) and should not be overwritten.

Profiling Flow in EDK

This section is a step-by-step guide to profiling applications in Xilinx® Platform Studio Software Development Kit (SDK). For information about profiling using Xilinx Platform Studio (XPS), refer to Appendix A, “Profiling Applications Using XPS for Software Development.”

Setting the Profile Configuration

Before performing the following steps, ensure that your hardware has a timer that is available for exclusive use by the profiling libraries and that you have enabled profiling in the Software Platform Settings dialog box in SDK.

To set the profile configuration:

1. Select the project name containing the Executable and Linkable Format (ELF) file to profile.
2. Select **Run > Run** to open the Run dialog box.
3. Click **Xilinx C/C++ ELF** in the Configurations list and click **New** to add a new run configuration.
4. In the **Name** field, type a name for the configuration or use the pre-filled name.
5. In the **Main** tab, browse to or type the project name and the C/C++ Application file.
6. Click the **XMD Target Connection** tab.
7. From the **Connection Type** drop-down list, select the preferred target connection type.
8. Click **Advanced Options** to specify advanced connection options for the selected connection type.
9. In the Profiler tab, select the **Enable Profiling** check box to enable the profiler.

   Note: If the tab appears disabled, read the message in blue print. To verify that the ELF file can be profiled on the selected embedded target, refer to the SDK Help System topic “Profiling the Program.” If you are profiling on a hardware target such as MicroBlaze™ MDM, MicroBlaze XMDStub, or PowerPC® processor hardware, you must also specify the Profiling Parameters for Hardware targets:
   - Sampling frequency for the histogram, in Hertz
   - Histogram bin size for programs, in words
   - Scratch memory address to collect Profile data
10. Click **Apply** to save the profile configuration settings.
11. Click **Run** to run and profile the ELF file on the selected embedded target.

For steps on how to control the ELF file as it runs, refer to the SDK Help System topic “Controlling the Program that is Running on an Embedded Target.”

Profile Perspective Navigation

The Xilinx Profiler Perspective presents a collection of views to visualize the statistics for the profiled ELF file. Refer to Chapter 4, “Navigating the Profile Perspective in SDK.”

Navigating the Profile Perspective in SDK

The SDK Profiler Perspective presents a collection of views to visualize the statistics for the profiled Executable and Linkable Format (ELF) file.

Launching the Perspective

SDK automatically launches the Xilinx® Profiler Perspective if the **Show Profile Results Immediately** check box is selected in the Profiler Launch Configuration.

You can also launch the Xilinx Profiler Perspective by selecting **Window > Open Perspective > Xilinx Profiler**.

Perspective Composition

The perspective is composed of the following views:

- C/C++ Projects Management View (default location: top-left of perspective)
- C/C++ Editor View (default location: top-middle of perspective)
- C/C++ source code Outline View (default location: top-right of perspective)

The tabbed sections at the bottom of the perspective contain the following views, which are described further in the next sections.

- 3D Bar Chart — Number of Calls View (default location: tabbed on bottom-left of perspective)
- 3D Pie Chart — Percentage of Time Spent View (default location: tabbed on bottom-left of perspective)
- Time Spent in Function (Self Time) View (default location: tabbed on bottom-left of perspective)
- Call Table View (default location: tabbed on bottom-right of perspective)
- Flat Profile View (default location: tabbed on bottom-right of perspective)
- Console Window (to control the ELF file running on the embedded target (default location: tabbed on bottom-right of perspective).

3D Bar Chart — Number of Calls View

The 3-D Bar Chart view shows the profiling result of the number of calls for each non-zero call function. All the functions are shown in a graphical three-dimensional bar chart with the number of calls in the Y axis and function names in the X axis.

The functions in the X axis are sorted according to the decreasing percentage of time taken, so that the function on the left is taking more time than the function on the right.

When you mouse over each bar, the detailed information for each function displays in a tool tip. The information includes:

- Number of calls
- Percentage of time taken
- Total time taken, in seconds
- Self time taken, in seconds
- Self time taken per call
- Total time taken per call

3D Pie Chart — Percentage of Time Spent View

The 3-D Pie Chart view shows the profiling result of the percentage of time spent for each non-zero function. All the functions are shown in a graphical, three-dimensional pie chart with the function name and the percentage of time in the label.

