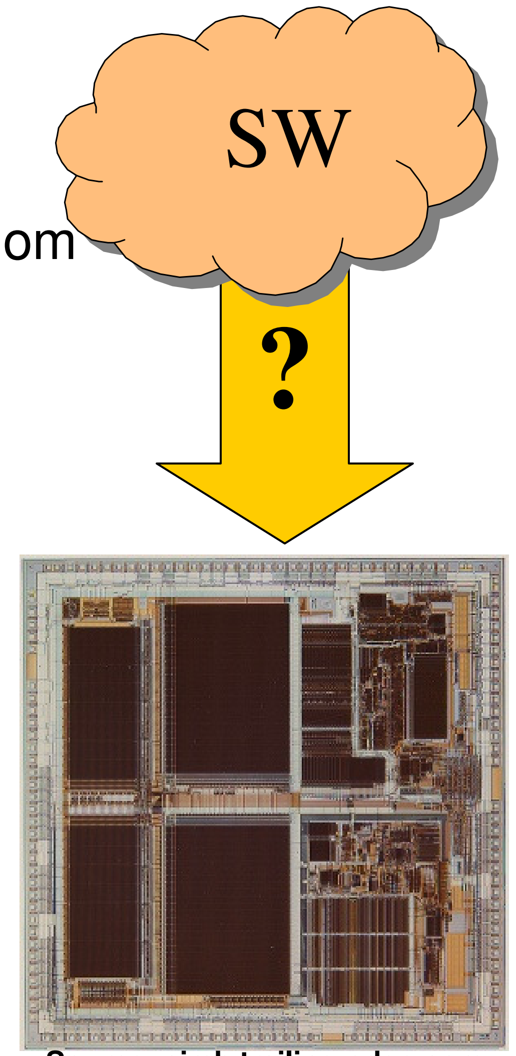
Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications

Gunar Schirner, Andreas Gerstlauer
and Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine

Motivation

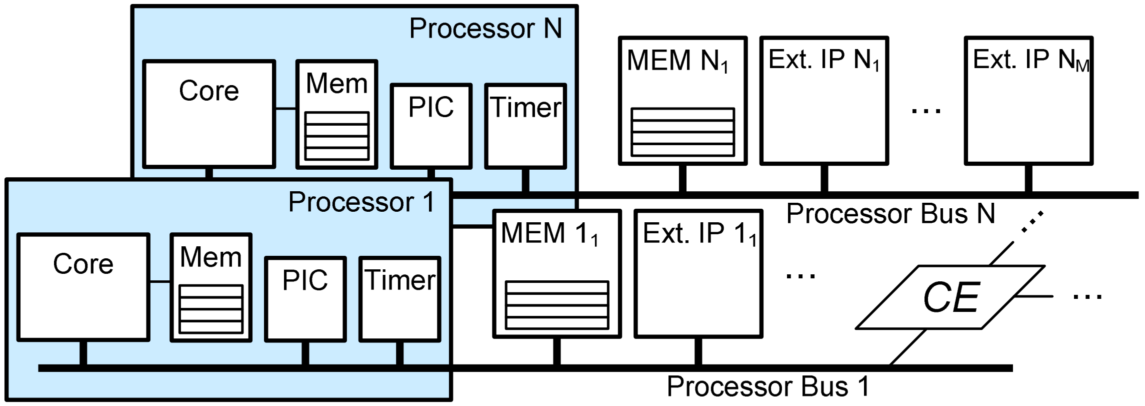
- Increasing Complexity of MPSoCs
  - Feature demands
  - Production capabilities + Implementation freedom
- System Level Design
  - Transaction Level Models (TLMs)
    - Design space exploration
    - Early development
- Increasing software content
  - Flexible solution
  - Addresses complexity
- How to create SW for MPSoC?
  - Avoid break in ESL flow:
    - Synthesize SW from abstract models



Source: simh.trailing-edge.com

Problem Definition

- Generate SW binaries for MPSoC from abstract specification
  - Eliminate tedious, error prone SW coding
  - Support wide range of system sizes
    - with RTOS
    - without RTOS (i.e. interrupt driven)
- Execution on
  - Real HW
  - Virtual Platform (ISS-based)



