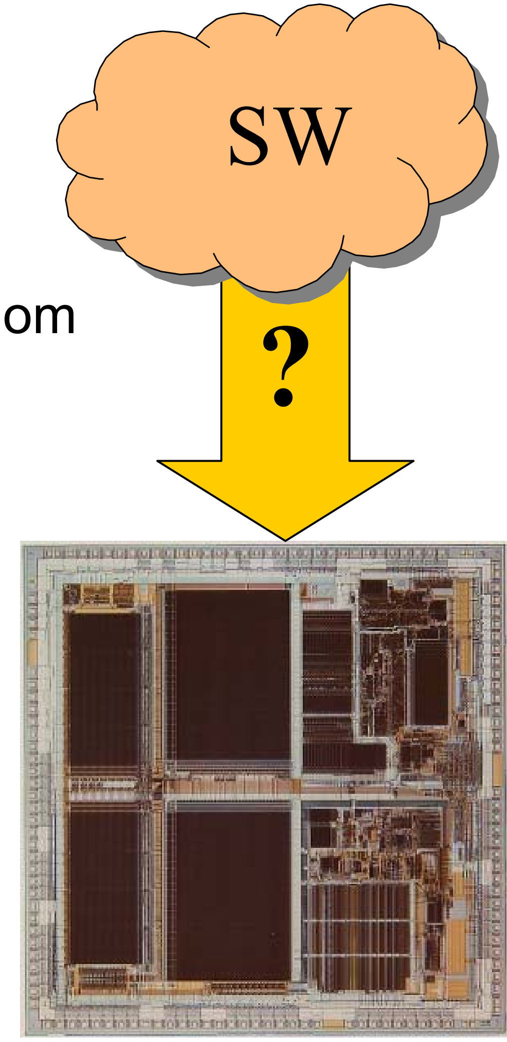
System-Level Development of Embedded Software

Gunar Schirner[1], Andreas Gerstlauer[2], Rainer Dömer[3]

1) ECE, Northeastern University, Boston, MA
2) University of Texas, Austin
3) CECS, University of California, Irvine

Motivation

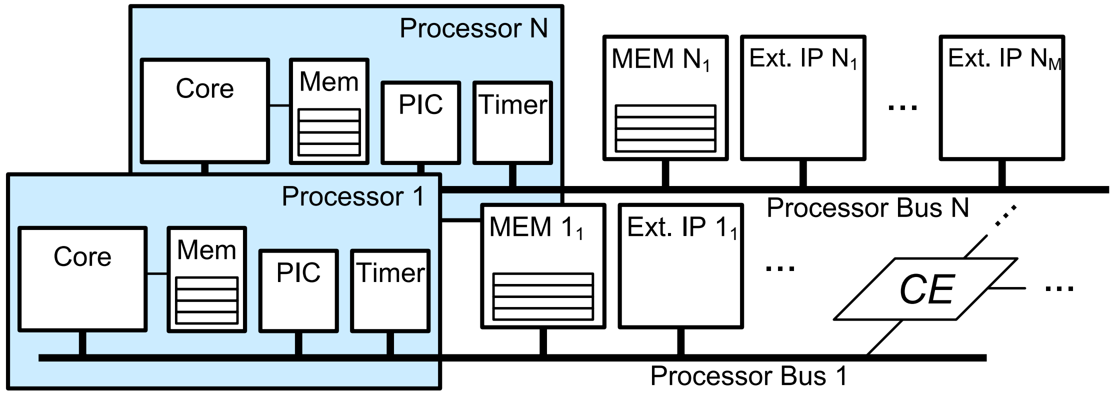
- Increasing complexity of Multi-Processor System-on-Chip (MPSoCs)
  - Feature demands
  - Production capabilities + Implementation freedom
- Increasing software content
  - Flexible solution
  - Addresses complexity
- How to create SW for MPSoC?
  - Avoid break in ESL flow:
    - Synthesize SW from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



Source: simh.trailing-edge.com

Goals

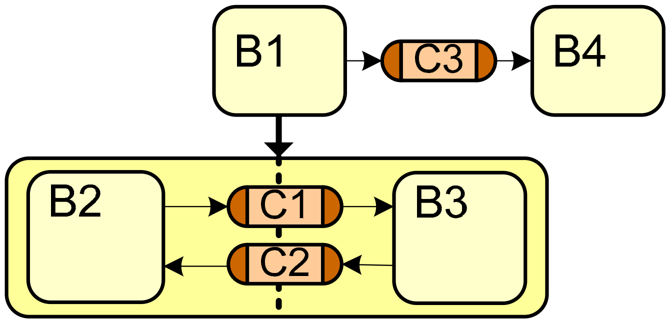
- Generate SW binaries for MPSoC from abstract specification
  - Eliminate tedious, error prone SW coding
  - Rapid exploration
  - High-level application developtment
  - Support wide range of system sizes

