

Systematic Generation of Embedded Software from High-level Models

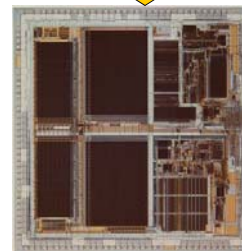
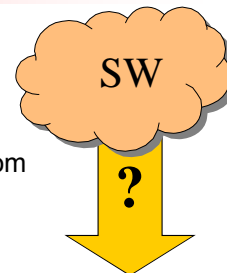
Gunar Schirner

Center for Embedded Computer Systems
University of California, Irvine



Motivation

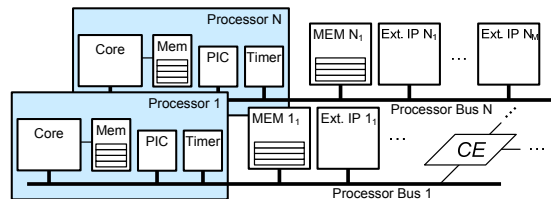
- Increasing complexity of Multi-Processor System-on-Chip (MPSoCs)
 - Feature demands
 - Production capabilities + Implementation freedom
- Increasing software content
 - Flexible solution
 - Addresses complexity
- How to create SW for MPSoC?
 - Avoid break in ESL flow:
 - Synthesize SW from abstract models



Source: simh.trailing-edge.com

Goals

- Generate SW binaries for MPSoC from abstract specification
 - Eliminate tedious, error prone SW coding
 - Rapid exploration
 - High-level application development
 - Support wide range of system sizes
 - with RTOS / without RTOS (i.e. interrupt-driven)



Gunar Schirmer, 2008

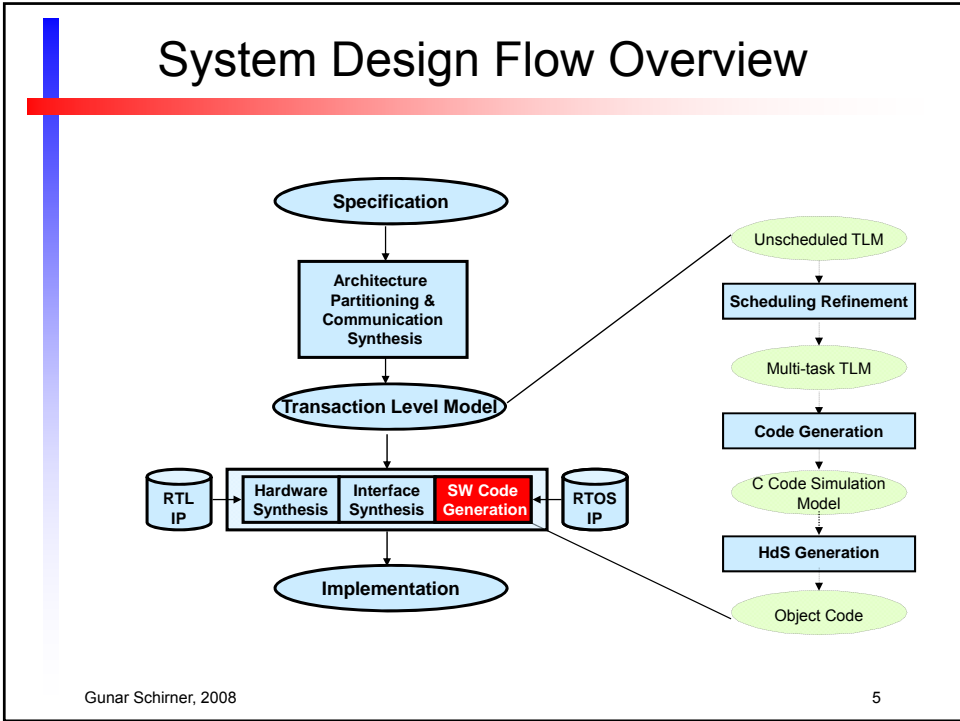
3

Outline

- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
- Experimental Results
- Conclusions

