Real Time Embedded Systems

"System On Programmable Chip"

Profiling methodology

René Beuchat

Laboratoire d'Architecture des Processeurs

rene.beuchat@epfl.ch

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Contents

- Introduction
- Software/hardware profiling
- Trace
- Profiling laboratory

Reference:

- *http://www.altera.com/literature/an/an391.pdf*
- *http://www.altera.com/literature/hb/nios2/qts_qii55001.pdf*

- *http://www-list.cea.fr/labos/fr/LSL/test/pathcrawler/wcet.html*

Introduction

- In a lot of case in an Embedded system it is necessary to know the execution time of a function, a thread or some part of a program.
- Especially for process scheduling, the Worst Case Execution Time (***WCET***) is an important parameter.
- The "*Real Time programming course*" (JD Decotignie, EPFL) details the algorithms used.

Introduction

- In this chapter, we will have a look and practical laboratory on software and hardware profiling.
- Profiling is a method for a software / hardware designer to evaluate the time spends in part of a program.
- Profiling can be done in software only or with the help of part of hardware.

Introduction

- Time estimation can be done.
  - **statically** by the way of simulator, graph analysis, the big challenges are :
    - estimation of cache miss/hit,
    - memory access, memory organisation
    - burst in memory access
    - interruptions (asynchronous events) time execution and rate
    - DMA bus time use
    - synchronization suspend time (semaphore, flags, …)
  - **Dynamically** by the execution on real system

Introduction

- In dynamically test, it is very difficult to be sure that the Worst Case is detected, but some intervals can be found. Margins need to be taken in account for security.

- A very difficult estimation time is when floating number are used and calculated by software.

Software profiling

- Software profiling is done by specifying to the compiler to add some instructions at the start and end of a function. Information will be saved in a file and accumulated time is registered.
- Precise time, depending on timer accuracy and period, can be registered.
- Unfortunately, there are a big number of instructions needed for this possibility.

Software profiling

- Each time there is an input to a function, address and time are saved
- Each time there is a return or end of a function, execution time is cumulated for this function in memory.
- The compiler need to support this functionality, gcc do it with a special compiler switch and the *xxx-elf-gprof* the **profiler** tool.

Software profiling

- Statistical dynamic profiling can be done with a timer.
- Every σ provided by an interruption and generated by a timer, the tool looks the address of the PC. It can known the function executed and increment an associated counter.
- If the sampling time is big enough and not synchronized with the scheduler timer, good execution statistics can be available.
- Here again, time is lost by the profiling system and reduce the real execution time.

Hardware profiling

- Dynamic profiling can be done with the help of some hardware part.
- Counters with big capacity (64 bits) and resolution (MHz .. 100 MHz) can be added to the system (and thus use some hardware resources). The counters are accessible as programmable interface.
- The principle is the same as for software profiling, the compiler add access at beginning (START) and end (STOP) of a function to the specific counter action associated to it.
- After some time, the total time spends to the function can be known.

Hardware profiling

- We can imagine min-max function for this counter.
- The time added to access the counter is less than a full software profiling
- The number of counter is limited by the available space in the FPGA
- Big precision can be obtain, but we can NOT guaranty that the WCET is met.

Performance counter core

Offset	Register Name	Bit Description		
		Read	Write	
		31 … 0	31 … 1	0
0	$T[0]_{lo}$	global clock cycle counter [31: 0]	*(1)*	0 = STOP 1 = RESET
1	$T[0]_{hi}$	global clock cycle counter [63:32]	*(1)*	0 = START
2	Ev[0]	global event counter	*(1)*	*(1)*
3	–	*(1)*	*(1)*	*(1)*
4	$T[1]_{lo}$	section 1 clock cycle counter [31: 0]	*(1)*	0 = STOP
5	$T[1]_{hi}$	section 1 clock cycle counter [63:32]	*(1)*	0 = START
6	Ev[1]	section 1 event counter	*(1)*	*(1)*
7	–	*(1)*	*(1)*	*(1)*
8	$T[2]_{lo}$	section 2 clock cycle counter [31: 0]	*(1)*	0 = STOP
9	$T[2]_{hi}$	section 2 clock cycle counter [63:32]	*(1)*	0 = START
10	Ev[2]	section 2 event counter	*(1)*	*(1)*
11	–	*(1)*	*(1)*	*(1)*
. . .	. . .	. . .	. . .	. . .
$4n + 0$	$T[n]_{lo}$	section *n* clock cycle counter [31: 0]	*(1)*	0 = STOP
$4n + 1$	$T[n]_{hi}$	section *n* clock cycle counter [63:32]	*(1)*	0 = START
$4n + 2$	Ev[n]	section *n* event counter	*(1)*	*(1)*
$4n + 3$	–	*(1)*	*(1)*	*(1)*

