Embedded Software Generation from System Level Design Languages

Haobo Yu, Rainer Doemer and Daniel Gajski
Center for Embedded Computer Systems
University of California, Irvine
http://www.cecs.uci.edu

Outline

- **Introduction**
  - Related work
- **Design flow**
- **Embedded software generation**
  - Task generation
  - Code generation
  - Operating System targeting
- **Experimental results**
- **Summary & conclusions**

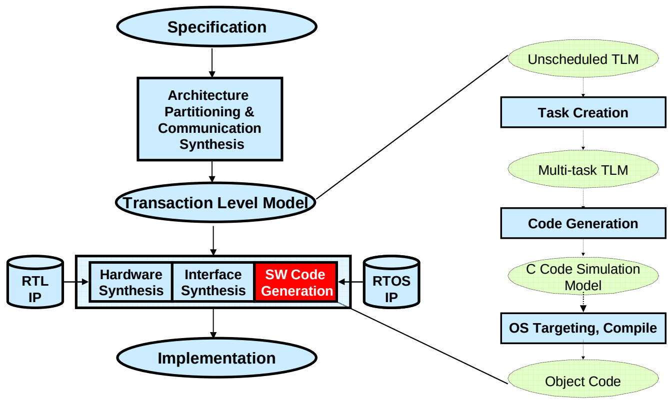
Introduction

- **Increasing Significance of Embedded SW**
  - ⇨ Most embedded software is still created manually after HW/SW partitioning
  - ⇨ Generation from system level design language (SLDL) is one solution to increase productivity
- **Embedded SW Generation within System Design Flow**
  - Sequence of refinement steps
  - Well-defined intermediate models
- **Implementing SLDL language elements using ANSI C**
  - Hierarchy, concurrency, communication
  - Modules, processes, channels, port mappings

Related Work

- **Code generation**
  - From abstract model (UML) [Rational]
  - From graphical finite state machine (StateCharts) [Harel90]
  - From synchronous programming languages (Esterel)[Boussinot91]
- **POLIS approach [Baladrin97]**
  - Mainly focused on reactive real time systems
  - Not easily extended for other more general frameworks
- **Software generation from SystemC SLDL**
  - Redefinition and overloading of SystemC class [Herrera03]
    - Requires C++ compiler and introduces SLDL language overhead
  - Substituting SystemC modules with C structures [Groetker03]
    - Requires special SystemC modeling styles

Embedded Software Generation in System Design Flow



ASPDAC 2004, Yokohama Copyright © 2004 H. Yu, R Doemer, D.Gajski 5

Embedded Software Generation Steps

1) Task creation

- Creates multiple tasks from specification
- Determine scheduling algorithm, task priorities

2) Code generation

- Create C code for each task from its SLDL description

3) Operating system targeting

- Implement task management, inter-task communication

- **Code optimization**

ASPDAC 2004, Yokohama Copyright © 2004 H. Yu, R Doemer, D.Gajski 6

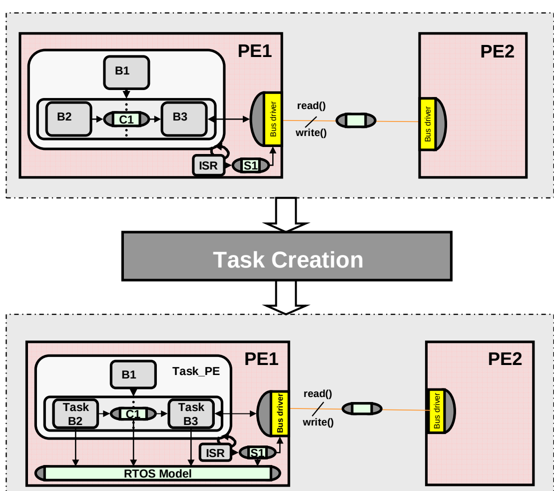
Task Creation (a)

- **Concurrency**
  - Conversion of concurrent behaviors into tasks
  - Fork child tasks dynamically inside a parent task

- **Communication**
  - SLDL channels are replaced by channels from RTOS Lib
    - semaphore, queue, handshake, ...

