# **Real Time Embedded Systems**

### "System On Programmable Chip"

### **Profiling methodology**

René Beuchat

Laboratoire d'Architecture des Processeurs

rene.beuchat@epfl.ch



RB - 2008 - EPFL/LAP

### Contents

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

#### Reference:

- <u>http://www.altera.com/literature/an/an391.pdf</u>
- <u>http://www.altera.com/literature/hb/nios2/qts\_qii55001.pdf</u>
- http://www-list.cea.fr/labos/fr/LSL/test/pathcrawler/wcet.html



- 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.



- 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.



- 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



- 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.



- 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.



- 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**

		Bit Description							
Offset	Register Name	Read	Write						
		31 0	31 1	0					
0	T[0] <sub>10</sub>	global clock cycle counter [31: 0]	(1)	0 = STOP 1 = RESET					
1	T[0] <sub>hi</sub>	global clock cycle counter [63:32]	(1)	0 = START					
2	Ev[0]	global event counter	(1)	(1)					
3	-	(1)	(1)	(1)					
4	T[1] <sub>10</sub>	section 1 clock cycle counter [31: 0]	(1)	0 = STOP					
5	T[1] <sub>hi</sub>	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] <sub>10</sub>	section 2 clock cycle counter [31: 0]	(1)	0 = STOP					
9	T[2] <sub>hi</sub>	section 2 clock cycle counter [63:32]	(1)	0 = START					
10	Ev[2]	section 2 event counter	(1)	(1)					
11	_	(1)	(1)	(1)					
•		•	•						
	•		•						
4 <i>n</i> + 0	T[n] <sub>10</sub>	section n clock cycle counter [31: 0]	(1)	0 = STOP					
4 <i>n</i> + 1	T[n] <sub>hi</sub>	section n clock cycle counter [63:32]	(1)	0 = START					
4 <i>n</i> + 2	Ev[n]	section <i>n</i> event counter	(1)	(1)					
4 <i>n</i> + 3	_	(1)	(1)	(1)					



RB - 2008 - EPFL/LAP

### **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.				
<pre>perf_print_formatted_report()</pre>	Sends a formatted summary of the profiling results to stdout.				
<pre>perf_get_total_time()</pre>	Returns the aggregate global profiling time in clock cycles.				
<pre>perf_get_section_time()</pre>	Returns the aggregate time for one section in clock cycles.				
<pre>perf_get_num_starts()</pre>	Returns the number of counter events.				
alt_get_cpu_freq()	Returns the CPU frequency in Hz.				



#### perf\_print\_formatted\_report( (void \*)PERFORMANCE COUNTER BASE,

// Peripheral's HW base address

alt get cpu freq(), 3,

"pc overhead",

"ts overhead");

// How many sections to print

// defined in "system.h"

"1st checksum test", // Display-names of sections



RB - 2008 - EPFL/LAP

#### **Performance counter functions**

	formance Counter Report Time: 2.07711 seconds (103855534 clock-cycles) + tion   %   Time (sec)  Time (clocks)  Occurrences					
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			

