System-Level Design (and Modeling for Embedded Systems)

Lecture 7 – Computation Modeling & Refinement

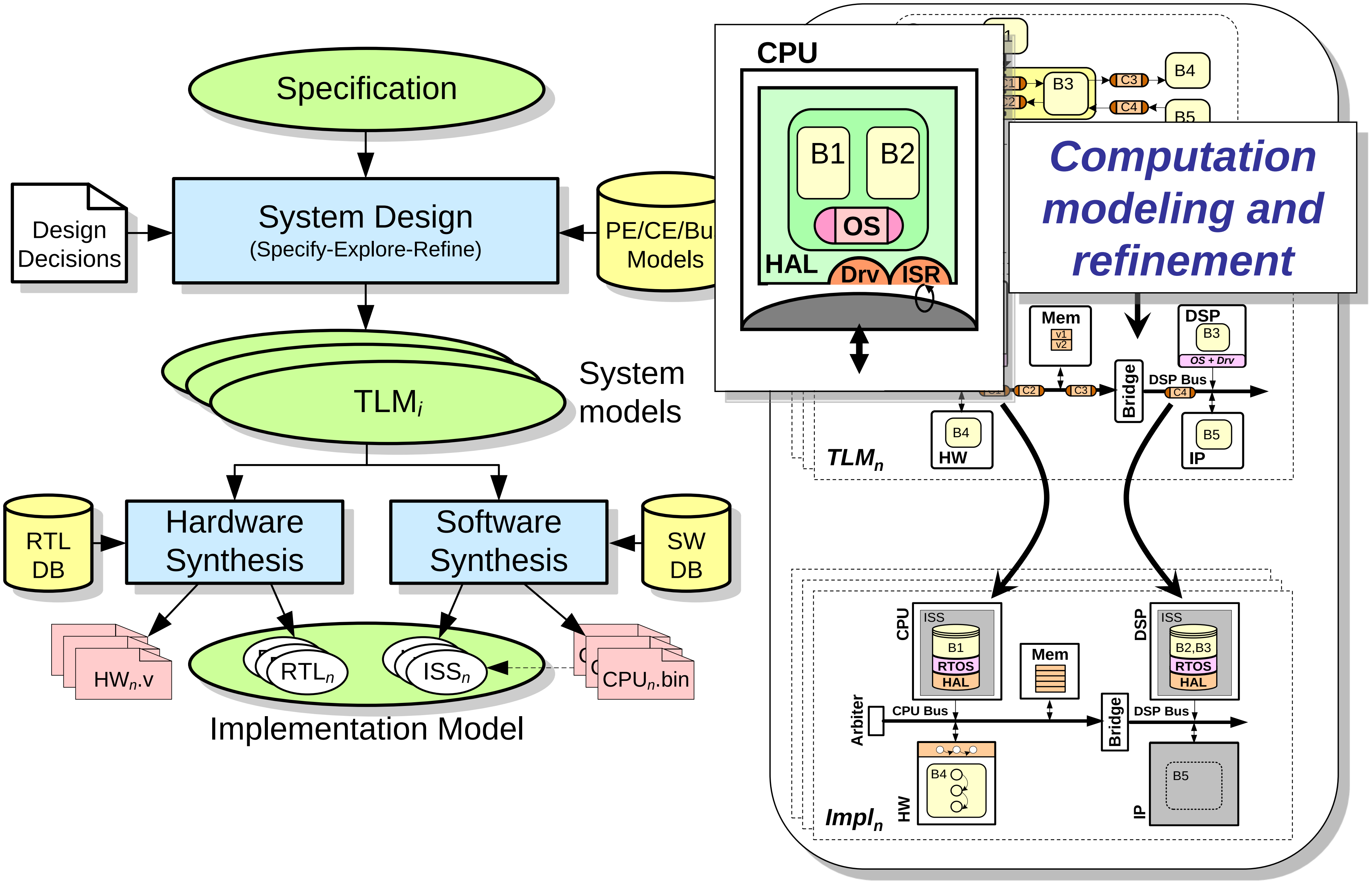
Kim Grüttner kim.gruettner@dlr.de
Henning Schlender henning.schlender@dlr.de
Jörg Walter joerg.walter@offis.de

System Evolution and Operation
German Aerospace Center (DLR)
&
Distributed Computation and Communication
OFFIS

Lecture 7: Outline

- **Processor layers**
  - Application
  - Task/OS
  - Firmware
  - Hardware
- **Processor synthesis**
  - Software synthesis
  - Hardware synthesis

System-On-Chip Environment (SCE)



Multi-Processor System-On-Chip (MPSoC)

- **Growing system complexities and sizes**
  - Heterogeneous multi-processor systems (MPSoC)
- **Increasing significance of embedded software**
  - Growing software content

➢ **System design at higher levels of abstraction**
  - Validation and analysis
  - Concurrent hardware and software development
  - Implementation synthesis

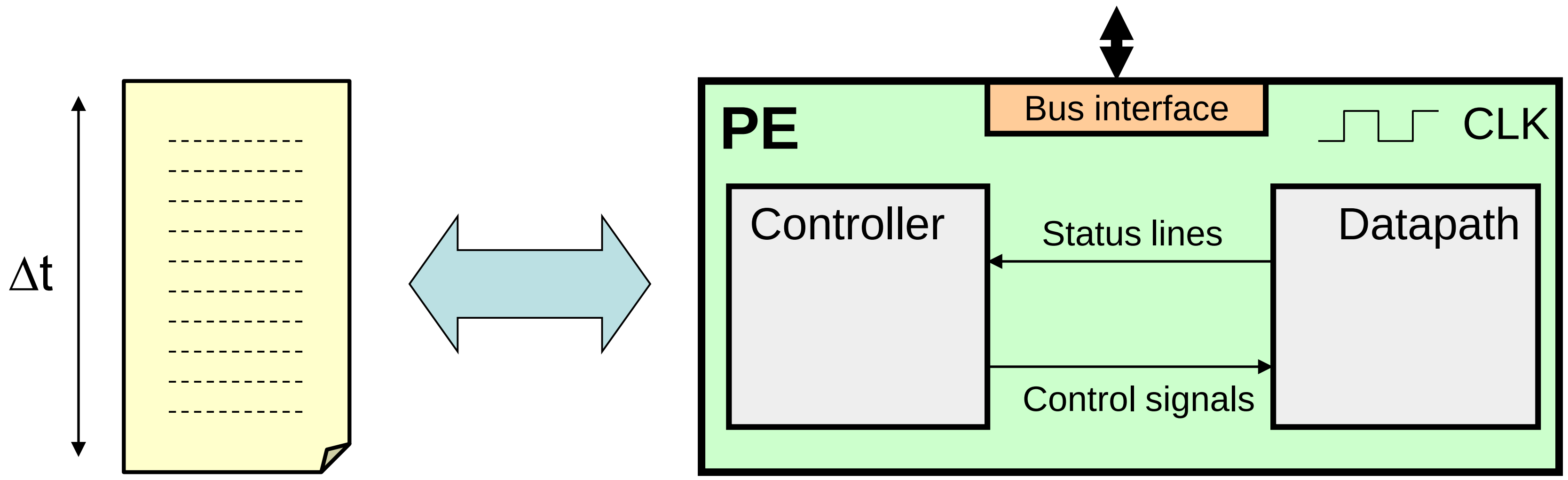
➢ **Design of embedded software and processors**
  - Large influence on system performance, power, etc.
  - Actual SW on ISS is accurate but slow

  ➢ High-level models for early and accurate feedback

  ➢ Software synthesis

General Processor Micro-Architecture

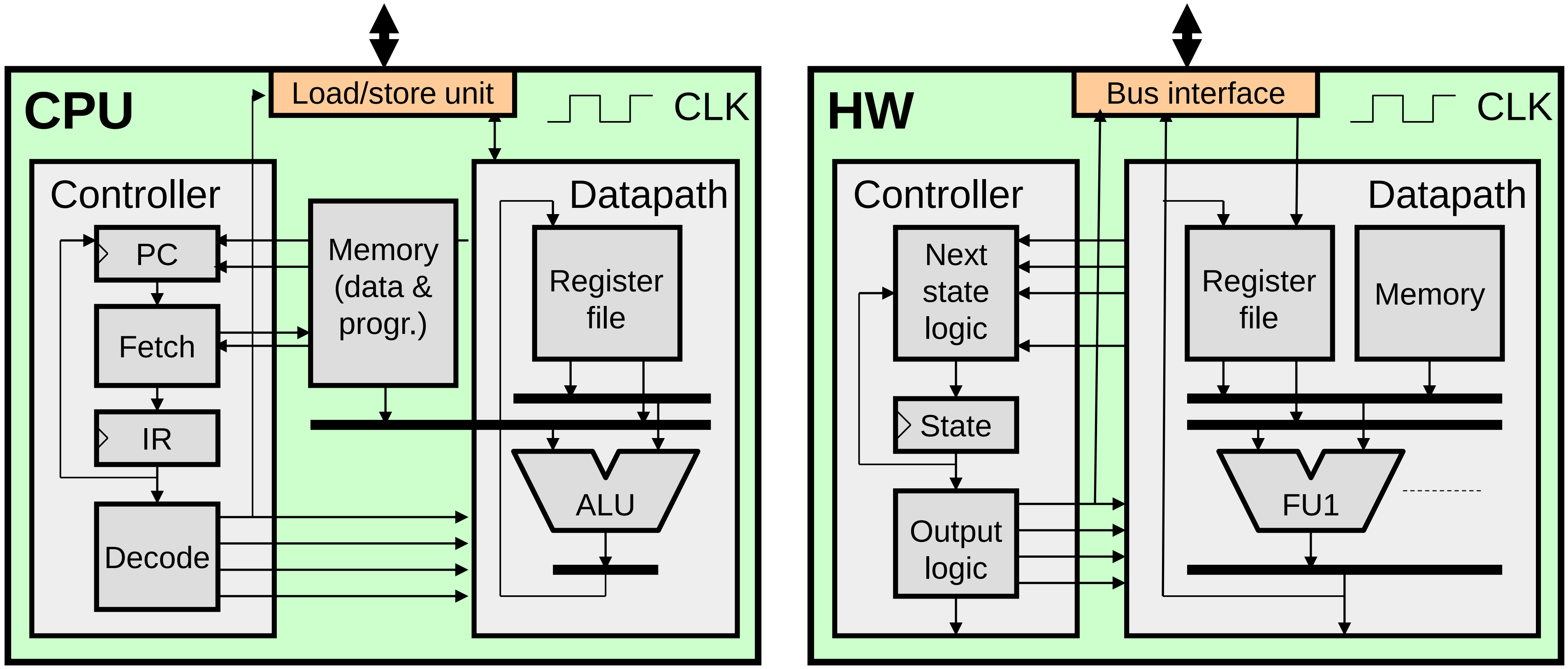
- **Basic system component is a *processor (PE)***
  - Programmable, general-purpose software processor (CPU)
  - Programmable special-purpose processor (e.g. DSPs)
  - Application-specific instruction set processor (ASIP)
  - Custom hardware processor



➢ **Functionality *and* timing**

Processor Models (1)

