Lecture 10: Outline

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

- **Processor synthesis**
  - Software synthesis
  - Hardware synthesis
General Processor Micro-Architecture

- **Basic computation 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 (and power and …)**
Computation Modeling (1)

- Structural RTL models
  - Sub-cycle accurate

Software processor

Hardware processor

Computation Modeling (2)

- Behavioral RTL models (FSMD)
- Instruction-set simulation (ISS) models
  - Purely functional or micro-architectural

Instruction set simulation (ISS)
Computation Modeling (3)

- **Host-compiled models**
  - Source-level application model
  - Back-annotate timing and other metrics
  - Abstract OS and processor models
  - Transaction-level model (TLM) backplane
  - C-based discrete-event simulation kernel [SpecC, SystemC]

➢ Fast and accurate full-system simulation


Host-Compiled Computation Layers

- **Application**
  - Process execution (C code)
  - Execution timing

- **OS & processor**
  - 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
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

Retargetable Back-Annotation

- Back-annotation flow
  - Intermediate representation (IR)
    - Frontend optimizations [gcc]
    - IR to C conversion
  - Target binary matching
    - Cross-compiler backend [gcc]
    - Control-flow graph matching
  - Timing and power estimation
    - Micro-architecture description language (uADL) or RTL
    - Cycle-accurate timing
    - Reference power model [McPAT]
- Back-annotation into IR
  - Basic block level

Binary-to-Source/IR Mapping

- Compiler optimizations
  - Frontend
    - Control flow optimizations
  - Backend
    - Instruction scheduling/percolation
- Mismatches
  - Capture frontend by annotating at IR, not source
  - Establish binary-IR mapping for back-annotation

- Graph matching heuristic
  - Synchronized, recursive depth-first traversal
    - Compatibility: loop and branch nesting levels
    - Cost: sum of unmatched nodes in subgraphs rooted at node
    - Return least-cost mapping between all successors (incl. skips)
  - Resolve ambiguities using debug information

Timing/Energy Characterization

- Basic block characterization
  - Execution depends on state
    - Pipeline stalls in case of hazards
    - Pipeline overlaps in multi-issue
  - Pairwise characterization
    - Over all immediate predecessors
    - Across function hierarchy
  - Timing & energy
    - First-to-last instruction fetch time
    - Resource utilization statistics

- Back-annotation into IR
  - Path-dependent metrics
    - Capture static branch prediction
Source-Level Simulation: Speed

- **Automatic timing and energy back-annotation**
  - Telecom & security applications [MiBench]
    - SHA, ADPCM, CRC32 & custom Eratosthenes’ Sieve
    - Small and large data sets, 10 to 700 million instr.
  - One-time back-annotation
    - 3min. to 3s BA runtime

  - Back-annotated source vs. traditional ISS
    - 2000 MIPS vs. 0.8 MIPS
    - Close to native source-level speeds

Source-Level Simulation: Accuracy

- **Source-level power and performance simulation**
  - Single- (z4-like) and dual-issue (z6-like) e200 PowerPC
    - No cache, static branch prediction
  - Compare against cycle-accurate reference [ISS+McPAT]
    - >99% average timing and energy accuracy @ 2000 MIPS

  - Integrate back-annotation of other metrics
    - Performance, energy, reliability, power, thermal (PERPT)
OS Modeling

- **High-level RTOS abstraction**
  - Specification is fast but inaccurate
    - Native execution, truly concurrent model
  - Traditional ISS-based validation infeasible
    - Accurate but slow (esp. in multi-processor context), requires full binary

➤ Model of operating system (task interleaving in time)
  ➤ 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

Operating System 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
Abstract RTOS Model

- Emulate the sequential execution of concurrent tasks
  - Task scheduler
    - Maintain task queues, determine task(s) to run & perform context switch
  - Timing model
    - Simulate back-annotated task delays, call scheduler to allow for preemptions

RTOS Model Implementation

- RTOS model
  - OS, task, event management
    - Descriptors & queues
  - Context switching
    - Block all but active task on SLDL level
  - Scheduling
    - Select and dispatch task based on algorithm
  - 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**

```
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);
};
```

- **OS management**
- **Task management**
- **Event handling**
- **Delay modeling**

