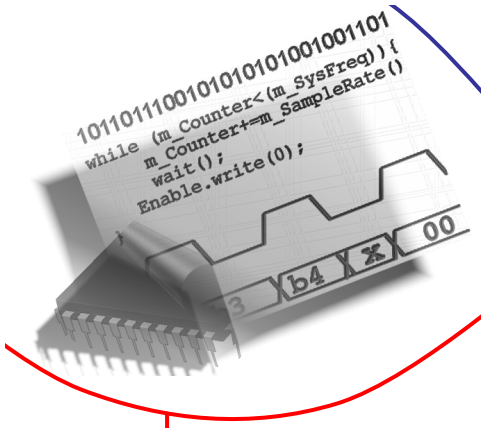


Ilia Oussorov Yuri Fonin

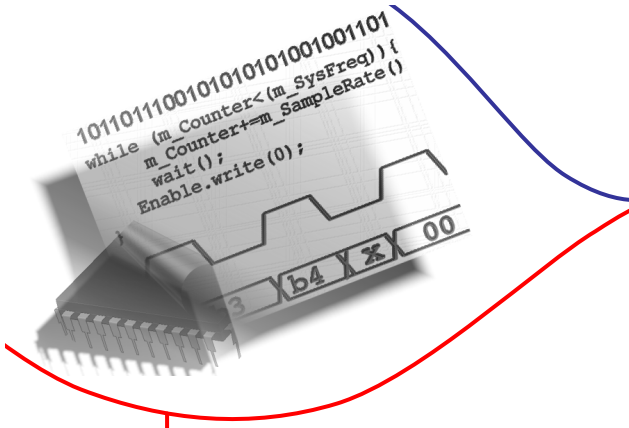
Development of the Embedded Software

(including Operating System) for SoCs in SystemC v. 2

The 10. European SystemC Users Group Meeting,
Munich, Germany

October 12th, 2004

Source code to this talk can be downloaded on our web site
(under terms of the Gnu Public License)



Embedded software with SystemC?

Embedded software is:

- -Application software
- -Device drivers
- -Operating system

SystemC levels for SW: (uTF-application)
(TLM – simple drivers)

Disagree?
TLM Speed is
100-1000x faster then RTL

Today's SoCs: RISC+DSP+MMU+Bus+Periph+ HW accelerators

RTL ~100-1KHz

SystemC TLM ~10-100K (cycle accurate)

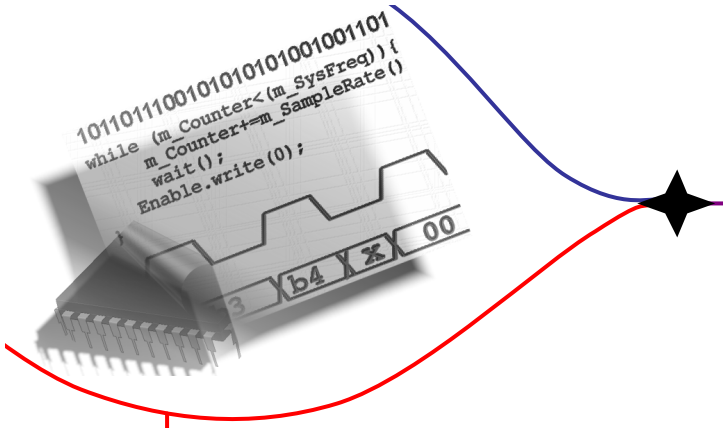
Future SoCs:

Many processors, communication networks, periphs, HW accelerators..

Today software developers need >1Mcyc/sec (better >10MCyc/sec)

Empty clocked SC_METHOD <0.7MCyc/sec (P4-2.6,512MB,VS2003–Ox)

Our experience - embedded in SystemC real life one ISS 30K-100K



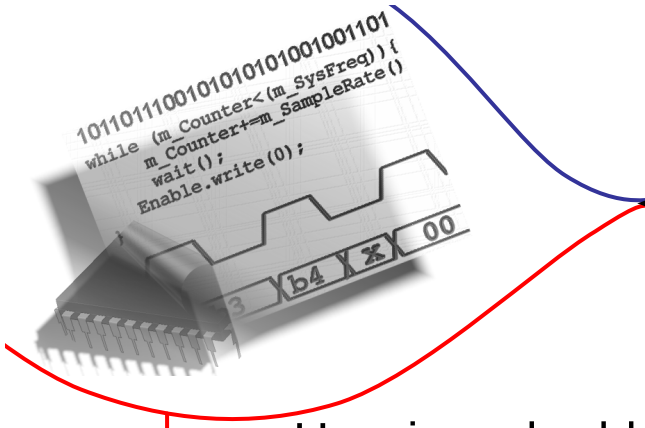
State of the art

Mr. Kunkel (VP of engineering Synopsys) : „... the system is more and more software these days. So the next thing you need to is to integrate the hardware into the software“

