Detection and Resolution of Real-Time Issues using TimeDoctor

Embedded Linux Conference

David Legendre & François Audéon
NXP Semiconductors – Business Line Digital Video Systems
Linz, November 2, 2007

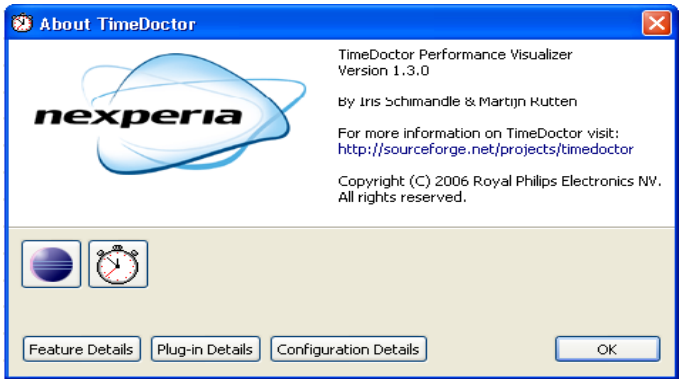
Presentation outline

- Everything you have always wanted to know about TimeDoctor
  - Key features
  - Setup on a Linux based, embedded system
- Example use cases
  - Detection and resolution of real time issues
  - Performance monitoring and analysis
- Conclusions and future work

Introduction to TimeDoctor

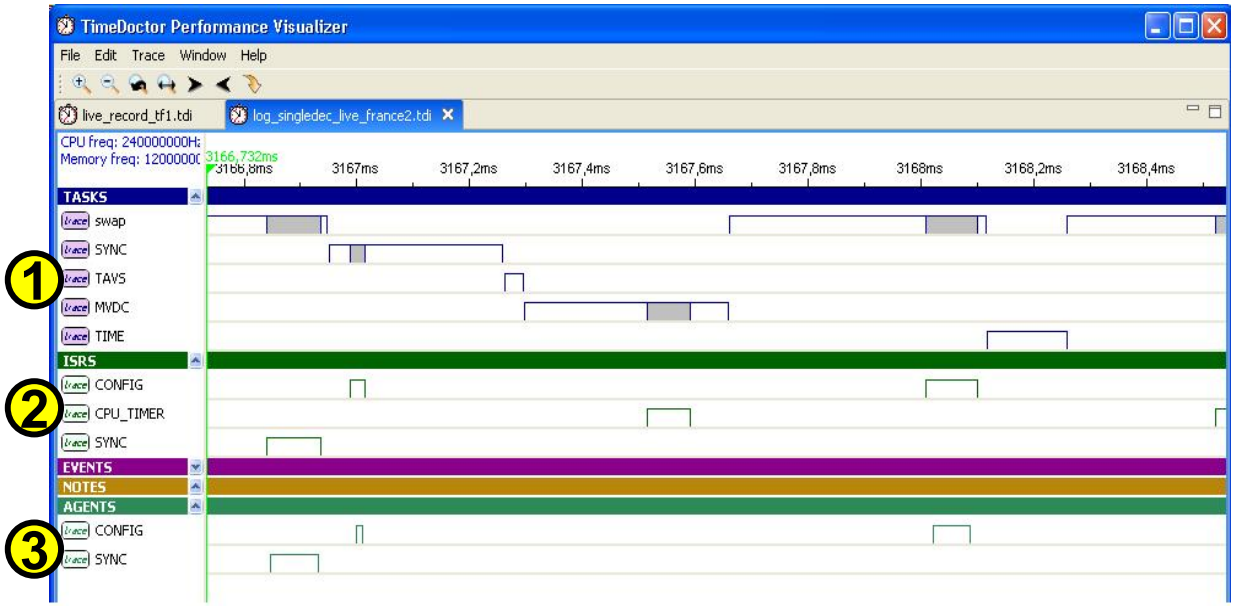
Features

- Graphical tool for visualizing time stamped debug information
  - Available as an ECLIPSE plug-in or a standalone executable
  - Philips/NXP development, recently made available under open source license
    - http://sourceforge.net/projects/timedoctor
- Real Time objects monitoring
  - Tasks
  - Events
  - Semaphores
  - Message Queues
  - Interrupts



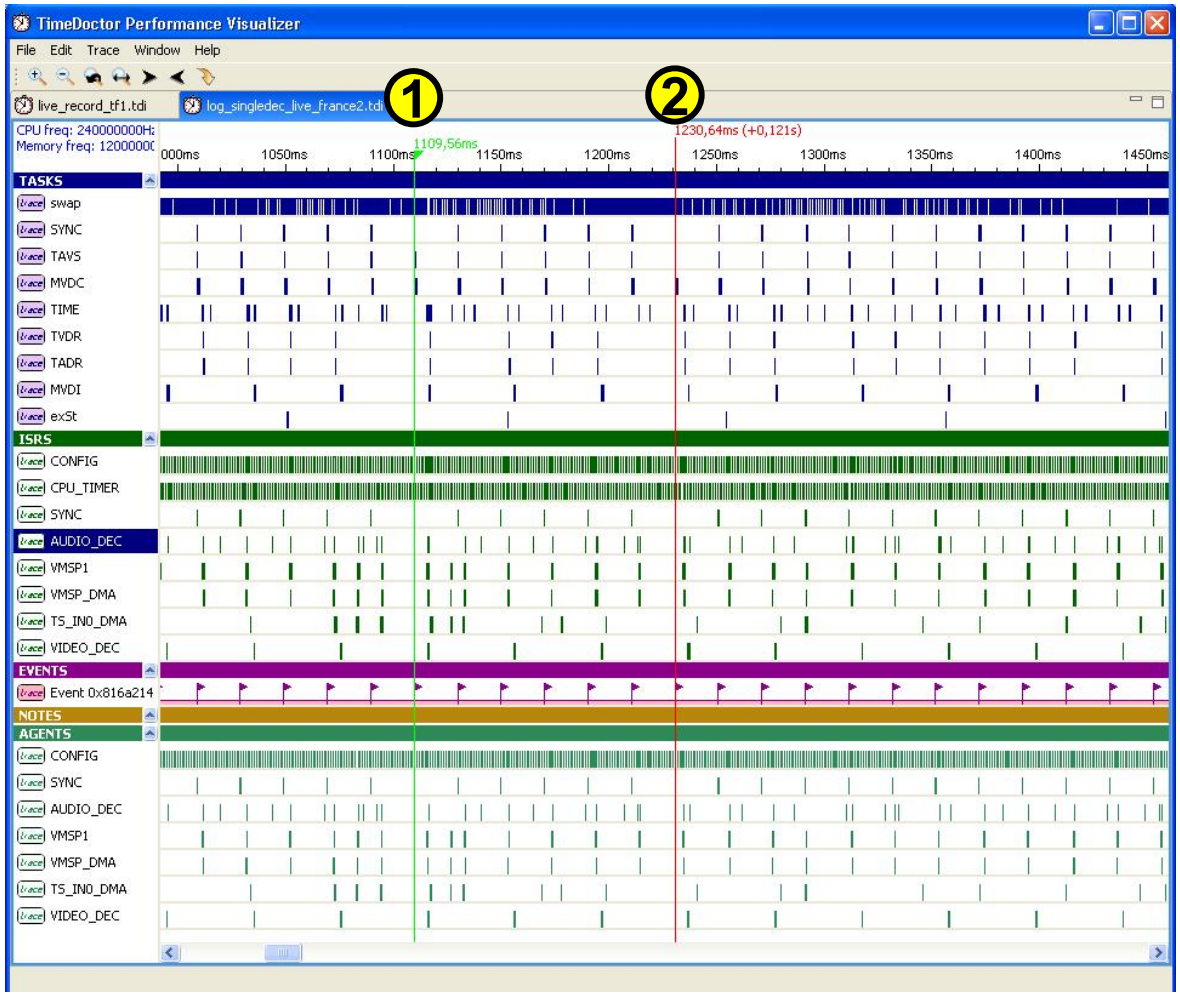
- Statistics computing for CPU time spent in task and interrupts
  - For a defined period of time
  - For all the recorded samples
- Collection of general purpose information
  - Agents

