ECE 587 – Hardware/Software Co-Design
Lecture 14 System Synthesis, Software Synthesis

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

March 3, 2025

Reading Assignment

- This lecture: 4,5
- Next lecture: 6

Outline

System Synthesis

Software Synthesis

Constructing System Models

- System models are constructed via combining layers of computation and communication models.
  - Different choices of layers lead to different system models.
  - Partial features from certain layer, or merged features from multiple layers can be included.
- Specification model
  - Various MoCs specified as hierarchical, sequential/parallel composition of processes communicating through abstract variables and message -passing channels.
- Transaction-level models (TLMs)
  - Mappings of processes and channels to processors and busses
  - Additional details but still in abstracted forms
- Cycle-accurate model (CAM)
  - Cycle-accurate computation as RTL/instructions
  - Pins and wires with driven protocols

System Models

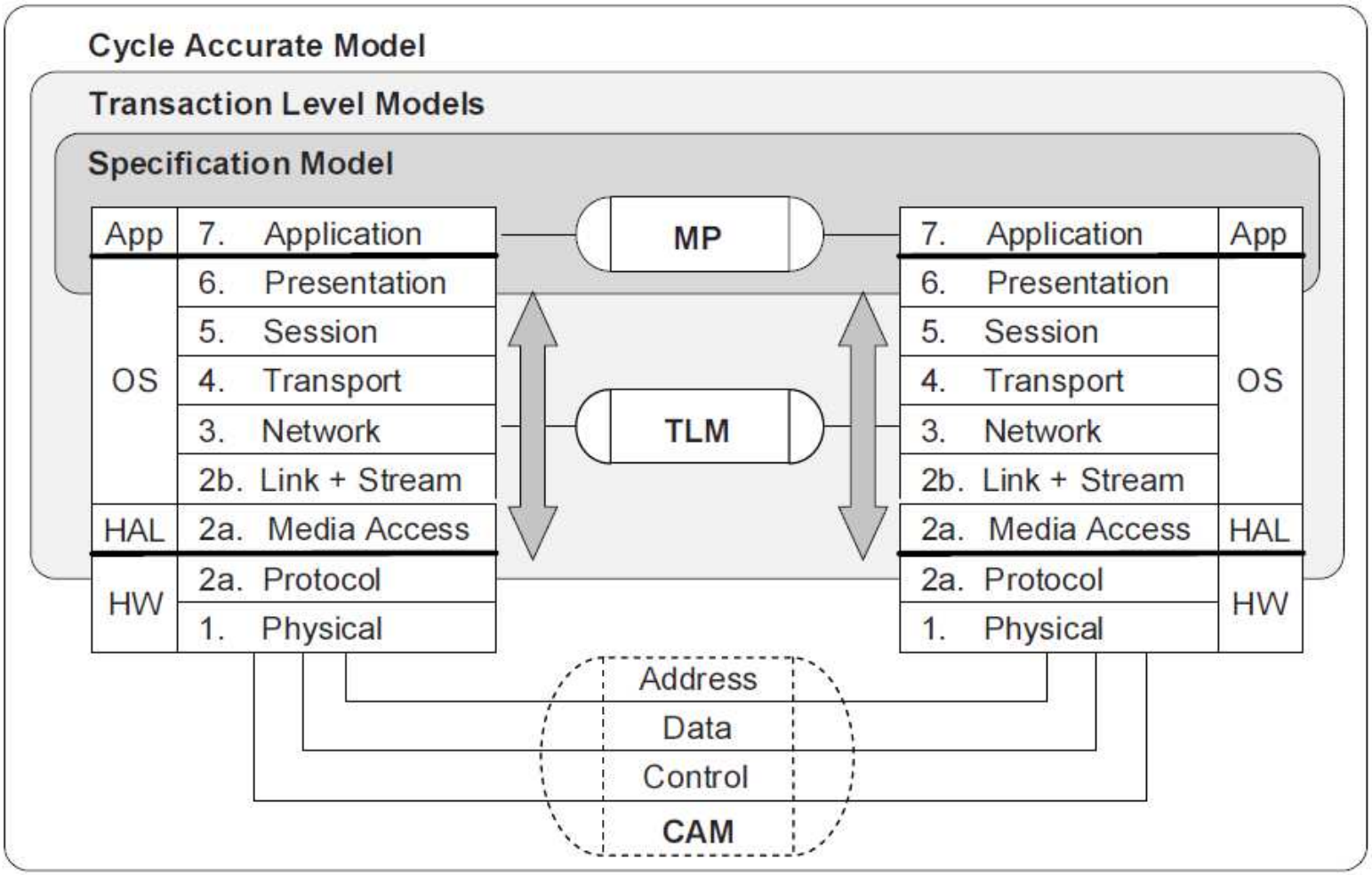


FIGURE 3.28 System models

(Gajski et al.)

Traditional Board-Based System Design I

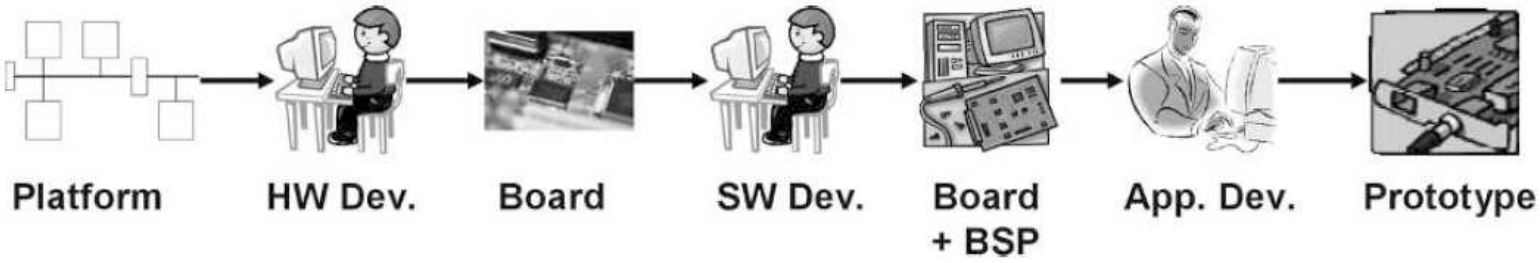


FIGURE 4.1 A traditional board-based system design process.

- Platform architect first defines a high-level platform including type of processors and communication architecture. (Gajski et al.)
  - Application characteristics are considered.
  - e.g use DSP for multimedia codec designs and use embedded processors for control intensive applications
- Then, the number of each component should be decided.
  - Depends on the number of independent tasks or available parallelism in the application.
  - Without an evaluation model, the platform architect depends on his or her experience and the application profile.
- The platform may be chosen based on legacy considerations.
  - New components are added for product updates.