- **Structural RTL models**

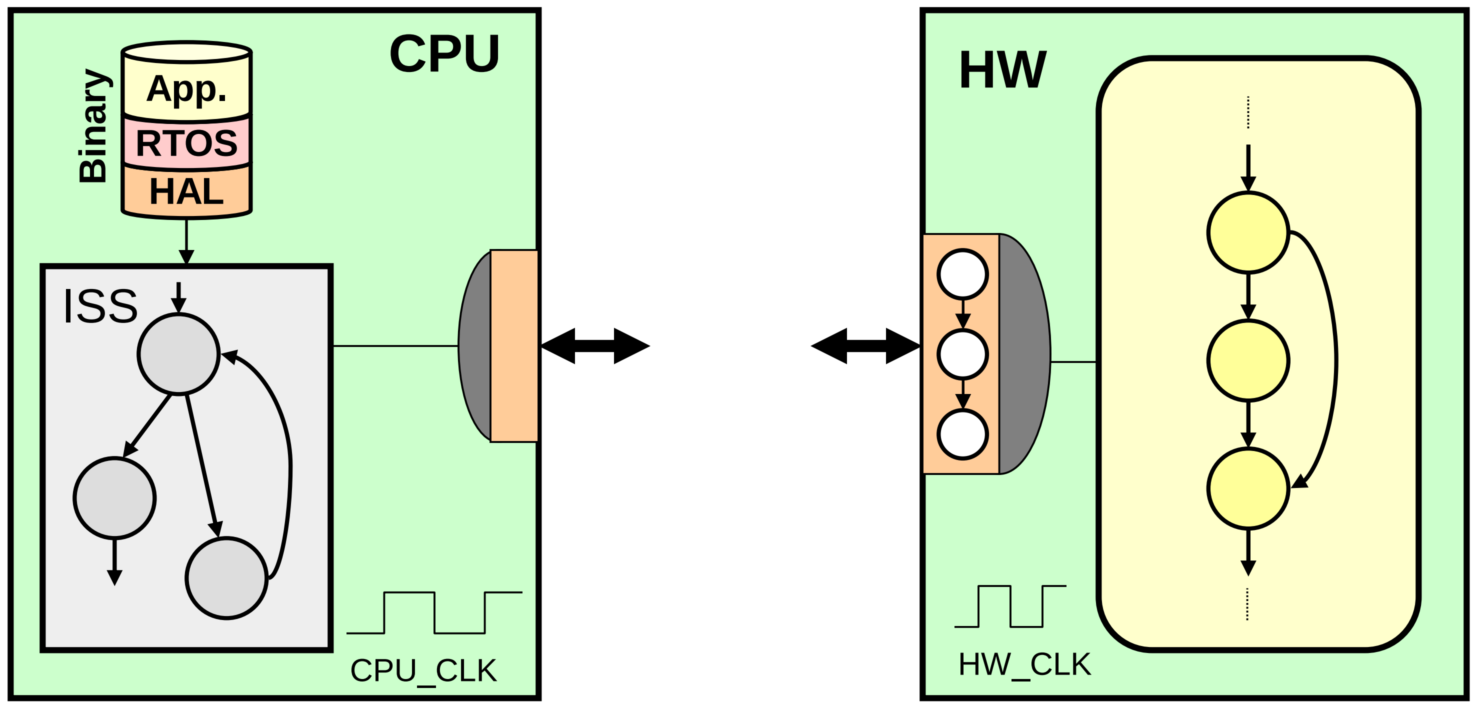


Software processor　　　　Hardware processor

➢ **Sub-cycle accurate**

Processor Models (2)

- **Behavioral RTL/IS models**

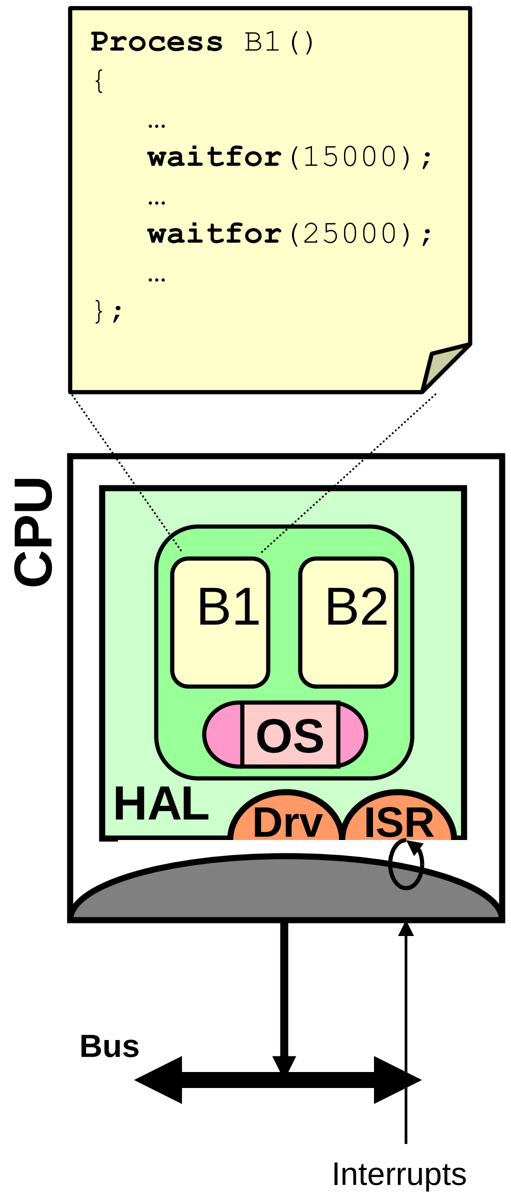


Instruction set simulation (ISS)

FSMD

➢ **Cycle accurate**

High-Level Computation Modeling



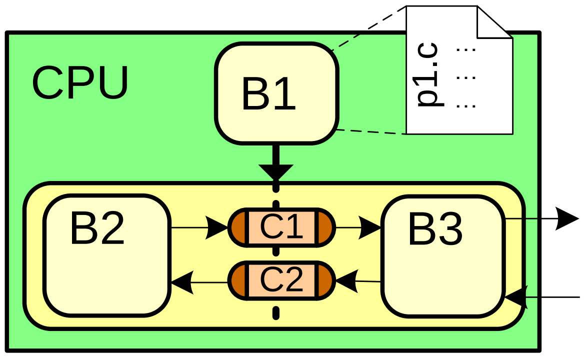
```
Process B1()
{
    …
    waitfor(15000);
    …
    waitfor(25000);
    …
};
```

- **Application modeling**
  - Native process execution (C code)
  - Back-annotated execution timing
- **Processor modeling**
  - Operating system
    - Real-time multi-tasking (RTOS model)
    - Bus drivers (C code)
  - Hardware abstraction layer (HAL)
    - Interrupt handlers
    - Media accesses
  - Processor hardware
    - Bus interfaces (I/O state machines)
    - Interrupt suspension and timing

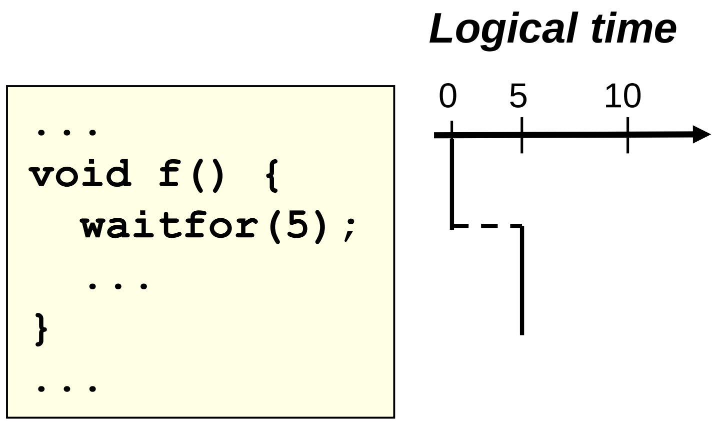
Source: G. Schirner, A. Gerstlauer, R. Doemer. “Abstract, Multifaceted Modeling of Embedded Processors for System Level Design,” ASPDAC07

Processor Model: Application Layer

- **High-level, abstract programming model**
  - Hierarchical process graph
    - ANSI C leaf processes
    - Parallel-serial composition
  - Abstract, typed inter-process communication
    - Channels
    - Shared variables



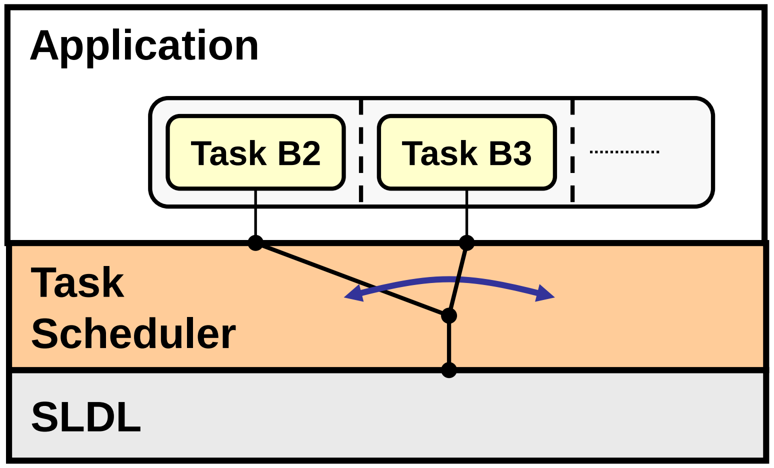
- ➢ **Timed simulation of application functionality (SLDL)**
  - Back-annotate timing
    - Estimation or measurement (trace, ISS)
    - Function or basic block level granularity
  - Execute natively on simulation host
    - Discrete event simulator
    - Fast, native compiled simulation



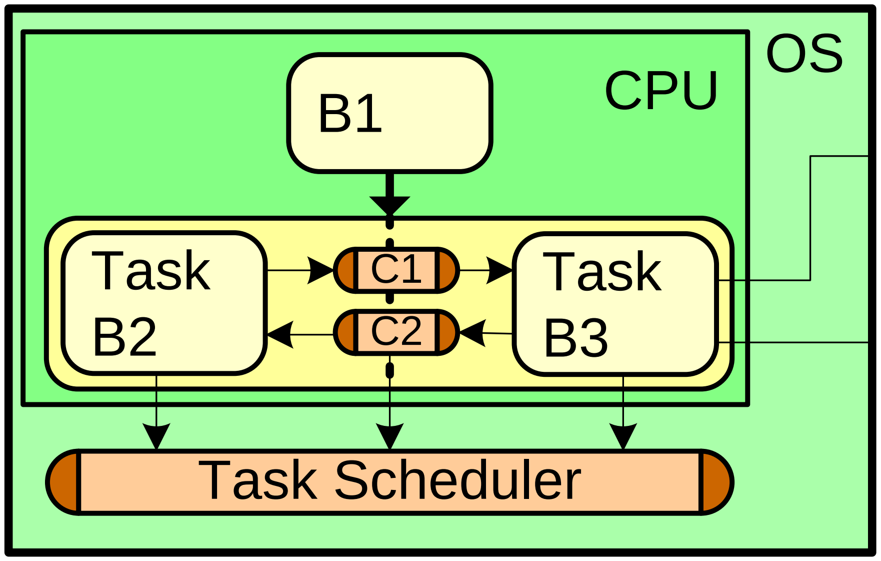
```
...
void f() {
  waitfor(5);
  ...
}
...
```

Processor Model: Task Layer

- **Scheduling**
  - Group processes into tasks
    - Static scheduling
  - Schedule tasks
    - Dynamic scheduling, multitasking
    - Preemption, interrupt handling
    - Task communication (IPC)

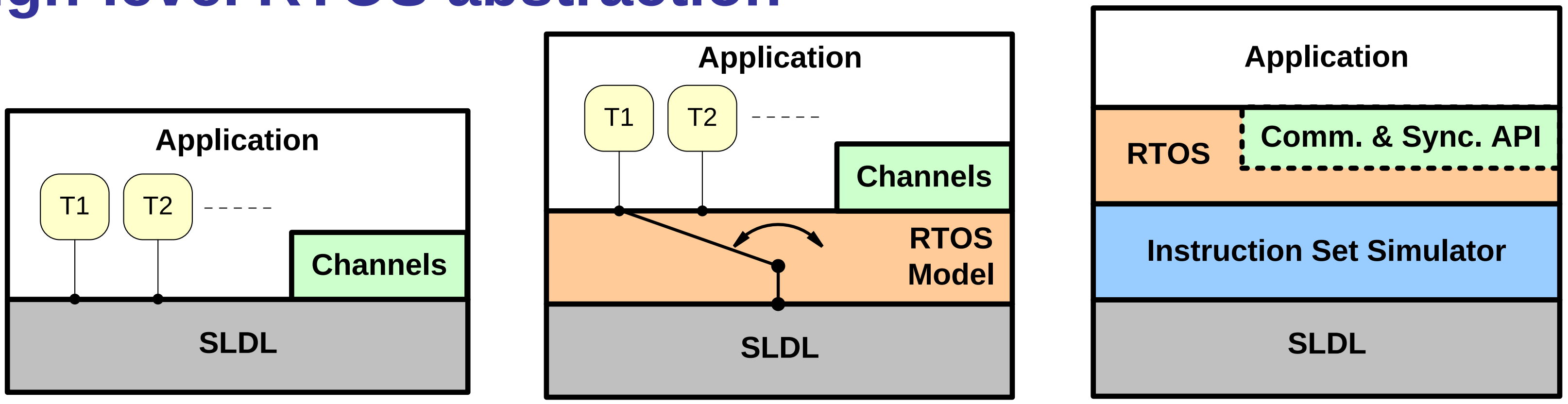


- **OS model on top of standard SLDL**
  - Wrap around SLDL primitives, replace event handling
    - Block all but active task
    - Select and dispatch tasks
  - Target-independent, canonical API
    - Task management
    - Channel communication
    - Timing and all events