Performance counter functions

Name	Summary
PERF_RESET()	Stops and disables all counters, resetting them to 0.
PERF_START_MEASURING()	Starts the global counter and enables section counters.
PERF_STOP_MEASURING()	Stops the global counter and disables section counters.
PERF_BEGIN()	Starts timing a code section.
PERF_END()	Stops timing a code section.
perf_print_formatted_report()	Sends a formatted summary of the profiling results to stdout.
perf_get_total_time()	Returns the aggregate global profiling time in clock cycles.
perf_get_section_time()	Returns the aggregate time for one section in clock cycles.
perf_get_num_starts()	Returns the number of counter events.
alt_get_cpu_freq()	Returns the CPU frequency in Hz.

Performance counter functions

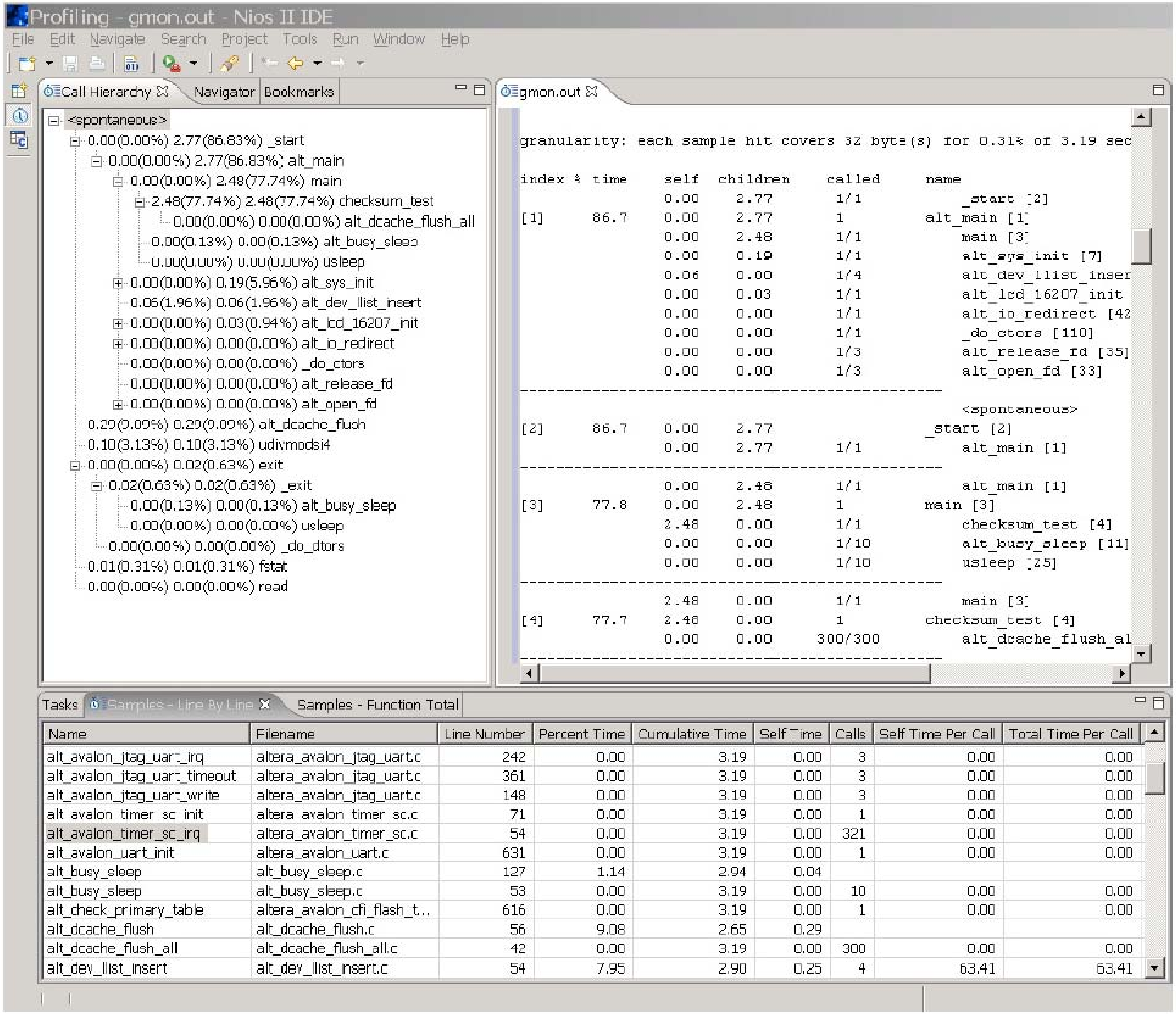
```
perf_print_formatted_report(
(void *)PERFORMANCE_COUNTER_BASE,
                                // Peripheral's HW base address
alt_get_cpu_freq(),             // defined in "system.h"
3,                              // How many sections to print
"1st checksum_test",            // Display-names of sections
"pc_overhead",
"ts_overhead");
```