Traditional Board-Based System Design II

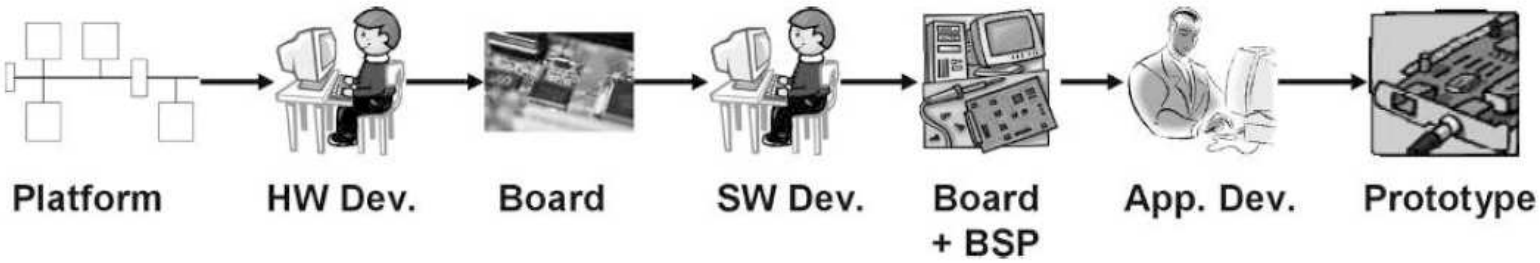


FIGURE 4.1 A traditional board-based system design process.

(Gajski et al.)

- HW engineers implement the platform and deliver the board before SW developers develop the system software.
  - Include HDL models of custom blocks and configurations of processors and other IPs.
- Before the application's SW development could begin, the development of boards and board support packages (BSPs) is required.
- The system prototype is produced after application SW is ready using the BSPs and downloaded to the board.

Traditional Board-Based System Design III

- Sequential development of HW and SW causes delays.
  - Both the application SW and the HW/BSP development take several months.
  - The system prototype is not ready until more than a year after specification.
- Additional verification issues may result in further delays.
  - Who should be accountable for bugs during software development? Buggy hardware design or improper use of the hardware due to poor understanding of software developers?
  - Time will be wasted between interactions between HW/SW teams and ad-hoc decisions.

Model Based System Design

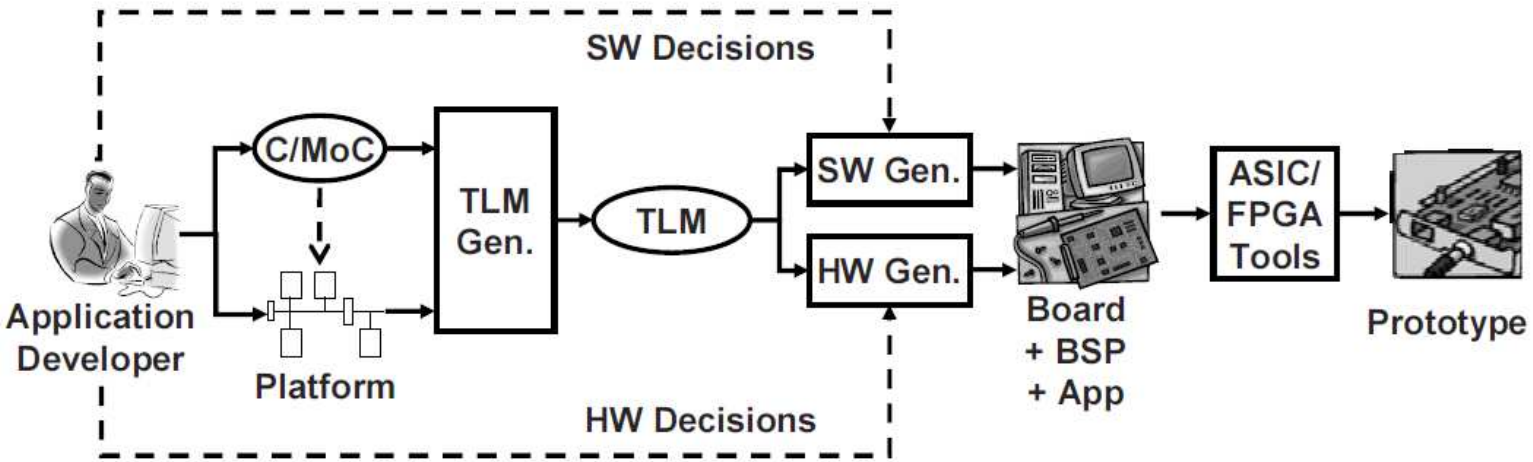


FIGURE 4.3 A model based development flow of the future.

(Gajski et al.)

- Instead of being postponed to the end of design process, application drives platform selection and HW/SW generation.
- TLM is first generated automatically from high level platform/application specifications.
- Platform is then defined/derived and application is mapped.
- Modification of system architecture or application is possible even late in the design process as everything will be generated automatically.

TLM Based Design Flow

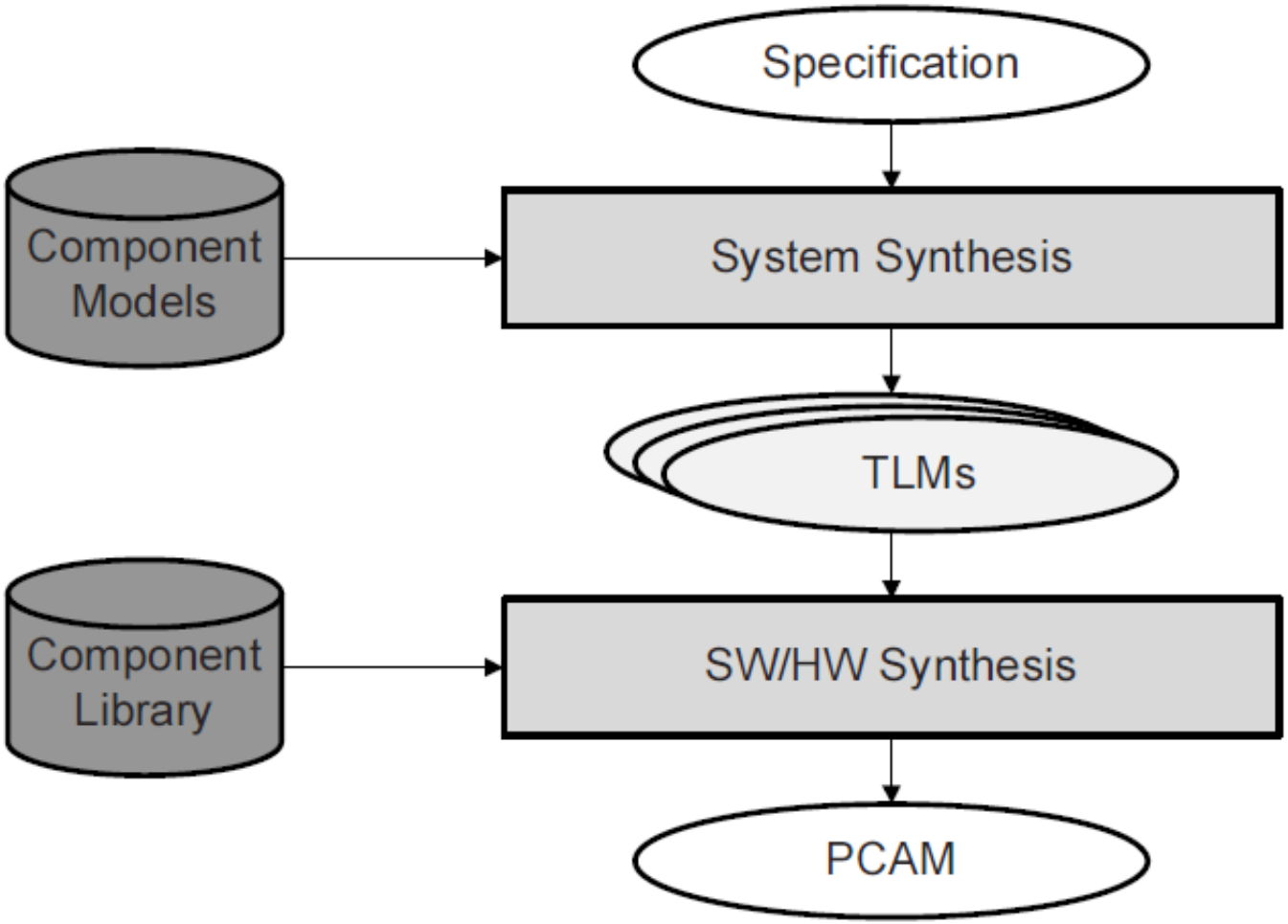


FIGURE 4.4 TLM based design flow.