Outline

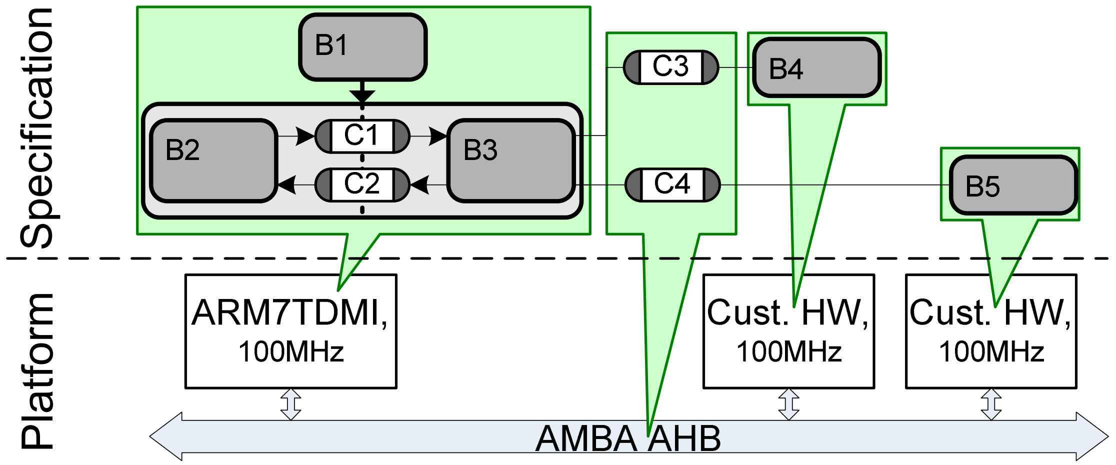
- Introduction
- Related Work
- System Design Flow Overview
- Hardware dependent Software Synthesis
  - Communication
  - Multi-Tasking
  - Binary Image Creation
- Experimental Results
- Conclusions
- Future Work

Related Work

- System Level Design Languages (SLDL)
  - SystemC® [Groetker et. al, 2002], SpecC [Gajski et. al, 2000]
- SLDL for SW Modeling + ISS Virtual Platform
  - [Yu et. al, CODES+ISSS 2003], [Kempf et. al, DATE 2005], [Desmet et. al, DAC 2000], [Bouchhima et. al, ASPDAC 2005]
  - [Benini et. al, JVLSI 2005]
- SW Synthesis from very specific input models
  - POLIS [Balarin et. al, 1997], DESCATES [RITZ et. al, 1993]
- Partial SW Synthesis Solution
  - [Herrera et. al, DATE 2003] replicates simulation primitives on target
  - [Krause et. al, DAES 2005] application mapping to real RTOS
  - [Gauthier et. al, TCAD 2001] focus on OS
  - [Yu et. al, ASPDAC 2004] focus on application only

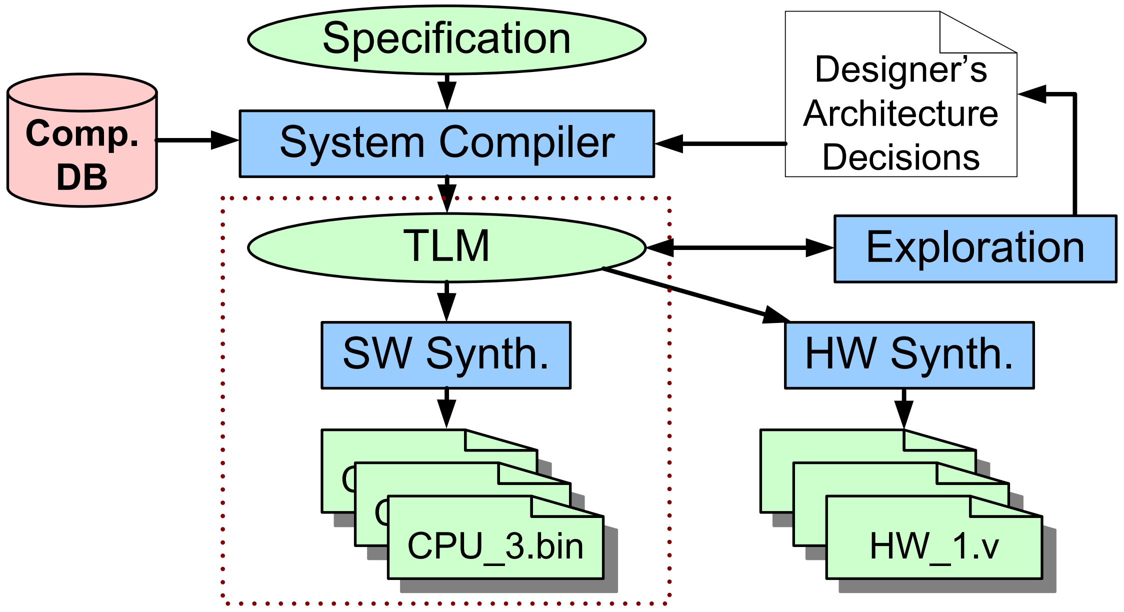
Specification

- 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)
  - Define platform and application mapping