Gunar Schirmer, 2008

4

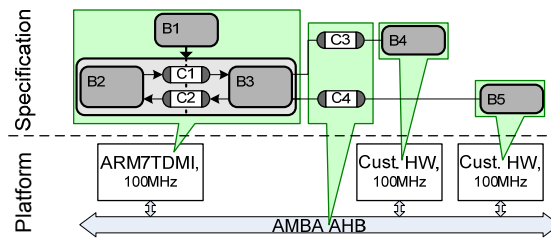


- ### Outline
- Introduction
 - System Design Flow Overview
 - ➔ Processor Modeling
 - Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
 - Experimental Results
 - Conclusions
- Gunar Schirmer, 2008 6

System Design Flow Overview

Input Specification

- Capture Application in C (SpecC SLDL)
 - Computation
 - Organize code in behaviors (processes)
 - Communicate through point-to-point channels
 - Select from feature-rich selection
 - Synchronous / Asynchronous
 - Blocking / Non-Blocking
 - Synchronization only (e.g. semaphore, mutex, barrier)
- Architecture decisions:
 - Processor(s)
 - HW component(s)
 - Busses
 - Mapping
 - ...



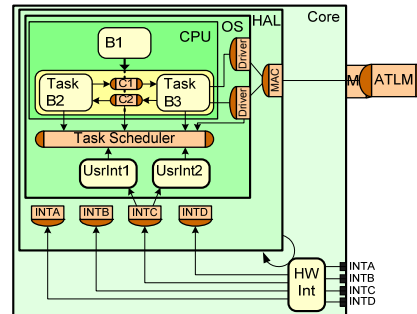
Gunar Schirner, 2008

7

Processor Model: Overview

Automatic TLM Generation

- System Compiler (SCE) generates System TLM
 - Model contains complete implementation information
- TLM Generation, processor modeling described in:
 - [Gerstlauer et. al, TCAD 09/2007],
 - [Shin et. al, CODES+ISSS 2006],
 - [Peng et. al, ASPDAC 2002],
 - [Schirner et. al, ASPDAC 2007]

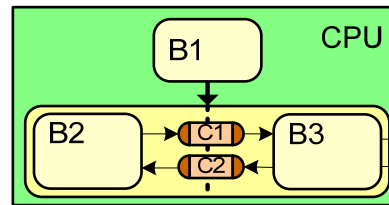
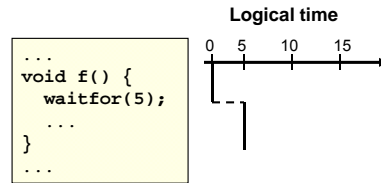


Gunar Schirner, 2008

8

Processor Model: Application (1/5)

- Goal: Timed execution
- **Application Level**
 - User code in SLDL
 - Hierarchical set of behaviors
 - Channels
 - High-level, typed message passing
 - Timed execution
 - Back annotate C code with wait-for-time statements
 - At function level
 - Many other possibilities
 - Execute natively on simulation host
 - Discrete event simulator
 - Fast execution speed

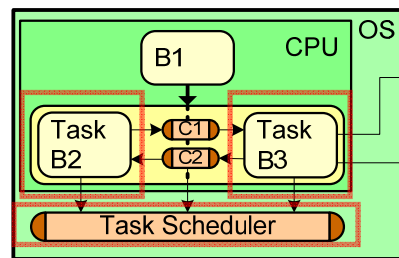
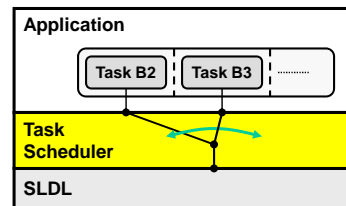


Gunar Schirmer, 2008

9

Processor Model: Task (2/5)

