RTOS Modeling for System Level Design

Design, Automation and Test in Europe Conference and Exhibition (DATE’03)

Andreas Gerslauer, Haobo Yu, Daniel D. Gajski

2010. 1. 20

Presented by Jinho Choi

ⓒ KAIST SE LAB 2010
Introduction (1/2)

- Problems of systems-on-chip design
  - Design complexity
  - Time-to-market pressure

- Raising abstraction level to solve the problem
  - SLDL(System Level Design Language)
    - SystemC, SpecC
Motivation

- Limitation of most SLDLs
  - No support for modeling the dynamic real-time behavior
- Dynamic real-time behavior
  - Only support by a RTOS (Real Time Operating System)

Proposal

- Design flow with RTOS model for system level
  - Capturing the RTOS behaviors in system level models
  - Written on top of SpecC language
  - Support all the key concepts in RTOS
  - Only a minimal modeling effort
  - Evaluating a system design at early design space exploration
SpecC

- A system-level description and specification
- A superset of ANSI-C
- Program is set of behaviors, channels, and interfaces
- Execution starts from behavior `Main.main()`

```
/* HelloWorld.c */
#include <stdio.h>

void main(void)
{
    printf("Hello World!\n");
}
```

```
// HelloWorld.sc
#include <stdio.h>

behavior Main
{
    void main(void)
    {
        printf("Hello World!\n");
    }
};
```
SpecC (cont’d)

```c
interface I1
{
    bit[63:0] Read(void);
    void Write(bit[63:0]);
};

channel C1 implements I1;

behavior B1(in int, I1, out int);

behavior B(in int p1, out int p2)
{
    int v1;
    C1 c1;
    B1 b1(p1, c1, v1),
        b2(v1, c1, p2);

    void main(void)
    { par { b1.main();
            b2.main();
        }
    }
};
```

Behavior        Ports        Channel        Interfaces

Child behaviors

Variable

© KAIST SE LAB 2010
SpecC (cont’d)

Sequential execution

**B_seq**

- b1
- b2
- b3

FSM execution

**B_fsm**

- b1
- b2
- b3
- b4
- b5
- b6

Concurrent execution

**B_par**

- b1
- b2
- b3

Pipelined execution

**B_pipe**

- b1
- b2
- b3

---

```c
behavior B_seq {
    B b1, b2, b3;
    void main(void) {
        b1.main();
        b2.main();
        b3.main();
    }
}

behavior B_fsm {
    B b1, b2, b3,
    b4, b5, b6;
    void main(void) {
        fsm {
            b1:{...}
            b2:{...}
        }
    }
}

behavior B_par {
    B b1, b2, b3;
    void main(void) {
        par(b1.main);
        b2.main();
        b3.main();
    }
}

behavior B_pipe {
    B b1, b2, b3;
    void main(void) {
        pipe(b1.main);
        b2.main();
        b3.main();
    }
}
```
System-Level Design Flow (2/4)

- **Specification Model**
  - Behavioral description of the system
  - Free of any implementation details
    - Without any implications about the structure of the implementation
  - No notion of time

[Diagram of system-level design flow with nodes and connections labeled b1, b2, b3, v1, v2, v3, c1.]

© KAIST SE LAB 2010
System-Level Design Flow (3/4)

- Architecture model
  - The result of mapping behaviors onto actual processing elements
Implementation model
- The output of the backend design process
- A structural description of the micro-architecture of each processor at the RTL
High-level RTOS abstraction

- Model standard RTOS concepts
  - Multi-tasking, time-sharing, preemption
  - Real-time scheduling
  - Task synchronization, task communication
interface RTOS {

void init(void);
void start(int sched_alg);
void interrupt_return(void);

proc task_create(char *name, int type,
    sim_time period, sim_time wcet);
void task_terminate(void);
void task_sleep(void);
void task_activate(proc tid);
void task_endcycle(void);
void task_kill(proc tid);
proc par_start(void);
void par_end(proc p);

evt event_new(void);
void event_del(evt e);
void event_wait(evt e);
void event_notify(evt e);

void time_wait(sem_time nsec);
};
RTOS Model (3/7)

- Model refinement

Unscheduled model

Architecture model
Task refinement

Step 1.
- Convert behaviors in the specification into RTOS-based tasks

```c
behavior B2() {
    void main(void) {
        ...
        waitfor(500);
        ...
    }
}
```

Unscheduled model

```c
behavior task_B2(RTOS os) implements Init {
    proc me;
    void init(void) {
        me = os.task_create("B2", APERIODIC, 0, 500);
    }
    void main(void) {
        os.task_activate(me);
        ...
        os.time_wait(500);
        ...
        os.taskTerminate();
    }
}
```

Architecture model
RTOS Model (5/7)

- Task refinement (cont’d)
  - Step 2
    - Create child tasks in a parent task

Unscheduled model

```
...  
par  
{    
  b2.main();
  b3.main();
}  
...  
```

Architecture model

```
...  
  task_b2.init();
  task_b3.init();
  os.par_start();
par {    
  task_b2.main();
  task_b3.main();
}  
  os.par_end();
...  
```
Synchronization refinement

- All events primitives are replaced with event handling routines of the RTOS model

```c
channel c_queue() {  
    event eRdy, eAck;
    void send(...)
    {
        ...
        notify eRdy;
        wait(eAck);
        ...
    }
};
```

```
channel c_queue(RTOS os) {  
    evt eRdy, eAck;
    void send(...)
    {
        ...
        os.event_notify(eRdy);
        os.event_wait(eAck);
        ...
    }
};
```

Unscheduled model

Architecture model
RTOS Model (7/7)

- Preemptive multi-task system modeling

Unscheduled model

Architecture model
The design of a voice code

- Two tasks for encoding and decoding
- Motorola DSP56600 processor
- A small custom RTOS

---

Lines of Code 13,475 15,552 79,096

Execution time 24.0 s 24.4 s 5 h

Context switches 0 326 326

Transcoding delay 9.7 ms 12.5 ms 11.7 ms
A RTOS model for system level design

- Not require any specific language extensions
- Support all the key concepts in RTOS
- Only a minimal modeling effort
- Validate the dynamic real-time behavior of multi-task systems in the early stage of system design
  - Providing accurate results with minimal overhead
Discussion

❖ Pros
■ Early validation of dynamic scheduling effects at system design
■ The first attempt to model RTOS features in system design

❖ Cons
■ Is it useful to know the number of context switch?
■ Hard to estimate the accurate system behavior.