When you mouse over each region, the detailed information for each function displays in a tool tip. The information includes:

- Percentage of time spent
- Total time taken, in seconds
- Self time taken, in seconds
- Number of calls
- Self time taken per call
- Total time taken per call

Time Spent in Function (Self Time) View

The Time Spent in Function view shows the profiling result of the self time spent per call in each non-zero function. All the functions are shown in the Y axis of the graphical, two-dimensional bar chart. The time unit is shown in the X axis.

The functions in the Y axis are sorted from top to bottom, according to the decreasing percentage of time taken, which means the function on the top is taking more total time than the function on the bottom.

When you point at each bar, SDK displays the detailed information for each function in a tool tip. The information includes:

- Number of calls
- Percentage of time taken
- Total time taken, in seconds
- Self time taken, in seconds
- Number of calls
- Total time taken per call

Call Table View

The Call Table View shows the profiling result for all the functions for the program in a table format. It also specifies the parent and child function call for a particular function. This view contains a call table, and a **Find** field that enables you to locate the specific string inside the table.

Finding a Specific String In the Table

To find a specific string inside the table:

1. Type the specific string in the **Find** field and click the arrow button next to the field.
2. Click the arrow button again to find the next matching string in the table.

The call table contains the following nine columns:

- **Function Name and Index**: Shows the name of the function and its index in the call table.
- **Called By**: Shows which function calls the indexed function.
- **Calls**: Shows all the functions called by the indexed function.
- **Time Taken (%)**: The percentage of time taken for this function in the program.
- **Self Time (seconds)**: How much time was spent on the function itself, in seconds.
- **Children Seconds**: How much time was spent on the child function, in seconds.
- **Calls Made**: Number of calls made from the parent functions or to the child functions.
- **Total Calls Made**: Total number of calls for this function.
- **Name Index**: Red indexes represent the current index of the specified function. Blue indexes represent either the index of the parent or child functions. Double-click the blue index to bring the index function into view.

Code Navigation

To edit a specific function:

1. Double-click a row in the Call Table View to open the code for the function in the editor. The corresponding file opens.
2. Scroll to the specific function and edit the file.

Flat Profile View

The Flat Profile View shows the profiling result of all the functions in a table format for the program. It contains a table, as well as a **Find** field that enables you to find the specific string inside the table.

Finding a Specific String Inside the Table

To find a specific string inside the table:

1. Type the string in the **Find** field and click the arrow button next to the text field.
2. Click the arrow button again to find the next matching string in the table.

The call table contains the following seven columns:

- **Time Taken (%)**: Percentage of time taken for this function in the program.
- **Function Name**: Name of the function.
- **Total Time (seconds)**: Total time taken for this function, in seconds.
- **Self Time (seconds)**: Time spent on the function itself, in seconds.
- **Number of Calls**: How many times this function has been called.
- **Nanoseconds/call (Self)**: Self time taken per call for this function, in nanoseconds.
- **Nanoseconds/call (Total)**: Total time taken per call for this function, in nanoseconds.

Code Navigation

To edit a specific function in the editor:

1. Double-click a row in the Flat Profile View to open the code for the function in the editor. The corresponding file opens.
2. Scroll to the specific function and edit the file.

Console Window

For information on using this view, refer to the "Controlling the Program that is Running on an Embedded Target" help topic in the SDK Help.

Profiling Applications Using XPS for Software Development

This appendix explains the steps involved in profiling applications when using Xilinx® Platform Studio (XPS) for software development instead of Platform Studio Software Development Kit (SDK). The profiling flow is very similar to the flow in SDK, except that in XPS some of the steps are not automated.

Setting Up the Hardware and BSP

Create a proper hardware setup by following the steps listed in "Setting Up the Hardware for Profiling" in Chapter 2. Then set up the software, following the first two steps in "Setting Up the Software to Support Profiling" in Chapter 2.