- Goal: Model dynamic scheduling effects
- **Task Level**
 - Group behaviors to tasks
 - Schedule tasks by abstract scheduler
 - Channel in SLDL
 - Wrap primitives that could trigger scheduling
 - Task start
 - Channel communication
 - wait-for-time

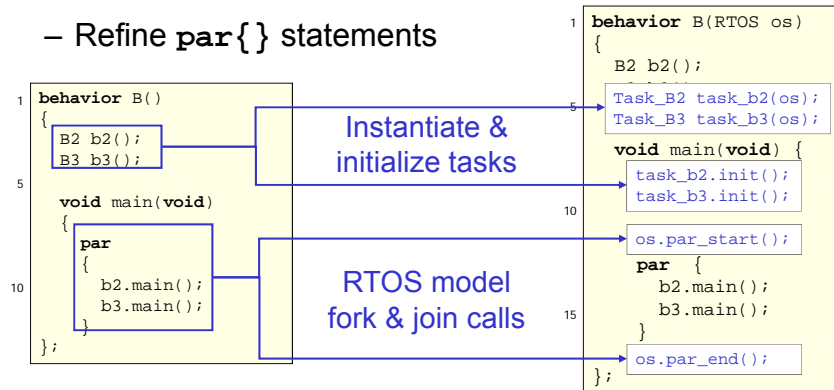


Gunar Schirmer, 2008

10

Processor Model: Task (2/5)

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



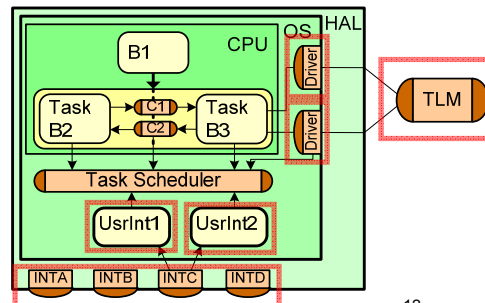
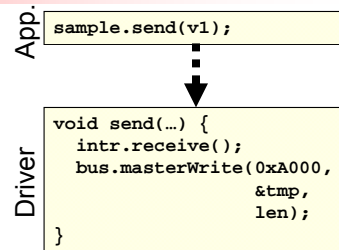
Gunar Schirmer, 2008

Source: H. Yu, R Doemer, D.Gajski 2004

11

Processor Model: Firmware (3/5)

- Goal: External Communication
- **Firmware Level**
 - Software Drivers
 - Presentation, Session, Packeting
 - Synchronization (e.g. Interrupts)
 - TLM Bus model
 - User transactions
 - However, interrupts are unscheduled

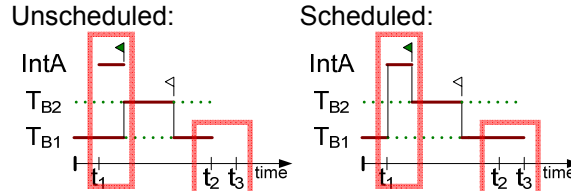


Gunar Schirmer, 2008

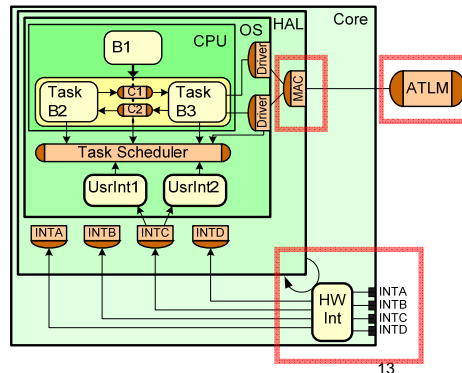
12

Processor Model: TLM (4/5)

- Goal: Interrupt Scheduling



- 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
 - Complete model!