Mr. McCafferty (president of Beach Solutions) : “..We’ll go to TI and they’ll have one-to-one hardware to software engineer. We’ll go to Nokia and they’ll have 1 hardware guy and 100 software guys.
... Hardware-software co-design is one of the bigger challenges ahead. “

Mr. Rizzatti, U.S. general manager of EVE: „..They only begin to test the software on engineering samples because traditional tools would be horrendously slow that a developer would need. We see a trend to provide software tools that you can use to model an entire system using a very high level of abstraction. ... When you test software between the software and the hardware, those tools don’t cut it.

Source: ElectronicNews, 9/24/2004
http://www.reed-electronics.com/electronicnews/article/CA455657?pubdate=09%2F19%2F2004



State of the art (continued)

How is embedded SW typically developed today?

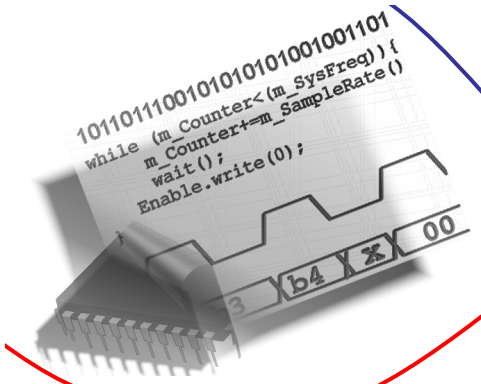
Flow 1: Wait until HW is available (FPGA/ eng. sample/ ...)

Flow 2:

- OS and device drivers – on the virtual models (cycle based or compiled - faster simulation techniques then SystemC)

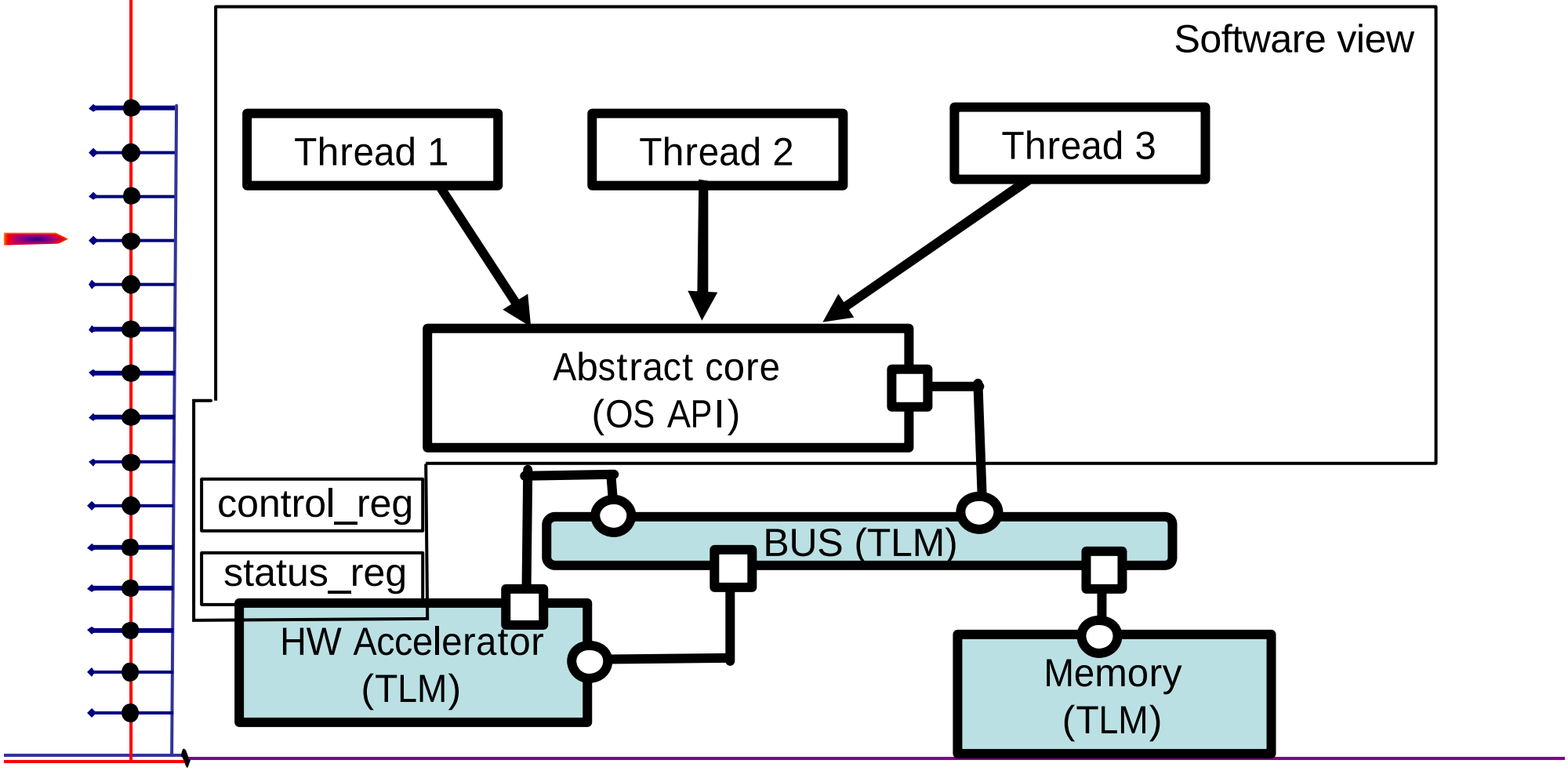
- Application – 1. Algorithms on the PC
  2. Integration – on the virtual models (cycle based or compiled - faster simulation techniques thenSystemC)