(Gajski et al.)

System Specification Model

- Application model
  - Purely functional w/o implementation details
  - Typically executable for early functional validation
- SW/HW platform
  - Structural: components and their connectivity
  - Usually not executable
- Mapping application and platform
  - Computational elements (processes) to processors
  - Communication elements (channels) to buses or routes

System Synthesis

- Generate TLM according to application and its mapping on the platform
- Generate the mapping or the platform if necessary
- Utilize a database describing components
  - Functionality?
  - Performance metrics?
  - Configuration settings?
- The ideal TLM output is one that provides a reasonable balance between fast simulation and accurate estimation.
  - Though the actual TLM output depends upon the design methodology and the availability of component models.

The Next Step: from TLM to PCAM/CAM

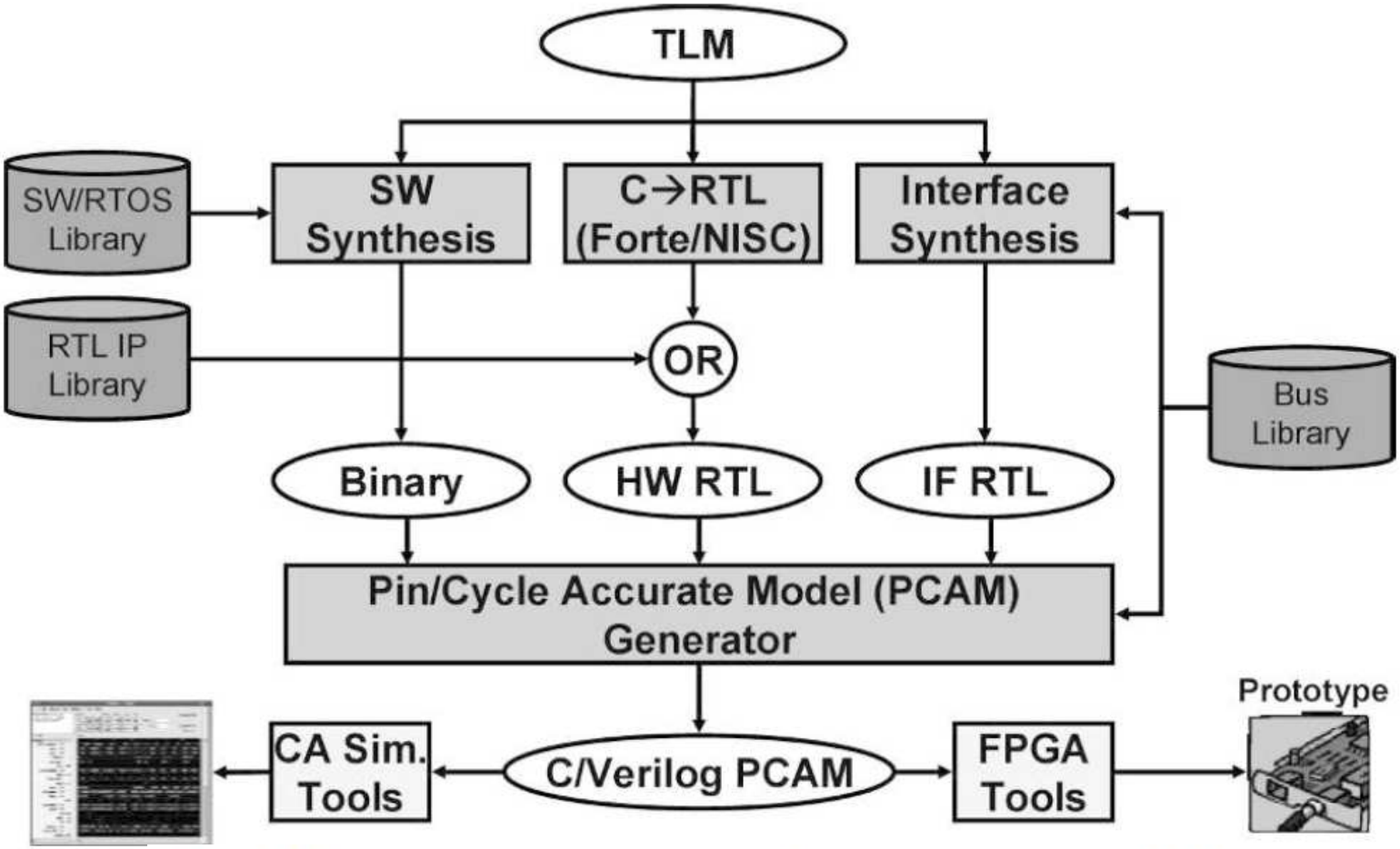


FIGURE 4.27 Cycle accurate model generation from TLM. (Gajski et al.)

Outline

System Synthesis

Software Synthesis

Software Design

- Software development dominates the design cost of modern computing systems.
- Complexity of software increases.
  - Overall system complexity increases.
  - Designers prefer software-centric implementations because of productivity and flexibility.

Automated Software Synthesis

- Generate software from models.
  - The final products are the binaries for processors.
- Benefits
  - No tedious and error-prone manual code writing
  - Demands less processor- and platform-specific knowledge from the designer
  - Each synthesis step can be individually verified.
- Increase productivity by reducing time for development and debugging

Challenges for Software Development/Synthesis

- Coupling to underlying hardware and external processes
- Timeliness
  - Real-time constraints extend to software implementation
  - Correctness not only means correct functionality, but also the ability to meet deadlines.
  - Predictable execution time is more important than fast execution.
- Concurrency
  - Scheduling with real-time constraints is complicated.
- Resource constraints
  - Memory, computing power, energy consumption, power dissipation, etc.

Software Synthesis and Programming Languages

- Software synthesis leverages existing software design tools.
- Software can be generated in existing programming language(s)
  - Instead of binaries, which can be produced by existing tools
  - Allow designers to have better control over the final code
- What languages are available?

Programming Languages

- Assembly: provide fine-grained control over processors
  - Processor dependent and overly verbose for large projects
- C: provide low-level features
  - Processor independent, require minimal run-time support
- C++: compatible with C, provide higher level abstractions
  - Complicated, large runtime overhead if not used properly
- Java: a simplified derivative of C++
  - Prevent unnecessary mistakes (both functional and performance-wise) by being less flexible
  - JVM provides supports of concurrency and communications at language level, though slow speed is a concern.
  - To meet deadlines is challenging due to garbage collection.
- And there are many more other programming languages.