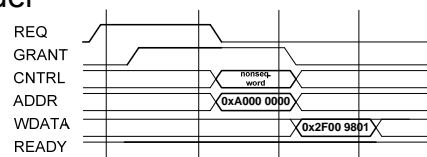


Gunar Schirmer, 2008

13

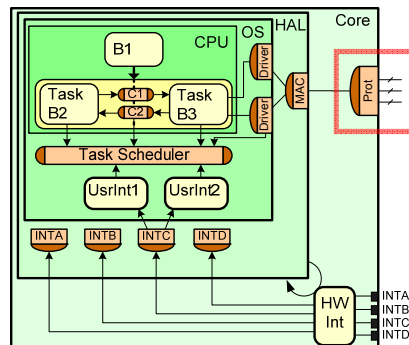
Processor Model: BFM (5/5)

- Goal: Accurate Bus Model



- Processor Bus Functional Model

- Pin-accurate model of processor
 - Cycle approximate for SW execution
- Bus model
 - Pin-accurate
 - Cycle-Accurate



Gunar Schirmer, 2008

14

Processor Model

- Summary of features:

Features	Level
Target approx. computation timing	Appl. ↓
Task mapping, dynamic scheduling	Task ↓
Task communication, synchronization	Firmware ↓
Interrupt handlers, low level SW drivers	TLM ↓
HW interrupt handling, int. scheduling	BFM ↓
Cycle accurate communication	BFM - ISS ↓
Cycle accurate computation	BFM - ISS ↓

Gunar Schirmer, 2008

15

Outline

- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
- Experimental Results
- Conclusions

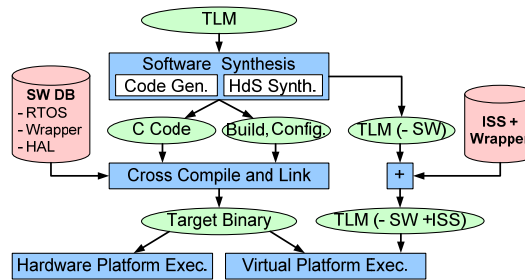
Gunar Schirmer, 2008

16

Software Synthesis

Second step: Software Synthesis

- Code Generation [Yu et. al, ASPDAC 2004]
 - Generate application code
 - Resolve behavioral hierarchy into flat C code
- Hardware-dependent Software (HdS) Synthesis
 - Communication Synthesis (drivers)
 - Multi-task Synthesis
 - Binary Image Generation
 - Generate ISS-based virtual platform



Gunar Schirmer, 2008

17

Outline

- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
- Experimental Results
- Conclusions

Gunar Schirmer, 2008

18

Code Generation (a)

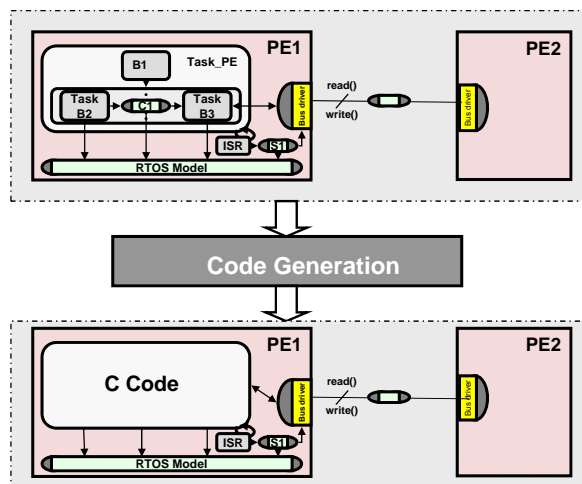
- Rules for C code generation
 1. Behaviors and channels are converted into C *struct*
 2. Child behaviors and channels are instantiated as C *struct* members inside the parent C *struct*
 3. Variables defined inside a behavior or channel are converted into data members of the corresponding C *struct*
 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

Gunar Schirmer, 2008

Source: H. Yu, R Doemer, D.Gajski 2004

19

Code Generation (b)



