EE382V: Embedded System Design and Modeling

Lecture 10 – Computation Modeling & Refinement

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu
Lecture 10: Outline

• **Processor layers**
  • Application
  • Task/OS
  • Firmware
  • Hardware

• **Processor synthesis**
  • Software synthesis
  • Hardware synthesis
System-On-Chip Environment (SCE)

Specification

System Design
(Specify-Explore-Refine)

Design
Decisions

System models

TLM$_i$

Hardware
Synthesis

Software
Synthesis

Implementation Model

Hardware Synthesis

SW

DB

CPU

B1

B2

OS

Drv

ISR

Implementation Model

Computation
modeling and
refinement

TLM$_n$

RTL

DB

Mem

DSP

Bridge

B3

B4

B5

CPU

ISS

RTOS

HAL

Impl$_n$

HW

SW

DB

RTL

DB

HW$_n$.v

CPU$_n$.bin

CPU

ISS

RTOS

HAL

Impl$_n$

HW

Mem

DSP

Bridge

DSP Bus

OS+DRV

CPU Bus

Arbiter

B5

IP

B4

B3

B2
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

- **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

• **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
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 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

Simulated Dynamic Behavior

Unscheduled

Scheduled

Inaccuracy due to timing granularity
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)
RTOS Model Interface

• Canonical, target-independent API

```c
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);
};
```
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

```c
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);
    }
}
```

```c
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);
}
```

```c
void send() {
    t = os.pre_wait();
    wait(ack);
    os.post_wait(t);
}
```
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
• **External communication**
  - Software Drivers
    - Presentation, Session, Packeting
    - Synchronization (e.g. Interrupts)
  - TLM Bus model
    - User transactions
  - However, interrupts are unscheduled

```c
void send(...) {
    intr.receive();
    bus.masterWrite(0xA000, &tmp, len);
}
```
Processor Model: TLM Layer

- **Processor TLM**
  - 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
**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

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

**Features**

<table>
<thead>
<tr>
<th>Target approx. computation timing</th>
<th>Appl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task mapping, dynamic scheduling</td>
<td></td>
</tr>
<tr>
<td>Task communication, synchronization</td>
<td></td>
</tr>
<tr>
<td>Interrupt handlers, low level SW drivers</td>
<td></td>
</tr>
<tr>
<td>HW interrupt handling, int. scheduling</td>
<td></td>
</tr>
<tr>
<td>Cycle accurate communication</td>
<td></td>
</tr>
<tr>
<td>Cycle accurate computation</td>
<td></td>
</tr>
</tbody>
</table>

**Images:**
- Processor layers diagram
- Processor hardware and features diagram
Lecture 10: 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

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**
  - Reintegrated 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 10: Outline

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

• Processor synthesis
  ✓ Software synthesis
  • Hardware synthesis
High-Level Synthesis (1)

• Allocation

High-Level Synthesis (2)

- **Scheduling**

```
......
y = 3*x;
i = 0;
do {
    d += y * i;
i++;
} while (i < 10);
h = d + d;
......
```

BFM (PAM) ➔ Control flow ➔ SFSMD ➔ Data flow ➔ FSMD
High-Level Synthesis (3)

- **Binding**

Unmapped RTL

\[ y = 3 \times x \]
\[ i = 0 \]
\[ t = y \times i \]
\[ d = t \]
\[ i++ \]
\[ h = 2 \times d \]

Mapped RTL

\[ \text{b2} = \text{RF}[1] \]
\[ \text{b3} = \text{FU}(\text{b2}, 3) \]
\[ \text{RF}[2] = \text{bus3} \]
\[ \text{RF}[0] = 0 \]
\[ \text{b1} = \text{RF}[2] \]
\[ \text{b2} = \text{RF}[0] \]
\[ \text{b3} = \text{FU}(\text{b1}, \text{b2}) \]
\[ \text{RF}[3] = \text{bus3} \]

Structural RTL

SCE Interactive RTL Synthesis

RTL Allocation

RTL Scheduling & Binding
**SpecC RTL Modeling**

### RTL Modeling Example

```specc
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*
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;         // Accelleraa style 1
        d = Inport * e;    // (unmapped)
        Outport = a;
        goto S2;
      }

    }
  }
};
SpecC RTL Modeling

**Mapped RTL Example**

```c
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,     // input ports
    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 :        {
            RF[0]=RF[1]+RF[2];   // Accellera style 2
            RF[3]=Inport*RF[4];  // (storage mapped)
            Outport = RF[0];     // storage mapped
            goto S2;
        }
    }
}

buffered[CLK] bit[32] RF[4];       // register file
```

Source: R. Doemer
SpecC RTL Modeling

**Mapped RTL Example**

```cpp
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[4];        // 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] = 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
```
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[4];  // 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;   // Accellera style 4
        else goto S0;
      }
      S1 : { BUS0 = RF[1];         // (connection mapped)
        BUS1 = RF[2];
        BUS3 = ADD0(BUS0,BUS1);
        RF[0]= BUS3;
        ... goto S2;
    }
  }
};
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 10: 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