Multi-Task Synthesis

- For dynamic on-line scheduling
  - Off-line scheduling can be represented as sequential composition of tasks and there is no need for multi-task support at run-time.
- RTOS-based multi-tasking
  - User tasks are executed on top of an off-the-shelf RTOS and are scheduled by the RTOS scheduler.
  - Preferred when there is enough resource due to its flexibility and maturity
- Interrupt-based multi-tasking
  - Applicable when off-the-shelf RTOS' are not suitable due to performance and resource constraints.
  - Also similar to how RTOS implements task management and task scheduling.

RTOS-Based Multi-Tasking

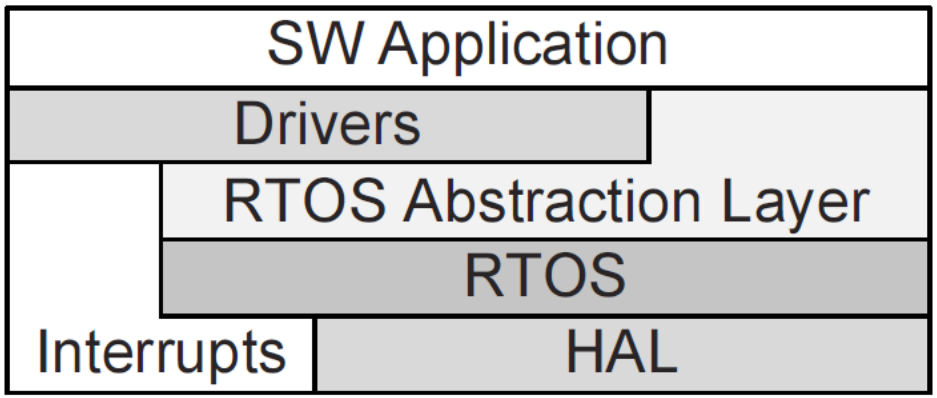


FIGURE 5.6 Software execution stack for RTOS-based multi-tasking (Gajski et al.)

- Off-the-shelf RTOS' are typically reliable and well-tested.
- Significant tool supports are available from the RTOS vendor.
- Highly configurable to reduce memory footprint.

The Software Stack

- HAL
  - Hide hardware differences from OS, e.g. different processors, interrupt controllers, timers
  - For OS to switch tasks, HAL could help to hide details for saving and restoring the processor's internal state.
- Interrupts provide synchronization with external devices
- RTOS provides services for task management, communication, and timing management.
- RTOS Abstraction Layer (RAL)
  - Provide a canonical OS interface for application portability, e.g. standardized APIs like POSIX.
  - Make the synthesis flow applicable to multiple RTOS' by decoupling synthesis and the target RTOS
- Drivers
  - Implement application-specific communication with external components using services provided at lower layers

Interrupt-Based Multi-Tasking

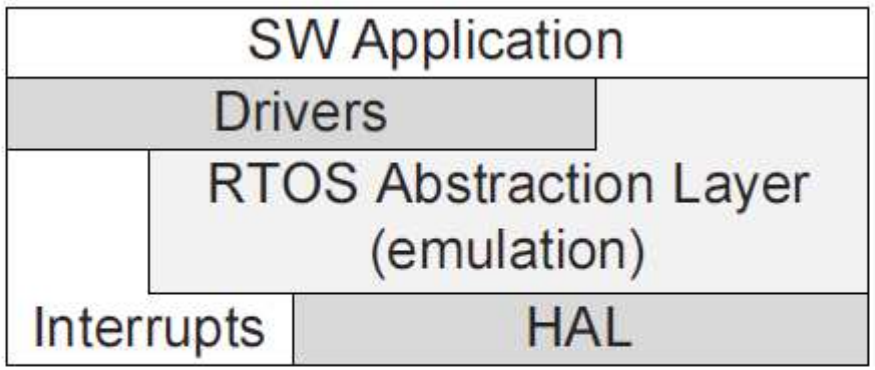


FIGURE 5.8 Software execution stack for interrupt-based multi-tasking (Gajski et al.)

- Suitable when there is not enough resource to accommodate an RTOS
- Work on a bare processor without any RTOS
- There is no RTOS layer while the RAL layer provide partial emulation.

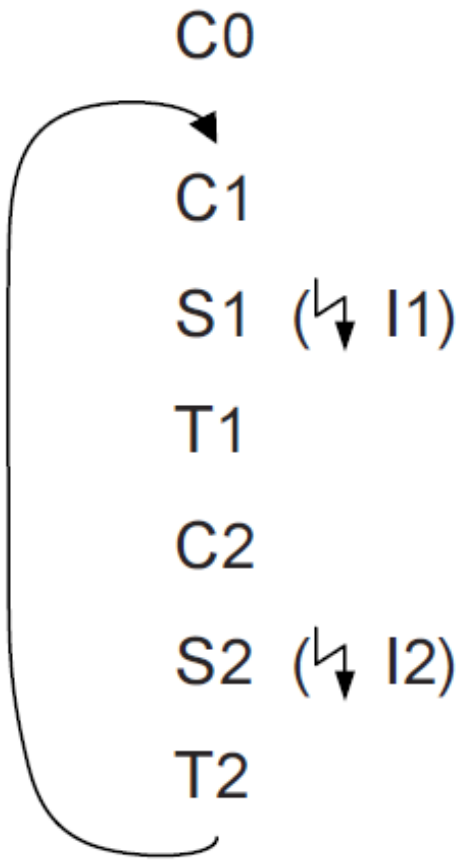
Scheduling and Processor State

- Execution flow of a task can be modeled as a FSM.
  - The state is a combination of processor state and memory state local to the task, i.e. the stack.
- Scheduling as saving and restoring processor state
  - Assume memory local to a task won't be modified by other tasks
  - Create a task: allocate memory for processor state storage and the stack
  - Suspend a task: save the processor state
  - Resume a task: restore the processor state
  - Terminate a task: release memory
- Combining with ISRs, this is how RTOS implements scheduling.
- However, this may not be suitable when there are a huge amount of processes.
  - Overhead of context switch.
  - Not enough memory for each task to have its own stack.

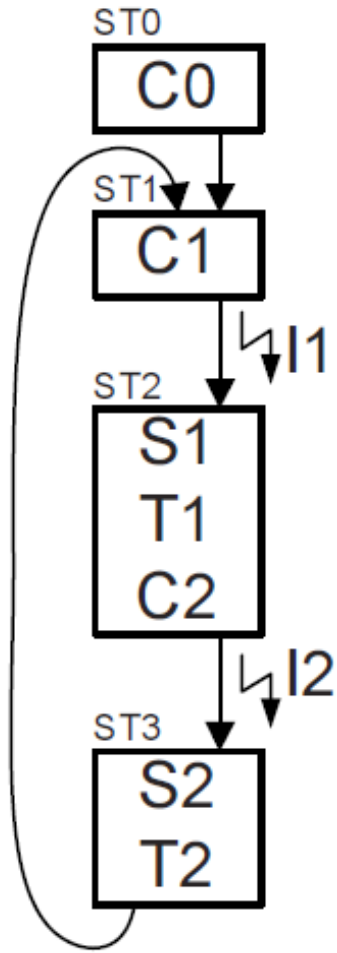
Specifying Tasks as FSMs

- The solution is to specifying tasks as FSMs explicitly.
  - As actors.
- Each task then manages its own state without referring to the processor state or the stack.
- Multiple tasks can then be scheduled to run "concurrently" on the same processor by interleaving their state transitions.
  - Executing times for state transitions should be constrained to achieve certain scheduling goals.
- If tasks are not already specified as actors, FSMs may be generated.