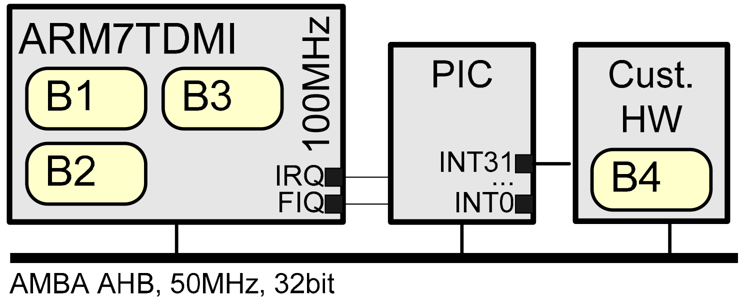


System Design Flow Overview

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
  - …

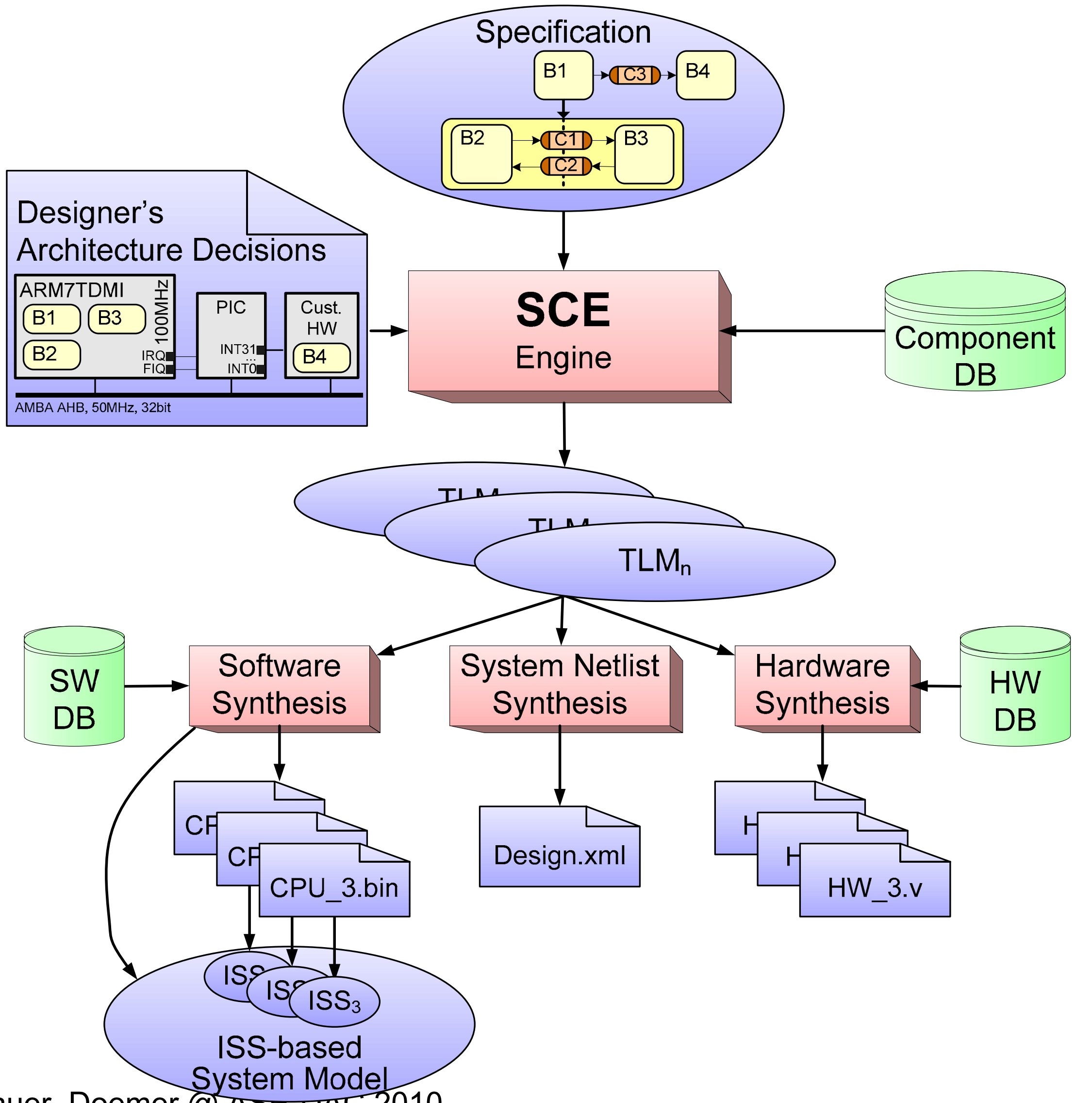




Outline

- 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

System Design Flow Overview



Outline

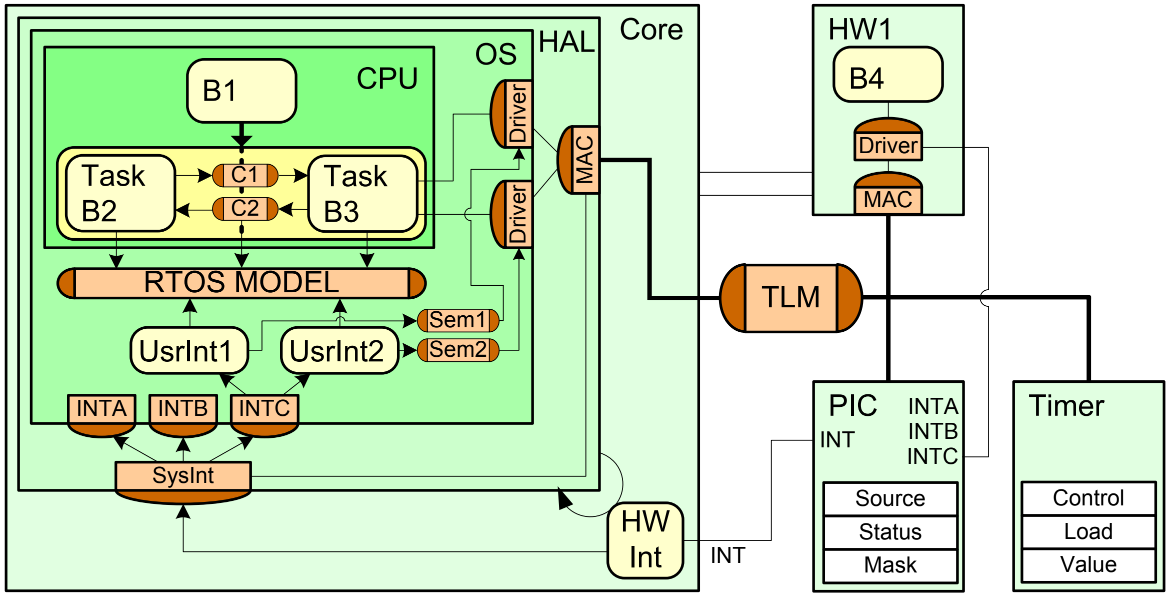
- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
  - Code Generation
  - Hardware-dependent Software Synthesis
    - Communication Synthesis