System Design Flow Overview

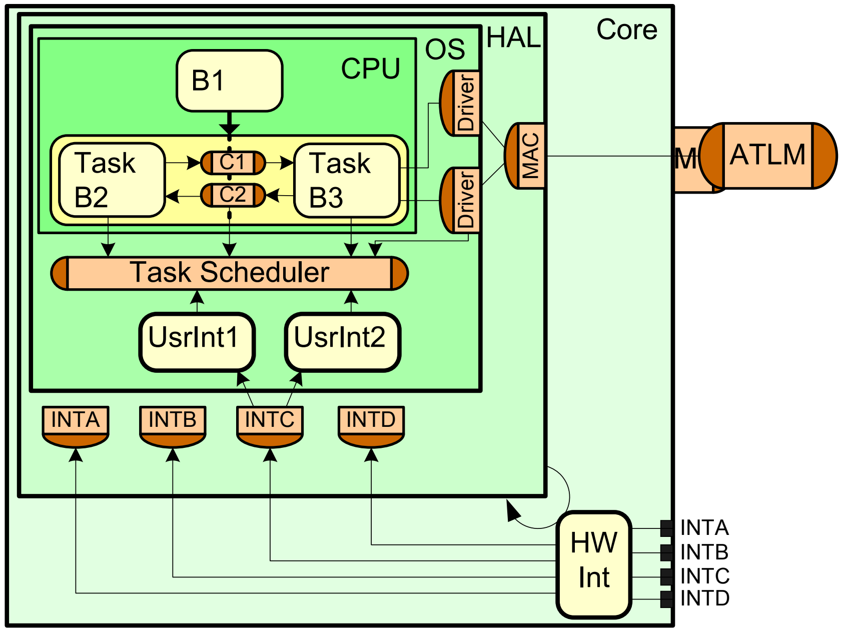
- Two Step Approach
  - TLM Generation
    - Use TLM for
      - Design Space Exploration
      - Analysis
      - Development / Debugging
  - Synthesis
    - Hardware
    - Software



Automatic TLM Generation

First step:

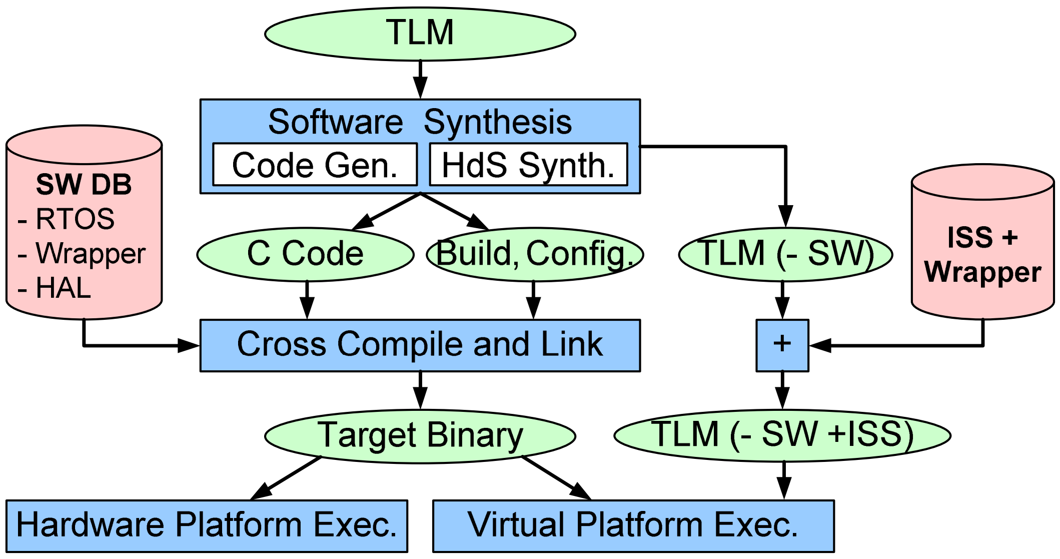
- System TLM Genration
  by system compiler (SCE)
  - Map application to processing elements
  - Group computation to tasks
  - Refine communication to OSI layer-based stack
  - Model contains complete implementation information
  - TLM Generation described in:
    - [Gerstlauer et. al, TCAD 09/2007],
      [Shin et. al, CODES+ISSS 2006],
      [Peng et. al, ASPDAC 2002], [Schirner et. al, ASPDAC 2007]