Example FSM Generation



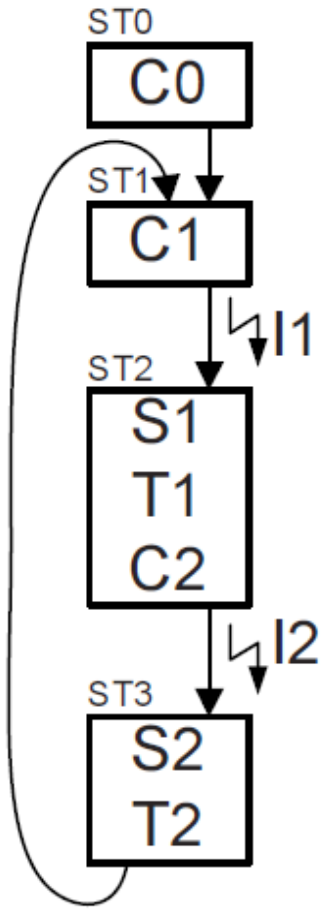
(a) Input



(b) Output

FIGURE 5.9 Interrupt-based multi-tasking example (Gajski et al.)

Interrupts-Based C Multi-Tasking



(b) Output

FIGURE 5.9

```
1  /* interrupt handler */
2  void intHandler_I1() {
3    release(S1);    /* set S1 ready */
4    executeTask0(); /* task state machine */
5  }
6  /* task state machine */
7  void executeTask0() {
8    do { switch(Task0.State) {
9         /* ... */
10     case ST1: C1(...);
11               Task0.State = ST2;
12     case ST2: if(attempt(S1)) T1_receive(...);
13               else break;
14               C2(...);
15               Task0.State = ST3;
16     case ST3: /* ... */
17   } } while (Task0.State == ST1);
18 }
```

LISTING 5.5 State machine implementation

(Gajski et al.)

External Communications and Driver Synthesis

- Tasks on different processors communicate via devices like network interfaces that attached to the processors.
  - Devices are accessed via drivers managed by OS.
- To implement various drivers, it is reasonable to follow the 7-layer model as the objective is to support communications.

Presentation Layer

```
1  typedef struct stReq {
2    long          startTime;
3    short         coeff1;
4    unsigned short base;
5  } tReq;
```

LISTING 5.7 User type definition

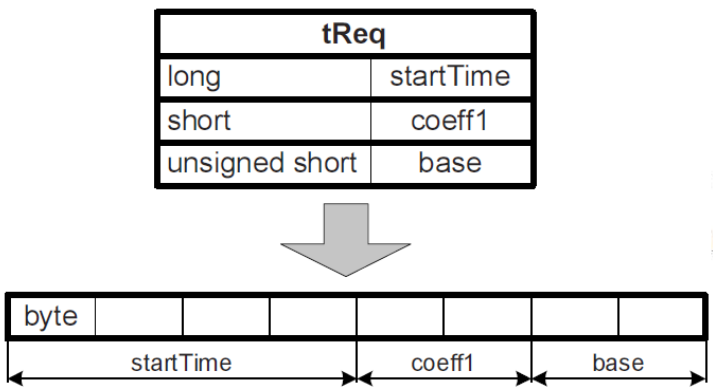


FIGURE 5.12 Marshalling example

```
1  void myCh_send(/* ...*/ *This, struct tReq *pD){
2    unsigned char *pB = This->buf;
3    htonlong(pB, pD->startTime);
4    pB += 4;
5    htonshort(pB, pD->coeff1);
6    pB += 2;
7    htonushort(pB, pD->base);
8    pB += 2;
9    DLink0_trans_send(/*...*/This->buf, 8);
10 }
```

LISTING 5.8 Marshalling code

- Use marshalling (and demarshalling) to create (and to retrieve data from) processor independent layout (Gajski et al.)
  - Host-to-network functions take care of byte endianness.
  - Marshalling/demarshalling code could be generated automatically.
- What if a more complicated data structure should be supported?

Transport Layer

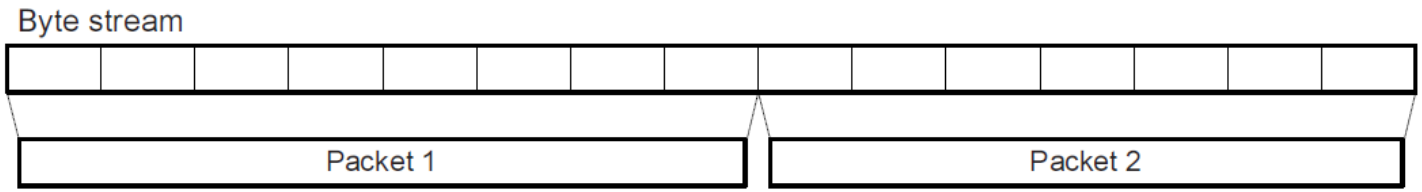


FIGURE 5.13 Packetization

```
DLink0_trans_send(void *pMsg, unsigned int len){
  unsigned char *pPos = pMsg;
  while(len) {
    unsigned long pktLen;
    /* length is minimum of max size and len */
    pktLen = min(len, CONFIG_PACKET_SIZE);
    DLink0_net_send(pPos, pktLen); /* transfer */

    len  -= pktLen; /* decr. transferred len */
    pPos += pktLen; /* advance pointer */
  }
}
```

LISTING 5.9 Packetization code example

(Gajski et al.)

- Utilize a low level service that supports only messages with fixed lengths via packetization.
- What if multiplexing is necessary?