Gunar Schirmer, 2008

Source: H. Yu, R Doemer, D.Gajski 2004

20

C Code Generation Example

```

1 behavior B1(int v)
{
2   int a;
3   void main(void)
4   {
5     a = 1;
6     v = a * 2;
7   }
8 }
9
10 behavior Task1
11 {
12   int x;
13   int y;
14   B1 b11(x);
15   B1 b12(y);
16   void main(void)
17   {
18     b11.main();
19     b12.main();
20   }
21 }
22
23 struct B1
24 {
25   int (*v) /*port*/;
26   int a;
27 };
28
29 void B1_main(struct B1 *this)
30 {
31   (this->a) = 1;
32   (*(this->v)) = (this->a) * 2;
33 }
34
35 struct Task1
36 {
37   int x;
38   int y;
39   struct B1 b11;
40   struct B1 b12;
41   void Task1_main(struct Task1 *this)
42   {
43     B1_main(&(this->b11));
44     B1_main(&(this->b12));
45     struct Task1 task1 =
46     {
47       0, /* x init value*/
48       0, /* y init value*/
49       { &(task1.x), /*port v of b11*/
50         0, /* a init value */
51       }, /*b11*/
52       { &(task1.y), /*port v of b12*/
53         0, /* a init value*/
54       }, /*b12*/
55     };
56     void Task1()
57     {
58       Task1_main(&task1);
59     }
60 }

```

(a) SpecC Code

(b) C Code

Gunar Schirmer, 2008

Source: H. Yu, R Doemer, D.Gajski 2004

21

Outline

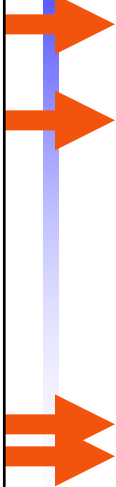
- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
- Experimental Results
- Conclusions

Gunar Schirmer, 2008

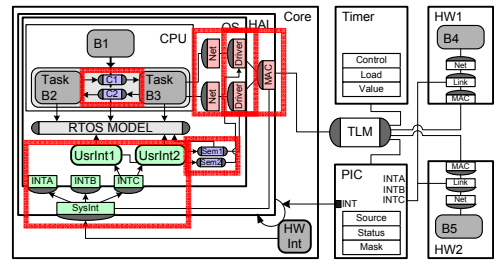
22

Hardware-dependent Software Synthesis

- Communication Synthesis



- Internal communication
 - Replace with target specific implementation
 - e.g. RTOS semaphore, event, msg. queue
- External communication
 - ISO / OSI layered to support heterogeneous architectures
 - Contains:
 - Marshalling
 - System addressing
 - Packetizing
 - MAC
- Synchronization
 - Polling, or
 - Interrupt



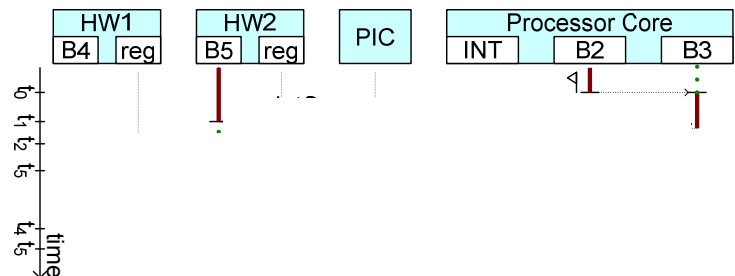
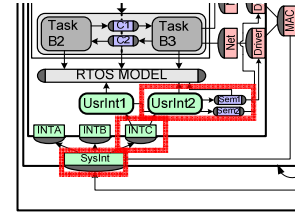
Gunar Schirmer, 2008

23

Hardware-dependent Software Synthesis

- Example: Synchronization by interrupt

- 1) Low level interrupt handler
 - Preempts current task
- 2) System interrupt handler
 - Checks PIC
- 3) User-specific interrupt handler
 - Handles shared interrupts
- 4) Semaphore
 - Releases task