    - Multi-Tasking
    - Binary Image Creation
- Experimental Results
- Conclusions

Processor Model: Overview

Automatic TLM Generation

- System Compiler (SCE) generates System TLM

  ➢ Model contains complete implementation information

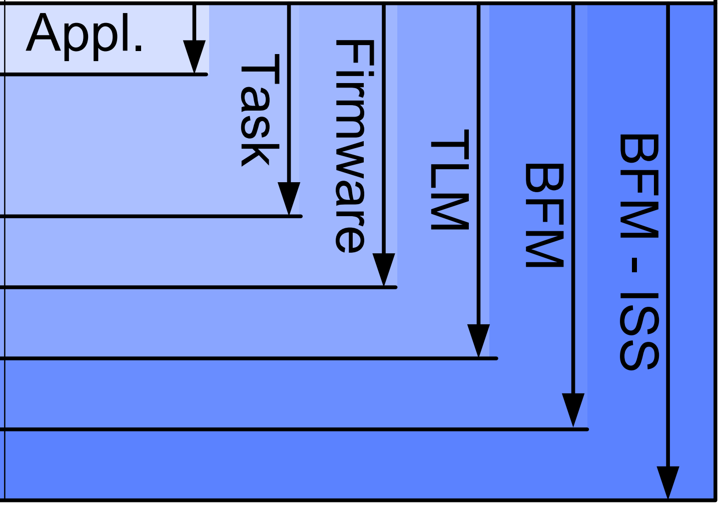


  – TLM generation, processor modeling described in:
    - [Doemer et. al, JES 07/2008], [Gerstlauer et. al, TCAD 09/2007], [Schirner et. al, TODAES 01/2010]

Processor Model

- Summary of features:

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



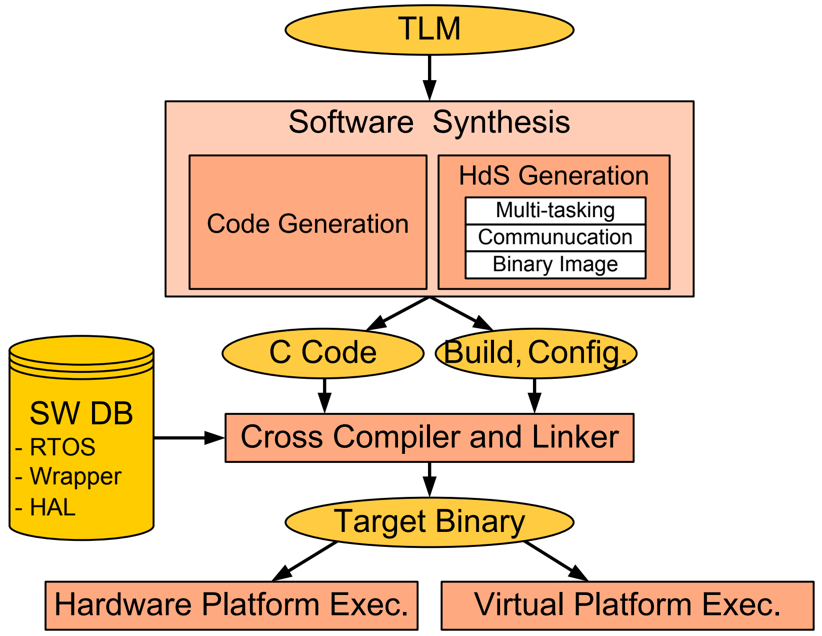
Outline

- 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

Software Synthesis

Second step: Software Synthesis

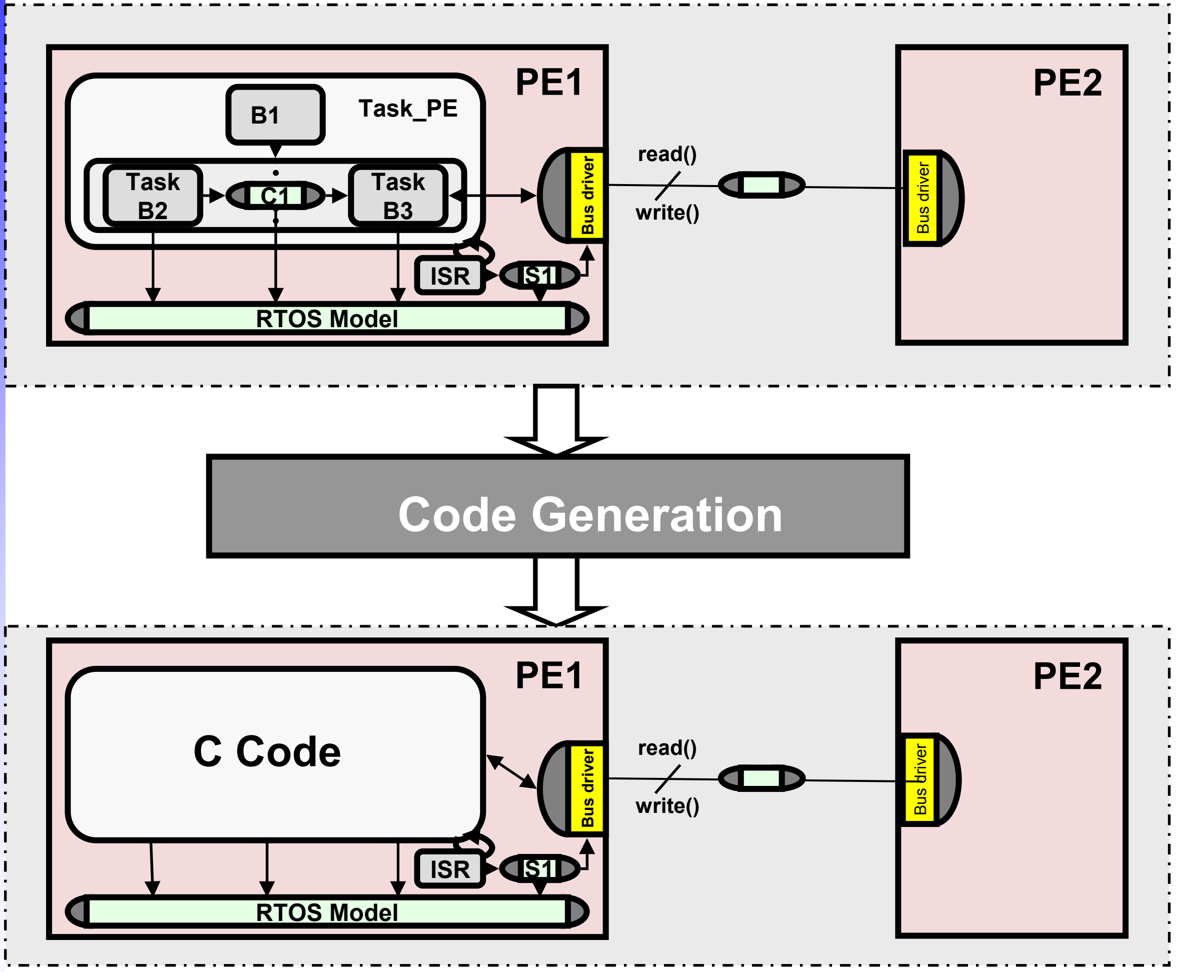
- Code Generation [Yu et. al, ASPDAC 2004]
  - Generate application code
  - Resolve behavioral hierarchy into flat C code
- Hardware-dependent Software (HdS) Synthesis
  - Multi-task Synthesis
  - Communication Synthesis
  - Binary Image Generation
  - Generate ISS-based virtual platform