Software Synthesis

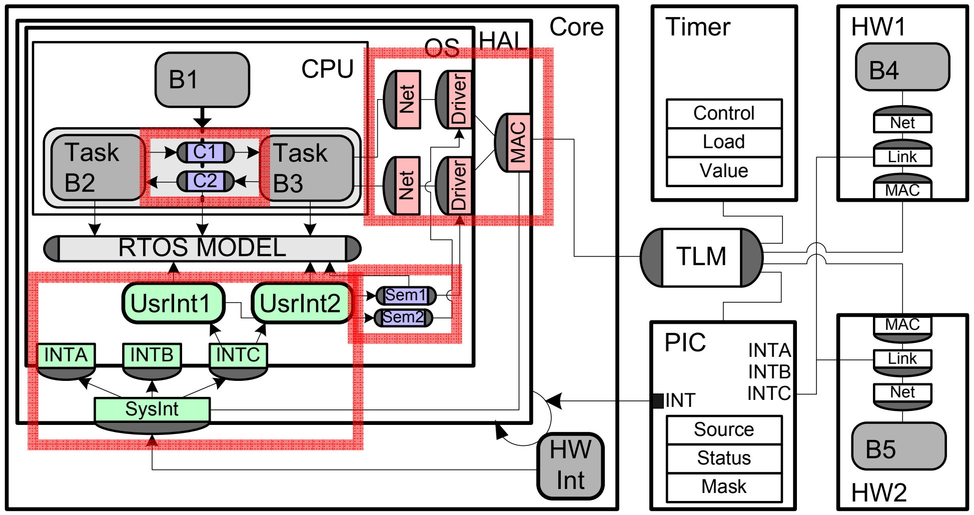
Second step: Software Synthesis

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



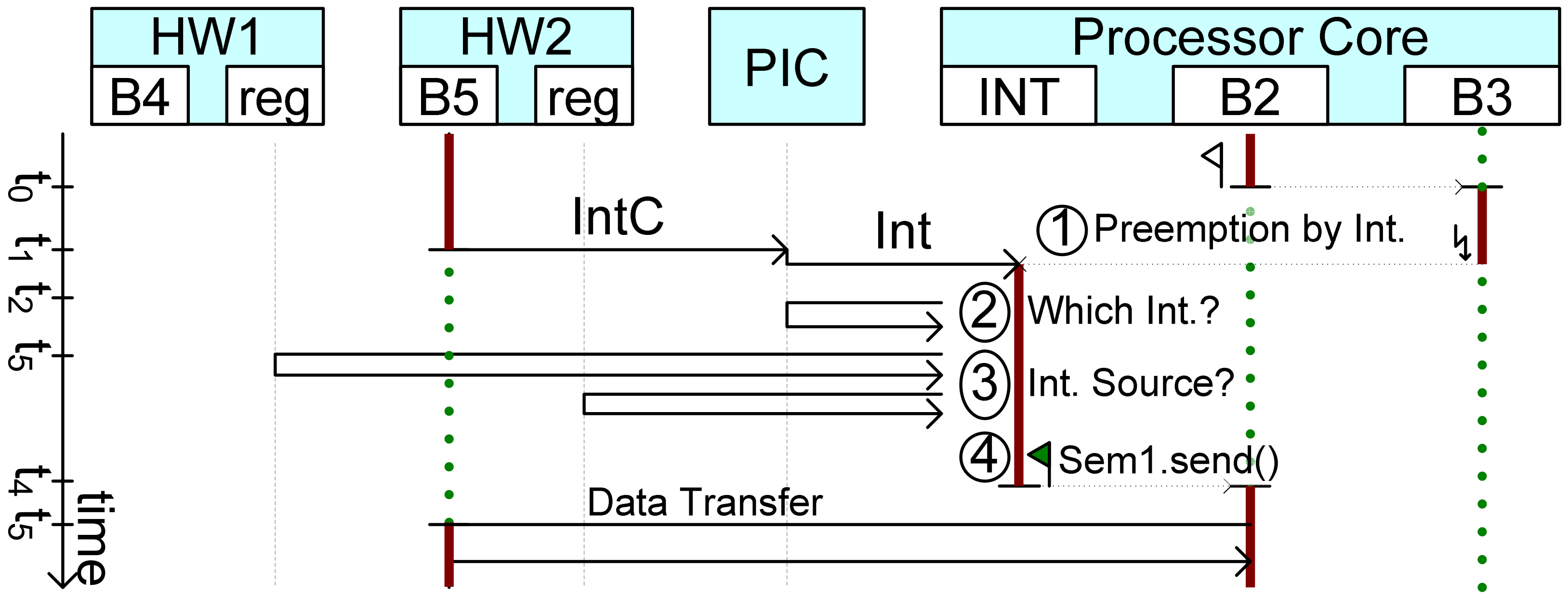
Hardware dependent Software

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



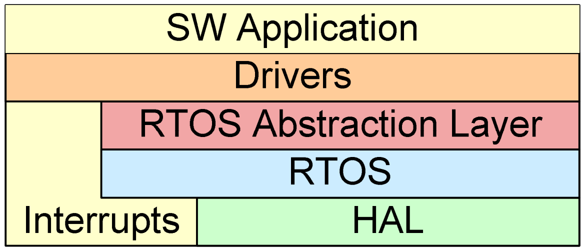
Hardware dependent Software

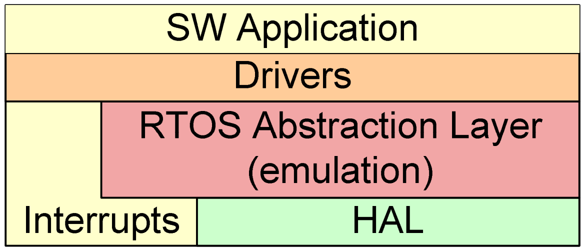
- Example synchronization with interrupt
  1) Low level ASM int. handler
  2) System interrupt handler
  3) User specific interrupt handler