At this point, the hardware and the BSP should be properly configured, and the application should have interrupts enabled.

Profiling the Application

Profiling the application comprises the following steps:

1. Compiling the Application
2. Running the Application
3. Using the Profile Command

Compiling the Application

The next step is to compile the application with the `-pg` compiler flag. To add the `-pg` flag to the compile line:

1. Right-click the application in the XPS Application tree view and select **Set Compiler Options**.
2. In the Advanced settings, add **-pg** in the Other Compiler Options to Append tab.

   Note: If profile data for Xilinx libraries (BSP and drivers) is also needed, use **-g -pg** as advanced compiler options in the Software Settings dialog box.

When libraries are generated, the standalone BSP automatically configures the code required for profiling, which becomes a part of the `libxil.a` library. The compiler inserts a call to the `_mcount` function after every software application function call. The `_mcount` function then gathers data on how often each of these software application functions are called. This function is also provided in the profiling library and handles collection of call graph data.

The profiling timers are initialized as part of software initialization, and the timer interrupt handlers collect information to provide the histogram data.

Running the Application

Once the application has been compiled for profiling, it must be run once to obtain profile data. This is a three-step process:

1. Specify the profiling options in XMD. These include the sampling frequency, histogram bin size, and the memory address at which the profile data will be stored.
2. Run the program for a sufficiently long interval.

   Note: Assuming the profile interrupts occur in the order of milliseconds, the length of profiling should be in the order of seconds.
3. Download the resulting `gmon.out` data stored in memory into a file, and process the file with `gprof`.

The following example execution session is done at the XMD command line prompt . Bold text indicates user commands.

```
XMD% connect mb mdm
Connecting to cable (Parallel Port - parport0).
.
.
.
Hard Multiplier Support...........on
Barrel Shifter Support............off
MSR clr/set Instruction Support....off
Compare Instruction Support........off
JTAG MDM Connected to MicroBlaze 1
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234

XMD% profile -config sampling_freq_hz 10000 binsize 4 profile_mem
0x22070000

XMD% dow executable.elf
        section, .text: 0x22000000-0x22001cac
        section, .rodata: 0x22001cac-0x22001d34
        section, .data: 0x22001d38-0x22001db8
        section, .sdata: 0x22001db8-0x22001dbc
        section, .bss: 0x22001dc0-0x220055e0
Downloaded Program dhrystone/executable.elf
Setting PC with program start addr = 0x22000000
Program dhrystone/executable.elf being Profiled on Hardware
Initialized Profile Configurations for the Program :
---------------------------------------------------
Sampling Frequency...............10000 Hz
Histogram Bin Size...............4 Words
Memory for Profiling used from...0x22070000
Memory Used for Profiling Data...2140 Bytes

XMD% bps exit
Setting breakpoint at 0x220000f4

XMD% con
Info:Processor started. Type "stop" to stop processor
```

```
RUNNING> XMD% Info:Software Breakpoint 0 Hit, Processor Stopped at
0x220000f4

XMD% profile
        Profile data written to gmon.out
```

The `profile config` command (**`profile -config sampling_freq_hz 10000 binsize 4 profile_mem 0x22070000`**) takes the `Sampling frequency`, `bin size`, and `Profile memory` arguments. Refer to "Generating and Viewing Profile Data" in Chapter 2 for more details on these parameters.

Using the Profile Command

Once the program is run, the `profile` command downloads the profile data stored in memory and stores it in the `gmon.out` file. This file can now be processed with `mb-gprof` or `powerpc-eabi-gprof` to obtain the call graph and histogram data. The following is an example profile processed with `mb-gprof`. For descriptions of these data columns, refer to "Flat Profile View" in Chapter 4.

```
$ mb-gprof dhrystone gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
 %       cumulative   self              self      total
 time    seconds      seconds   calls   ns/call   ns/call   name

 33.33   0.02         0.02      400000  50.00     50.00     Proc_8

 33.33   0.04         0.02                                  main

 16.67   0.05         0.01      400000  25.00     50.00     Proc_1

 16.67   0.06         0.01      400000  25.00     25.00     Proc_6

 0.00    0.06         0.00      1200000 0.00      0.00      Proc_7

 .

 .

 .
```

