Software Synthesis for System on Chip

Haobo Yu

Ph.D. Final Defense
Information and Computer Science

Committee Members:
Professor Daniel D. Gajski
Professor Rainer Doemer
Professor Tony Givargis

Outline

- Introduction
- Related work
- RTOS scheduling refinement
- Code generation
- RTOS targeting
- Experimental results
- Conclusions

Introduction

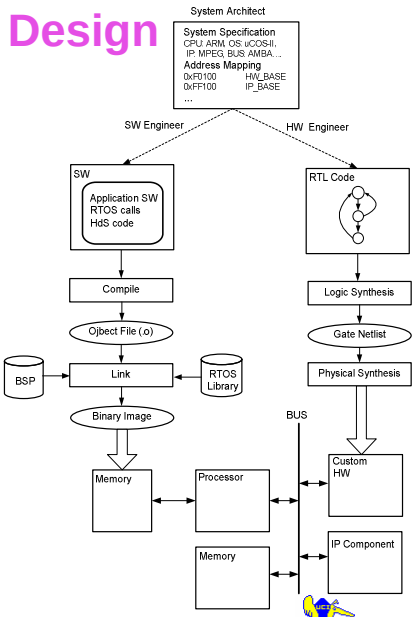
- System level design
  - System level description languages (SpecC, SystemC)
  - High level system models for exploration and synthesis
- Increasing significance of embedded SW in SoC
  - Embedded processors are widely used in SoC design
  - Typical SoC contains processor, HW and communication arch.
  - Complex SW is needed to drive the system
  - 50-70% of SoC design process is used for SW development (source: Virtual Socket Interface Alliance)
- New method for creating embedded software for SoC

Embedded Software for SoC

- Concepts
  - Application software
  - Real time operating system (RTOS)
  - Hardware dependent software (HdS)
- Development Tool
  - Processor IP vender (ARM:ADS)
  - RTOS venders (WindRiver:Tornado,GreenHill: MULTI)
  - FPGA tool (Altera:SOPC Builder, Xilinx:EDK)
  - DSP tool (MathWorks:MATLAB, Cadence:SPW)

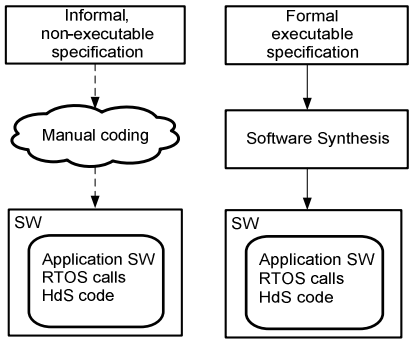
Embedded SW Design

- System Architect
  - Write system spec.
  - Allocate processors
  - Select HW / IP
  - HW / SW communication
  - Address allocation
- HW Engineer
  - Write RTL code
  - Logic synthesis
  - Layout, place / route…
- SW Engineer
  - Write application code
  - Write drivers
  - Compile / link / debug



Manual vs. Synthesis

- Manual Coding
  - Need low level HW information
  - Tedious and error-prone
  - Target specific & lack portability
  - Updating is hard
- Software Synthesis
  - From a high level functional specification to code implementation
  - Different from software compilation

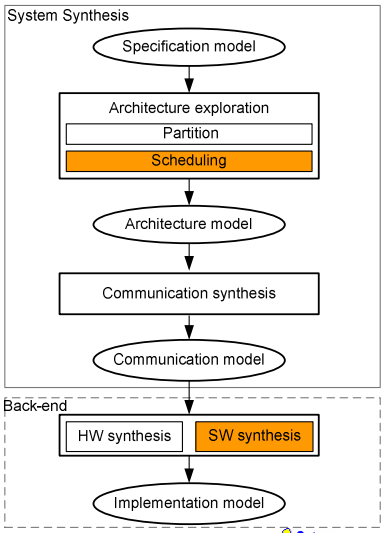


Related work

- RTOS Modeling
  - Specific target implementations [Tomiyama01]
  - Proprietary language & simulation engine [Desmet00]
- Software Synthesis and Code Generation
  - From abstract model (UML) [Rational]
  - From graphical finite state machine (StateCharts) [Harel90]
  - From synchronous programming languages (Esterel) [Boussinot91]
  - Reactive real time systems (POLIS) [Baladrin97]
  - Software scheduling [Cortadella00]
  - OS generation and application targeting [Gauthier01]
  - Redefinition and overloading of SystemC class [Herrera03]
  - Substituting SystemC modules with C structures [Groetker03]