OS Modeling

- **High-level RTOS abstraction**



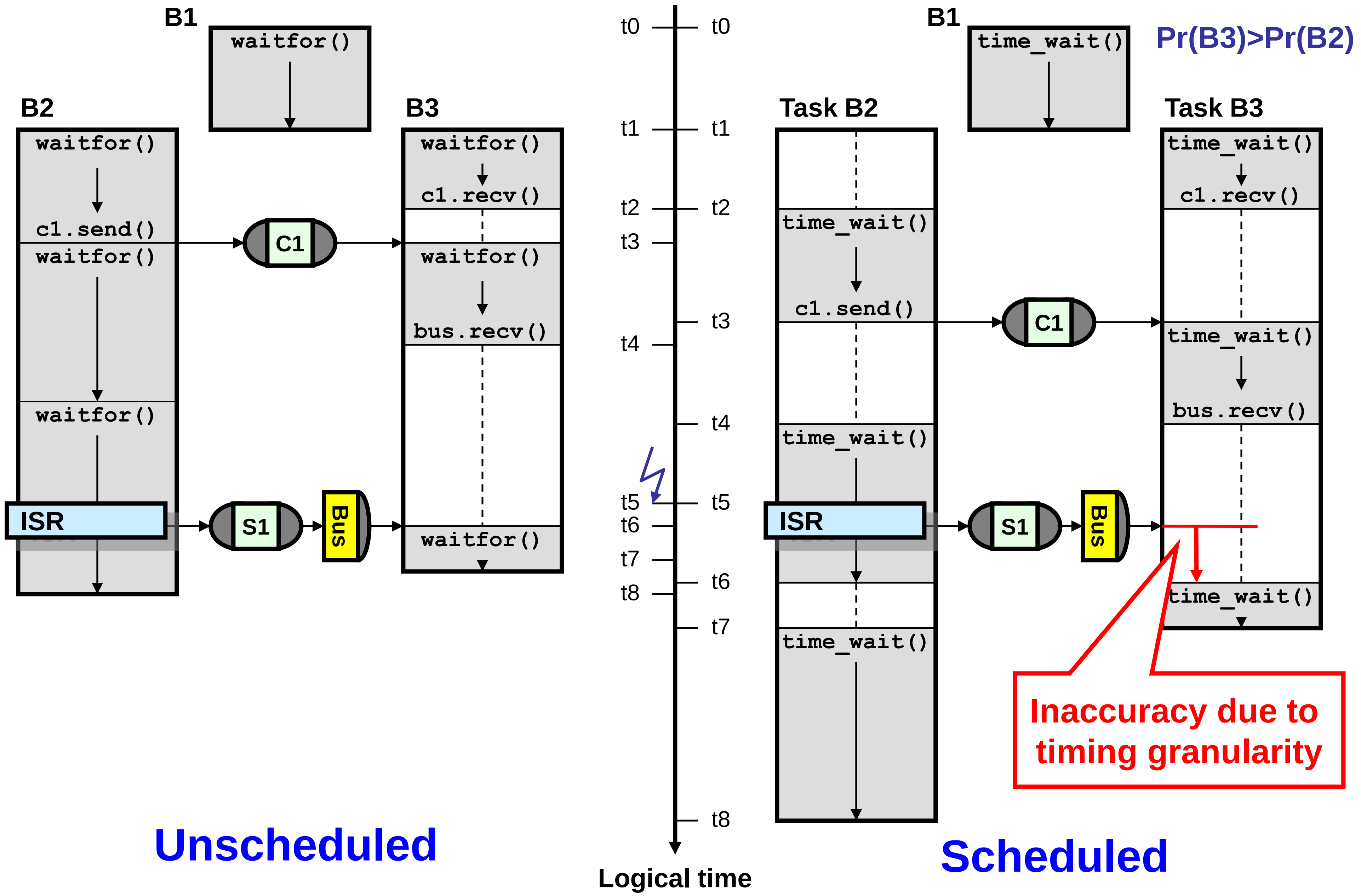
Specification *TLM* Implementation

- Specification is fast but inaccurate
  - Native execution, concurrency model
- Traditional ISS-based validation infeasible
  - Accurate but slow (esp. in multi-processor context), requires full binary

➢ Model of operating system
  - ➢ High accuracy but small overhead at early stages
  - ➢ Focus on key effects, abstract unnecessary implementation details
  - ➢ Model all concepts: Multi-tasking, scheduling, preemption, interrupts, IPC

Source: A. Gerstlauer, H. Yu, D. Gajski. "RTOS Modeling for System-Level Design," DATE03.

Simulated Dynamic Behavior



RTOS Model Implementation

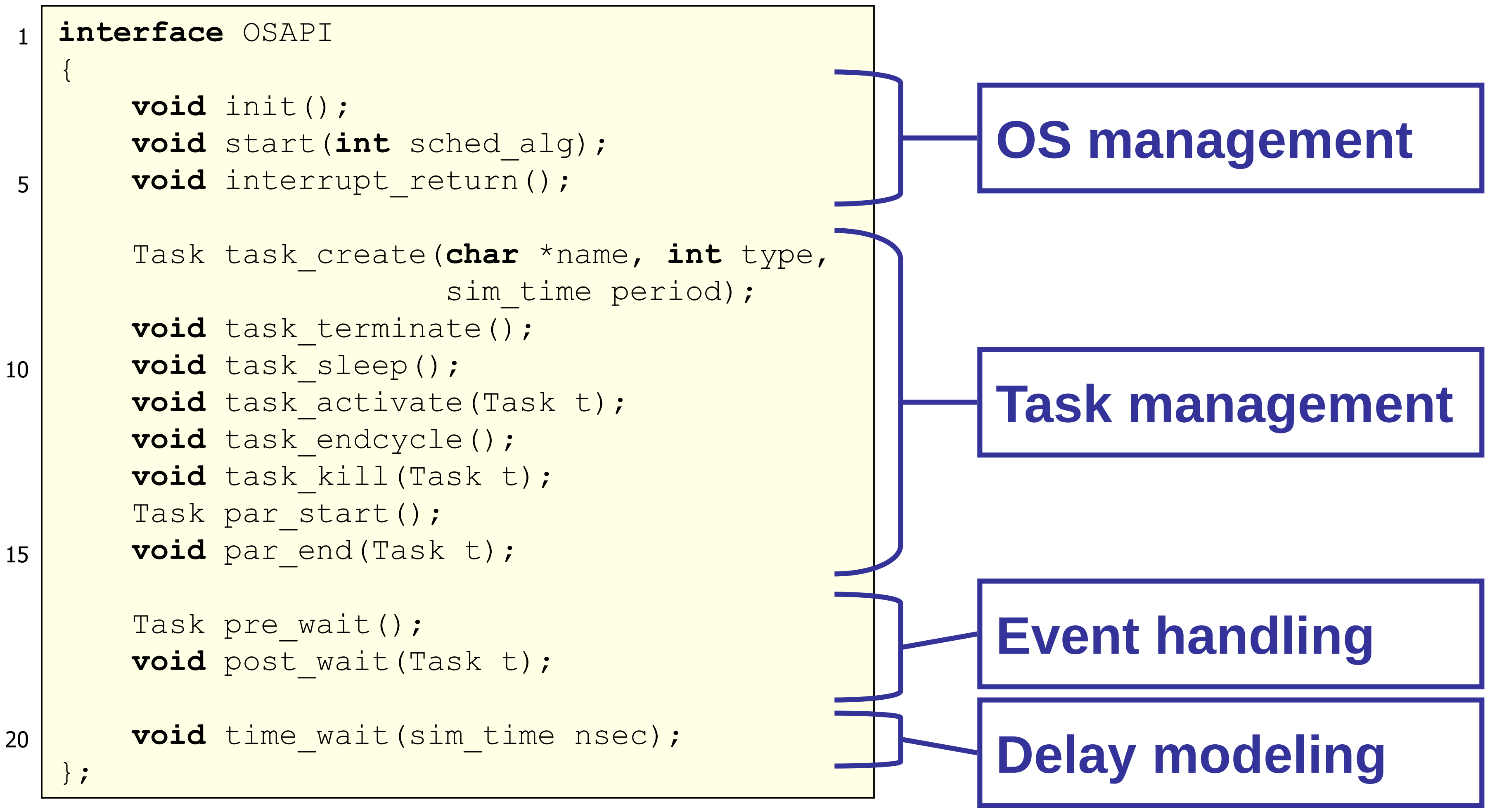
- **RTOS model**
  - OS, task, event management
    - – Descriptors & queues
  - Scheduling
    - – Select and dispatch task based on algorithm
    - – Block all but active task on SLDL level
  - Preemption
    - – Allow rescheduling at simulation time increases
  - Event handling
    - – Remove task temporarily from OS while waiting for SLDL event

➢ **RTOS model library**
  - RTOS models for different scheduling strategies
    - – Round robin, priority based
  - Parametrizable
    - – Task parameters (priorities)

```
channel OS implements OSAPI {
  Task current = 0;
  os_queue rdyq;

  void dispatch(void) {
    current = schedule();
    notify(current.event);
  }
  void yield() {
    task = current;
    dispatch();
    wait(task.event);
  }

  void time_wait(time t) {
    waitfor(t);
    yield();
  }

  Task pre_wait(void) {
    Task t = rdyq.get(current);
    dispatch(); return t;
  }
  void post_wait(Task t) {
    rdyq.put(t);
    wait(t.event);
  }
};
```