Appendix B

Profiling Restrictions

The following restrictions apply when profiling in EDK:

- Profiling does not measure the time spent in interrupt handlers because interrupt handlers typically disable further interrupts from occurring. Therefore, it is impossible for profiling interrupts to occur when the program is executing an interrupt handler.
- Profiling can only be done with the standalone platform; it cannot be done in the presence of an OS. This is because the profiling libraries are only available in the standalone BSP.
- Recursive functions are not supported.
- If the timer is directly connected to the processor (for example, when there is no interrupt controller), the software application requires additional setup to support profiling.
- The call graph for functions inside C and Math libraries (`libc` and `libm`) are not generated because these libraries are not compiled with the `-pg` compiler profiling option.
- Ensure that memory used for collecting profile data is not used by any other function in the application.
- If you are using a custom linker script for a PowerPC® processor, it must include a `.vectors` section. This is because profiling is based on interrupts, and using interrupts requires a `.vectors` section.
- Profiling cannot be done while debugging. Enable profiling only when selecting the **Run** configuration in SDK.

Glossary

B

BBD file

Black Box Definition file. The BBD file lists the netlist files used by a peripheral.

BFL

Bus Functional Language.

BFM

Bus Functional Model.

BIT File

Xilinx® Integrated Software Environment (ISE®) Bitstream file.

BitInit

The Bitstream Initializer tool. It initializes the instruction memory of processors on the FPGA and stores the instruction memory in blockRAMs in the FPGA.

block RAM (BRAM)

A block of random access memory built into a device, as distinguished from distributed, LUT based random access memory.

BMM file

Block Memory Map file. A BMM file is a text file that has syntactic descriptions of how individual block RAMs constitute a contiguous logical data space. Data2MEM uses BMM files to direct the translation of data into the proper initialization form. Because a BMM file is a text file, it is directly editable.

BSB

Base System Builder. A wizard for creating a complete design in Xilinx Platform Studio (XPS). BSB is also the file type used in the Base System Builder.

BSP

See Standalone BSP.

C

CDMAC

Communications Direct Memory Access Controller.

D

DCM

Digital Clock Manager

DCR

Device Control Register.

DLMB

Data-side Local Memory Bus. See also: LMB.

DMA

Direct Memory Access.

DOPB

Data-side On-chip Peripheral Bus. See also: OPB.

DRC

Design Rule Check.

DSPLB

Data-side Processor Local Bus. See also: ISPLB.

E

EDIF file

Electronic Data Interchange Format file. An industry standard file format for specifying a design netlist.

EDK

Xilinx Embedded Development Kit.

ELF file

Executable and Linkable Format file.

EMC

External Memory Controller.

EST

Embedded System Tools.

F

FATfs (XilFATfs)

LibXil FATFile System. The XilFATfs file system access library provides read/write access to files stored on a Xilinx SystemACE CompactFlash or IBM microdrive device.

FPGA

Field Programmable Gate Array.

FSL

MicroBlaze™ Fast Simplex Link. Unidirectional point-to-point data streaming interfaces ideal for hardware acceleration. The MicroBlaze processor has FSL interfaces directly to the processor.

G

GDB

GNU Debugger.

GPIO

General Purpose Input and Output. A 32-bit peripheral that attaches to the on-chip peripheral bus.

H

Hardware Platform

Xilinx FPGA technology allows you to customize the hardware logic in your processor subsystem. Such customization is not possible using standard off-the-shelf microprocessor or controller chips. Hardware platform is a term that describes the flexible, embedded processing subsystem you are creating with Xilinx technology for your application needs.

HDL

Hardware Description Language.

I

IBA

Integrated Bus Analyzer.

IDE

Integrated Design Environment.

ILA

Integrated Logic Analyzer.

ILMB

Instruction-side Local Memory Bus. See also: LMB.

IOPB

Instruction-side On-chip Peripheral Bus. See also: OPB.