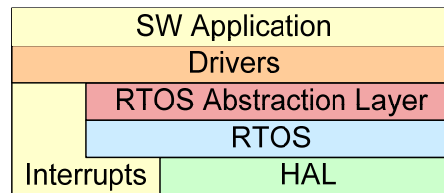


Gunar Schirmer, 2008

24

Hardware-dependent Software Synthesis

- Multi-Task Synthesis
 - RTOS-based Multi-Tasking
 - Based on off the shelf RTOS
 - e.g. μ C/OS-II, eCos, vxWorks
 - RTOS Abstraction Layer
 - Canonical Interface
 - » limit interdependency RTOS / Synthesis
 - Generate task management code
 - Internal communication



Gunar Schirmer, 2008

25

Hardware-dependent Software Synthesis

- Multi-task Synthesis
 - Example

```

1 behavior B2B3(RTOS os)
  {
    Task_B2 task_b2(os);
    Task_B3 task_b3(os);
5   void main(void) {
      task_b2.init();
      task_b3.init();
10  os.par_start();
      par {
        b2.main();
        b3.main();
15  }
    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;
  taskCreate(task_b2, NULL,
             B2_main, &this->task_b2);
  taskCreate(task_b3, NULL,
             B3_main, &this->task_b3);
  taskJoin(*task_b2, (void **)&status);
  taskJoin(*task_b3, (void **)&status);
  taskTerminate(NULL);
}

```

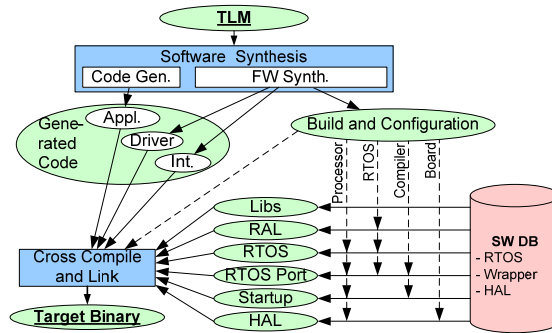
Gunar Schirmer, 2008

Source: H. Yu, R Doemer, D.Gajski 2004

26

Binary Image Generation

- Generate binary for each processor
 - Generate build and configuration files
 - Select software database components
 - Configure RTOS
 - Cross compile and link
 - Significant effort in DB design
 - Minimize components
 - Analyze dependencies
 - Goal: flexible composition



Gunar Schirmer, 2008

27

Outline

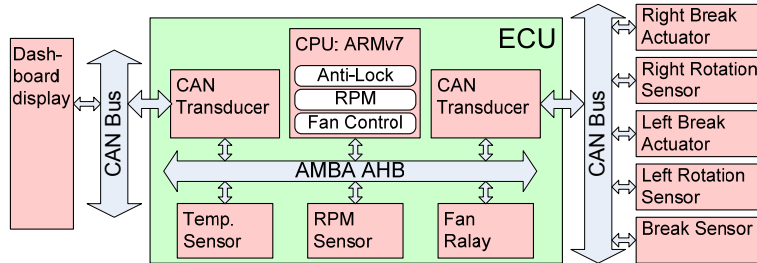
- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
 - Code Generation
 - Hardware-dependent Software Synthesis
 - Communication Synthesis
 - Multi-Tasking
 - Binary Image Creation
- ➔ Experimental Results
- Conclusions

Gunar Schirmer, 2008

28

Experimental Results

- Automotive Example
 - ARM7TDMI / 2x CAN / Anti-Lock Brakes, RPM, Fan



Generated System's Statistics

Footprint	36224 Bytes
Alloc. Stacks	4096 Bytes
CPU Busy Cycles	6.706 MCycles
Latency RPM Task	1794 Cycles
# Interrupts	1478

Experimental Results

- Synthesis Results
 - 6 Applications
 - diff. complexities
 - e.g. 2 ... 14 ISRs
 - All functionally validated
 - Generated HdS
 - 200 ... 1200 lines
 - Within less than 1s
- Manual implementation would take days!