RTOS Model Interface

- **Canonical, target-independent API**

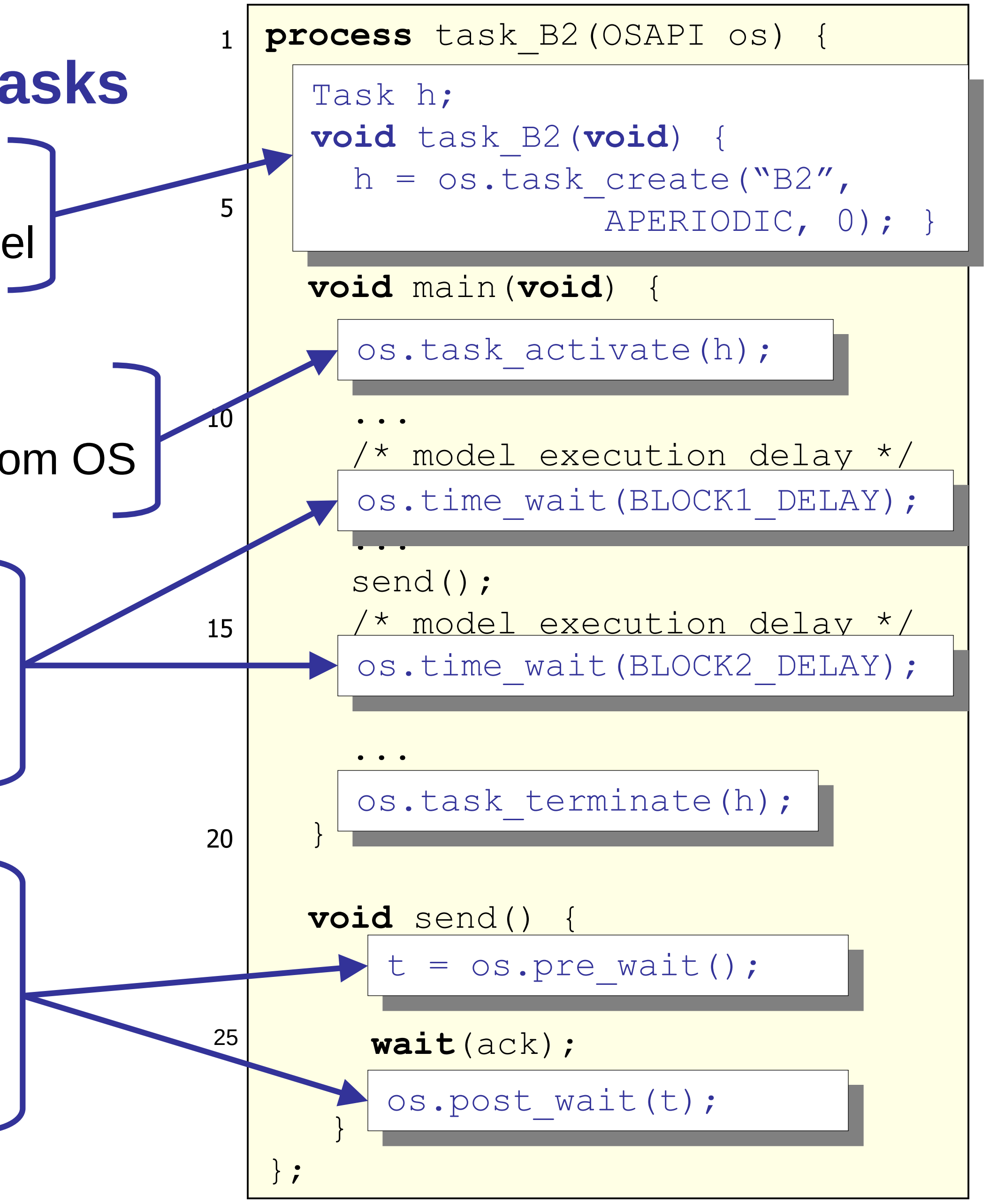


```
interface OSAPI
{
    void init();
    void start(int sched_alg);
    void interrupt_return();

    Task task_create(char *name, int type,
                     sim_time period);
    void task_terminate();
    void task_sleep();
    void task_activate(Task t);
    void task_endcycle();
    void task_kill(Task t);
    Task par_start();
    void par_end(Task t);

    Task pre_wait();
    void post_wait(Task t);

    void time_wait(sim_time nsec);
};
```

Back

Task Refinement

- **Convert processes into tasks**
  - Task initialization
    - – Register task with OS model
  - Task activation
    - – Wait for task start trigger from OS
  - Replace delay model
    - – Trigger rescheduling in OS
    - ➢ Preemption points
  - Communication and synchronization
    - – Wrap around SLDL event handling

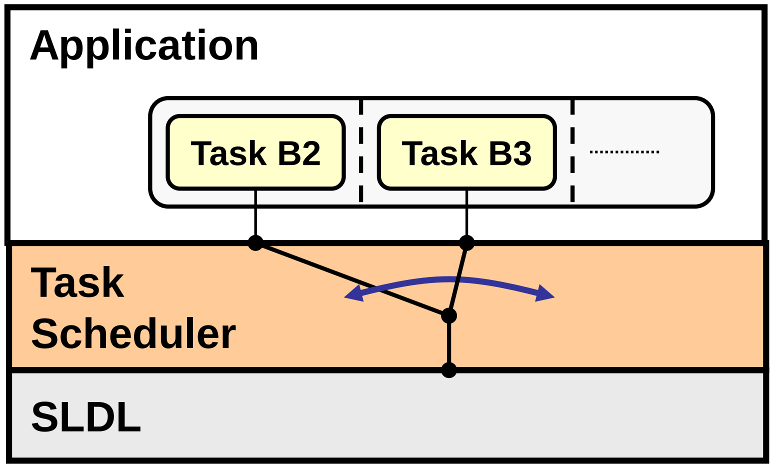


```
process task_B2(OSAPI os) {
  Task h;
  void task_B2(void) {
    h = os.task_create("B2",
                APERIODIC, 0); }
  void main(void) {
    os.task_activate(h);
    ...
    /* model execution delay */
    os.time_wait(BLOCK1_DELAY);
    ...
    send();
    /* model execution delay */
    os.time_wait(BLOCK2_DELAY);

    ...
    os.task_terminate(h);
  }

  void send() {
    t = os.pre_wait();
    wait(ack);
    os.post_wait(t);
  }
};
```

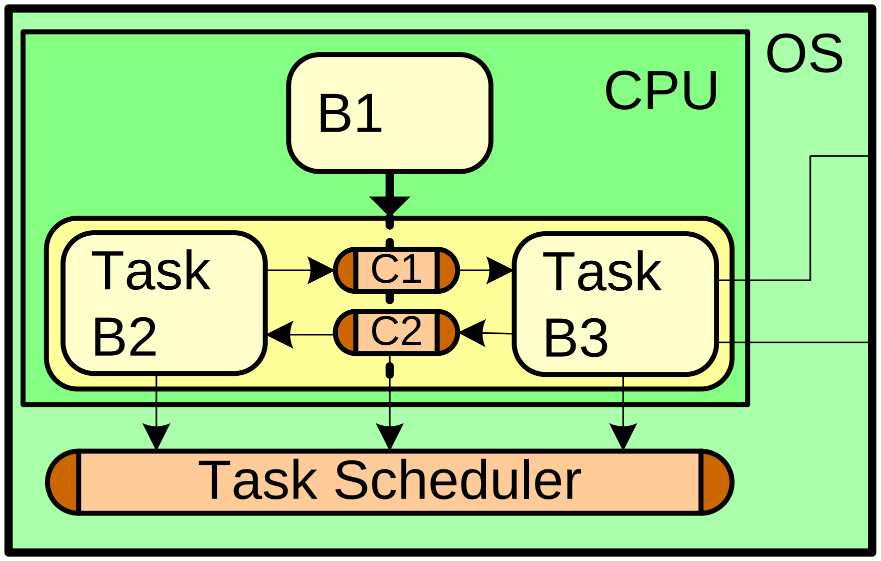
Back

Processor Model: Task Layer

- **Scheduling**
  - Group processes into tasks
    - Static scheduling
  - Schedule tasks
    - Dynamic scheduling, multitasking
    - Preemption, interrupt handling
    - Task communication (IPC)



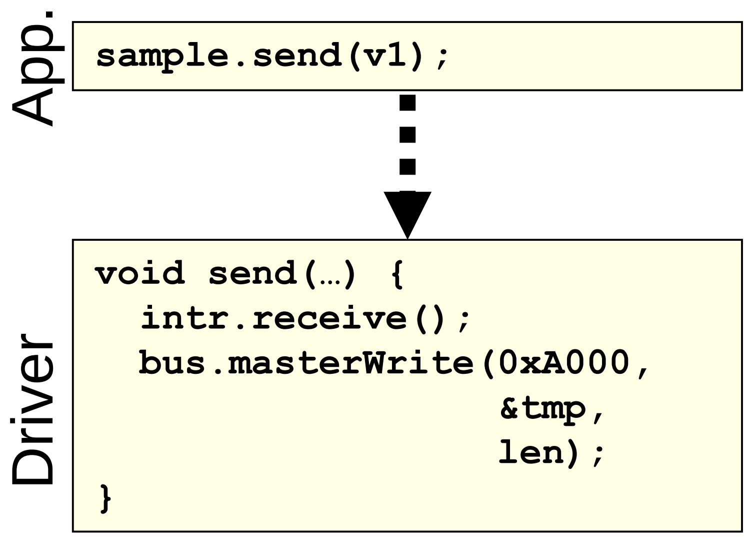
- ➢ **Scheduling refinement**
  - Flatten hierarchy
  - Reorder behaviors
- ➢ **OS refinement**
  - Insert OS model
  - Task refinement
  - IPC refinement



Processor Model: Firmware Layer

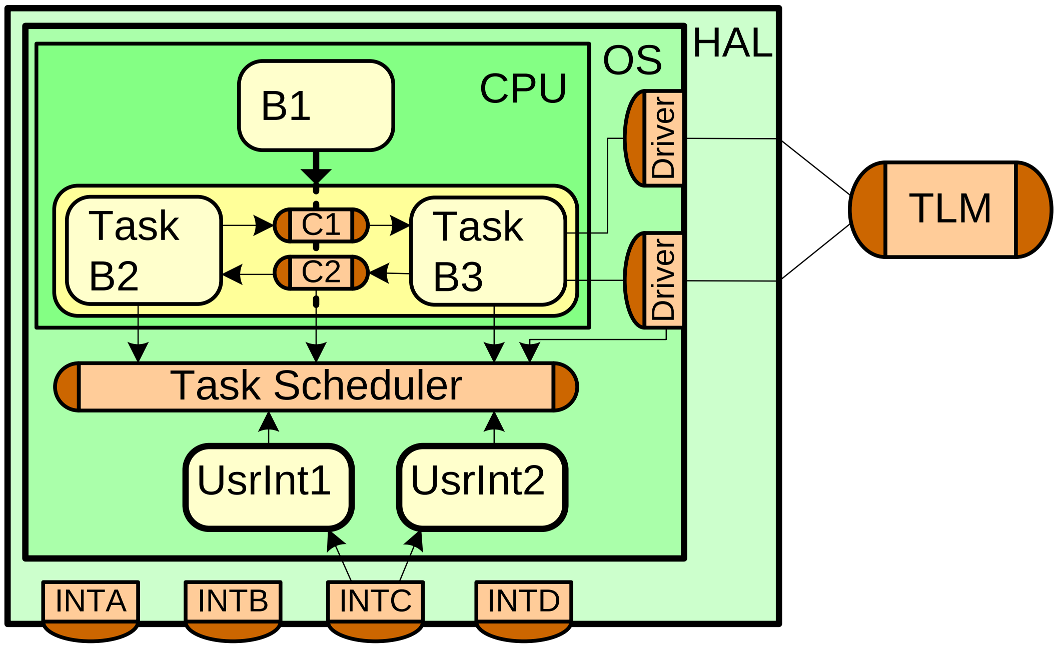
Hardware Abstraction Layer (HAL)

- **Interrupt handling**
- **External communication**
  - Software Drivers
    - Presentation, Session, Packeting
    - Synchronization (e.g. Interrupts)
  - TLM Bus model
    - User transactions
  - However, interrupts are still unscheduled



```
sample.send(v1);
```

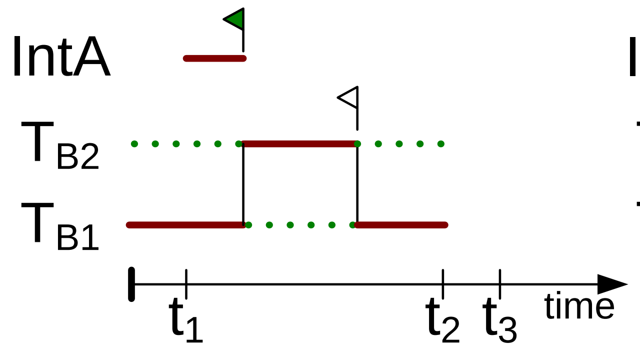
```
void send(…) {
  intr.receive();
  bus.masterWrite(0xA000,
                  &tmp,
                  len);
}
```