#### 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);
  }
}
```
Simulated Dynamic Behavior

OS Modeling Results

• Configurable, generic and flexible OS model
  • Configurable scheduling strategies and parameters
    – Round-robin or priority-based scheduling
  ➢ Scheduling exploration
    – Artificial periodic task sets, uniformly distributed periods & utilizations
    – Back-annotation at 1μs, 10μs, 100μs, or 1000μs granularity
    – Dual-core MIPS Malta reference platform w/ Linux 2.6 SMP kernel [OVP]
Speed and Accuracy Tradeoffs

- Errors in discrete preemption models
  - Potentially large preemption errors
    - Not bounded by simulation granularity

- Automatic Timing Granularity Adjustment (ATGA)
  - Observe system state to predict preemption points
  - Dynamically and optimally control timing model
  - Transparently integrated into OS model
    - Eliminate preemption errors

ATGA Model Execution Example

- Priorities: \( T_{\text{Int}} \), \( T_H \), \( T_M \), \( T_L \)
- States: Sleep, Idle, Ready
- Time points: \( t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7 \)
- OS Mode: Predictive, Fall-back, Predictive

ATGA Results

- **ATGA OS model**
  - Artificial periodic task sets
  - Vs. conventional model at varying granularity

- **Reference platform [OVP]**
  - MIPS-Malta
  - Linux 2.6

- **Optimally navigate speed vs. accuracy tradeoff**
  - As fast as coarse grain (100μs)
  - As accurate as fine grain (1μs) simulation

---

Operating System Layer

- **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
Hardware Abstraction Layer (HAL)

- **External communication**
  - Software Drivers
    - Presentation, session, network communication layers
    - Synchronization (interrupts)
  - Hardware/software boundary
    - Low-level HW access
    - Bus drivers and interrupt handlers
    - Canonical HW/SW interface
- **External interface**
  - Bus transactions (TLM)
  - Interrupt trigger

```java
void send(...) {
    intr.receive();
    bus.masterWrite(0xA000, &tmp, len);
}
sample.send(v1);
```

Hardware Layer (1)

- **Processor TLM**
  - HW interrupt handling
    - Interrupt logic
      - Suspend user code
    - Interrupt scheduling
      - Priority, nesting
  - Peripherals
    - Interrupt controller
    - Timers
  - TLM bus model
    - Bus transactions
Hardware Layer (2)

- **Cache modeling**
  - Pure behavioral modeling
    - Tag state
    - Hits/misses
    - Replacement policy
  - Integrated into back-annotation
    - Called with accessed address trace
    - Update cache state
    - Return delay penalties
  - Implemented as SpecC channel
    - < 200 lines of code


Cache-Aware Back-Annotation

- **Host-compiled functional model**
  ```c
  void f(void) {
      BB1: ...
      os.wait(BB1_DELAY);
      if (c) goto BB3;
      BB2: a[i][j] += sum;
      alist[__idx] = A_BASE + 4*(i*A_WID+j);
      ...
      miss = cache.upd(__alist, __idx);
      os.wait(BB2_DELAY + miss);
      BB3: ...
      os.wait(BB3_DELAY);
      drv.write(res);
  }
  