Outline of the tool (1/4)



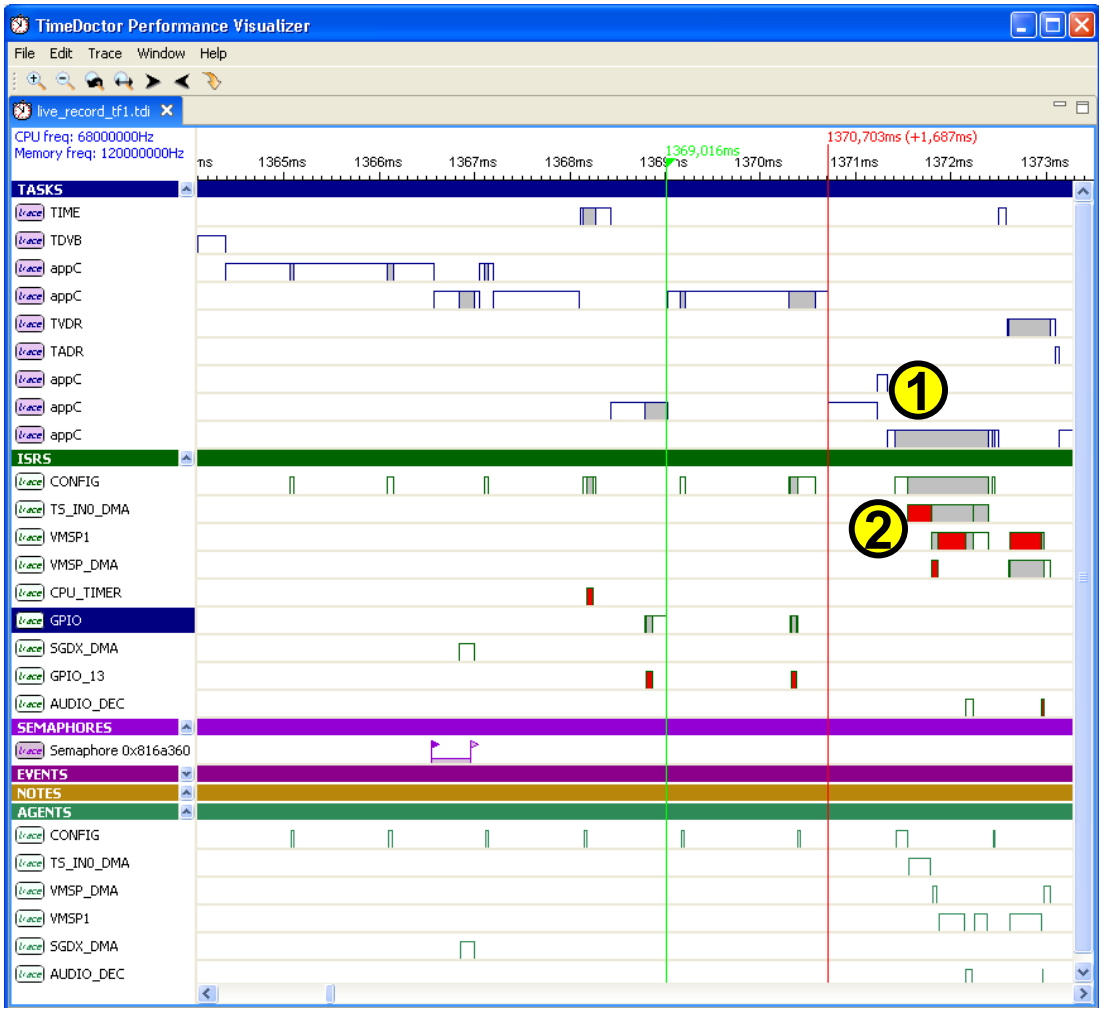
- (1) Tasks:
  - Kernel thread
  - User space thread
- (2) ISRs:
  - Kernel interrupt handlers
- (3) Agents:
  - General purpose (Here top-half interrupt handlers for platform specific drivers)

Outline of the tool (2/4)