Link/HAL Layer: Synchronization by Interrupts

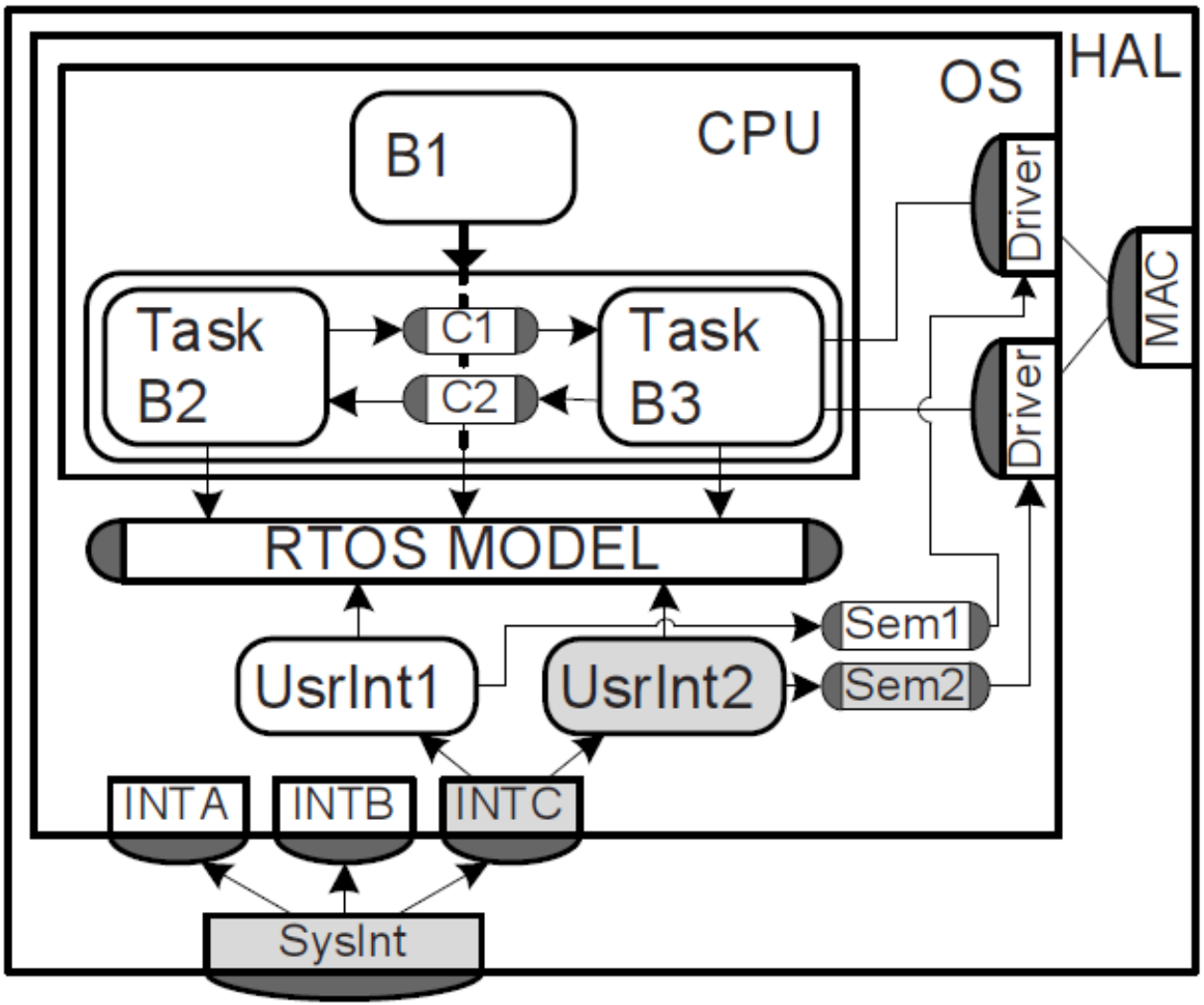


FIGURE 5.14 Chain for interrupt-based synchronization (Gajski et al.)

Events in Interrupt-Based Synchronization

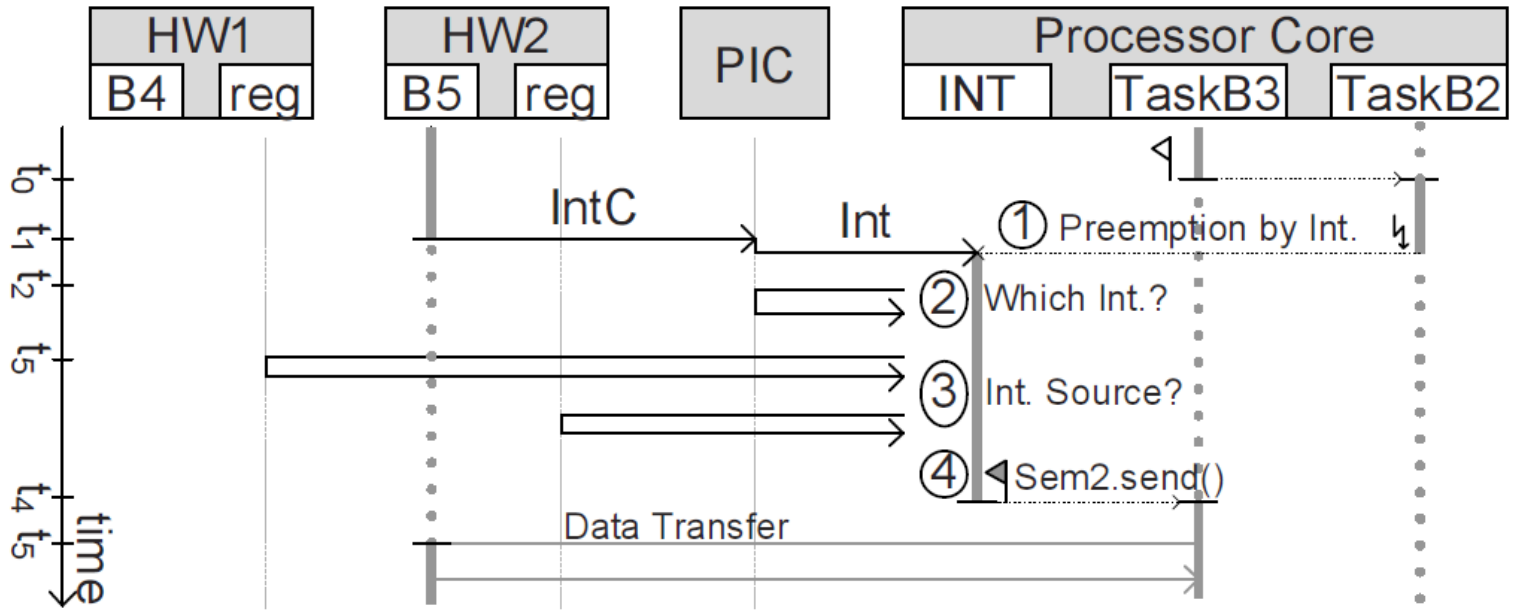


FIGURE 5.15 Events in interrupt-based synchronization

(Gajski et al.)

- Each driver component handles a set of events.
  - Hardware ISR preempts the current task.
  - SysInt queries PIC for detailed interrupt data.
  - INTC queries hardware status and triggers UsrInt2 that post the semaphore.