  4) Semaphore releasing task



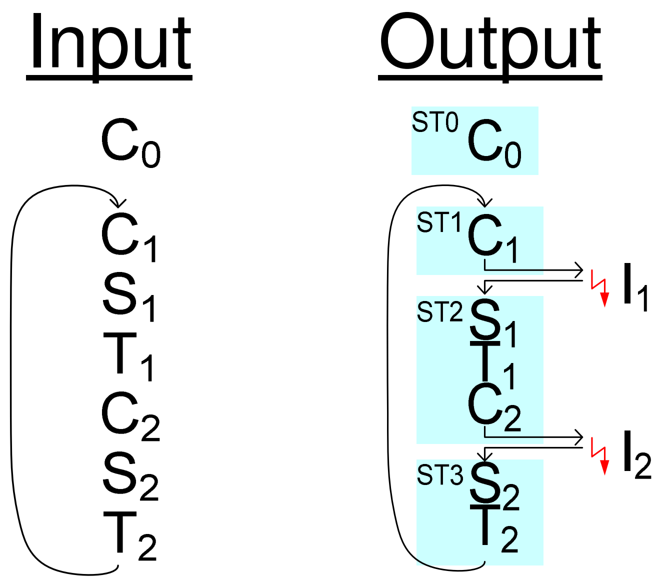
Hardware dependent Software

- Multi-Task Synthesis
  - RTOS-based Multi-Tasking
    - Based on off the shelf RTOS (e.g. µC/OS-II)
    - RTOS Abstraction Layer
      - Limit interdependency RTOS / Synthesis
      - Canonical Interface
    - Generate task management code
    - Internal communication
  - Interrupt-based Multi-Tasking
    - Alternative when no RTOS desired
      - Availability, footprint, simplicity
    - Use interrupts for multiple threads execution