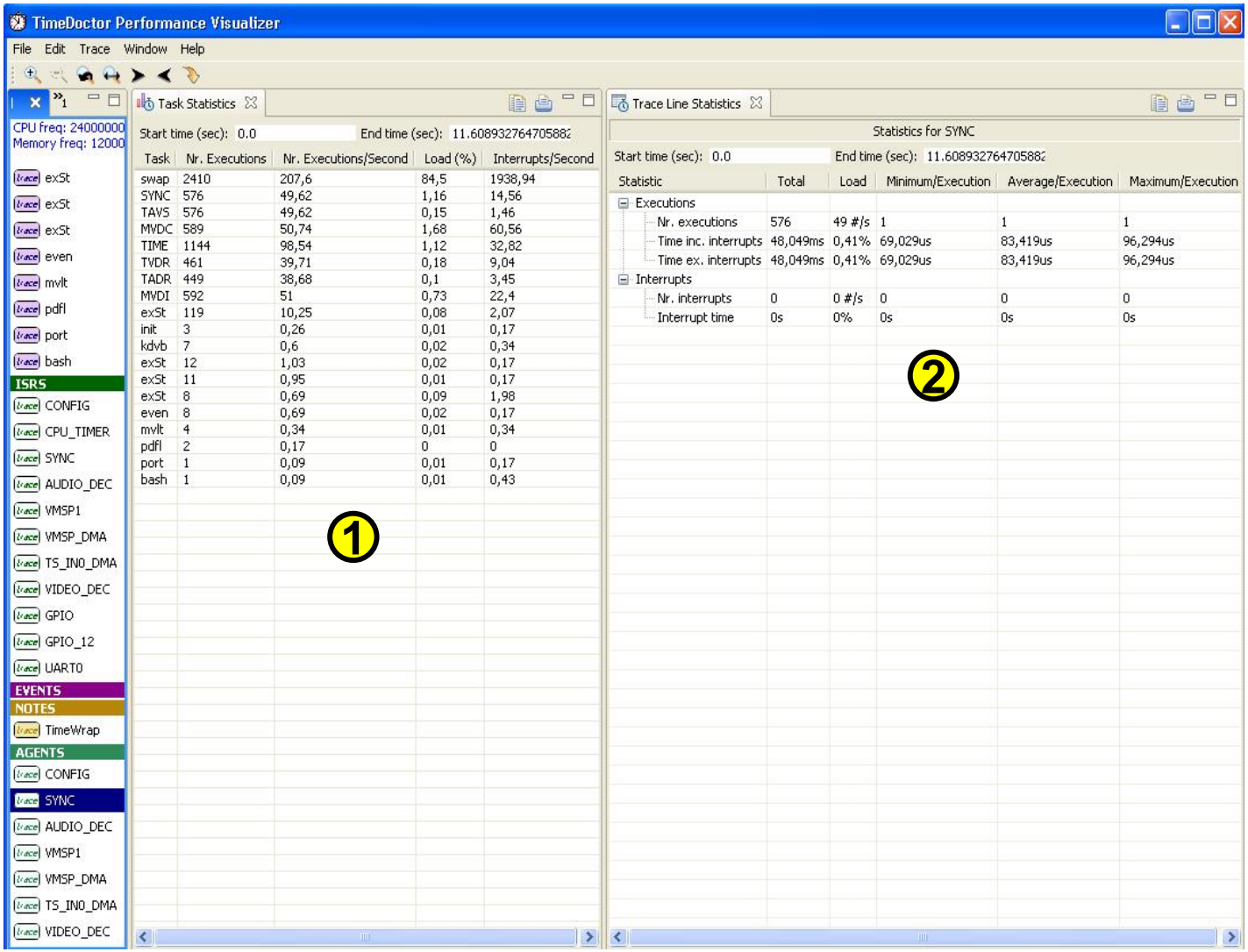
- (1)-(2) Timing measurement
- Zoom in & out

Outline of the tool (3/4)



- (1) Preemption between tasks and ISRs
- (2) Preemption between ISRs

Outline of the tool (4/4)

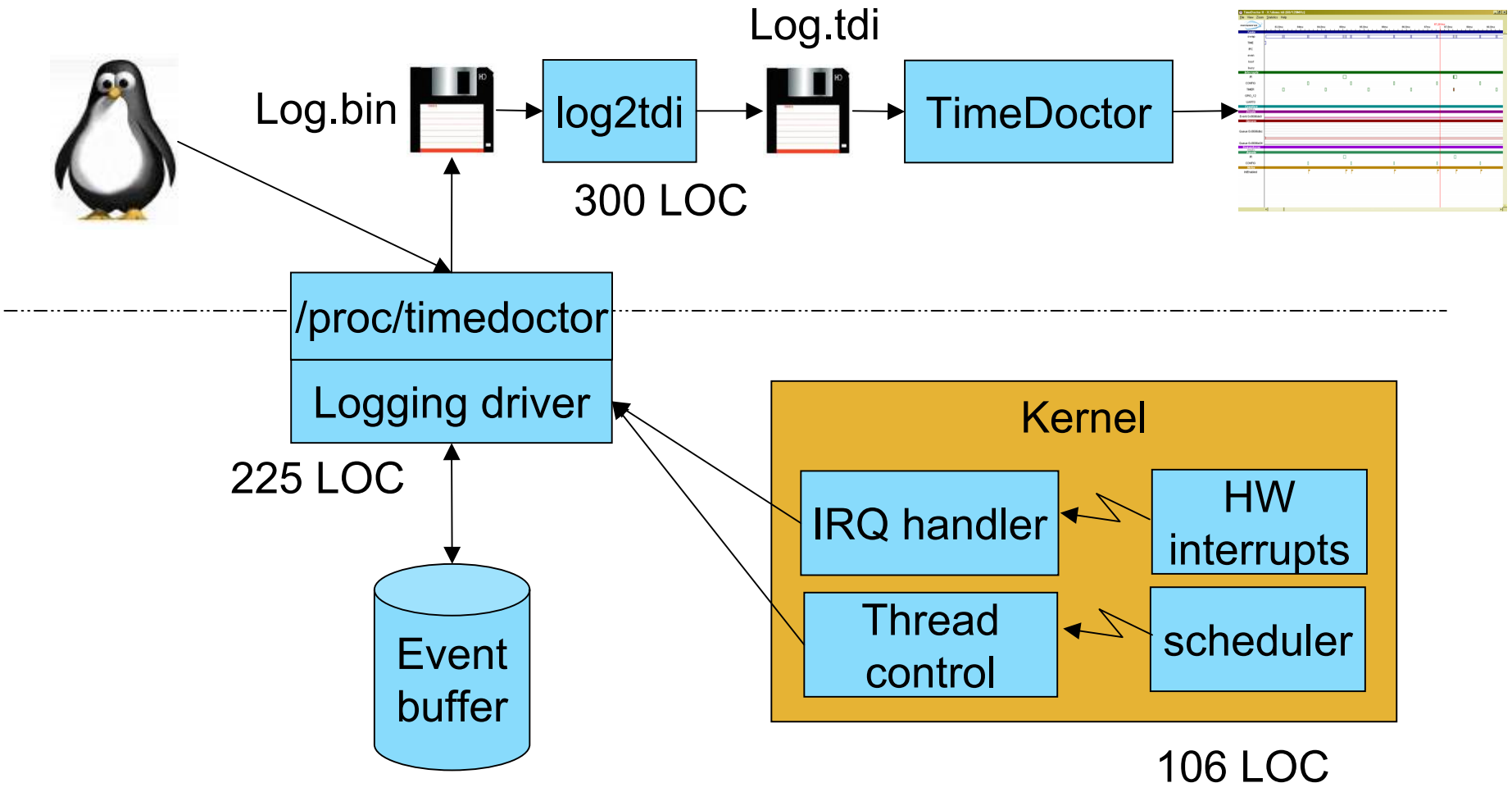


- (1) Tasks summary
  - Execution
  - Load
  - Interrupts
- (2) Detailed statistics
  - Load
  - Minimum
  - Average
  - maximum