Link/HAL Layer: Synchronization by Polling

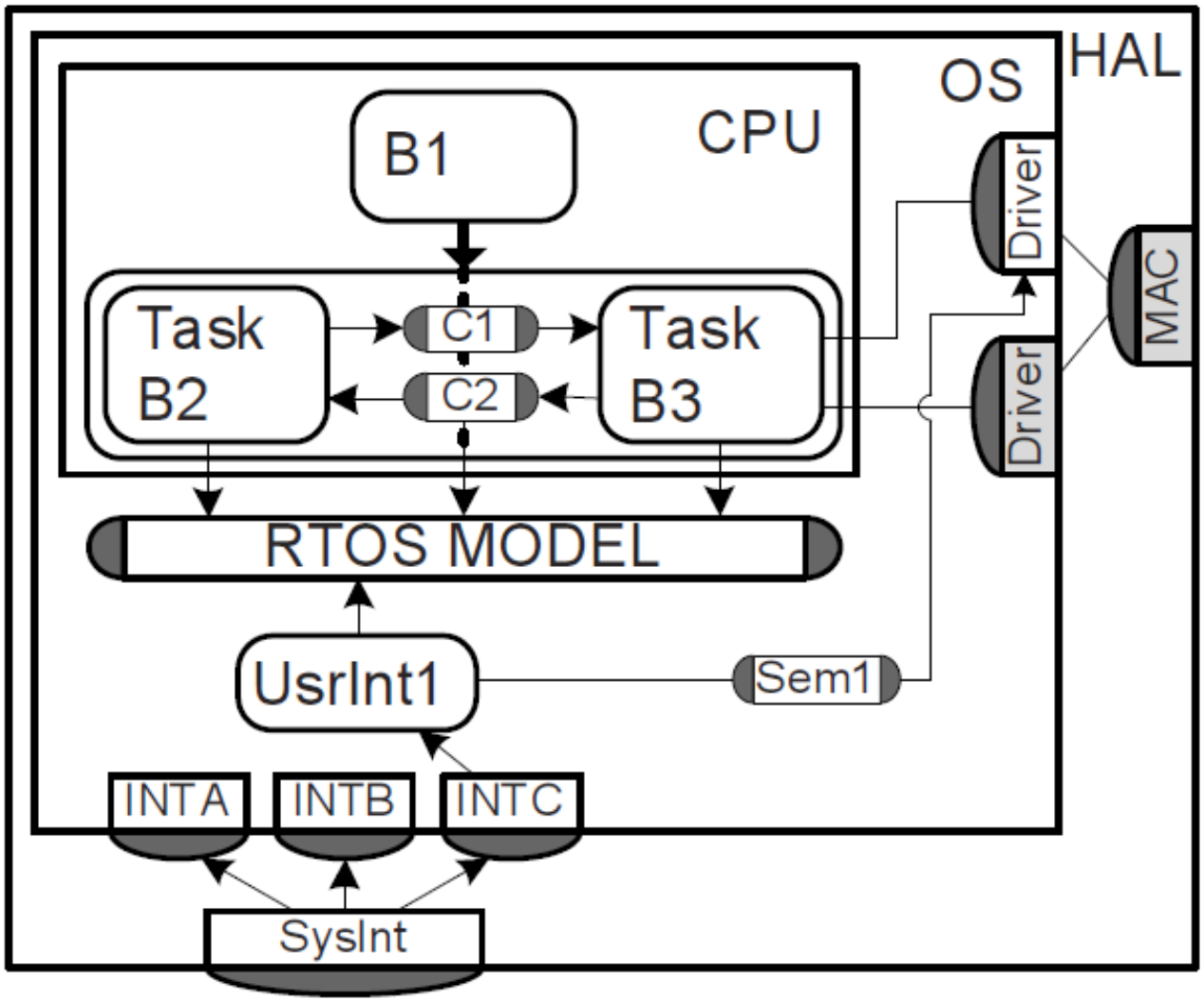


FIGURE 5.16 Polling-based synchronization (Gajski et al.)

Events in Polling-Based Synchronization

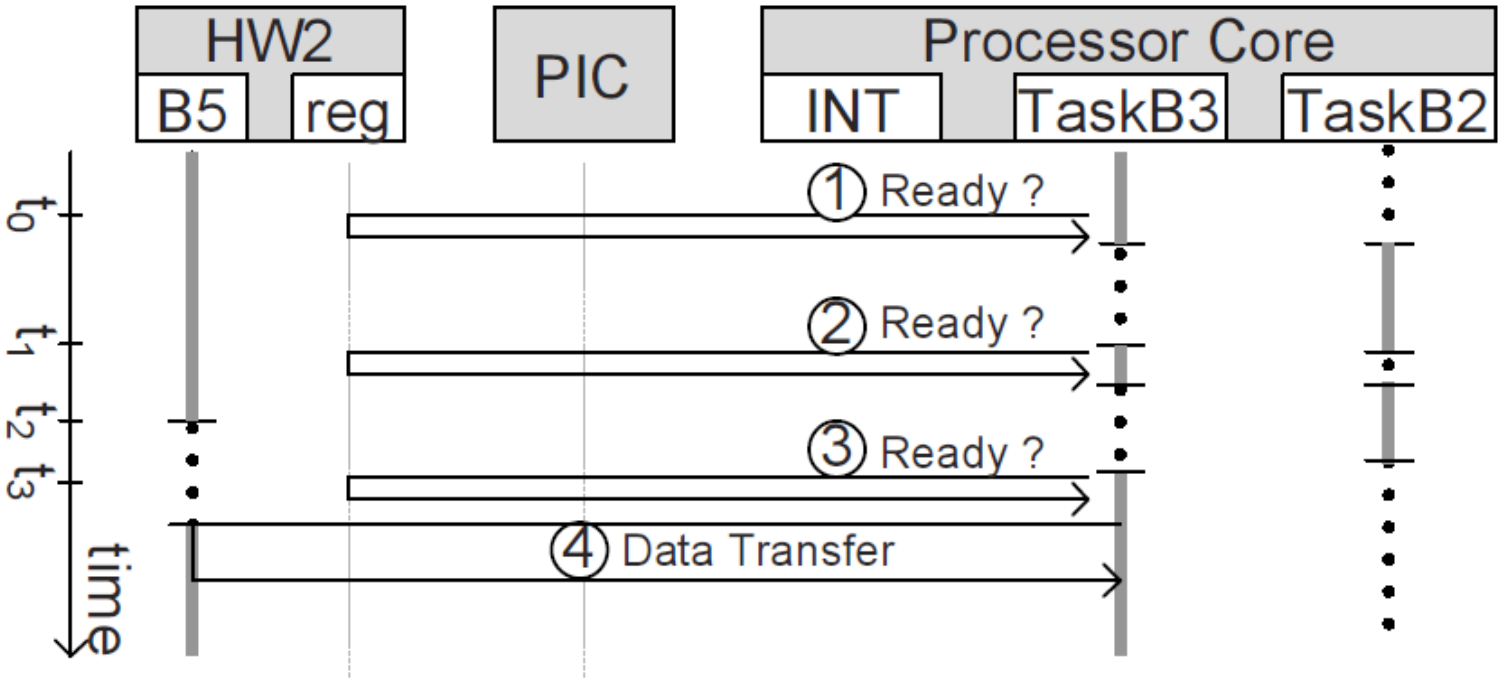


FIGURE 5.17 Events in polling-based synchronization

(Gajski et al.)