  void main(void) {
      os.task_create(&f, "Task 1", PRIO0);
  }
  ```

- **Hybrid timing model**
  - Static + dynamic
    - Runtime cache model
Hardware Layer (3)

- **Bus-functional model (BFM)**
  - Pin-accurate processor model
    - Timing-accurate bus and interrupt protocols
  - Bus model
    - Pin- and cycle-accurate
    - Driving and sampling of bus wires

Processor Models

- **Processor layers**
  - Application
    - Native, host-compiled C
    - Back-annotation
  - OS
    - OS model
    - Middleware, drivers
  - HAL
    - Firmware
  - Processor hardware
    - Bus interfaces
    - Interrupts
    - Cache

Processor Model Example

- 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 Model 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)
Multi-Core Models

- Multi-core OS model
  - SMP scheduler model
    - Global or partitioned queue
  - Configurable parameters
    - Number of cores
    - FIFO, round-robin, priority-based scheduling policies
    - Priorities, affinity, time slice (for round-robin)

- Multi-core processor model
  - Multi-core interrupt handling chain models
    - Interrupt handlers & tasks
    - Configurable generic interrupt controller (GIC) model
  - TLM bus interfaces


Multi-Core OS Model

- Global or partitioned SMP scheduling
  - Replicated or shared Ready, Idle, Sleep & Wait queues
- Processor suspension and interrupt handling
  - Interrupt handlers as highest-priority OS-internal tasks
Multi-Core Processor Model

- Emulate the processor hardware/software interface
  - OS & hardware abstraction layers
    - I/O drivers, interrupt suspension
  - Hardware layer
    - TLM bus interface, interrupt routing logic & interrupt controller models

Dual-Core Processor Model Example

- Errors in preemption model due to discrete timing
  - Integrate multi-core ATGA approach
Multi-Core ATGA Model

- **Enhanced fallback mode check**
  - Only fall back when ext. event triggers interrupt task with higher priority than current task
    - Potential task switch
    - Allow for delayed interrupts otherwise

- **Model inter-core interrupt notifications**
  - Adjust predicted times or switch to fallback
  
  - **Accurate interrupt response times while maintaining speed**
  - But: high-priority interrupt-driven tasks degrade performance

Multi-Core Cache Model

- **Application model**
  - Per core memory access list
    - Address, mode, time stamp

- **Cache interface**
  - Hardware layer of processor model

- **Generic cache model**
  - Emulate cache state
    - Only tags, no values
    - Return hit & miss info
  - Parameterizable
    - Cache size, line size, associativity, replacement & write-back policy

Multi-Core Cache Simulation

- Directly committing accesses in simulation order
  - Globally out-of-order in discrete timing model

- Delayed reordering of aggregated requests
  - Multi-Core Out-of-Order Cache (MOOC) model

- 100% accurate results @ coarse-grain speeds
MPCSoC Platform Simulation

- Cellphone baseband MPCSoC
  - Design space exploration: mapping & scheduling

- Full-system simulation in close to real time
  - 1400 MIPS at > 99% timing accuracy
Lecture 9: 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

---

**Lecture 9: Outline**

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

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

- C-to-RTL high-level synthesis (HLS)
  - Allocation, scheduling, binding

\[
\begin{align*}
  & y = 3 \times x; \\
  & i = 0; \\
  & \text{do} \\
  & \quad \text{d} += y \times i; \\
  & \quad i++; \\
  & \text{while} \ (i < 10); \\
  & \quad h = d + d; \\
  & \quad ...
\end{align*}
\]


SCE Interactive RTL Synthesis
Modeling of Hardware in SoC Design

- **RTL Modeling**
  - State modeling: *Accellera RTL Semantics Standard*
    - **Style 1: unmapped**
      - \( a = b \cdot c; \)
    - **Style 2: storage mapped**
      - \( R1 = R1 \cdot RF2[4]; \)
    - **Style 3: function mapped**
      - \( R1 = \text{ALU1}(\text{MULT}, R1, RF2[4]); \)
    - **Style 4: connection mapped**
      - \( \text{Bus1} = R1; \)
      - \( \text{Bus2} = RF2[4]; \)
      - \( \text{Bus3} = \text{ALU1}(\text{MULT}, \text{Bus1}, \text{Bus2}); \)
    - **Style 5: exposed control**
      - \( \text{ALU_CTRL} = 011001b; \)
      - \( \text{RF2_CTRL} = 010b; \)
      - ...


---

### SpecC RTL Modeling

#### RTL Modeling Example

```spec
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,   // Start
    signal out bit[31:0] Outport, // output ports
    signal out bit[1] Done)    // 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

```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 : {
                a = b + c;         // state actions
                d = Inport * e;    // (register transfers)
                Outport = a;
                goto S2;
            }
        }
    }
}
```

Source: R. Doemer

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 : {
                a = b + c;         // state actions
                d = Inport * e;    // (register transfers)
                Outport = a;
                goto S2;
            }
        }
    }
}
```

Source: R. Doemer
SpecC RTL Modeling

Mapped RTL Example

```c
behavior FSM 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,  // state input
    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
        
        if (RST) { goto S0; }          // reset actions
        S0 : { if (Start) goto S1; else goto S0; }

        S1 : { RF[0] = 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

```c
behavior FSM 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,  // state input
    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
        
        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)
                BUS2 = ADD0(BUS0,BUS1);
                RF[0] = BUS0;
                ...
                goto S2; }
    }
}
```

Source: R. Doemer
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,
  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 :

  if (RF_CTRL = 011000b) // Accellera style 5
    ADD0_CTRL = 01b;    // (exposed control)
  MUL0_CTRL = 11b;

  ...

  goto S2;

};
```

Source: R. Doemer

Lecture 10: Summary

- **Host-compiled computation 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