Implementation (1/2): modification of code

- In Kernel space:
  - A Linux driver: 225 LOC (include and source files)
    - Low-intrusion logging functions to record the debug events (assembly and C)
    - Control of the event buffer
    - Public control API (e.g reset/start/stop the logging)
  - Kernel patches (fork.c, sched.c, arch/xxxx/interrupts.c): 106 LOC
    - Thread creation and context switching
    - Interrupt occurrences
- In user space:
  - A /proc interface to control the logging and to dump the event buffer
  - A Perl script (~300 lines) to convert the event buffer into a TimeDoctor compliant input file.

Implementation: dataflow



Summary

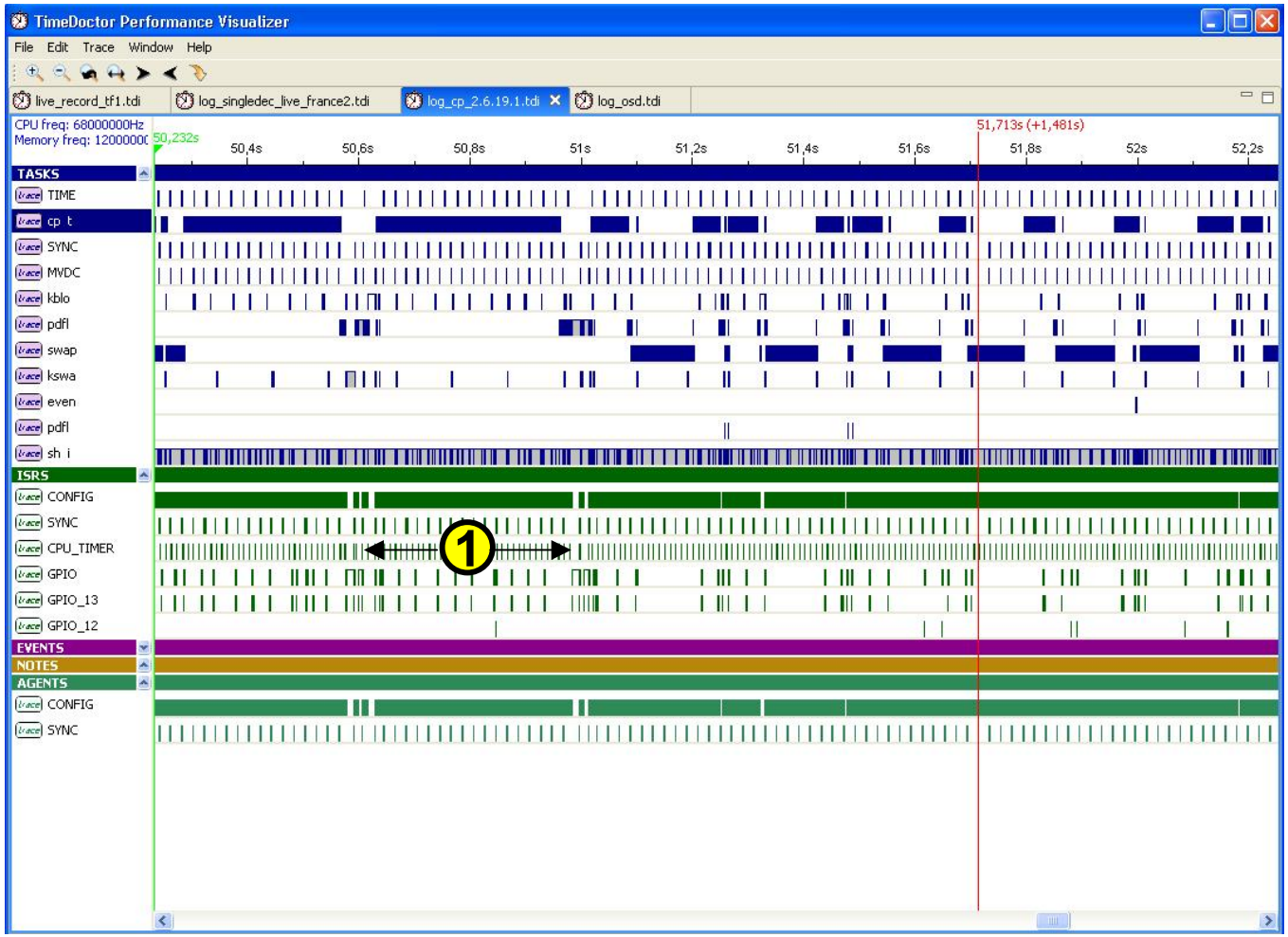
- Tool enabling monitoring of CPU activity through an easy to use, graphical user interface
- Helps with performance analysis and debug
- Easy to customize to log specific messages
- Embedded part is easy to adapt to a new OS, and has a very low and predictable overhead

More concretely? Example use cases

First issue: HDD access (1/3)