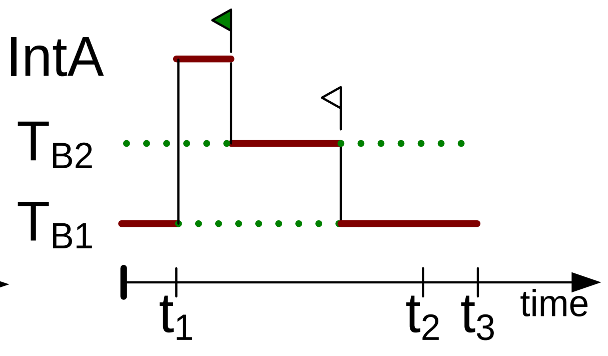
Processor Model: Hardware Layer

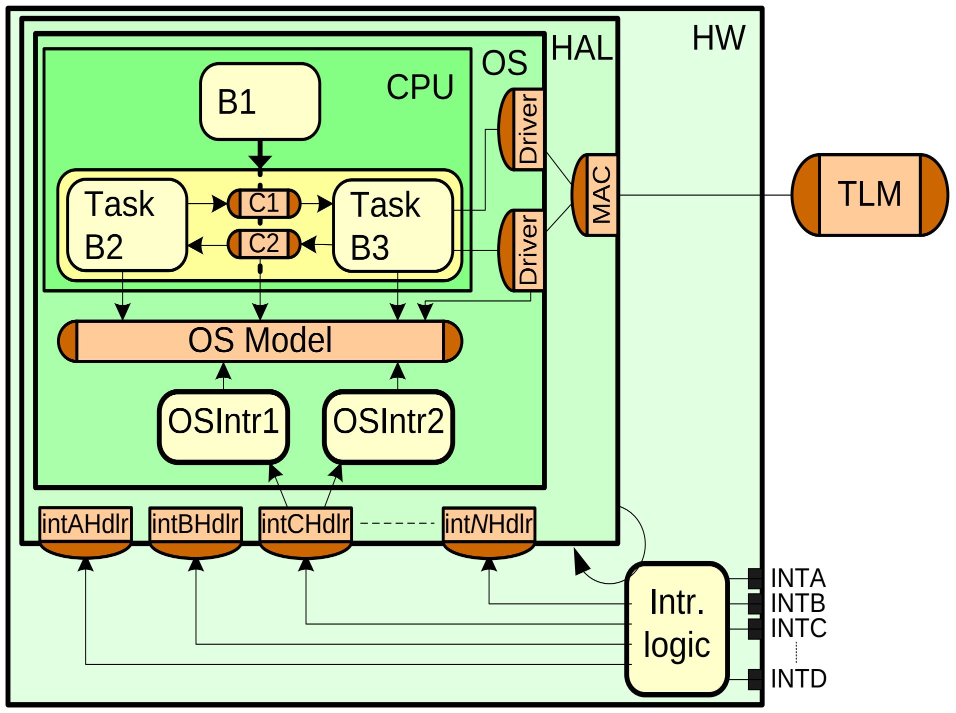
- **Hardware Layer**
  - Hardware interrupt handling
    - Interrupt Scheduling
      - » Suspend user code
      - » Priority, Nesting
  - Media Access Control (MAC) for bus interface
    - Split user transaction into bus transaction
  - Arbitrated TLM bus model

Unscheduled (HAL):



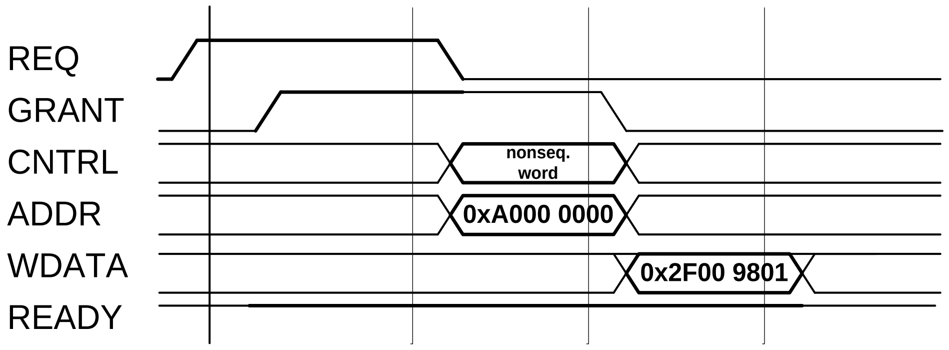
Scheduled (HW):

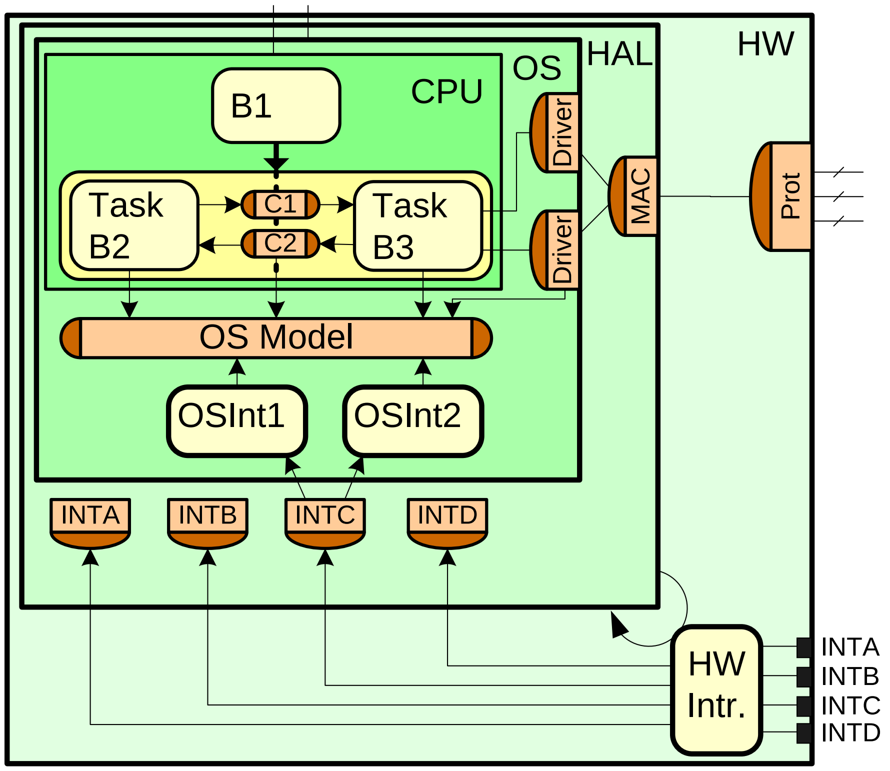




Processor Model: Bus-Functional Layer

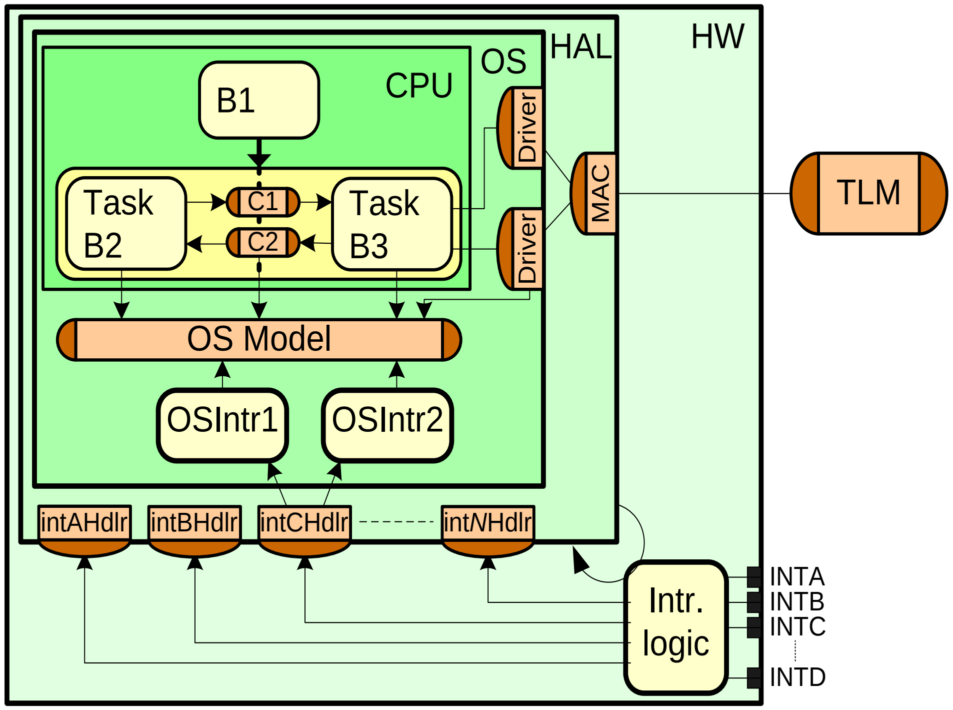
- **Processor bus-functional model (BFM)**
  - Pin-accurate model of processor
    - Cycle approximate for SW execution
  - Bus model
    - Pin-accurate
    - Cycle-Accurate

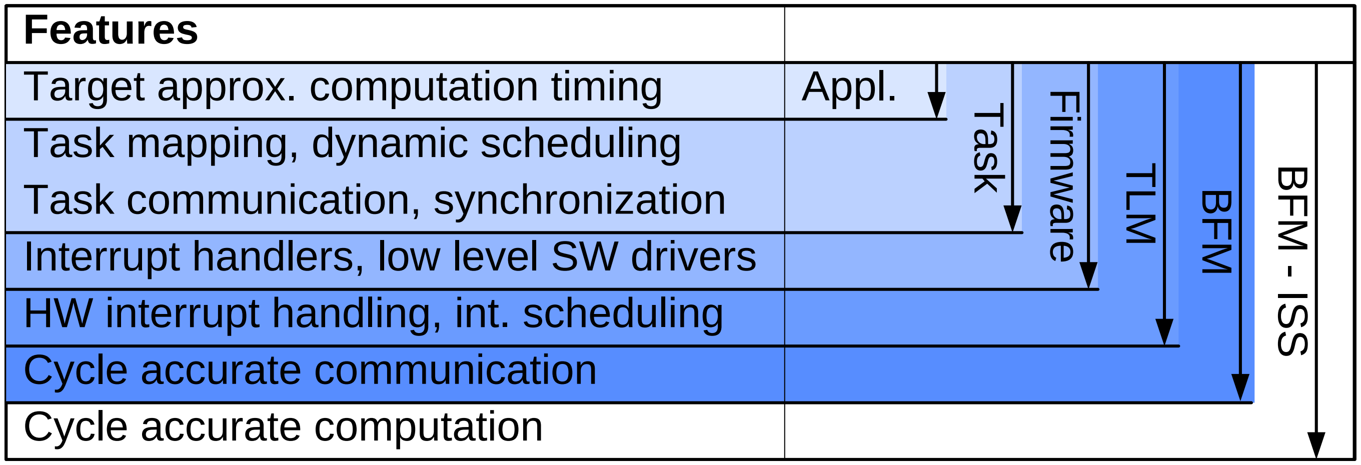




Processor Model - Summary

- **Layered model**
  - Feature levels
- **Processor layers**
  - Application
    - Native C
  - Task
    - OS model
  - Hardware abstraction
    - Middleware
  - Processor hardware
    - Bus I/F
    - Interrupts, suspension