Performance counter functions

```
--Performance Counter Report--
Total Time: 2.07711 seconds (103855534 clock-cycles)
+-----------------+--------+-----------+---------------+-----------+
| Section         |      % | Time (sec)| Time (clocks) |Occurrences|
+-----------------+--------+-----------+---------------+-----------+
|1st checksum_test|     50 |  1.03800  |      51899750 |        1  |
+-----------------+--------+-----------+---------------+-----------+
| pc_overhead     |1.73e-05|  0.00000  |            18 |        1  |
+-----------------+--------+-----------+---------------+-----------+
| ts_overhead     |4.24e-05|  0.00000  |            44 |        1  |
+---------------+-------+---------+-------------+----------+
```

- 3 sections, relative and absolute timing information

Profiling display (hierarchical/flat)



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Hardware profiling

- Using external hardware, as big memory, it is possible to TRACE the program execution by registering all addresses. A post processing can be executed to analyze the registered information and profiling the program execution. Some filters can be added on the memorization system.
- This information can be very useful for debugging purpose

Hardware profiling

- Traces registration of hardware signals (*as with Altera signal tap analyzer or Xilinx chipscope*), as memory access, can determine the good or bad utilization of memory bandwidth, and thus function execution time.

Hardware profiling

- With the help of a precise timer, execution time between an event and synchronized process access can be obtain.
- Interrupt response, interrupt latency can be precisely measure by this way.
- With he help of parallel port and external logic analyzer, function time execution, jitter and variation can be observed. Some instruction to access the parallel port are necessary and need to be provided by the programmer or the compiler with pragma informations.

Profiling

- Profiling can be very useful to help to optimize time execution. The programmer can have information were to optimize the program and/or where to search to accelerate some part by hardware accelerator or specific optimized instructions.

GNU profiler advantage/disadvantage

- The software profiler gives a complete view of the program profile without the programmer intervention.
- Software is added thus reducing the real execution time. Each function is larger in code. It access another function to collect information → **Cache will not works as without profiling, more cache miss will be available.**

GNU profiler advantage/disadvantage

- Profiling by timer sampling is not available when the processor is interrupt disabled. Thus time spend in interrupt routine is not take in account, if timer interrupts are not enabled.
- Software Profiling is done for the entire system, not for one specific function.

Hardware profiling

- The hardware trace needs a specific hardware interface and the analyzing software (ex. from ***FS2*** or ***Lauterbach***, for NIOS2 systems)

Laboratory

- To exercise profiling make the Altera tutorial
- Be careful to NOT use the small library, as floating point values are printed. The full library is needed for that → with FPGA4U external SDRAM is thus necessary.