Interrupt-based Multi-Tasking

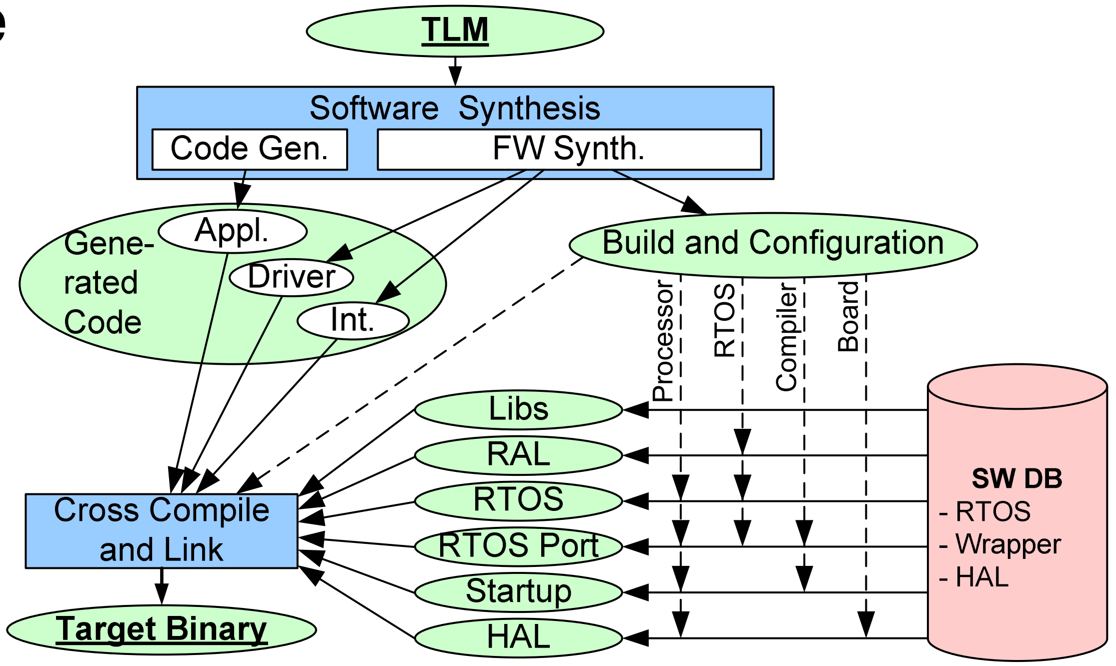


C_n	Computation
S_n	Synchronization
T_n	Transfer (Data)
I_n	Interrupt

- Assume only interrupt based synchronization
- Code composed of C_n, S_n, T_n
- Convert task code to state machine
  - Each synch. point starts new state
- Execute state machine in interrupt

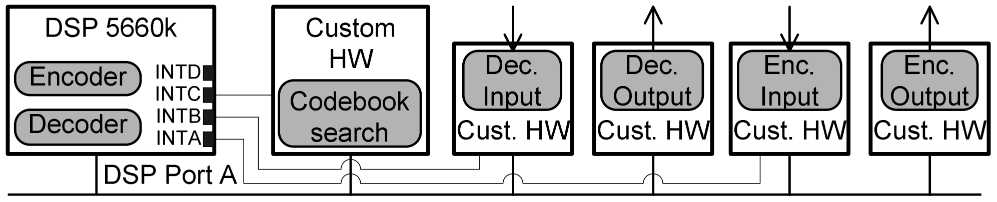
Binary Image Generation

- Generate binary for each processor
  - Generate build and configuration files
    - Select software database components
    - Configure RTOS
  - Cross compile and link