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

Code Generation

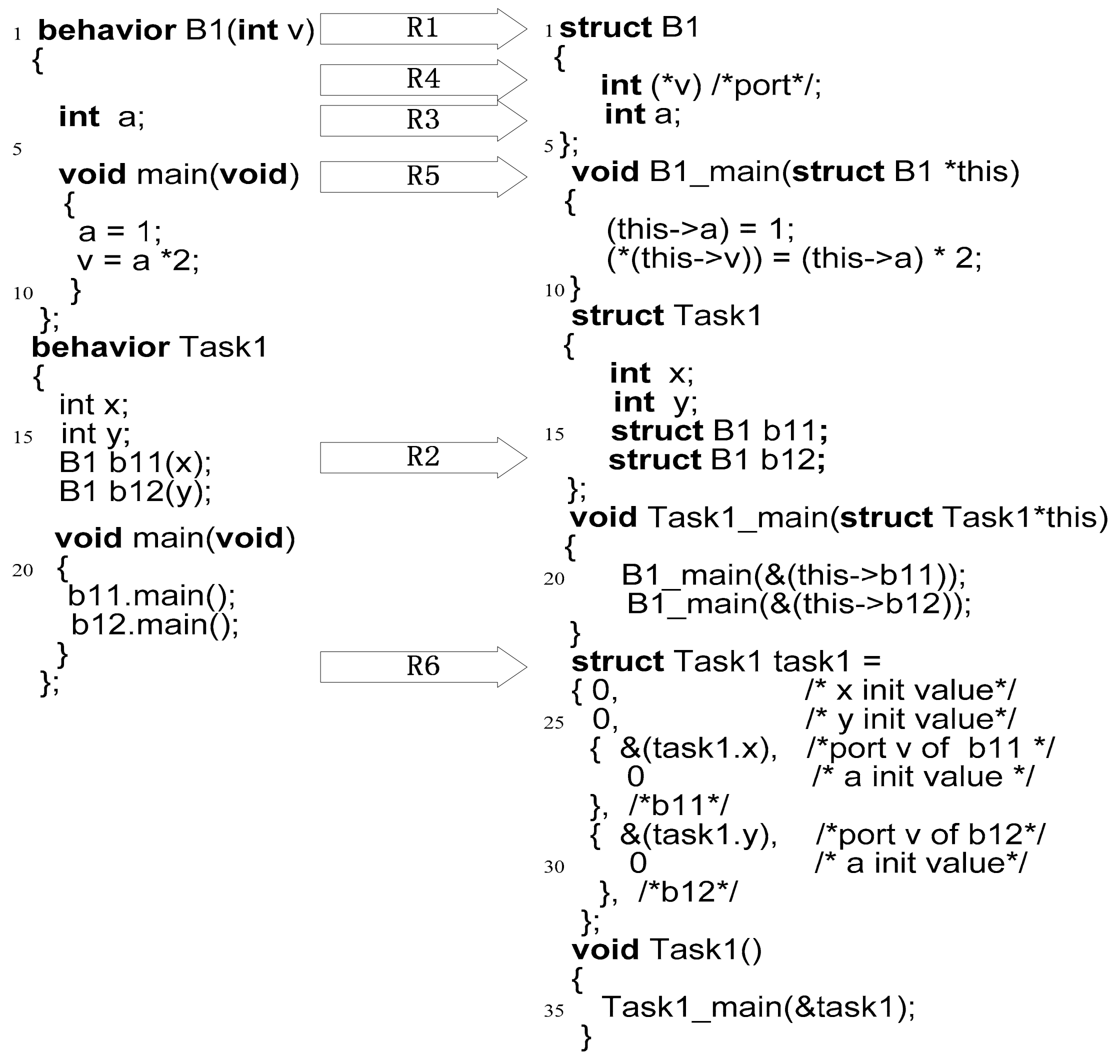


Rules for C code generation

- Behaviors and channels are converted into C *struct*
- Child behaviors and channels are instantiated as C *struct* members inside the parent C *struct*
- Variables defined inside a behavior or channel are converted into data members of the corresponding C *struct*
- Ports of behavior or channel are converted into data members of the corresponding C *struct*
- Functions inside a behavior or channel are converted into global functions
- 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

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

C Code Generation Example



```
behavior B1(int v)
{

  int  a;

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

  void main(void)
  {
   b11.main();
   b12.main();
  }
};
```