• 3 sections, relative and absolute timing information



RB - 2008 - EPFL/LAP

### **Profiling display (hierarchical/flat)**

Profiling - gmon.out - N Edit Navigate Search Pro	iject Tools <u>R</u> un <u>W</u> indow	Нер								
- 🗉 📥 🛛 💁 - 🛛 🛷	] *= � <del>•</del> +									
👌 Call Hierarchy 😂 🛛 Navigat	or Bookmarks		Imon.out 🖾 🔪							E
										+
	196) etart		ranularitu.	sech sem	nia h	it cover	. 37	hute(g) for D	31% of 3.19 se	
		2	,ranularicy.	cach Sai	ipic II	IC COVEL	5 52 .	byce(b) for o.	51% OF 5.15 2C	
			ndex % time	self	chil	dren	calle	d name		
			Index 3 Lime			2.77 1/1				
E-2.48(77.74%) 2.48(77.74%) checksum_test			1] 86.7			77	1		rt [2]	
	6)	ish_ali L	[1] 00.1	0.00		48	1/1	alt_main		-
	).00(0.13%) alt_busy_sleep			0.00		19	1/1	main	NO 470 8 7	
	).00(0.00%) usleep								sys_init [7]	
÷-0.00(0.00%) 0.19	9(5.96%) alt_sys_init			0.06		00	1/4		dev_llist_inse:	r
-0.06(1.96%) 0.06	i(1.96%) alt_dev_llist_insert			0.00		03	1/1		lcd_16207_init	
€ 0.00(0.00%) 0.03(0.94%) alt_cd_16207_nit € 0.00(0.00%) 0.00(0.00%) alt_o_redirect				0.00	0.00 0.00		1/1		alt_io_redirect [42	
							1/1		ctors [110]	
	)(0.00%) _do_ctors			0.00		S. C. 1953.	1/3		alt_release_fd [35]	
	)(0.00%) alt release fd			0.00	0.00		1/3	alt_	alt_open_fd [33]	
	)(0.00%) alt_open_fd	-								
	· · · · · · · · · · · · · · · · · · ·							10 The second	ntaneous>	
-0.29(9.09%) 0.29(9.09%		6	2] 86.7	- 180.55 T	2.77	_start [2]				
-0.10(3.13%) 0.10(3.13%				0.00	2.	77	1/1	alt_	main [1]	
□ 0,00(0.00%) 0.02(0.63%)										
0.02(0.63%) 0.02(0.6				0.00		48	1/1	A. 194 (1977) 198 (1977)	main [1]	
0.00(0.13%) 0.00	)(0.13%) alt_busy_sleep	0	3] 77.8			48	1	main [3]		
0.00(0.00%) 0.00	)(0.00%) usleep			2.48		00	1/1		checksum_test [4]	
0.00(0.00%) 0.00(0.	00%) _do_dtors			0.00	0.00 1	1/10	alt_	_busy_sleep [11]	]	
-0.01(0.31%) 0.01(0.319	6) fstat			0.00	σ.	00	1/10	usle	ep [25]	
0.00(0.00%) 0.00(0.00%	6) read	57		2.48 0.00 2.48 0.00 0.00 0.00					503	
			41 99 9						main [3]	
		Ļ	[4] 77.7						checksum_test [4]	
						00 300/300		0 alt_dcache_flush_al		<u>ا ج ا</u>
			•						•	Г
r Tasks 🗴 Samples - Line By Lin	Samples - Function T	otal								-
Name	Filename	Line Number	Percent Time	Cumulative	Time	SelfTime	Calls	Self Time Per Call	Total Time Per Ca	11 🕒
alt_avalon_jtaq_uart_irg	altera_avalon_jtaq_uart.c	242	0.00		3.19	0.00	3	0.00	0.0	D
alt avalon jtag uart timeout	altera avalon jtag uart.c	361	0.00		3.19	0.00	3	0.00	0.0	0
alt_avalon_jtag_uart_write	altera_avalon_jtag_uart.c	148	0.00		3.19	0.00	3	0.00	0.0	0 -
alt_avalon_timer_sc_init	altera_avalon_timer_sc.c	71	0.00		3.19	0.00	1	0.00	0.0	D
alt_avalon_timer_sc_irg	altera_avalon_timer_sc.c	54	0.00		3.19	0.00	321	0.00	0.0	0
alt_avalon_uart_init	altera_avalon_uart.c	631			3.19	0.00	1	0.00	0.0	D
alt_busy_sleep	alt_busy_sleep.c	127	1.14		2.94	0.04				
alt_busy_sleep	alt busy sleep.c	53	0.00		3.19	0.00	10	0.00	0.0	0
alt_check_primary_table	altera_avalon_cfi_flash_t	616			3.19	0.00	1	0.00		
alt_dcache_flush	alt_dcache_flush.c	56			2.65	0.29	1			
alt dcache flush all	alt dcache flush all.c	42			3.19	0.00	300	0.00	0.0	D
alt dev list insert	alt dev llist insert.c	54			Z.90	0.25	4	63.41		
	Parc_dor_moc_hoor de		1.55	1	2.50	0,23	7	05,71	00,7	



- 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



 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.



RB - 2008 - EPFL/LAP

- 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.



- 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.



- 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.



 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.