- Significant productivity gain
- Fast regeneration after platform modification

Example	GSM	Car	JPEG	Mp3 SW	Mp3 HW	Mp3 HW + JPEG
Complexity						
IO/HW/Bus	4/1/1	9/2/3	2/0/1	2/0/1	2/3/4	6/3/4
SW Behaviors	112	10	34	55	54	90
Channels	18	23	11	10	26	47
Tasks/ISRs	2/3	3/5	1/2	1/3	1/8	3/14
Lines of Code, RTOS-based						
Application	-	153	818	13914	12548	13480
HdS	-	649	210	299	763	1186
Lines of Code, Interrupt-based						
Application	5921	210	797	13558	12218	-
HdS	377	575	187	256	660	-
Execution, RTOS-based						
CPU Cycles	-	6.7M	127.7M	185.8M	44.5M	174.6M
CPU Load	-	0.9%	100.0%	100.0%	30.9%	86.6%
Interrupts	-	1478	805	4195	1144	1914
Execution, Interrupt-based						
CPU Cycles	42.0M	5.1M	126.7M	182.3M	43.3M	-
CPU Load	42.5%	0.7%	100.0%	100.0%	30.5%	-
Interrupts	3451	1027	726	4078	1054	-

Conclusions

- Presented Embedded Software Generation
 - Including: Hardware-dependent Software
 - Communication synthesis
 - Multi-task synthesis
 - Binary image creation
- Integrated into ESL flow
 - Seamless solution
- Complete: from abstract model to implementation!
 - Completes ESL flow for software
 - Eliminates tedious and error prone manual HdS development
 - Significant productivity gain
 - Enables rapid design space exploration

Gunar Schirmer, 2008

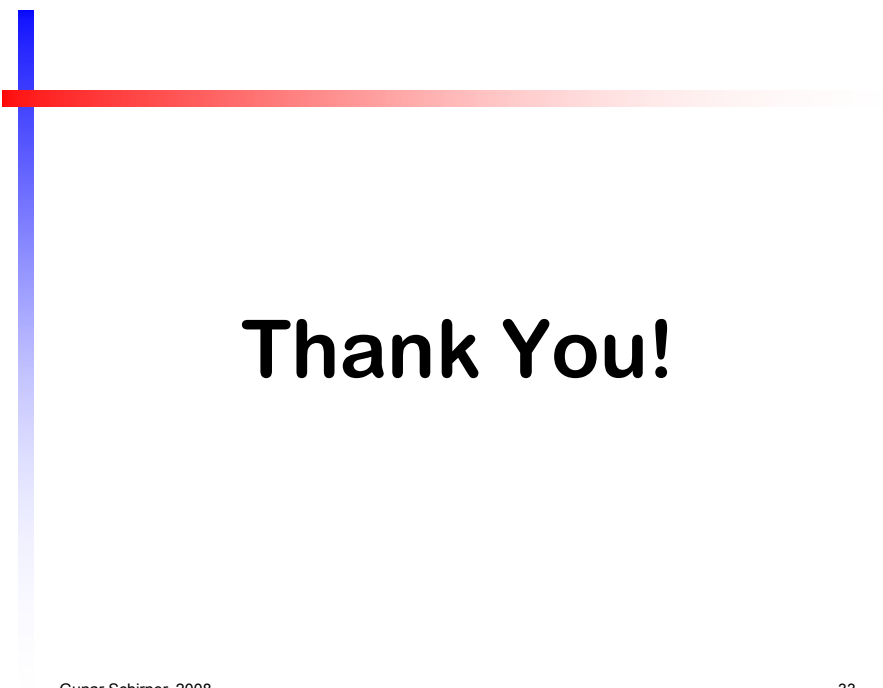
31

Acknowledgements

- SCE Research Team
 - Thanks for your support of the SCE environment

Gunar Schirmer, 2008

32



Thank You!

Gunar Schirner, 2008

33