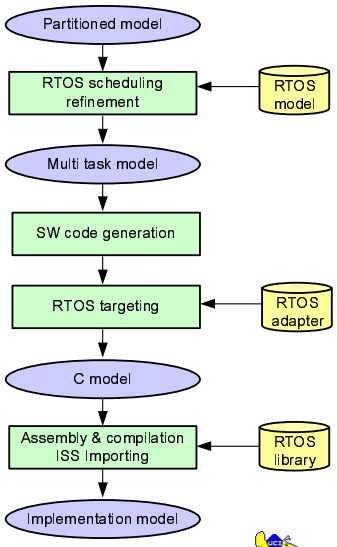
SW Synthesis in System Level Design

- Architecture exploration
  - Schedule SW behaviors
  - Create SW tasks
  - Evaluate different scheduling algorithms
- Back-end
  - Generate C code
  - Generate bus drivers
  - Target for RTOS
  - Compile & Link



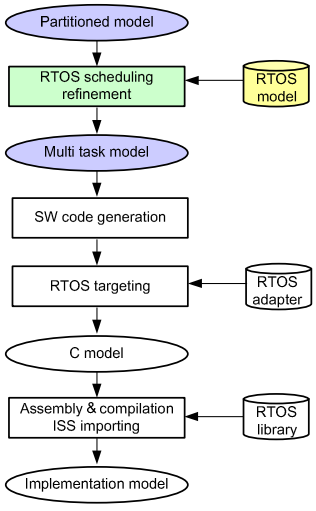
Software Synthesis Flow

- Two intermediate models
  - *Multi task model*
  - *C model*
- Four model refinement steps
  - Scheduling refinement
  - Code generation
  - RTOS targeting
  - Compiling & importing
- Three libraries
  - RTOS model library
  - RTOS adapter library
  - Target RTOS library (vendor)



Scheduling Refinement

- Behaviors need scheduling
  - Static scheduling
  - Dynamic scheduling
- Dynamic scheduler
  - Actual RTOS is available only for final implementation
  - Model RTOS in SpecC
  - Provide general task management functionality
- Scheduling refinement
  - Behaviors => SW tasks
  - Multi task model
  - Scheduling algorithm



Scheduling Refinement Steps

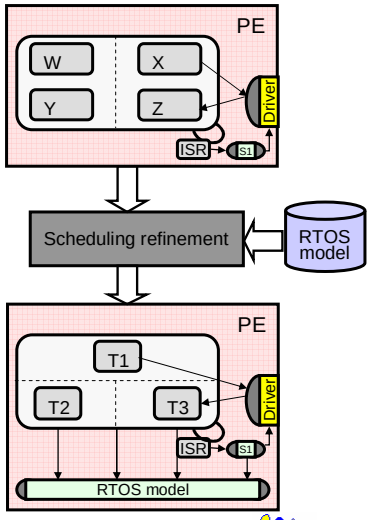
- Insert RTOS Model
  - RTOS model instantiation
  - Create main task
- Task Refinement
  - Create software tasks from parallel behaviors
- Synchronization Refinement

  os.pre_wait();