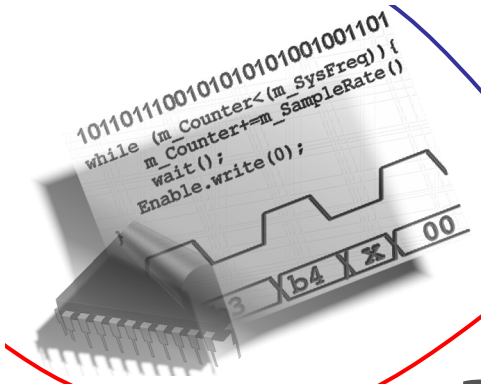
Currently SystemC seems to be outside of real life SW development



Can SystemC address embedded SW?

Suggestion: introduce intermediate abstraction level between TF and TLM - Processor Abstraction Level (PAL)





To be concrete: abstract OS

xxx.h

```
OS_MODULE(Processor_1){
        DECL_OS_THREAD(app_thread1);
        DECL_OS_THREAD(app_thread2);
        void boot_proc(void *);
        Processor_1(){
                SC_THREAD(boot_proc);
        }
}
```

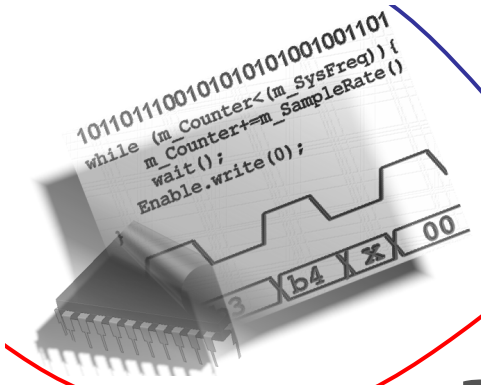
Rewrite header to remove class stuff or just use preprocessor #ifdef ON_EMBEDDED_TARGET!

xxx.cpp

```
OS_START_THREAD(Processor_1,boot_proc){
        OS_HANDLE h1,h2;
        my_param_t _params1, _params2;
        h1=Create_Thread(app_thread1, &_params1,...);
        h2=Create_Thread(app_thread2, &_params2,...);
}

OS_THREAD(Processor_1,app_thread1){
        my_param_t *Param=(my_param_t *) arg;
        do_something(..);
        Wait_Mutex( hMutex1);
        do_others(..);
        SetEvent(hEvent1);
};
```

Write ANSI C! Later compile with the C compiler for your embedded target!



To be concrete: hardware accelerator driver

xxx.h

```
OS_MODULE(Processor_1){
        sc_export<bus_port_if> bus_port;
                DECL_OS_THREAD(acc_control_thread);
                void boot_proc(void *);
                Processor_1(){
                        SC_THREAD(boot_proc);
                }
        }
```

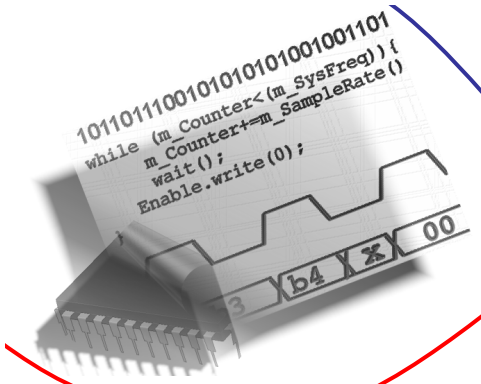
Rewrite header to remove class stuff or just use preprocessor #ifdef ON_EMBEDDED_TARGET!