(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

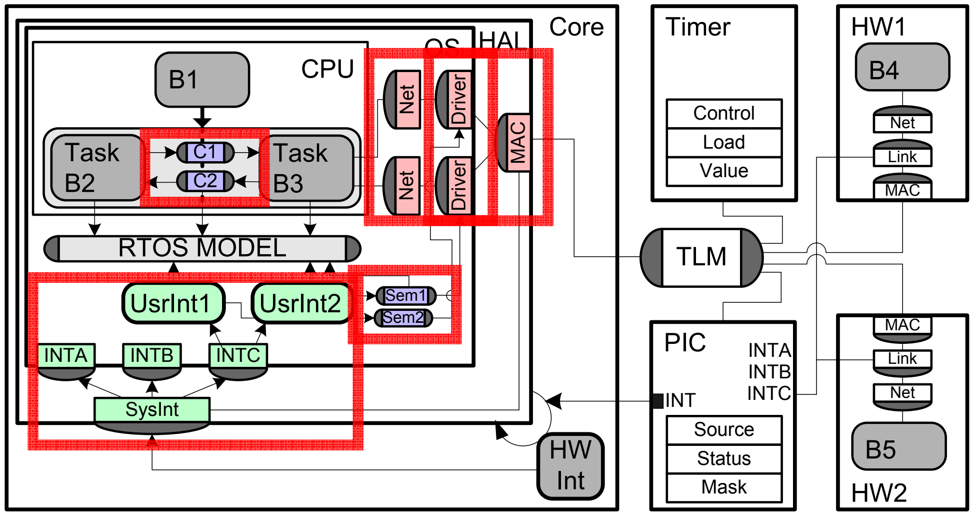
Source: H. Yu, R Doemer, D.Gajski 2004

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

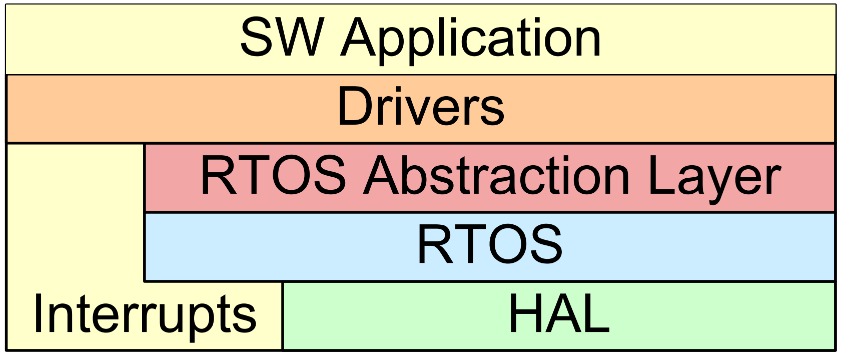
Hardware-dependent Software Synthesis

- 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



Hardware-dependent Software Synthesis

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



Hardware-dependent Software Synthesis

- Multi-task Synthesis
  - Example

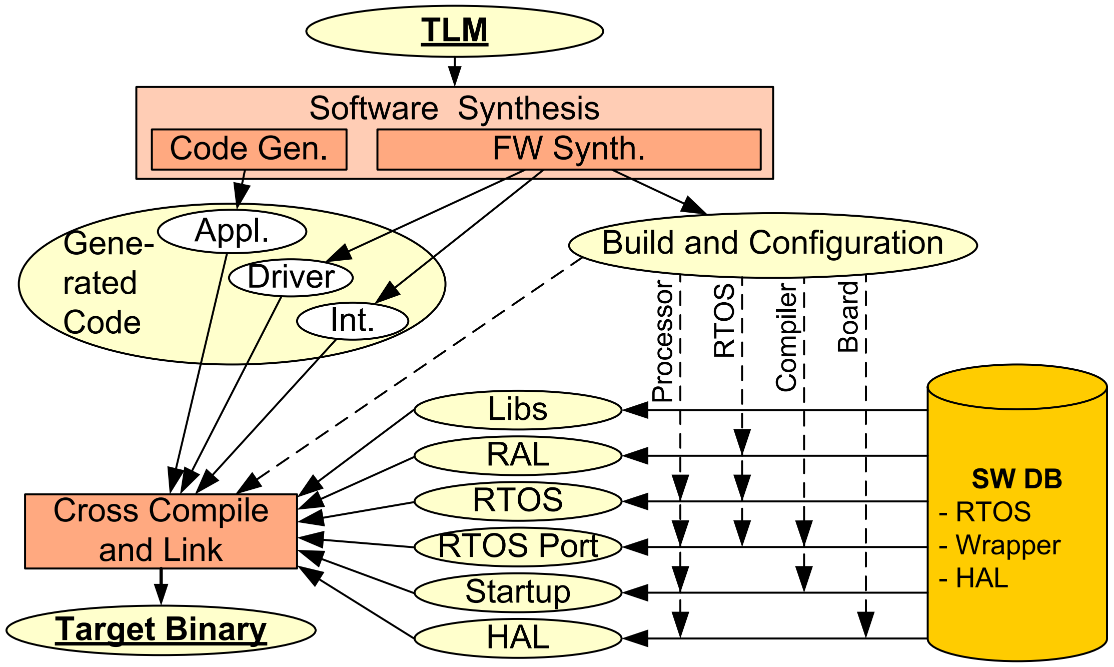
```
behavior B2B3(RTOS os)
{
  Task_B2 task_b2(os);
  Task_B3 task_b3(os);
  void main(void) {
    task_b2.init();
    task_b3.init();
    os.par_start();
    par   {
      b2.main();
      b3.main();
    }
    os.par_end();
};
```

```
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;
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);
}
```