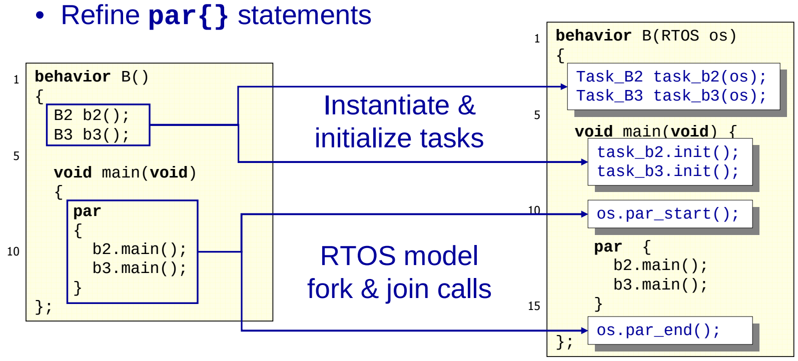
- **Multi-task system scheduled by abstract RTOS model**
  - Choose scheduling algorithm and set task priority
  - Simulate and check timing properties for the SW part

Task Creation (b)



Task Creation (c)

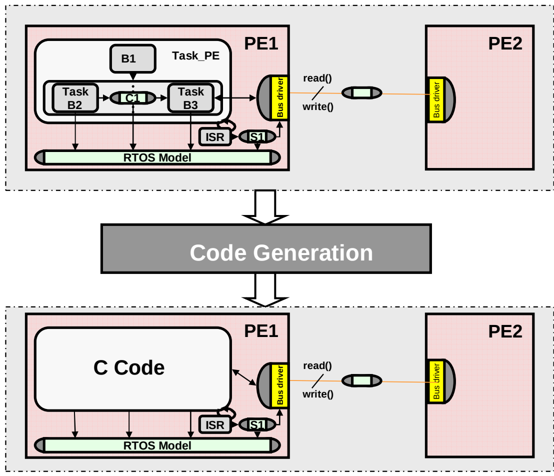
- **Dynamic task creation**
  - Refine **par{}** statements



Code Generation (a)

- **Rules for C code generation**
  1. Behaviors and channels are converted into C *struct*
  2. Child behaviors and channels are instantiated as C *struct* members inside the parent C *struct*
  3. Variables defined inside a behavior or channel are converted into data members of the corresponding C *struct*
  4. Ports of behavior or channel are converted into data members of the corresponding C *struct*
  5. Functions inside a behavior or channel are converted into global functions
  6. A static *struct* instantiation for each PE is added at the end of the output C code to allocate/initialize the data used by SW

Code Generation (b)



Code Generation (c)

```
behavior B1(int v)          --R1-->
{                           --R4-->
  int  a;                   --R3-->

  void main(void)           --R5-->
  {
   a = 1;
   v = a *2;
  }
};
behavior Task1
{
  int x;
  int y;
  B1 b11(x);                --R2-->
  B1 b12(y);

  void main(void)
  {
   b11.main();
   b12.main();
  }
};                          --R6-->
```