xxx.cpp

```
OS_START_THREAD(Processor_1,boot_proc){
        OS_HANDLE h1;
        my_param_t _params1; params1.a=;
        h1=Create_Thread(acc_control_thread, &_params1,...);
}

                OS_THREAD(acc_control_thread,app_thread1){
                        unsigned int _status;
                        Wait_Event(evStartMP4);
                        WRITE4B(ACC_DATA_ADDR_REG,DATA_BUFFER_START);
                        WRITE4B(ACC_CONTROL_ADDR, ACC_START);
                        READ4B(ACC_STATUS_REG, _ status);
                        while(_ status!= ACC_TAKEN_DATA)
                                READ4B(ACC_STATUS_REG, _ status);
                };
```

Write ANSI C! Later compile with the C compiler for your embedded target!



Data flow between abstract processors and HW accelerators

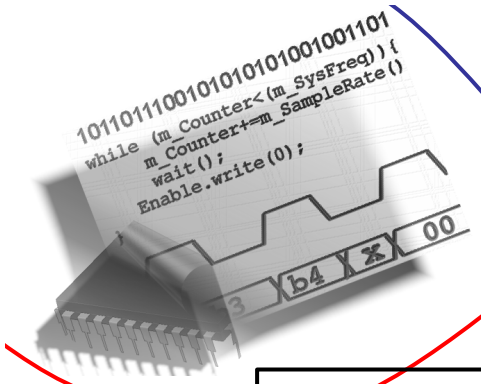
Abstract processors refer to the simulation host memory
HW accelerators refer to TLM memory models

Synchronization of the data buffers must be provided

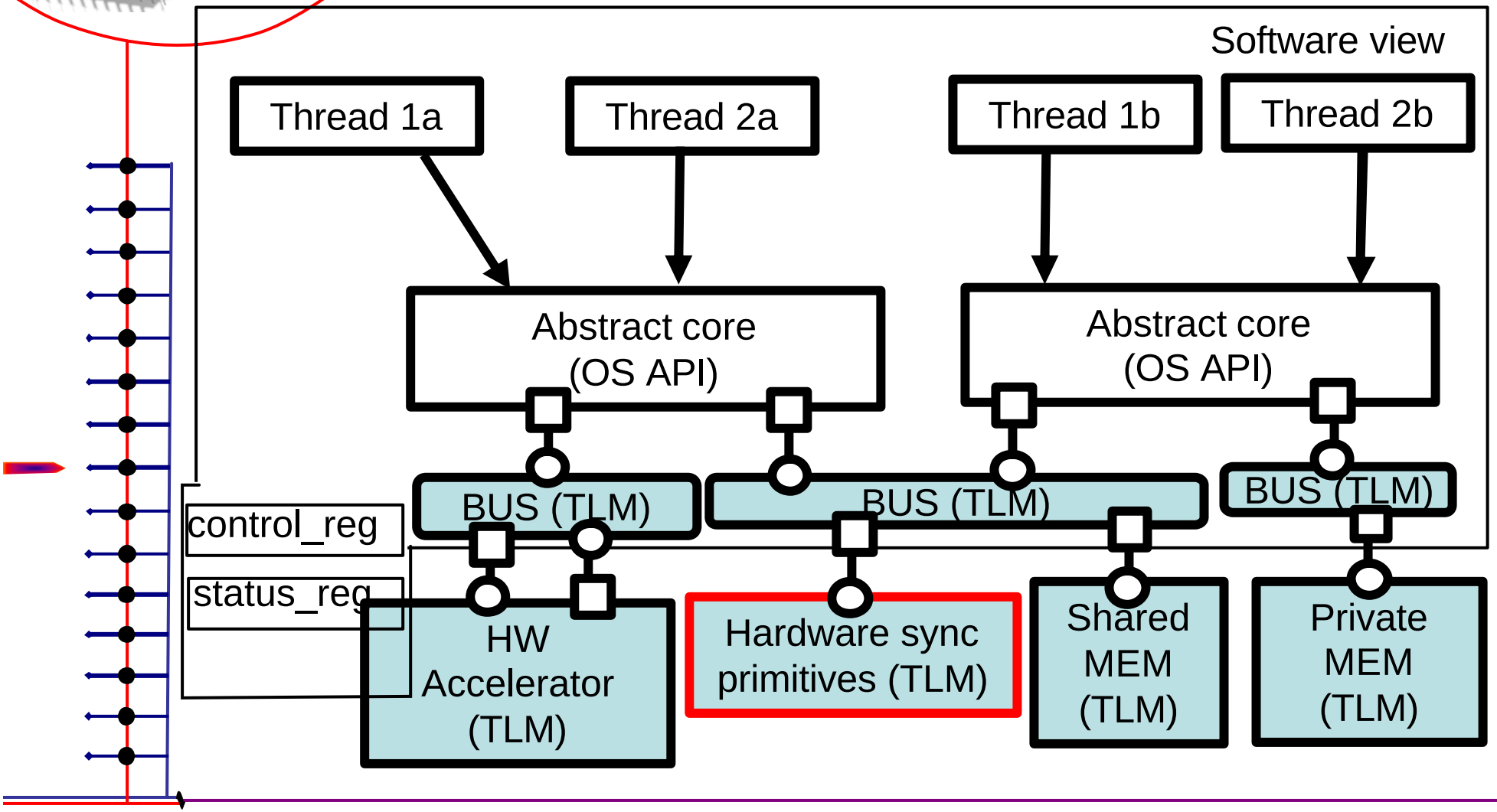
How?