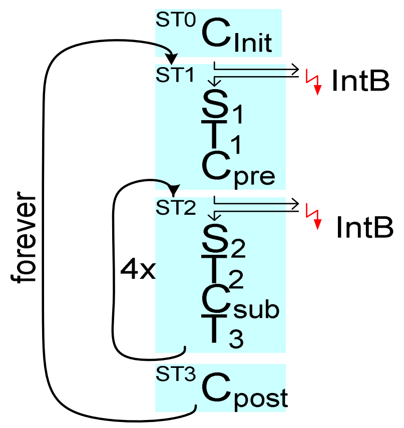


Experimental Results

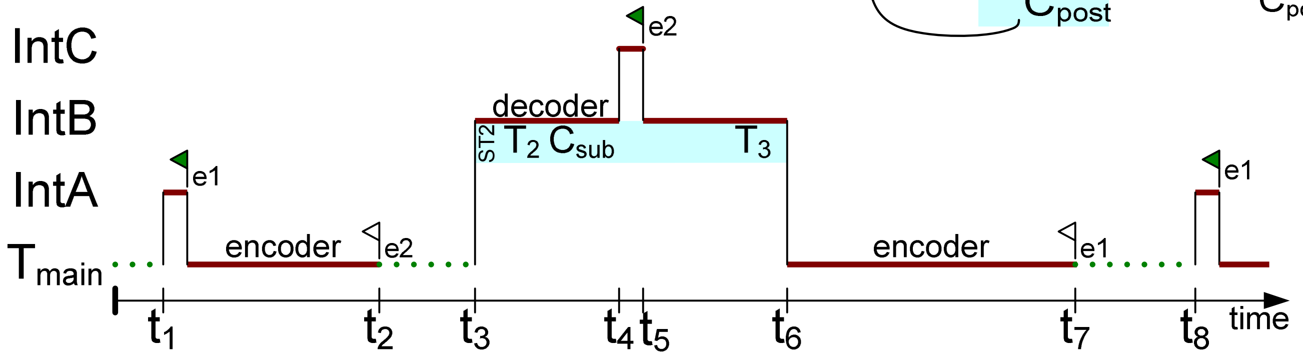
- Interrupt-based example:
  - GSM transcoding
    - Motorola DSP 5660
    - Interrupt driven, no RTOS



Decoder state machine

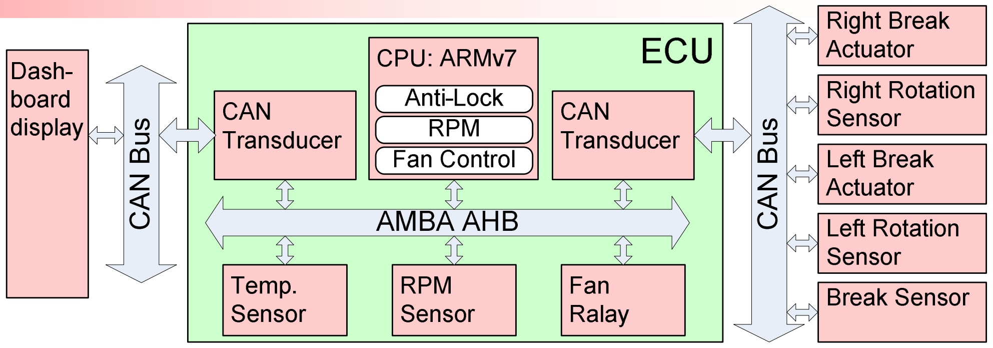


C_{init}	Initialize the decoder
S_1	Parameters available
T_1	Receive parameters
C_{pre}	Preprocessing (linear pred.)
S_2	New sub frame ready
T_2	Receive sub frame
C_{sub}	Decode sub frame
T_3	Output sub frame
C_{post}	Post processing



Experimental Results

- Automotive Example
  - ARM7TDMI
  - 2x CAN



- Generated both multi-tasking approaches
- Interrupt-based:
  - Less memory (OS-code + stack)
  - Fewer busy cycles
  - Lower latency

Multi-tasking	RTOS-based	Interrupt-based
Footprint	36224 Bytes	21052 Bytes
Alloc. Stacks	4096 Bytes	1024 Bytes
CPU Busy Cycles	6.706 MCycles	5.106 MCycles
Latency RPM Task	1794 Cycles	1001 Cycles
# Interrupts	1478	1027

Experimental Results

- Synthesis Results
  - 6 Applications
    - diff. complexities
    - e.g. 2 … 14 ISRs
- Generated HdS
  - 210 … 1186 lines
- HdS generated in less than 1s
- Manual implementation would take 12 to 79h
- Significant productivity gain

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 Systematic HdS synthesis approach
  - Communication synthesis
  - Multi-task synthesis
  - Binary image creation
- HdS synthesis integrated into ESL flow
  - Seamless solution
- 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

Acknowledgements

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

Thank You!

Interrupt-based Multi-Tasking

- Source code excerpt:

```
void intHandler_I1() {
  release(S1);        /* set S1 ready */
  executeTask0();     /* task state machine */
}
void executeTask0() {
  do { switch(State) {
      /* ... */
     case ST1: C1(...);
               State = ST2;
     case ST2: if(attempt(S1)) T1_receive(...);
               else break;
               C2(...);
               State = ST3;
     case ST3: /* ... */
  } } while (State == ST1);
}
```