(a) SpecC Code

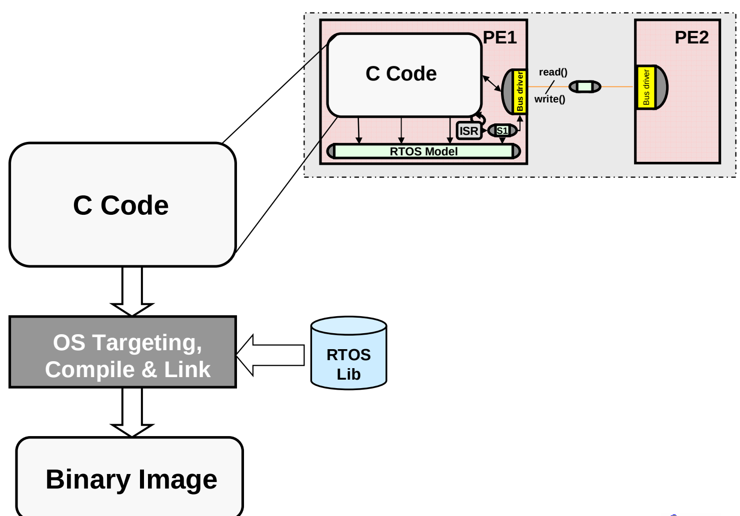
```
struct B1
{
   int (*v) /*port*/;
   int a;
};
void B1_main(struct B1 *this)
{
   (this->a) = 1;
   (*(this->v)) = (this->a) * 2;
}
struct Task1
{
   int  x;
   int  y;
   struct B1 b11;
   struct B1 b12;
};
void Task1_main(struct Task1*this)
{
    B1_main(&(this->b11));
    B1_main(&(this->b12));
}
struct Task1 task1 =
{ 0,                /* x init value*/
  0,                /* y init value*/
  { &(task1.x),     /*port v of  b11 */
     0              /* a init value */
  }, /*b11*/
  { &(task1.y),     /*port v of b12*/
     0              /* a init value*/
  }, /*b12*/
};
void Task1()
{
  Task1_main(&task1);
}
```

(b) C Code

Operating System Targeting (a)

- **Task management (Scheduling)**
  - Implementing the abstract RTOS model interfaces by specific RTOS library APIs
- **Task communication**
  - Replacing methods of abstract RTOS channels with equivalent services of the target RTOS library routines

Operating System Targeting (b)



Operating System Targeting (c)

- **Implement task management using pthread library**

```
behavior B2B3(RTOS os)
{
  Task_B2 task_b2(os);
  Task_B3 task_b3(os);

  void main(void) {
    task_b2.init();
    task_b3.init();

    os.par_start();

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

```
struct B2B3
{ struct Task_B2 task_b2;
  struct Task_B3 task_b3;};
void *B2_main(void *arg)
{ struct Task_B2 *this=(struct Task_B2*)arg;
  ...
  pthread_exit(NULL); }
void *B3_main(void *arg)
{ struct Task_B3 *this=(struct Task_B3*)arg;
  ...
  pthread_exit(NULL); }
void *B2B3_main(void *arg)
{  struct B2B3 *this= (struct B2B3*)arg;
   int status; pthread_t *task_b2, *task_b3;
 pthread_create(task_b2, NULL,
                  B2_main, &this->task_b2);
 pthread_create(task_b3, NULL,
                  B3_main, &this->task_b3);

 pthread_join(*task_b2, (void **)&status);
 pthread_join(*task_b3, (void **)&status);

 pthread_exit(NULL);
}
```


Experiment

- **GSM Vocoder (voice encoder for mobile phones)**
- **Input model: 11,557 lines of SpecC code**
- **HW/SW partitioning:**
  - HW : Custom hardware co-processor (codebook)
  - SW : ARM7DTI (other part of the spec)
- **Output:**
  - HW: 5540 lines of Verilog code
  - SW: 7882 lines of C code

Experimental Results

- **Implementation**
  - One task for voice encoding
  - Operating System uC-OSII
- **Code sizes**

	SPEC	TLM	SW(TLM)	C
Behavior/Channel	102	127	96	0
Operations	16,614	19,527	14,573	23,868
Lines (of C code)	11,557	12,606	10,920	7,882

- **Binary code for ARM**

	Code Size	Data Size
Object File from C	33KB	19KB
Final Image	47KB	28KB

Summary and Conclusion

- **Embedded SW Generation in System-level Flow**
  - Refinement steps and algorithms
  - Task creation, Code generation, OS targeting
- **Applicable to system models written in SLDL**
  - SpecC, SystemC, ...
- **Software Synthesis frees the designer from manual coding**
- **High productivity gain**
  - Automatic seconds
  - Manual months
- **Verification of the generated code becomes easier**
  - Refinement-based approach generates well-structured code
  - Intermediate models are well-defined
- **Future work**
  - Focus on SW/HW driver synthesis
  - Improvements on OS targeting part
- **Additional information**
  - `http://www.cecs.uci.edu/~cad/sce.html`