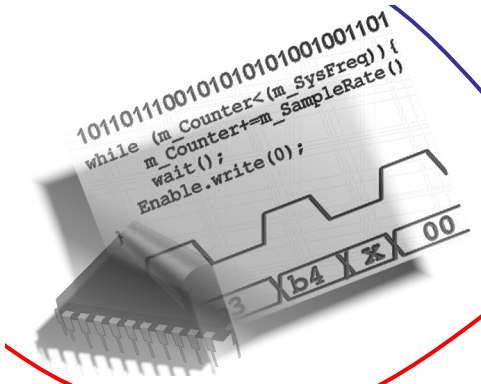
- If you develop your embedded libc yourself:
  - You will have 2 malloc‘s with different signatures:
    - for private memory (simulation host libc)
    - for shared memory (embedded libc: would allocates buffers inside of the host memory space of the shared memory TLM model and manage this memory)
- If you don‘t (want to/must) develop embedded libc or have no privatememory:
  - Copy data buffers by direct bus accesses before the syncronisation of the processor and hw accelerator (small overhead)

Multiprocessor systems



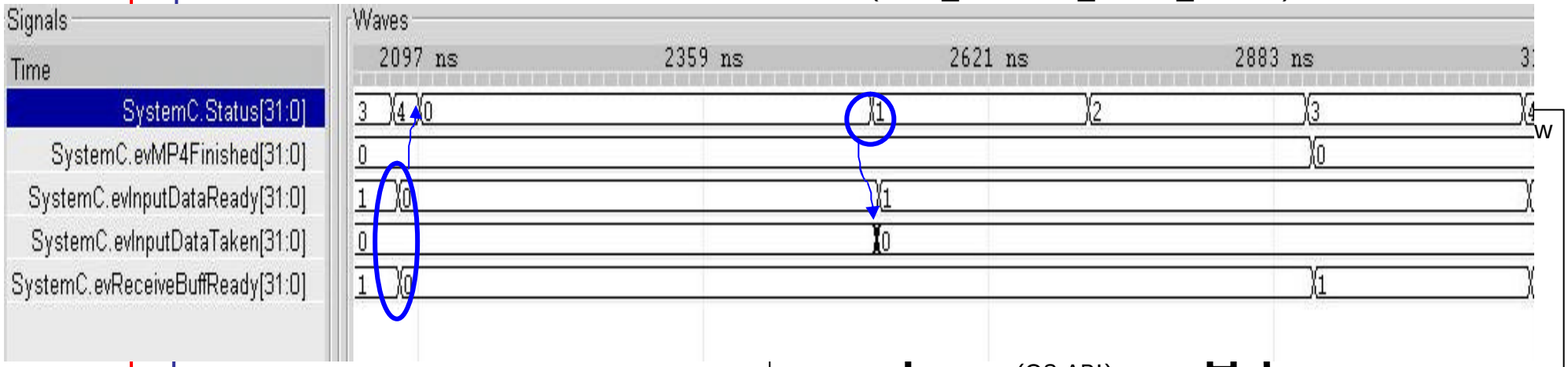
Remap synchronization primitives to the hardware accesses