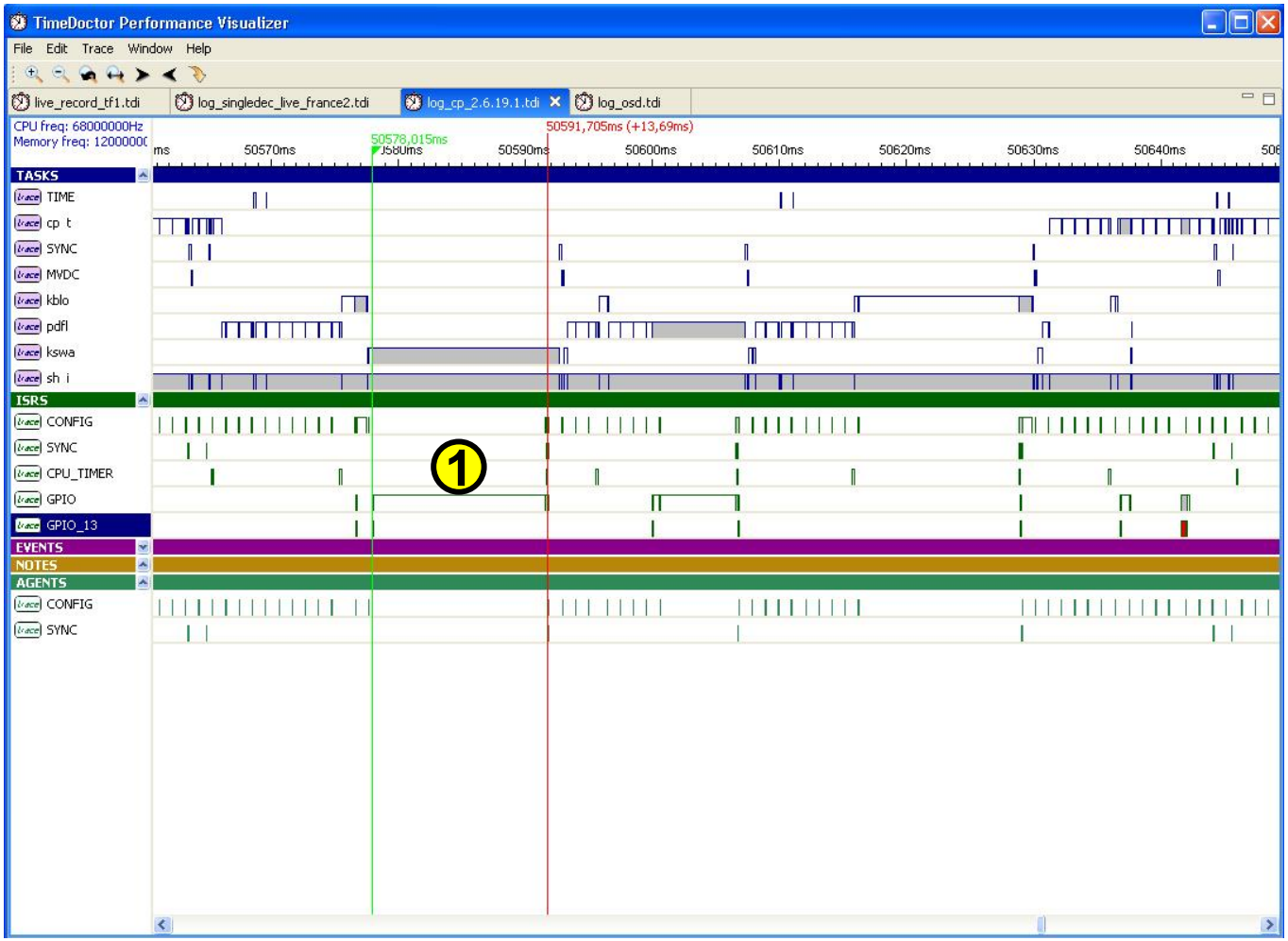
- The IDE driver was developped by a third party
- Basic testing was performed by the subcontractor:
  - HDD recognition was OK
  - Data transfer was OK
  - Integrity of the data was OK
- BUT, after integration in the system this one was not working properly (freezes, hick-ups, crashes)
- Let’s look at how TimeDoctor let us understand the root cause

First issue: HDD access (2/3)



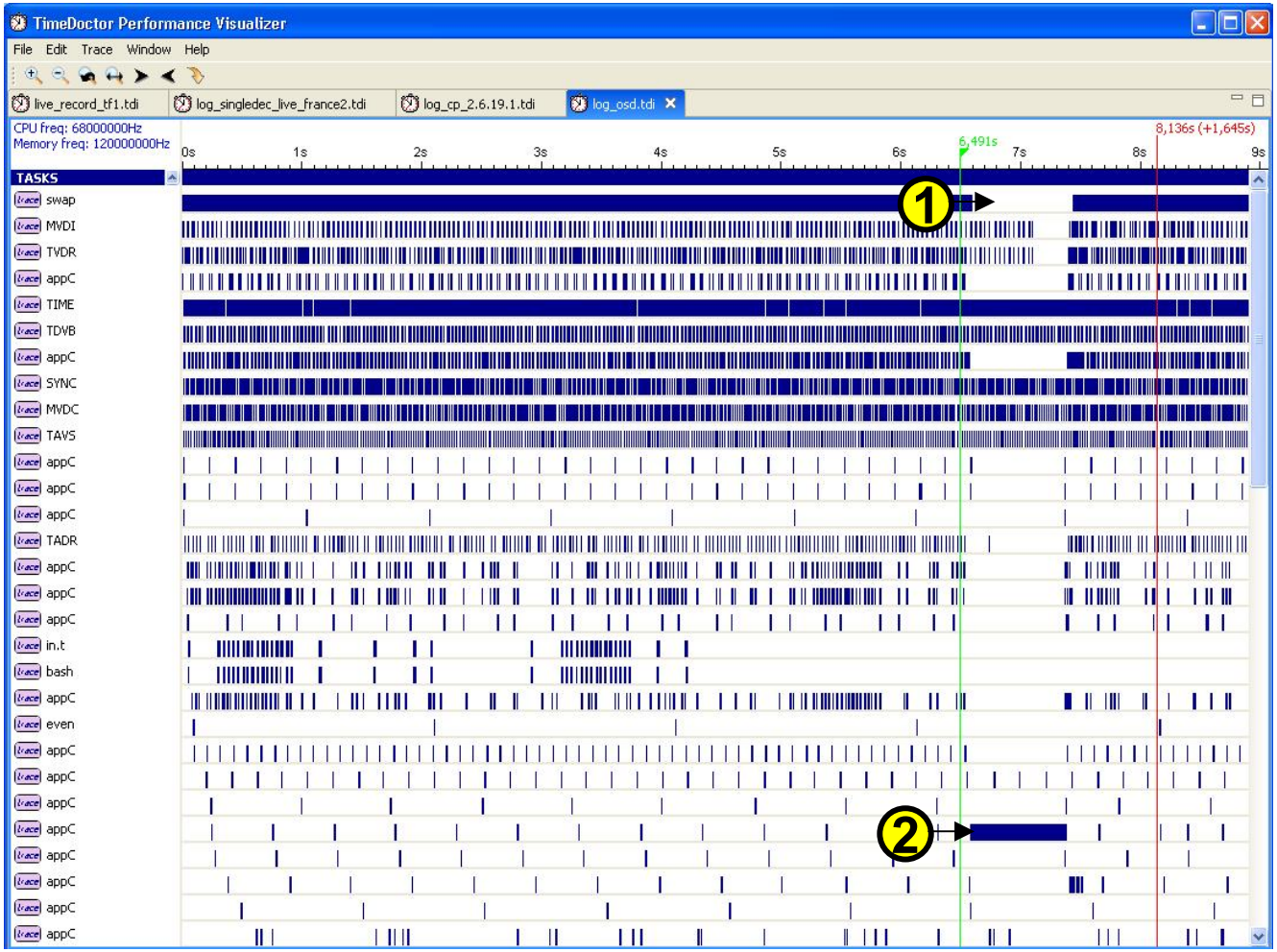
- This test is a simple copy of a file from the HDD to another file on the HDD
- (1) Look at CPU_TIMER interrupt (=kernel tick), you can detect hick-up in the rate of this ISR