Appl.
Task
Firmware
TLM
BFM
BFM - ISS

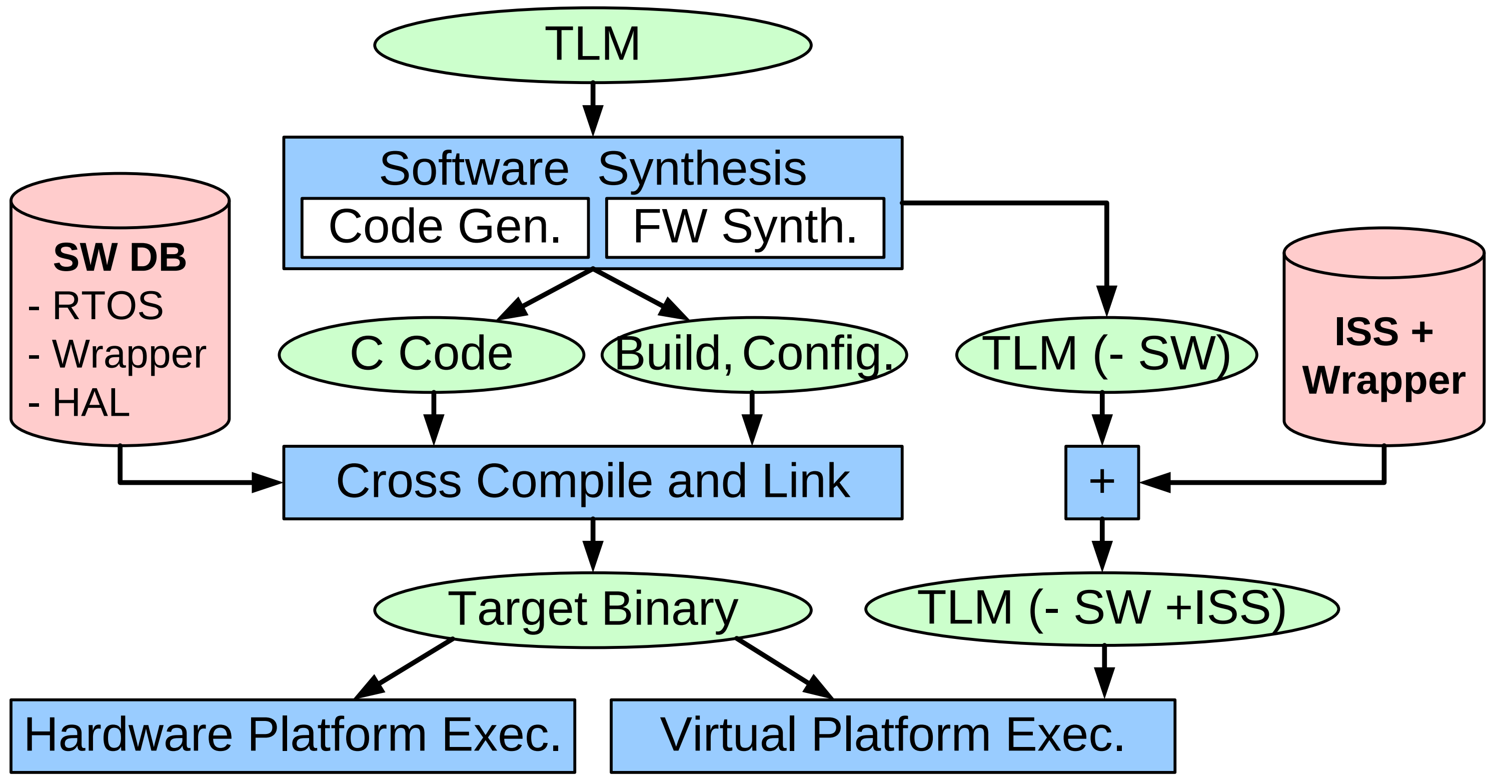
Features	
Target approx. computation timing	Appl.
Task mapping, dynamic scheduling	
Task communication, synchronization	
Interrupt handlers, low level SW drivers	
HW interrupt handling, int. scheduling	
Cycle accurate communication	
Cycle accurate computation	

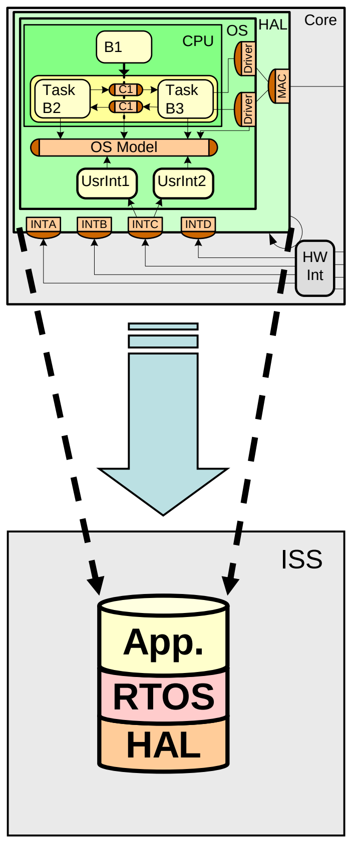
Lecture 7: Outline

- ✓ **Processor layers**
  - ✓ Application
  - ✓ Task/OS
  - ✓ Firmware
  - ✓ Hardware

- **Processor synthesis**
  - Software synthesis
  - Hardware synthesis

Software Synthesis

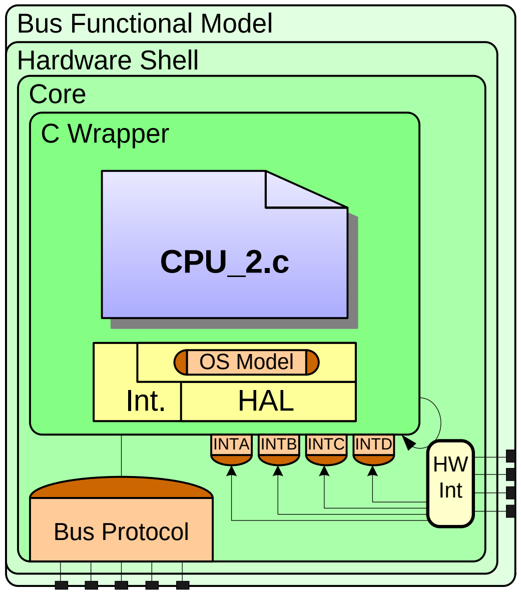




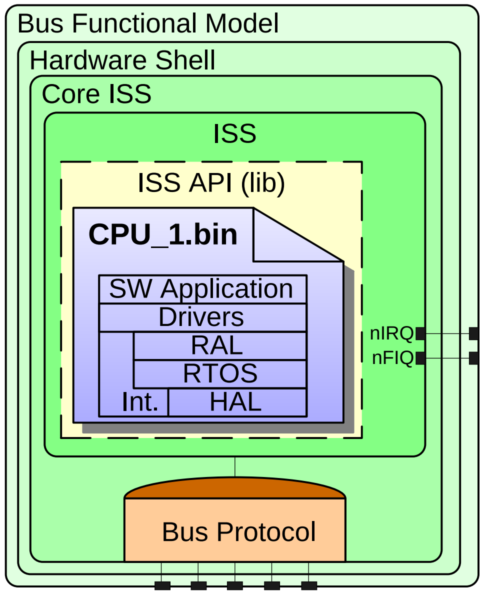
- **Automatically generate target binaries from TLM**
  - Generate code for application (tasks and IPC)
  - Synthesize firmware (drivers, interrupt handlers)
  - OS wrappers and HAL implementations from DB
  - Compile and link against target RTOS and libraries

Source: G. Schirner, A. Gerstlauer, R. Doemer. "Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications," ASPDAC08

Processor Implementation Models



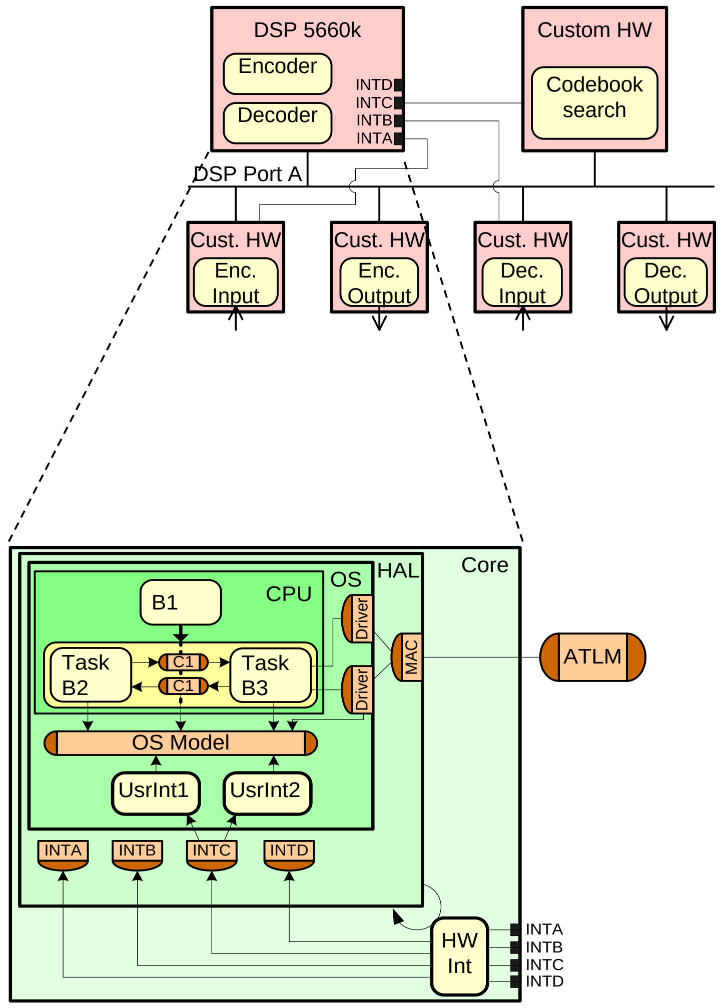
- **Software C model**
  - Generated application C code
    - Flat standard ANSI C code
  - Firmware and hardware models
    - RTOS model, HAL model
    - Low-level & hardware interrupt handling
    - External bus communication protocol/TLM



- **Software ISS model**
  - Reintegrared processor ISS
    - Bus-functional ISS wrapper
  - Running generated binary
    - Application, RTOS, drivers, HAL

Single-Processor Experiments

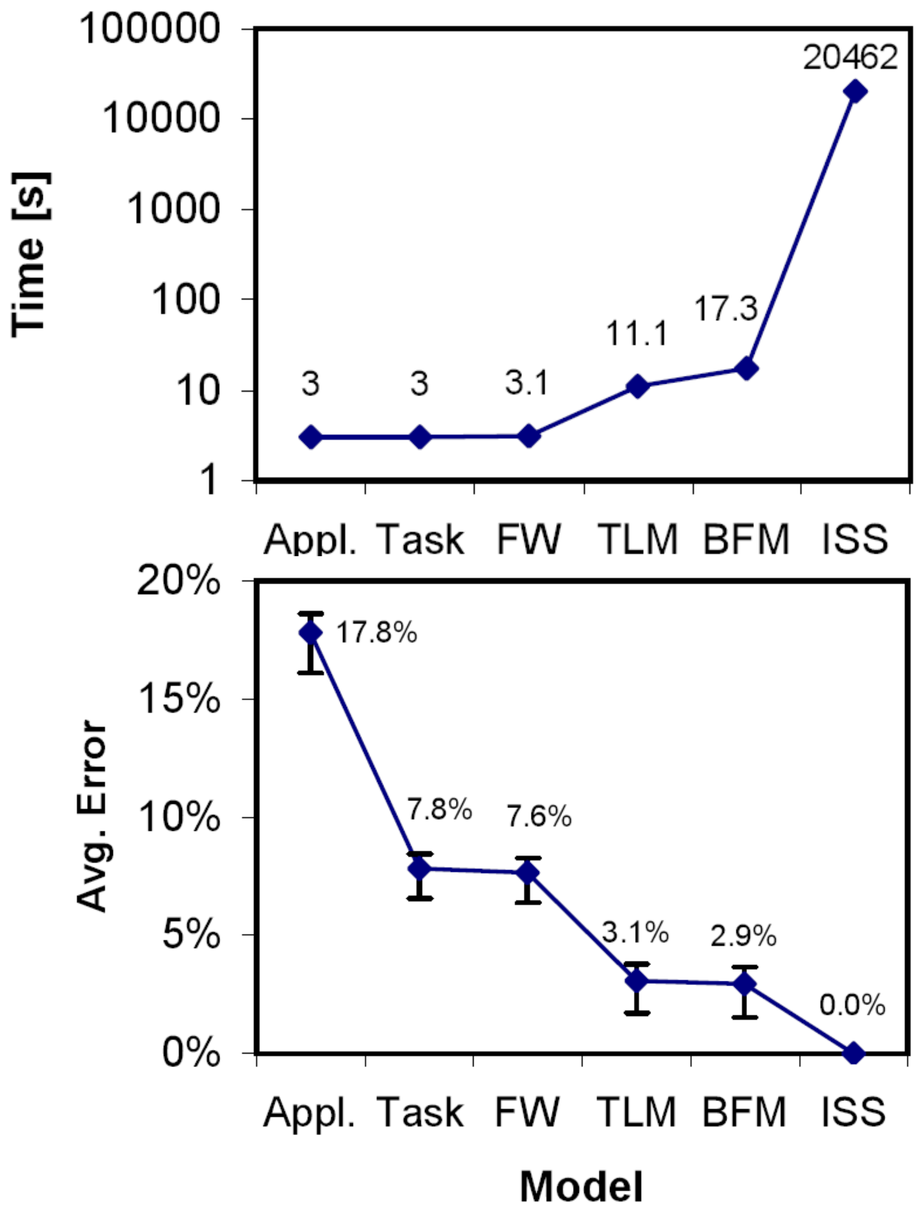
- **Voice encoding and decoding**
  - Motorola DSP 56600
    - Encoding & decoding tasks
    - custom OS
  - 4 custom I/O blocks
  - 1 custom HW co-processor
    - Codebook search
- **Processor models**
  - Perfect timing
    - Back-annotated from ISS
  - Priority-based OS model
    - EDF: Decoder > Encoder
  - HW interrupt scheduling
    - 4 non-preempted priority levels
- **Reference**
  - Motorola proprietary ISS