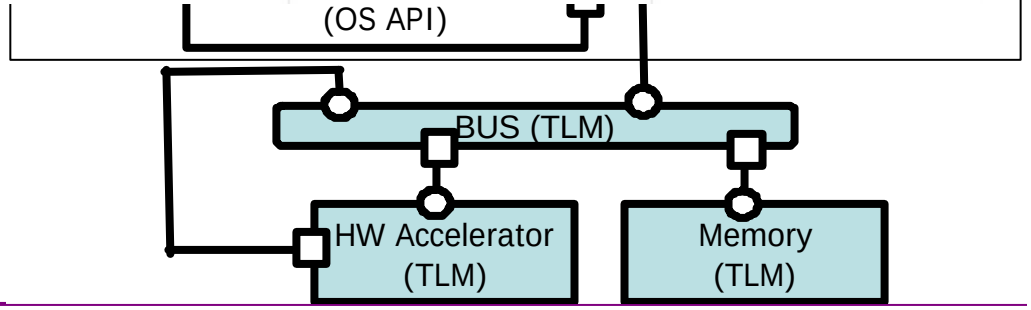


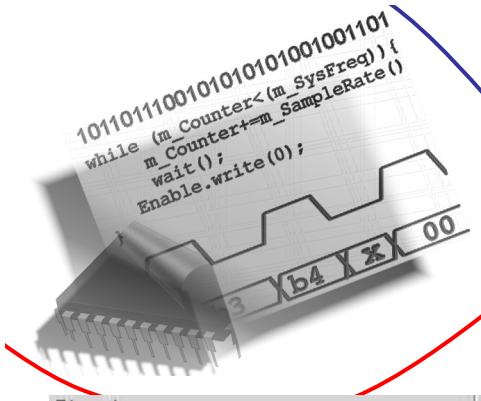


Multithreaded software debugging

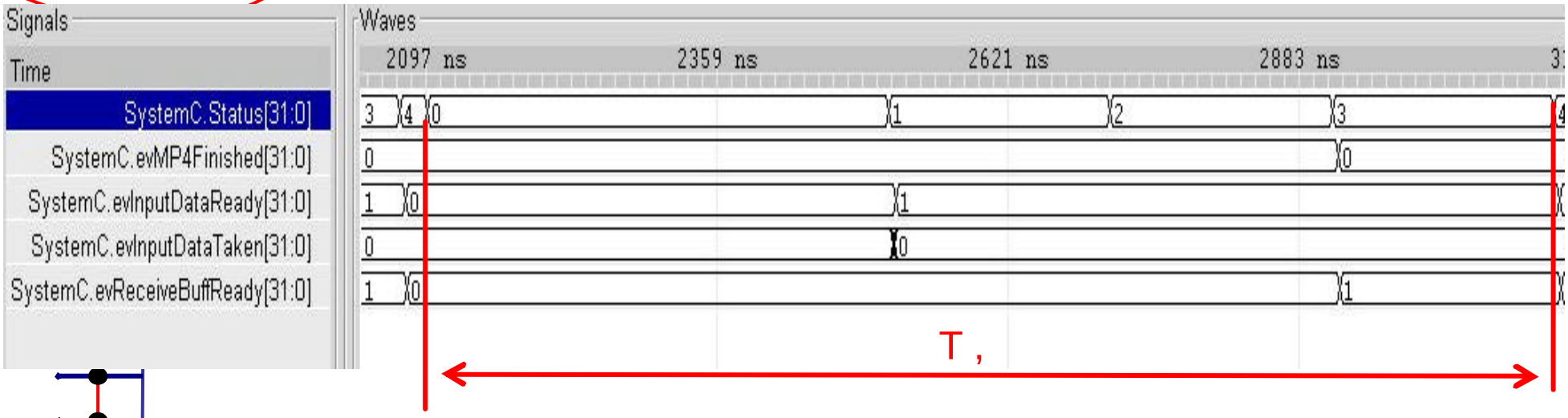
```
OS_THREAD(Processor_1,acc_control_thread){
        unsigned int _status;
        while (...){
                Wait_Event(evInpuDataReady);
                Wait_Event(evReceiveBuffReady);
                WRITE4B(ACC_DATA_ADDR_REG,DATA_BUFFER_STAR
                WRITE4B(ACC_CONTROL_ADDR, ACC_START);
                READ4B(ACC_STATUS_REG, _ status);
```