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

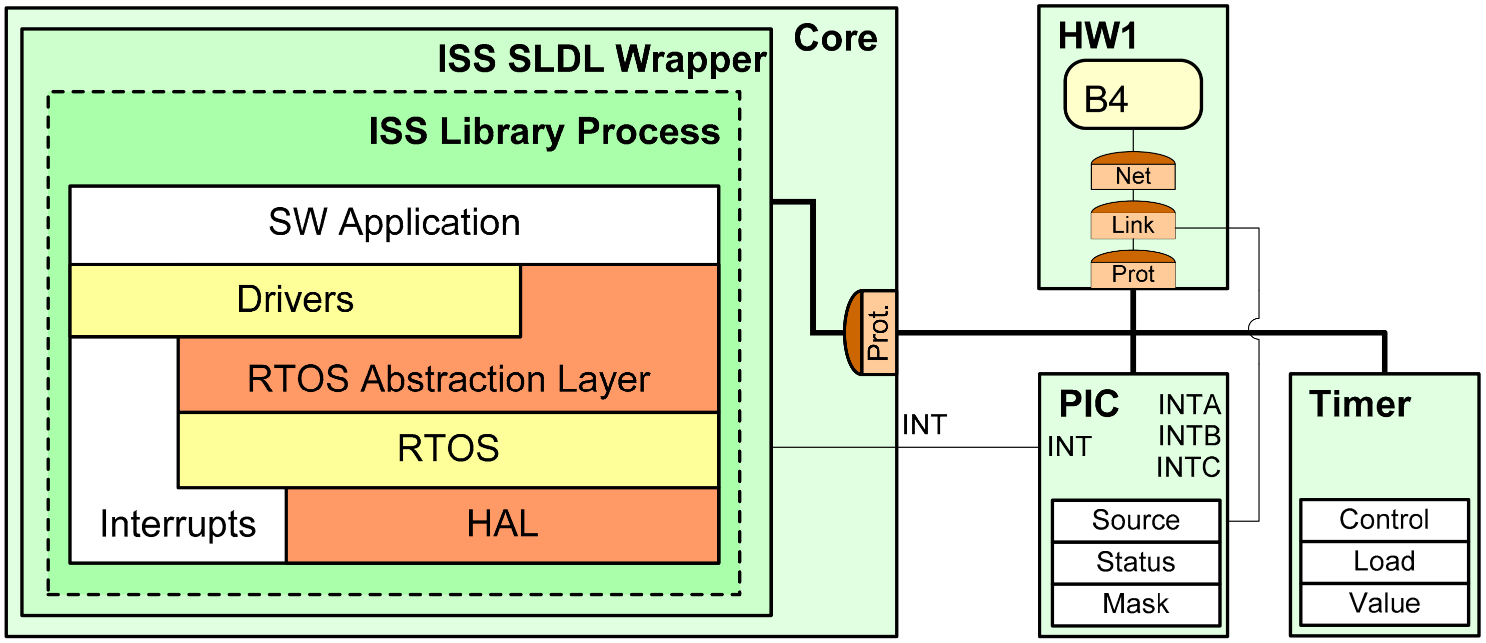
Binary Image Generation

- 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



Validation

- Validate generated binaries
  - Integrate ISS into system model
    - Execute target binary