- The task repeatedly checks hardware status and yields if cannot proceed.
- How often should the task check?
  - More often: overhead in scheduling, waste processor cycles
  - Less often: the hardware may need to wait when ready

Binary Image Generation

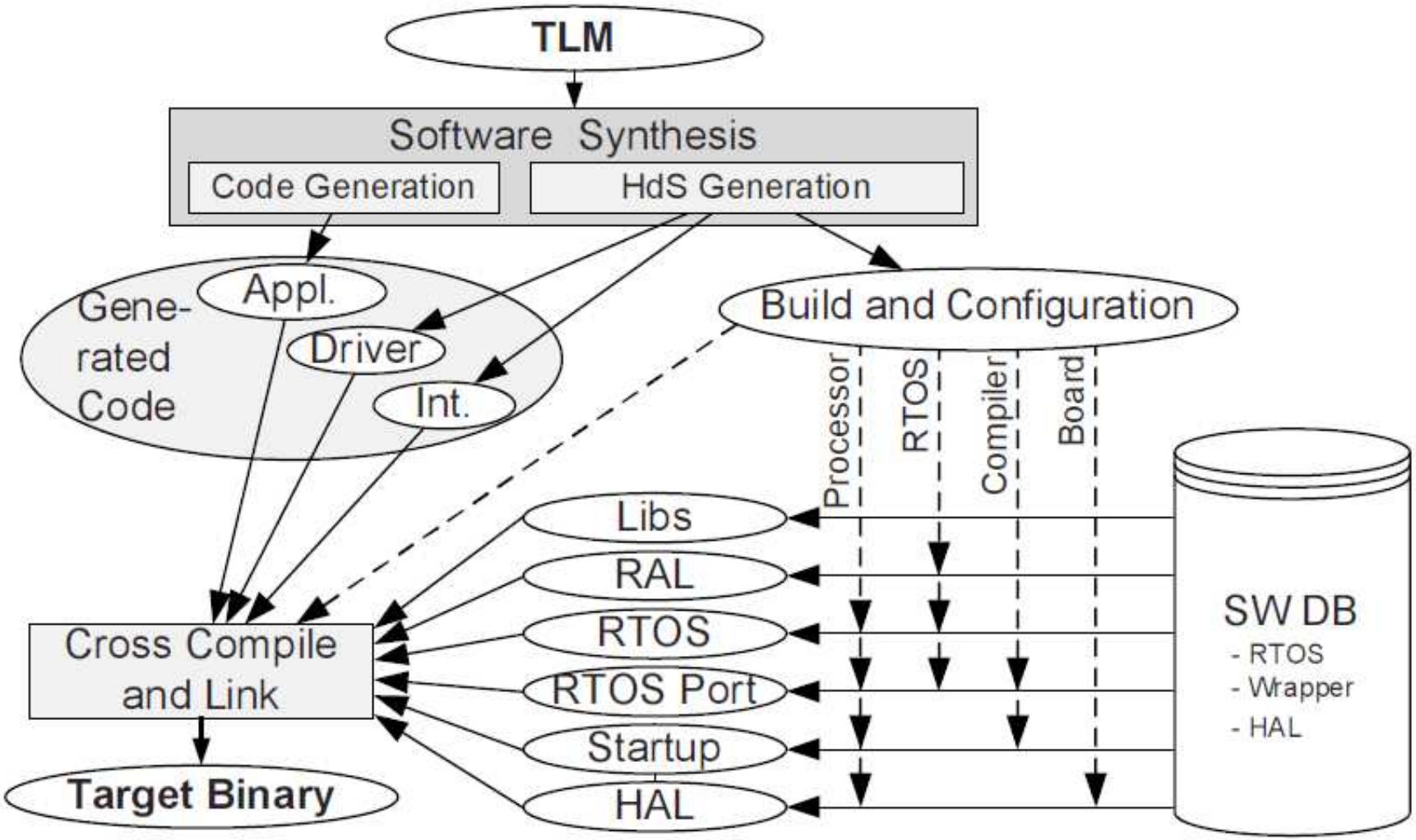


FIGURE 5.19 Binary image generation

(Gajski et al.)

Execution

- The produced binaries are downloaded onto the target platform and executed.
- The target platform may be implemented as an ASIC or using a FPGA prototyping platform.
  - Allow to validate cycle-accurate functionality.
  - Allow to validate signal timing in physical world.
- Alternatively, the binaries can be validated using a virtual platform.
  - Usually based on an instruction-set simulator (ISS) as cycle-accurate simulation will be too slow.
- Designers may utilize detailed feedback to further improve the system, and trigger the synthesis again for design iterations.
  - Automated synthesis speeds up the whole process so designers may explore more system implementations.

Virtual Platform

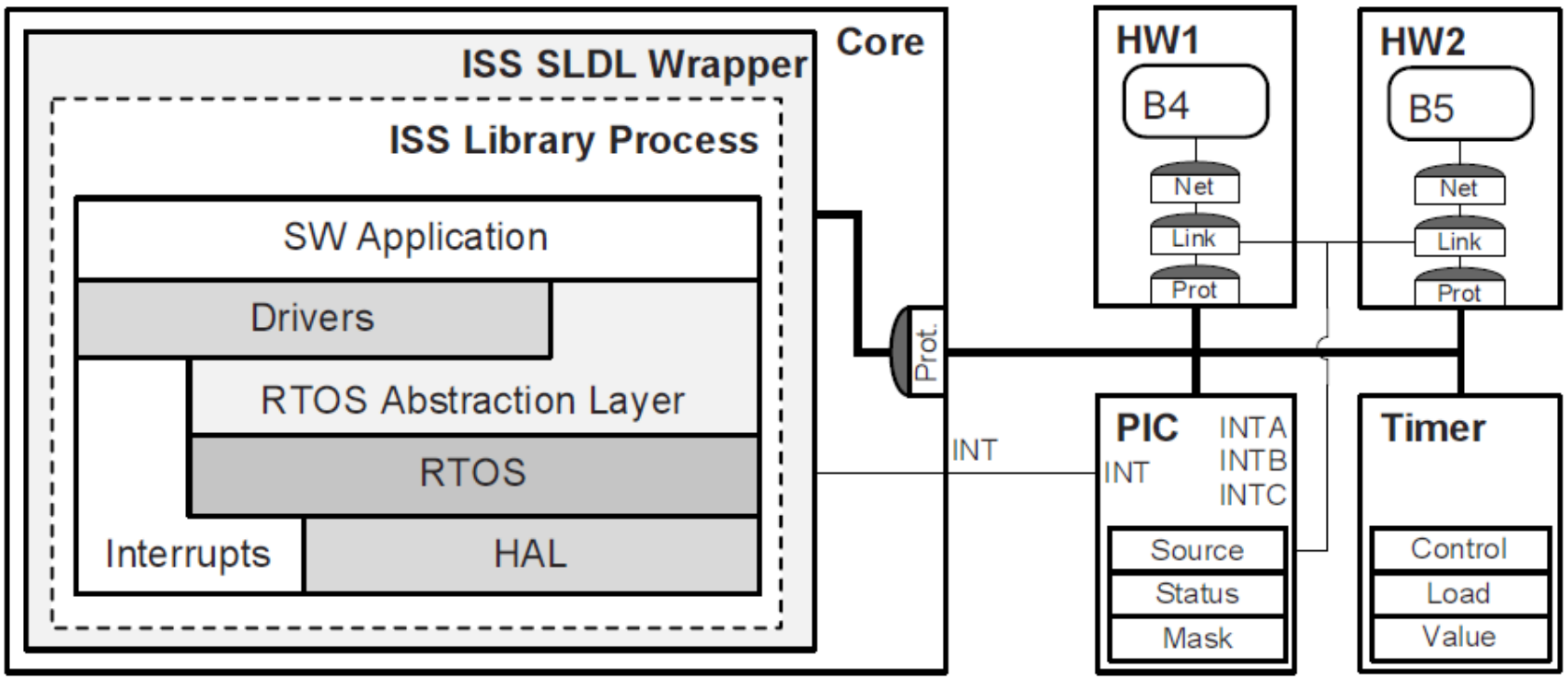


FIGURE 5.20 ISS-based Virtual platform

(Gajski et al.)