IPIC

Intellectual Property Interconnect.

IPIF

Intellectual Property Interface.

ISA

Instruction Set Architecture. The ISA describes how aspects of the processor (including the instruction set, registers, interrupts, exceptions, and addresses) are visible to the programmer.

ISC

Interrupt Source Controller.

ISE

Xilinx ISE Project Navigator project file.

ISPLB

Instruction-side Peripheral Logical Bus. See also: DSPLB.

ISS

Instruction Set Simulator.

J

JTAG

Joint Test Action Group.

L

Libgen

Library Generator sub-component of the Xilinx Platform Studio technology.

LMB

Local Memory Bus. A low latency synchronous bus primarily used to access on-chip block RAM. The MicroBlaze processor contains an instruction LMB bus and a data LMB bus.

M

MDD file

Microprocessor Driver Description file.

MDM

Microprocessor Debug Module.

MFS

LibXil Memory File System. The MFS provides user capability to manage program memory in the form of file handles.

MHS file

Microprocessor Hardware Specification file. The MHS file defines the configuration of the embedded processor system including buses,peripherals, processors, connectivity, and address space.

MLD file

Microprocessor Library Definition file.

MOST®

Media Oriented Systems Transport. A developing standard in automotive network devices.

MPD file

Microprocessor Peripheral Definition file. The MPD file contains all of the available ports and hardware parameters for a peripheral.

MSS file

Microprocessor Software Specification file.

N

NCF file

Netlist Constraints file.

NGC file

The NGC file is a netlist file that contains both logical design data and constraints. This file replaces both EDIF and NCF files.

NGD file

Native Generic Database file. The NGD file is a netlist file that represents the entire design.

NGO File

A Xilinx-specific format binary file containing a logical description of the design in terms of its original components and hierarchy.

NPI

Native Port Interface.

O

OCM

On Chip Memory.

OPB

On-chip Peripheral Bus.

P

PACE

Pinout and Area Constraints Editor.

PAO file

Peripheral Analyze Order file. The PAO file defines the ordered list of HDL files needed for synthesis and simulation.

PBD file

Processor Block Diagram file.

Platgen

Hardware Platform Generator sub-component of the Platform Studio technology.

PLB

Processor Local Bus.

PROM

Programmable ROM.

PSF

Platform Specification Format. The specification for the set of data files that drive the EDK tools.

S

SDF file

Standard Data Format file. A data format that uses fields of fixed length to transfer data between multiple programs.

SDK

Software Development Kit.

Simgen

The Simulation Generator sub-component of the Platform Studio technology.

Software Platform

A software platform is a collection of software drivers and, optionally, the operating system on which to build your application. Because of the fluid nature of the hardware platform and the rich Xilinx and Xilinx third-party partner support, you may create several software platforms for each of your hardware platforms.

SPI

Serial Peripheral Interface.

Standard C Libraries

EDK libraries and device drivers provide standard C library functions, as well as functions to access peripherals. Libgen automatically configures the EDK libraries for every project based on the MSS file.

Standalone BSP

Standalone Board Support Package. A set of software modules that access processor-specific functions. The Standalone BSP is designed for use when an application accesses board or processor features directly (without an intervening OS layer).

SVF File

Serial Vector Format file.

U

UART

Universal Asynchronous Receiver-Transmitter.

UCF

User Constraints File.

V

VHDL

VHSIC Hardware Description Language.

X

XBD File

Xilinx Board Definition file.

XCL

Xilinx CacheLink. A high performance external memory cache interface available on the MicroBlaze processor.

Xilkernel

The Xilinx Embedded Kernel, shipped with EDK. A small, extremely modular and configurable RTOS for the Xilinx embedded software platform.

XMD

Xilinx Microprocessor Debugger.

XMP File

Xilinx Microprocessor Project file. This is the top-level project file for an EDK design.

XPS

Xilinx Platform Studio. The GUI environment in which you can perform the develop your embedded design.

XST

Xilinx Synthesis Technology.

Z

ZBT

Zero Bus Turnaround™.