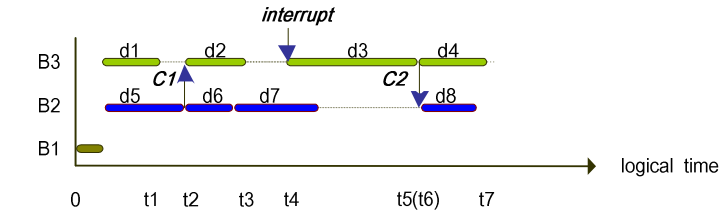
  wait event => *wait event;*

  os.post_wait();
- Preemption Point Creation

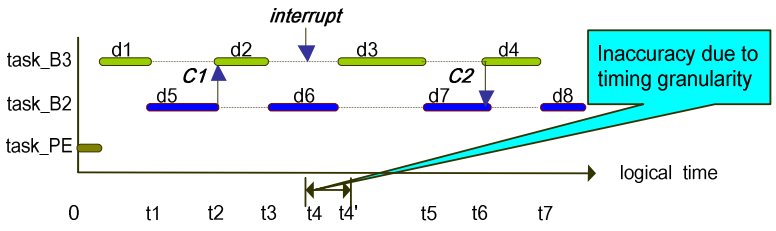
  waitfor(T) => *os.time_wait(T);*



Scheduling Refinement Example



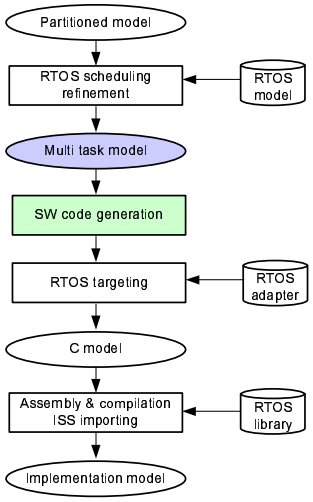
(a) unscheduled model



(b) scheduled model

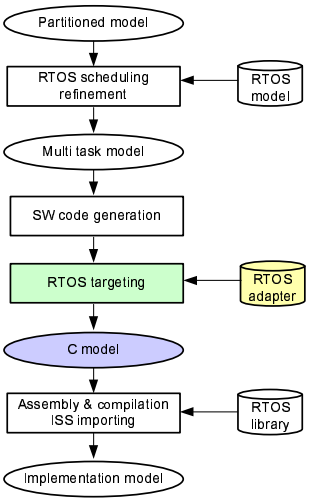
C Code Generation From SpecC

- Generate C code for each task
  - Hierarchical behaviors / channels
  - Most compiler accept C as input
- SpecC constructs
  - Behaviors
  - Channels
  - Interfaces
  - Methods
  - Hierarchy
  - Ports / Port mapping
  - Instantiation / Initialization
- C constructs
  - C struct
  - Functions
  - Pointers

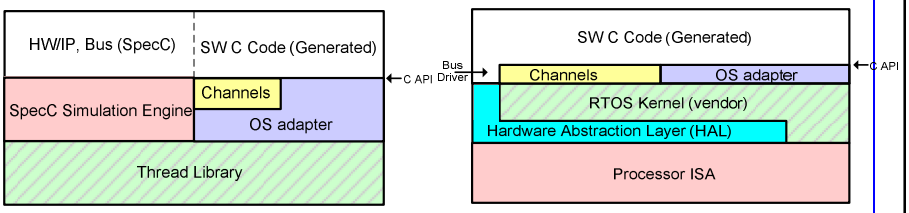


RTOS Targeting

- Needs OS support for C code
  - Task management API
  - Task communication API
  - Task synchronization API
- Create code for each platform
  - Knowledge about target RTOS API
  - Lacks portability
- A general RTOS API set
  - Insert general RTOS API to C code
  - Implemented by RTOS adapter
  - C code can be imported to the system model
  - *C Model*



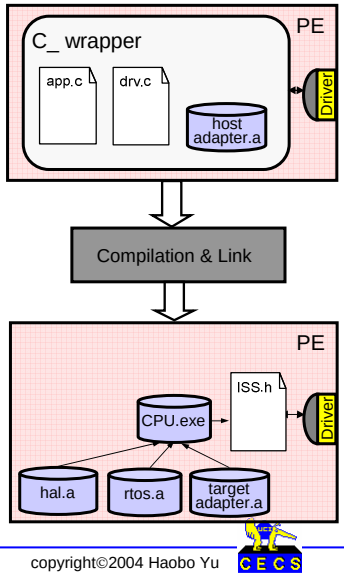
Host and Target RTOS Adapters



- Host RTOS adapter
  - Build on top of thread library
  - Link against SpecC simulation engine
  - Enable C code co-simulate with SpecC
- Target RTOS adapter
  - Middleware between application and RTOS kernel
  - Translate general RTOS API to target specific RTOS calls
  - Target RTOS adapter libraries for different RTOSs

Compile and Link

- Compile generated C code
  - Compiler for target processor
  - Modify / optimize C code
- Link object code
  - HAL library
  - Specific RTOS library
  - Target RTOS adapter
- Implementation model
  - Instruction set simulator
  - Import binary image
  - Cycle accurate simulation