First issue: HDD access (3/3)



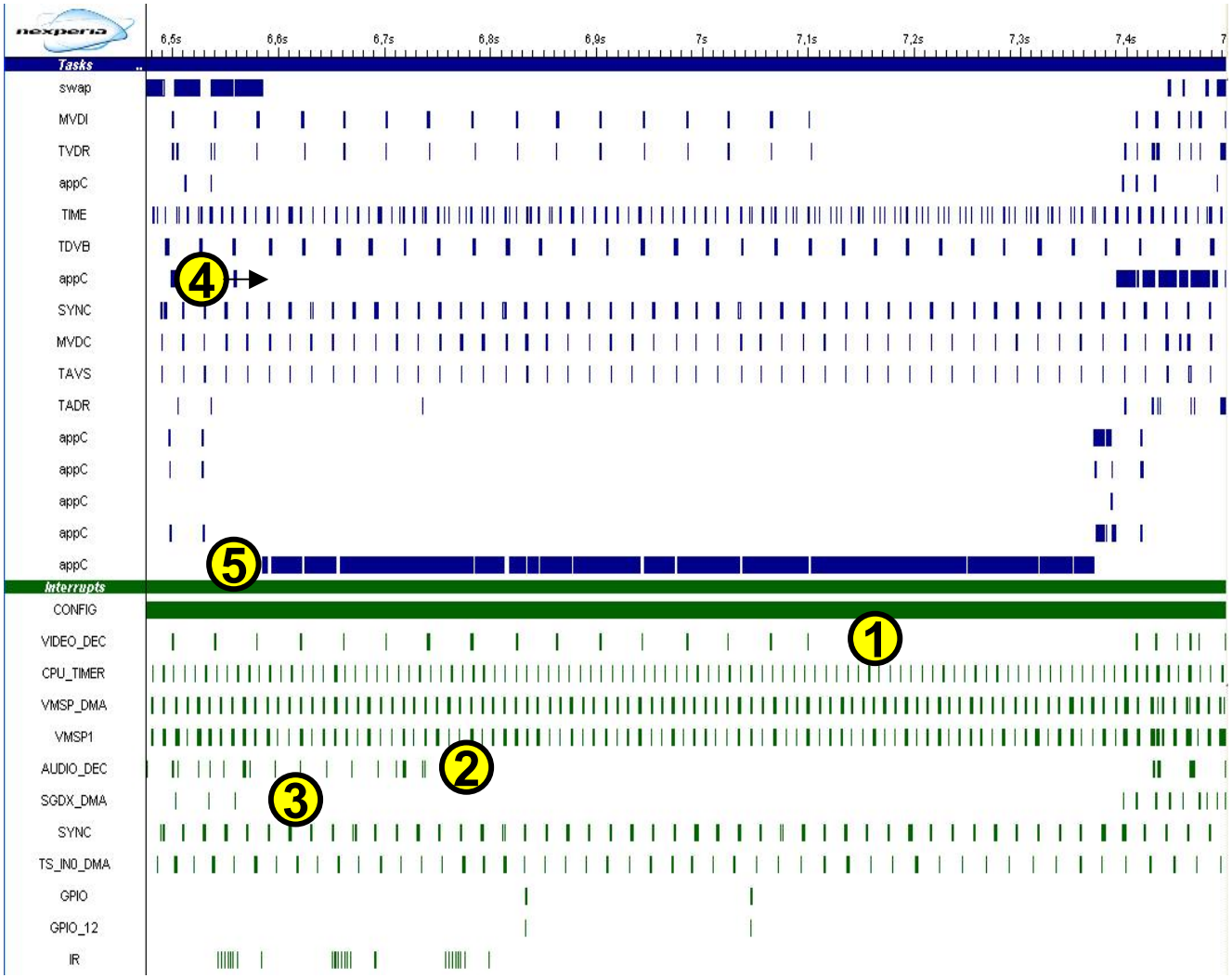
- (1) CPU is blocked in an interrupt for ~14 ms
- The root cause was an IDE bus conflict due to a wrong ordering of DMA requests and register accesses

Second issue: video freezes during execution (1/2)



- (1) Hum, look at the big hick-up in swap thread, (represents IDLE time)
- (2) One appC thread is blocking the CPU for almost 1s

Second issue: video freezes during execution (2/2)



- (1)(2)The video and audio decoders are starving
- (3) Because the demultiplexer stops feeding them
- (4) Because the thread that feeds the demux is blocked although declared as SCHED_FIFO
- (5) The root cause was that the priority of this thread was wrong