Processor Modeling Results

- **Execute on Sun Fire V240 (1.5 GHz)**
  - 163 speech frames
- **Speed vs. accuracy**
  - ➢ OS model (Appl ⇨ Task)
  - ➢ Interrupts (FW ⇨ TLM)

➢ **1800x speed w/ 3% error (vs. cycle-accurate ISS)**



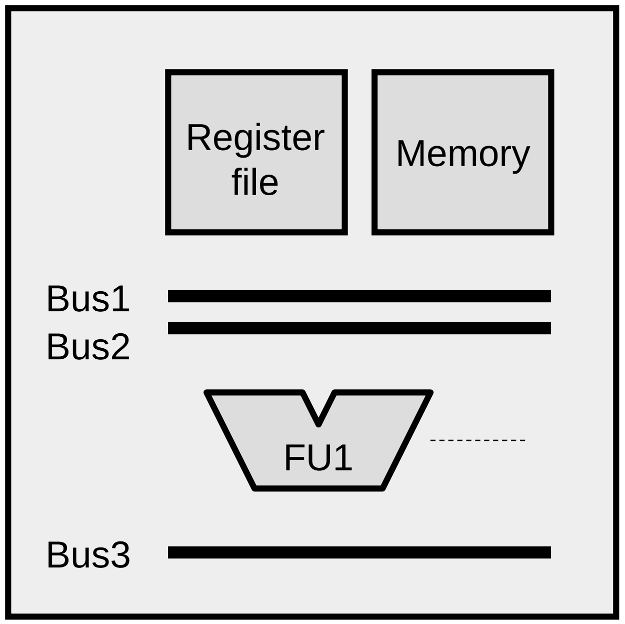
Lecture 7: Outline

- ✓ Processor layers
  - ✓ Application
  - ✓ Task/OS
  - ✓ Firmware
  - ✓ Hardware

- **Processor synthesis**
  - ✓ Software synthesis
  - Hardware synthesis

High-Level Synthesis (1)

- **Allocation**

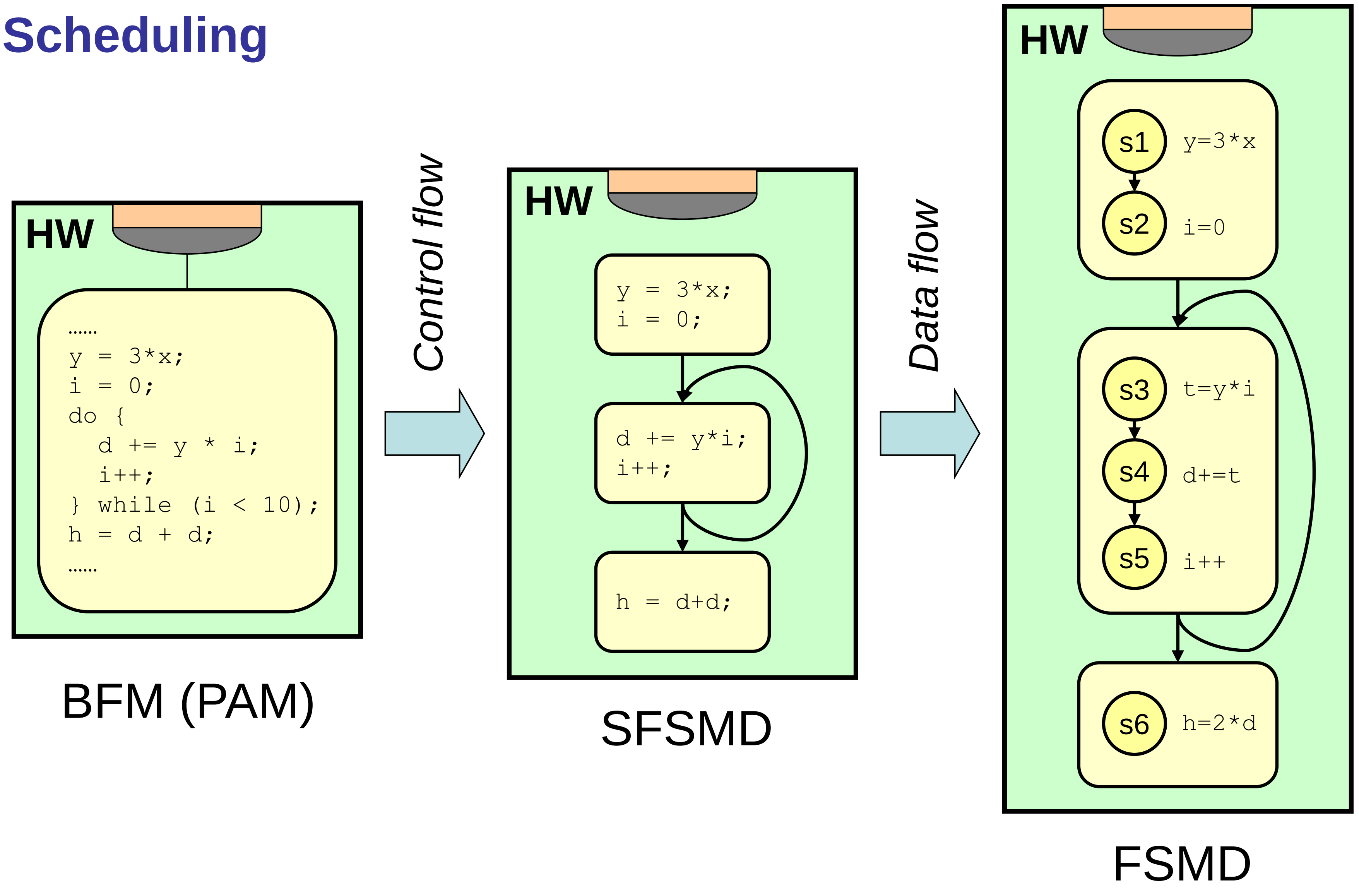


Datapath

Source: D. Shin, A. Gerstlauer, R. Dömer, D. Gajski, "An Interactive Design Environment for C-based High-level Synthesis of RTL Processors," TVLSI, April 2008.

High-Level Synthesis (2)

- **Scheduling**



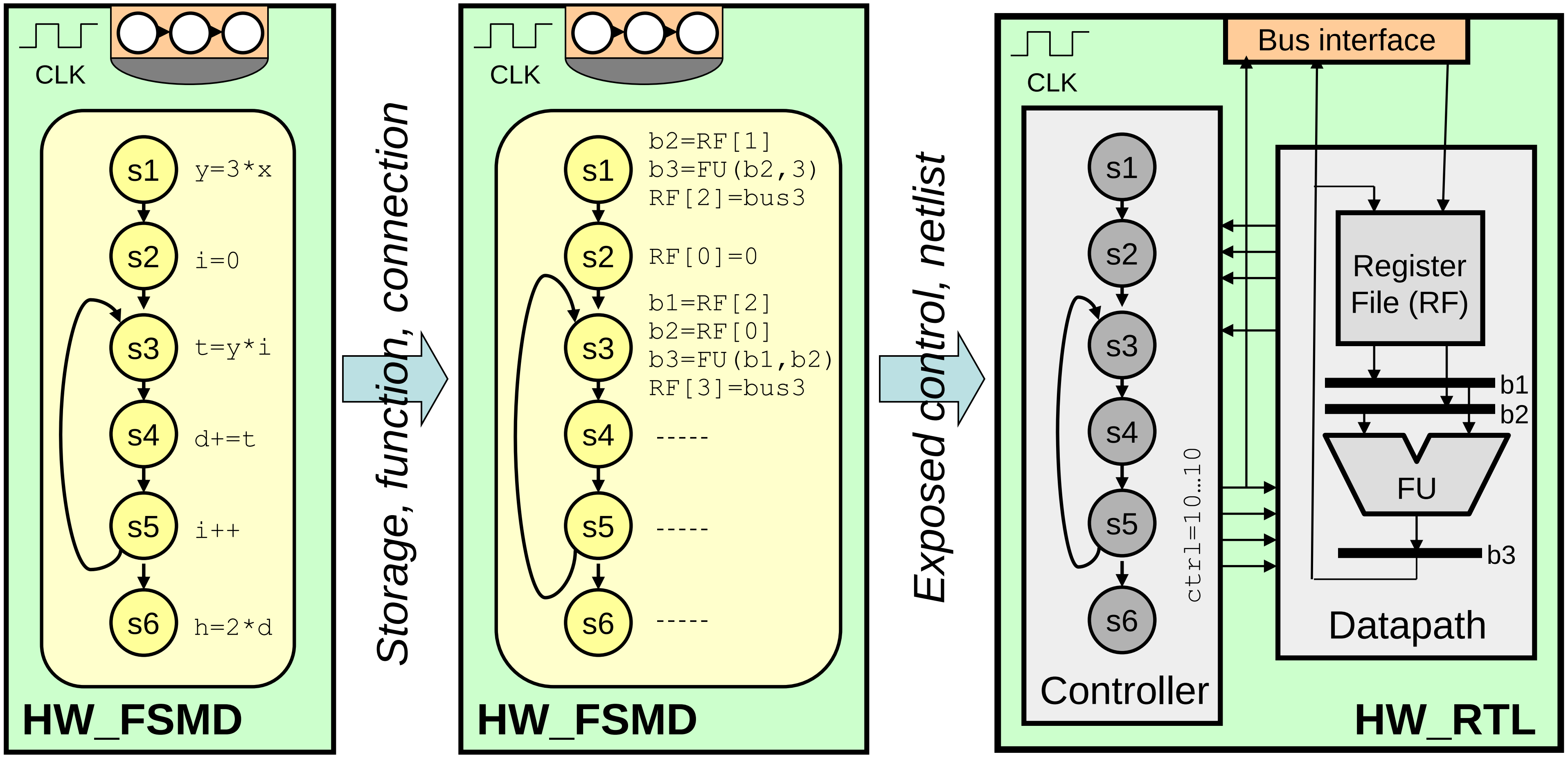
BFM (PAM)

SFSMD

FSMD

High-Level Synthesis (3)