Software Synthesis Example

```
behavior B1
{ void main(void)
 …}
behavior B2
{ void main(void)
 …}
behavior CPU()
{
  B1 b1();
  B2 b2();
  void main(void)
  {



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

Input Model

```
behavior Task_B1(RTOS os)
{ void main(void)
 …}
behavior Task_B2(RTOS os)
{ void main(void)
 …}
behavior Task_CPU(RTOS os)
{
  Task_B2 task_b2(os);
  Task_B3 task_b3(os);
  void main(void) {
    task_b2.create();
    task_b3.create();
    os.par_start();
    par  {
      b2.main();
      b3.main();
    }
    os.par_end();}
};
```

Multi Task Model

```
struct Task_B1{…}
void Task_B1_main(struct
            Task_B1 *This) {…}
struct Task_B2{…}
void Task_B2_main(struct
            Task_B2 *This) {…}
struct Task_CPU {
 struct Task_B1 task_b1;
 struct Task_B2 task_b2 }
void Task_CPU_main(struct
            Task_CPU  *This)
{
  TaskCreate(&Task_B1_main,
          &This->task_b1,0);
  TaskCreate(&Task_B2_main,
          &This->task_b2,0);



  TaskJoin(NULL);
}
```

C Model

Experiment

- Developed software synthesis tools for SpecC
  - Scheduling refinement tool scos
  - Code generation tool sc2c
  - Integrated in SoC Design Environment (SCE) toolset
- Examples
  - GSM Vocoder
  - JPEG encoder
  - Motor controller
  - Mp3 decoder
  - Inter-task communication examples

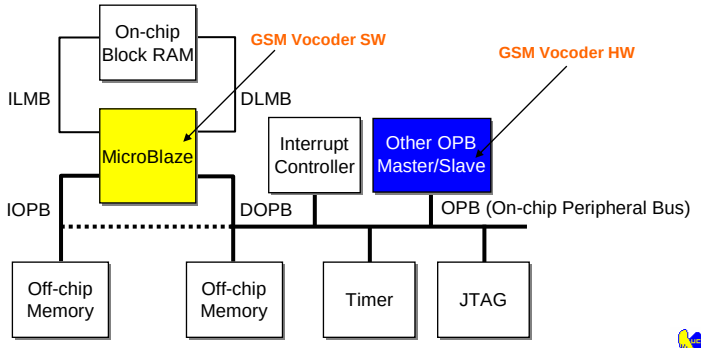
SW Code Generation Results

Design		Behaviors	Channels	SW Tasks	C Code (#LoC)	Code Gen.	Manual Coding
Vocoder	SW	109	0	1	9,805	1.41s	98h
	HW/SW	107	1	2	9,244	1.57s	92h
Mp3	SW	131	3	16	33,519	5.98s	335h
	HW/SW	147	7	34	32,092	9.24s	320h
JPEG	SW	36	0	1	1,655	0.21s	16h
	HW/SW	29	6	2	1,959	0.31s	19h
Motor	SW	29	3	9	2,245	0.28s	22h
	HW/SW	25	8	5	2,300	0.32s	23h

- Generate SW for different system arch. in seconds
- 1000x productivity gain

Implement GSM voice encoder on FPGA

- Xilinx Virtex-II FPGA
  - Processor: Microblaze
  - RTOS: uC/OS-II
  - Bus: OPB Bus



Conclusions

- Benefits of software synthesis
  - Relieves designers from the tedious, error-prone code writing process
  - Eliminates the need to maintain two versions of software
- Contributions of my dissertation
  - Software synthesis flow with well defined design steps
  - Demonstrated effectiveness and productivity gain through experiment
  - C and SpecC co-simulation enables validation of C code through fast C model simulation instead of using instruction set simulation
  - *C model* serves as a virtual system prototype for designers to add/debug/validate new application software to the system
  - Defined a way to model RTOSs in SpecC, which enables scheduling exploration in the early stages of system design
- Future work
  - Apply software synthesis to models in SystemC
  - Introduce more features for RTOS models in SpecC

Ph.D. Final Defense
copyright©2004 Haobo Yu