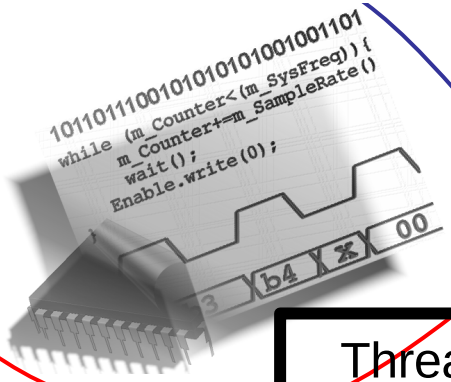
Requirement analysis

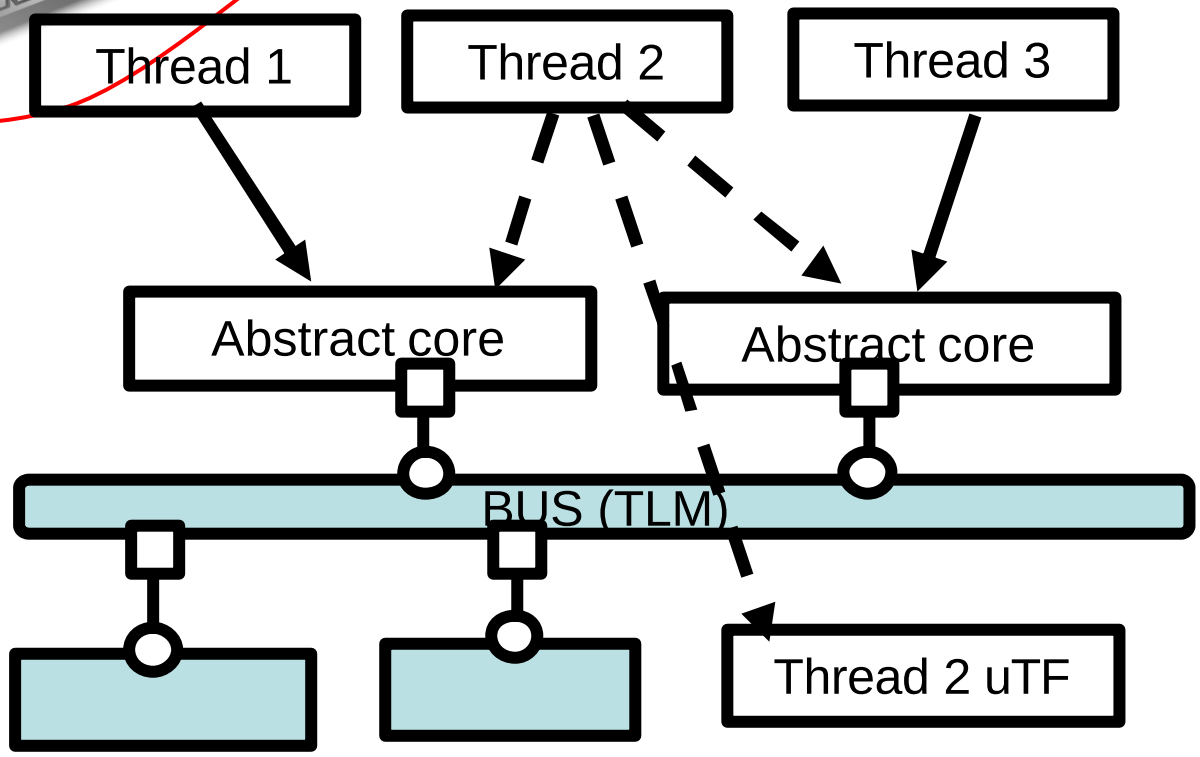


(Time, accelerator processes one block)

- Accelerator will pause, if the software executes slower then T
- Software optimization could be useless, if software executes faster then T
- Input buffers must be filled with appropriate frequency
- Output buffers must be taken with appropriate frequency