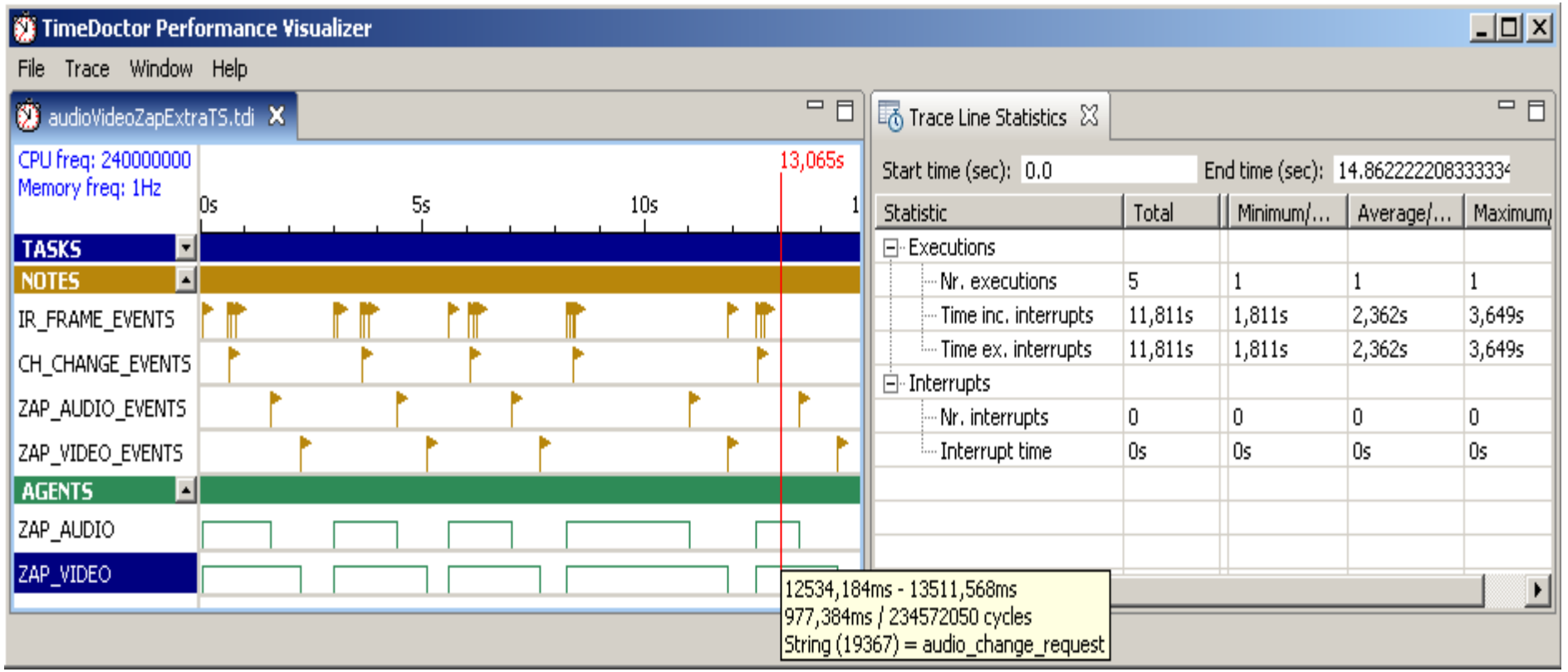
Summary

- TimeDoctor helps visualize the real time behavior of a complex application to detect and analyze issues
- Combine with expert knowledge of the system-under-test it gives precious hints to help isolate the root cause, thus saving considerable debug time

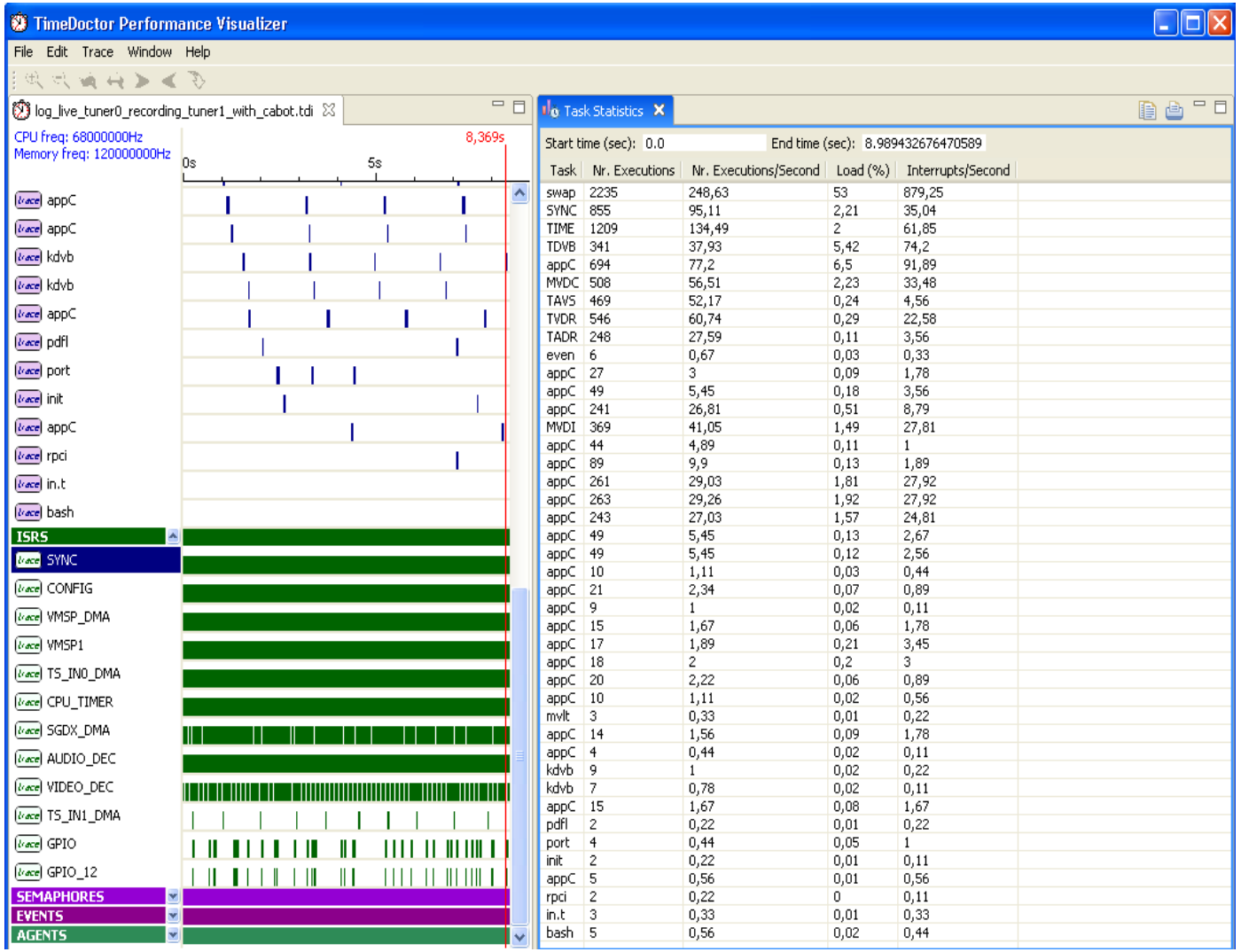
Performance analyzing and monitoring

Zapping time

- Measured on transitions between R1/R2 transponders (BFM / Arte)
  Average 2.4s, peak 3.6s for A/V program change