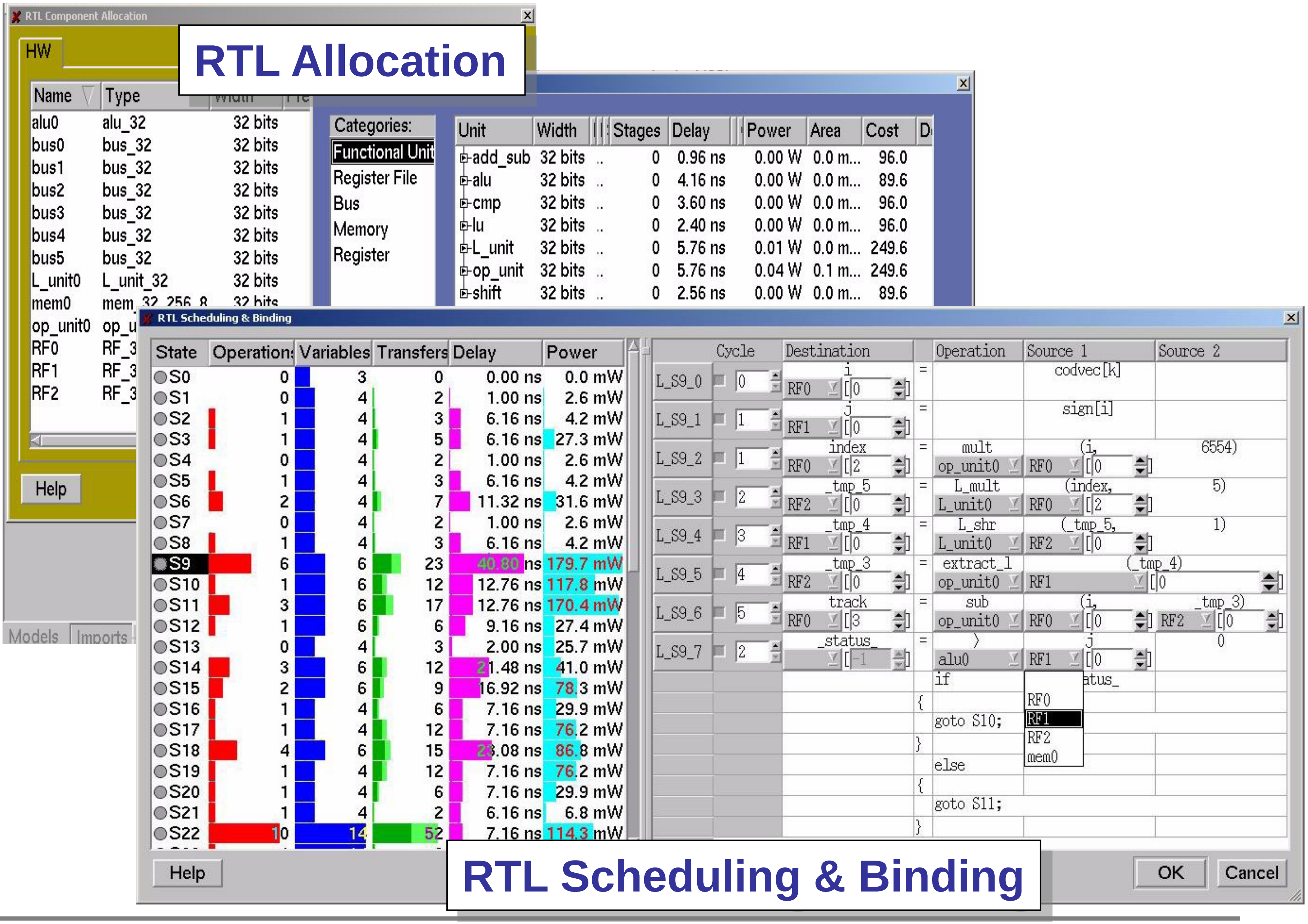
- **Binding**



Unmapped RTL | Mapped RTL | Structural RTL

Source: Accellera, "RTL Semantics," http://www.eda.org/alc-cwg/cwg-open.pdf

SCE Interactive RTL Synthesis



Name	Type	Width
alu0	alu_32	32 bits
bus0	bus_32	32 bits
bus1	bus_32	32 bits
bus2	bus_32	32 bits
bus3	bus_32	32 bits
bus4	bus_32	32 bits
bus5	bus_32	32 bits
L_unit0	L_unit_32	32 bits
mem0	mem_32_256_8	32 bits
op_unit0	op_u	
RF0	RF_3	
RF1	RF_3	
RF2	RF_3	

Categories:
- Functional Unit
- Register File
- Bus
- Memory
- Register

Unit	Width		Stages	Delay	Power	Area	Cost
add_sub	32 bits	..	0	0.96 ns	0.00 W	0.0 m...	96.0
alu	32 bits	..	0	4.16 ns	0.00 W	0.0 m...	89.6
cmp	32 bits	..	0	3.60 ns	0.00 W	0.0 m...	96.0
lu	32 bits	..	0	2.40 ns	0.00 W	0.0 m...	96.0
L_unit	32 bits	..	0	5.76 ns	0.01 W	0.0 m...	249.6
op_unit	32 bits	..	0	5.76 ns	0.04 W	0.1 m...	249.6
shift	32 bits	..	0	2.56 ns	0.00 W	0.0 m...	89.6

State	Operations	Variables	Transfers	Delay	Power
S0	0	3	0	0.00 ns	0.0 mW
S1	0	4	2	1.00 ns	2.6 mW
S2	1	4	3	6.16 ns	4.2 mW
S3	1	4	5	6.16 ns	27.3 mW
S4	0	4	2	1.00 ns	2.6 mW
S5	1	4	3	6.16 ns	4.2 mW
S6	2	4	7	11.32 ns	31.6 mW
S7	0	4	2	1.00 ns	2.6 mW
S8	1	4	3	6.16 ns	4.2 mW
S9	6	6	23	40.80 ns	179.7 mW
S10	1	6	12	12.76 ns	117.8 mW
S11	3	6	17	12.76 ns	170.4 mW
S12	1	6	6	9.16 ns	27.4 mW
S13	0	4	3	2.00 ns	25.7 mW
S14	3	6	12	21.48 ns	41.0 mW
S15	2	6	9	16.92 ns	78.3 mW
S16	1	4	6	7.16 ns	29.9 mW
S17	1	4	12	7.16 ns	76.2 mW
S18	4	6	15	23.08 ns	86.8 mW
S19	1	4	12	7.16 ns	76.2 mW
S20	1	4	6	7.16 ns	29.9 mW
S21	1	4	2	6.16 ns	6.8 mW
S22	10	14	52	7.16 ns	114.3 mW

	Cycle	Destination		Operation	Source 1	Source 2
L_S9_0	0	i RF0 [0	=		codvec[k]	
L_S9_1	1	j RF1 [0	=		sign[i]	
L_S9_2	1	index RF0 [2	=	mult op_unit0	(i, RF0 [0	6554)
L_S9_3	2	_tmp_5 RF2 [0	=	L_mult L_unit0	(index, RF0 [2	5)
L_S9_4	3	_tmp_4 RF1 [0	=	L_shr L_unit0	(_tmp_5, RF2 [0	1)
L_S9_5	4	_tmp_3 RF2 [0	=	extract_l op_unit0	RF1	(_tmp_4) [0
L_S9_6	5	track RF0 [3	=	sub op_unit0	(i, RF0 [0	_tmp_3) RF2 [0
L_S9_7	2	_status_ [-1	=	) alu0	j RF1 [0	0
				if	atus_	
			{		RF0	
				goto S10;	RF1	
			}		RF2	
				else	mem0	
			{			
				goto S11;		
			}			

SpecC RTL Modeling

RTL Modeling Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{   void main(void)
    {
      fsmd(CLK)                             // clock + sensitivity
        {
          bit[32] a, b, c, d, e;            // local variables

                        { Outport = 0;      // default
                          Done = 0b;        // assignments
                         }
          if (RST)      { goto S0;          // reset actions
                         }
          S0 :          { if (Start) goto S1;
                          else       goto S0;
                         }
          S1 :          { a = b + c;        // state actions
                          d = Inport * e;   // (register transfers)
                          Outport = a;
                          goto S2;
                         }
          ... }
     }
};
```

Source: R. Doemer

SpecC RTL Modeling

Mapped RTL Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity

          bit[32] a, b, c, d, e;            // unmapped variables

                      { Outport = 0;        // default
                        Done = 0b;          // assignments
                       }
          if (RST)    { goto S0;            // reset actions
                       }
          S0 :        { if (Start) goto S1;
                        else       goto S0;
                       }
          S1 :        { a = b + c;          // Accellera style 1
                        d = Inport * e;     // (unmapped)
                        Outport = a;
                        goto S2;
                       }
    }
};
```

Source: R. Doemer

SpecC RTL Modeling

Mapped RTL Example

```
behavior FSMD_Example(
        signal in  bool       CLK,         // system clock
        signal in  bool       RST,         // system reset
        signal in  bit[31:0]  Inport,      // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,     // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                             // clock + sensitivity
         buffered[CLK] bit[32] RF[5];      // register file

                    { Outport = 0;         // default
                      Done = 0b;           // assignments
                     }
       if (RST)     { goto S0;             // reset actions
                     }
       S0  :        { if (Start) goto S1;
                      else       goto S0;
                     }
       S1  :        { RF[0]=RF[1]+RF[2]; // Accellera style 2
                      RF[3]=Inport*RF[4];// (storage mapped)
                      Outport = RF[0];
                      goto S2;
                     }
    }
};
```

Source: R. Doemer

SpecC RTL Modeling

Mapped RTL Example

```
behavior FSMD_Example(
        signal in  bool       CLK,          // system clock
        signal in  bool       RST,          // system reset
        signal in  bit[31:0]  Inport,       // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,      // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                              // clock + sensitivity
       {
        buffered[CLK] bit[32] RF[5];        // register file

                      { Outport = 0;        // default
                        Done = 0b;          // assignments
                       }
         if (RST)     { goto S0;            // reset actions
                       }
         S0  :        { if (Start) goto S1;
                        else       goto S0;
                       }
         S1  :        { RF[0] =             // Accellera style 3
                         ADD0(RF[1],RF[2]);// (function mapped)
                        RF[3] =
                         MUL0(Inport,RF[4]);
                        Outport = RF[0];
                        goto S2;
                       }
    }
};
```

Source: R. Doemer

SpecC RTL Modeling

Mapped RTL Example

```
behavior FSMD_Example(
        signal in  bool       CLK,           // system clock
        signal in  bool       RST,           // system reset
        signal in  bit[31:0]  Inport,        // input ports
        signal in  bit[1]     Start,
        signal out bit[31:0]  Outport,       // output ports
        signal out bit[1]     Done)
{  void main(void)
   {
     fsmd(CLK)                               // clock + sensitivity
        {
        buffered[CLK] bit[32] RF[5];         // register file
        bit[32] BUS0, BUS1, BUS2;            // busses
                     { Outport = 0;          // default
                       Done = 0b;            // assignments
                     }
        if (RST)     { goto S0;              // reset actions
                     }
        S0 :         { if (Start) goto S1;
                       else       goto S0;
                     }
        S1 :         { BUS0 = RF[1];         // Accellera style 4
                       BUS1 = RF[2];         // (connection mapped)
                       BUS3 = ADD0(BUS0,BUS1);
                       RF[0]= BUS3;
                       ...
                       goto S2;
     }               }
};
```

Source: R. Doemer

SpecC RTL Modeling

Mapped RTL Example

```
behavior FSMD_Example(
           signal in  bool        CLK,          // system clock
           signal in  bool        RST,          // system reset
           signal in  bit[31:0]   Inport,       // input ports
           signal in  bit[1]      Start,
           signal out bit[31:0]   Outport,      // output ports
           signal out bit[1]      Done)
{   void main(void)
    {
      fsmd(CLK)                                 // clock + sensitivity
        {
           signal bit[5:0] RF_CTRL;             // control wires
           signal bit[1:0] ADD0_CTRL, MUL0_CTRL;
                          { Outport = 0;        // default
                            Done = 0b;          // assignments
                           }
           if  (RST)      { goto S0;            // reset actions
                           }
           S0  :          { if  (Start) goto S1;
                            else        goto S0;
                           }
           S1  :          { RF_CTRL = 011000b; // Accellera style 5
                            ADD0_CTRL = 01b;   // (exposed control)
                            MUL0_CTRL = 11b;
                            ...

                            goto S2;
        }                  }
};
```

Source: R. Doemer

Lecture 7: Summary

- **OS and Processor Modeling**
  - Model of software running in execution environment
    - Timed application, OS, bus drivers, interrupt handlers
    - Processor hardware model, suspension, bus interfaces
  - ➢ Virtual platform prototype
    - ➢ Embedded software development and validation
    - ➢ Viable complement to ISS-based validation

- **Backend processor synthesis**
  - Software synthesis
    - Code generation, RTOS targeting, cross-compilation & linking
    - Fully automatic final target binary generation
  - Hardware synthesis
    - High-level/behavioral synthesis: allocation, scheduling, binding
    - Interactive C-to-RTL synthesis flow

Student Assistant Job Offer

Henning.Schlender@dlr.de

- Tasks
  - Context: Autonomous Driving
  - ROS 2 Development (C++)
  - Implementing ROS 2 Components
  - Co-Simulation with Carla Simulator
  - Implementing Automated Driving Scenarios

- Requirements
  - Bachelor‘s Degree (desirable)
  - Experience in
    - C++
    - Gitlab
    - Linux