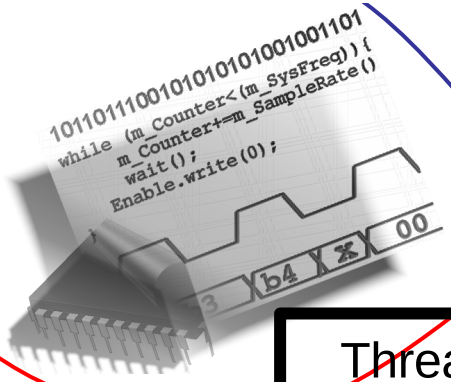
Partitioning

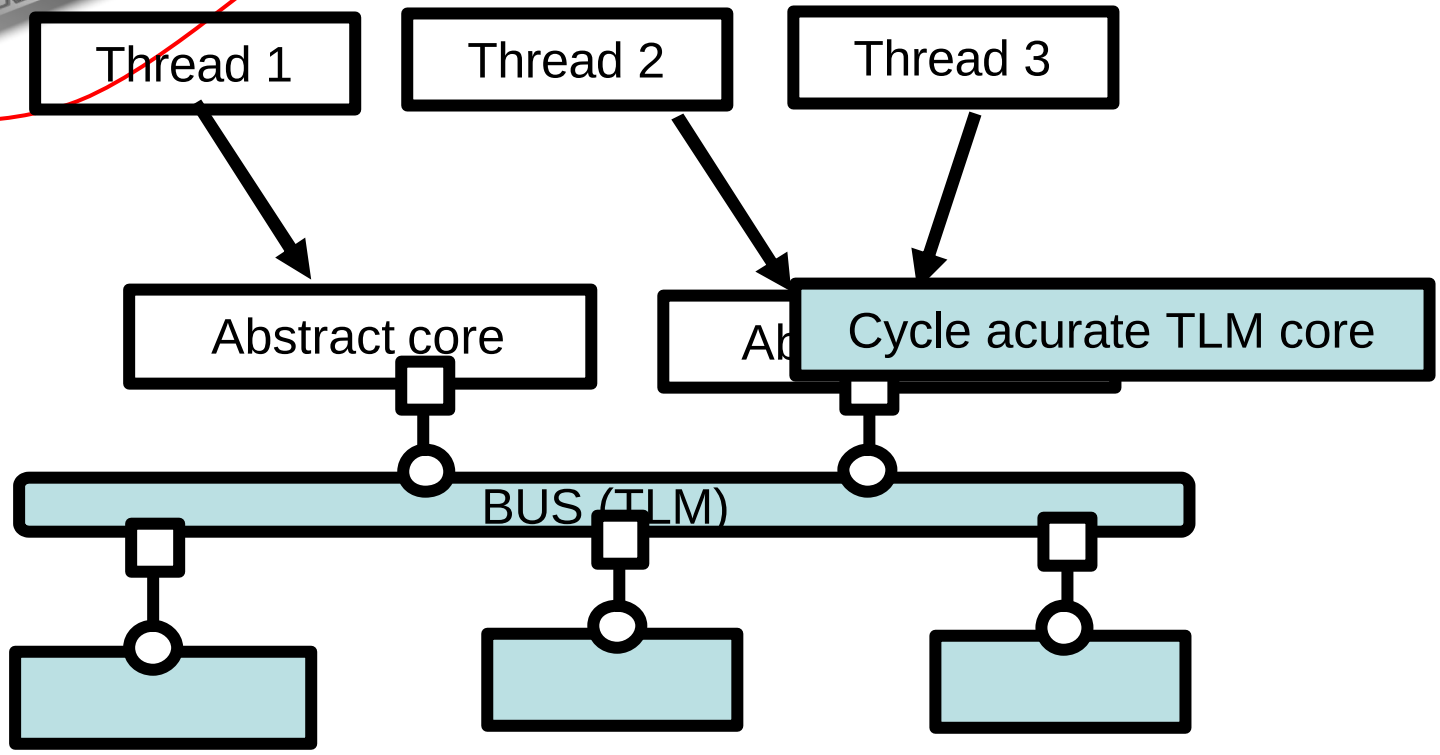




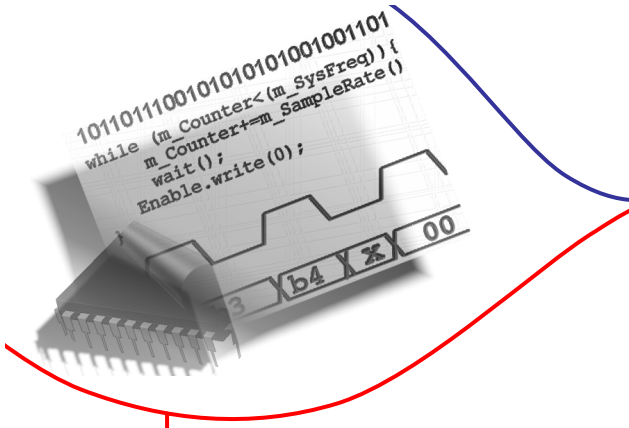
- Thread allocation

Profiling





Replace just one (some) core(s), compile the same software and profile

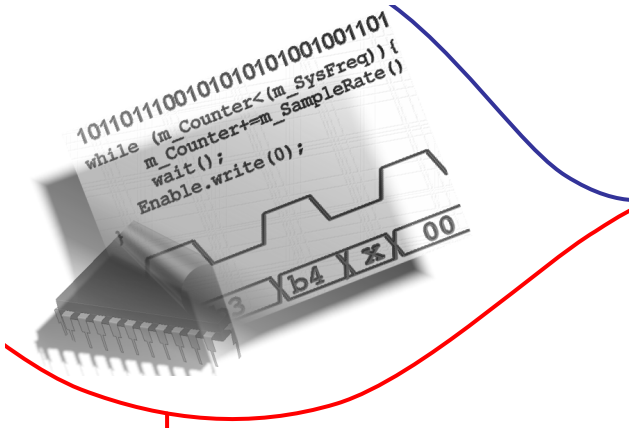


Summary

- Much faster simulation, because
  1. ISS Models are not simulated
  2. Less simulated cycles (cycles only consumed by bus transfers)
- Extensive regression tests are possible
- Requirements to the timing can be tested

- Whole SoC software including application software, OS, drivers can be simulated

  (except small processor dependent part, like boot routines, thread switching implementations..., which can be developed only on the ISS)



Thank you for your attention

Q&A

Visit our web page!
Try the OS library and examples!
Contact us, we will be happy to work with you!