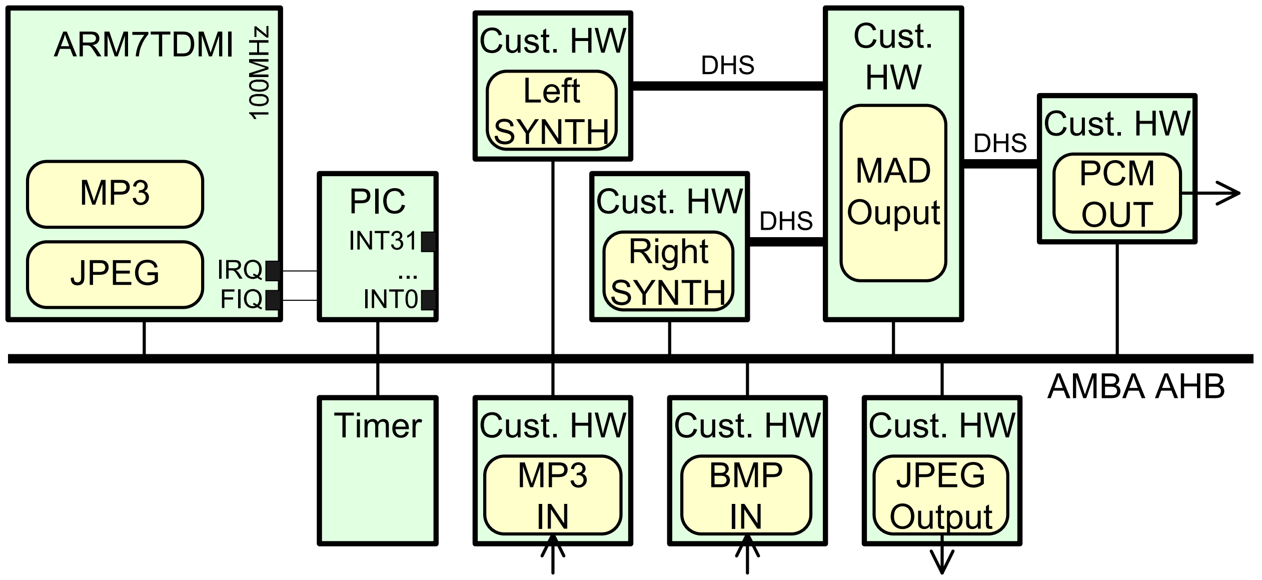


Outline

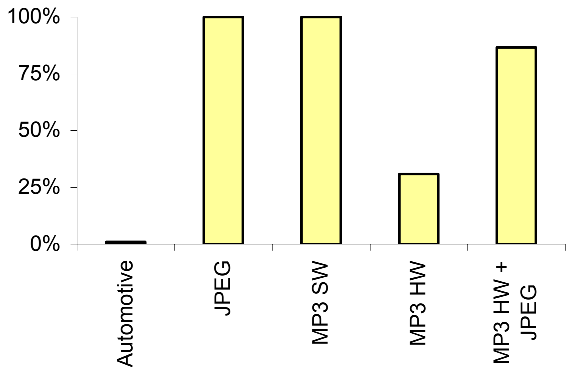
- Introduction
- System Design Flow Overview
- Processor Modeling
- Software Generation
  - Code Generation
  - Hardware-dependent Software Synthesis
    - Communication Synthesis
    - Multi-Tasking
    - Binary Image Creation
- Experimental Results
- Conclusions

Experimental Results

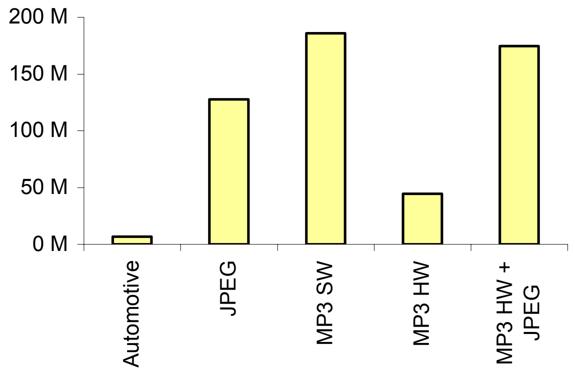
- Automotive Example
  - ARM7TDMI / 2x CAN / Anti-Lock Brakes, RPM, Fan
- Multimedia Examples
  - JPEG, MP3 (w/o, w/ HW support)



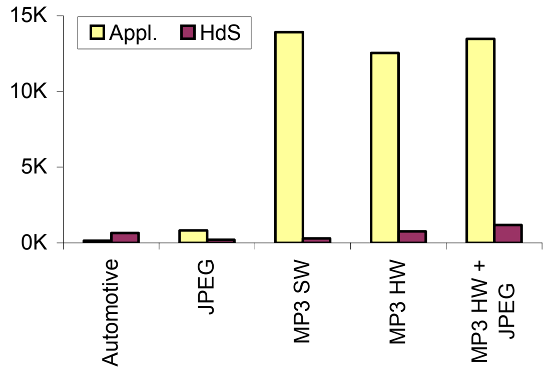
Experimental Results



CPU Utilization



CPU Cycles



Generated Lines (Appl., HdS)

Conclusions

- High-Level SW Development
  - Software Generation
    - Code Generation
    - Hardware-dependent Software
      - Communication synthesis
      - Multi-task synthesis
      - Binary image creation
  - Seamlessly intergraded into ESL Flow
    - High-level specification, generate low-level implementation
  - From abstract model to implementation!
    - Eliminates tedious and error prone manual SW development
    - Significant productivity gain
    - Enables rapid design space exploration

Acknowledgements

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

Thank You!