**Profiling Overview**

In SDK, you can profile a program running on embedded hardware. The Profiling feature is software-intrusive, and is based on the GNU gprof tool. 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

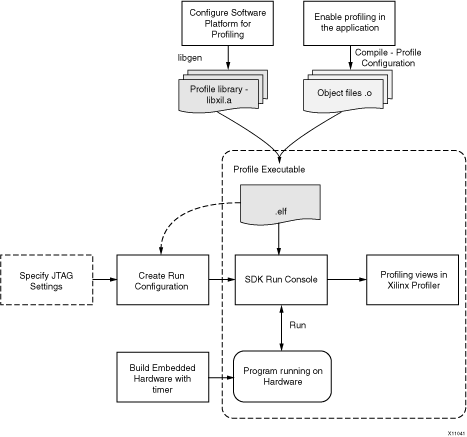
For additional information about GNU gprof, refer to <http://sourceware.org/binutils/docs-2.18/gprof/index.html>.

**How Profiling Works**

The execution flow of the program is altered so that gprof can obtain data. 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 profiling workflow is described in the following diagram:



**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 Zynq®-7000 ARM processors, the internal SCU timer should be used.

**Setting Up the Software for Profiling**

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

1. Enable profiling in the Software Platform to include profiling libraries. Profiling is supported only for standalone software platforms.
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.
3. Use an executable compiled for profiling (such as the -pg compiler switch). You should use the profile build configuration to build your application. Refer to [Build Configurations](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_build.htm#50655263_74172) for more information.

**Generating and Viewing Profile Data**

To profile the application, you must create a **run configuration** in SDK and provide options in the Profiler tab. Refer to [Run Configurations](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_debug_run_configuration.htm#50655260_65733) and [Profile Configuration](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_profile_configuration.htm#50655250_56360) for more information.

When profiling the program on a hardware target, SDK uses Xilinx Microprocessor Debugger (XMD) for communication to the processor.

The XMD **Run Console** allows you to stop the program execution. When the program completes execution, the Profiler automatically stops the program, then reads the profile data and processes it. You can also choose to stop program execution at any time.

The **Xilinx Profiler View** enables you to visualize the statistics for the profiled program. To open the Xilinx Profiler View, double-click the profile results data file in the Project Explorer view after the program execution completes or stops. Refer to [Xilinx Profiling View](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_profiler_perspective.htm#50655249_55081) for more information.

**Supported Profile Targets**

SDK supports profiling the application on all processor architecture targets running on a hardware board. As mentioned here, the hardware system should have a timer interrupt for generating periodic sampling interrupts.

**Related Concepts**

[Build Configurations](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_build.htm" \l "50655263_74172)

[Run Configurations](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_debug_run_configuration.htm" \l "50655260_65733)

[Profile Configuration](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_profile_configuration.htm" \l "50655250_56360)

[Xilinx Profiling View](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/SDK_Doc/concepts/sdk_c_profiler_perspective.htm" \l "50655249_55081)

Copyright © 1995-2014 Xilinx, Inc. All rights reserved.