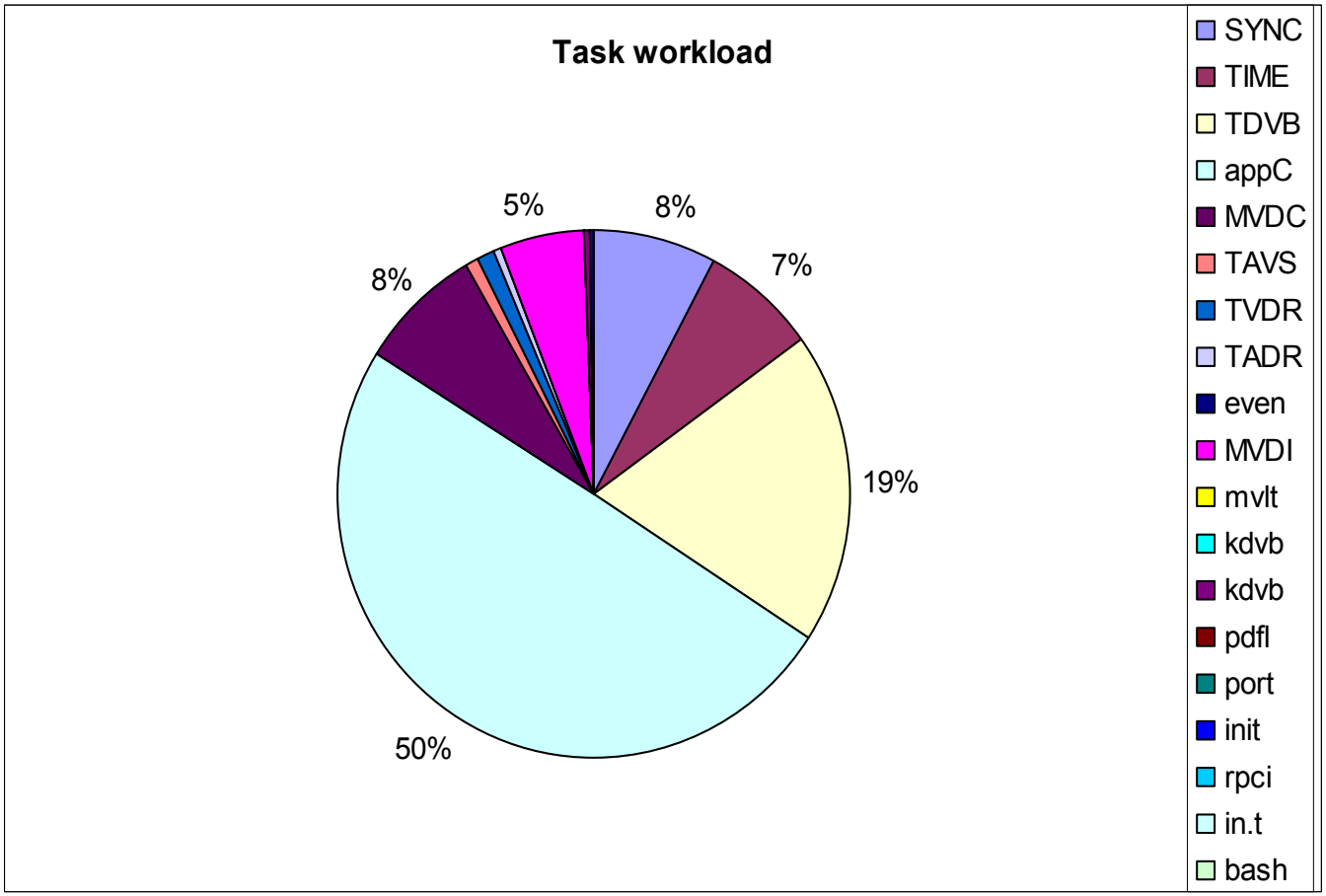
CPU budget (1/2)



Task	Nr. Executions	Nr. Executions/Second	Load (%)	Interrupts/Second
swap	2235	248,63	53	879,25
SYNC	855	95,11	2,21	35,04
TIME	1209	134,49	2	61,85
TDVB	341	37,93	5,42	74,2
appC	694	77,2	6,5	91,89
MVDC	508	56,51	2,23	33,48
TAVS	469	52,17	0,24	4,56
TVDR	546	60,74	0,29	22,58
TADR	248	27,59	0,11	3,56
even	6	0,67	0,03	0,33
appC	27	3	0,09	1,78
appC	49	5,45	0,18	3,56
appC	241	26,81	0,51	8,79
MVDI	369	41,05	1,49	27,81
appC	44	4,89	0,11	1
appC	89	9,9	0,13	1,89
appC	261	29,03	1,81	27,92
appC	263	29,26	1,92	27,92
appC	243	27,03	1,57	24,81
appC	49	5,45	0,13	2,67
appC	49	5,45	0,12	2,56
appC	10	1,11	0,03	0,44
appC	21	2,34	0,07	0,89
appC	9	1	0,02	0,11
appC	15	1,67	0,06	1,78
appC	17	1,89	0,21	3,45
appC	18	2	0,2	3
appC	20	2,22	0,06	0,89
appC	10	1,11	0,02	0,56
mvlt	3	0,33	0,01	0,22
appC	14	1,56	0,09	1,78
appC	4	0,44	0,02	0,11
kdvb	9	1	0,02	0,22
kdvb	7	0,78	0,02	0,11
appC	15	1,67	0,08	1,67
pdfl	2	0,22	0,01	0,22
port	4	0,44	0,05	1
init	2	0,22	0,01	0,11
appC	5	0,56	0,01	0,56
rpci	2	0,22	0	0,11
in.t	3	0,33	0,01	0,33
bash	5	0,56	0,02	0,44

- Use the «Statistics » tab for tasks (/ISRs) based profiling
- In this use-case, you can easily check that the CPU is 53% IDLE
- The results can be exported and processed with external tools for reporting

CPU budget (2/2)



- 50 % is consumed by the application. We should check this with our subcontractor ;-)
- 19 % is consumed by a kernel thread which is doing stream copies into LinuxDVB: decision has been taken to replace this by MMAP'ed HW buffers

Summary

- TimeDoctor can also be used to measure and analyse specific performances (zapping time, boot time,…)
- Helps you check the CPU usage is in line with your predictions or requirements and do a first level profiling
  - Helps you focus your optimization efforts
- Facilitates the generation of performance reports that can be easily automated as part of the build process
  - CPU consumption per thread, ISR

Conclusion

Conclusion

- TimeDoctor offers a convenient way to visualize and analyze the real time behaviour of embedded systems
- It shortens the debug cycle of complex issues by helping isolate most probable root causes
- It helps understand where you CPU cycles go and why
- The embedded bit has a very low overhead, is small and easy to port to virtually any OS
- It’s easy to learn and use
- It’s available now on SourceForge (Eclipse Public License)
- Yet it’s not the panacea, but just another tool in the embedded developer’s toolbox

Future work

- More meaningfull names for tasks
- Release the Linux driver and kernel patches
- Hook to code profilers
- Automation for use in context of automated non regression tests

Credits and acknowledgements

- I would like to thank the following colleagues for their valuable contributions
- David Legendre
- Pierre Le Pifre
- Alexis Watine
- STB220 DVR team
  - IFA2007 multi-room DVR demo

  See it in the showroom

  … although you'll not see much ;-(

Questions?