A Formal Model and Methodology for Hardware/Software Codesig

Harry Hsieh, Bassam Tabbara
Alberto Sangiovanni-Vincentelli
University of California, Berkeley, CA

SRC Task Review, April 15-16, 1997

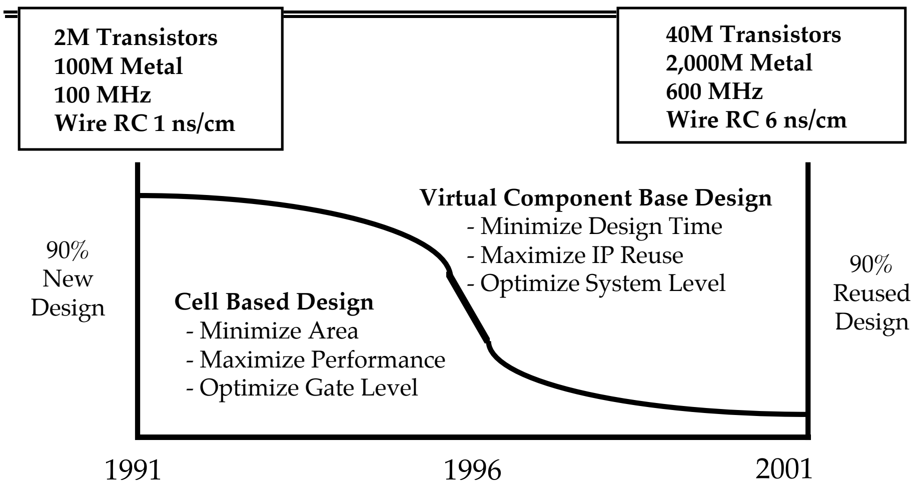
The Opportunity..

"The System Level Integration (SLI)* ASIC market will reach $14 billion by the year 2000 and account for about 60% of the ASIC Market."

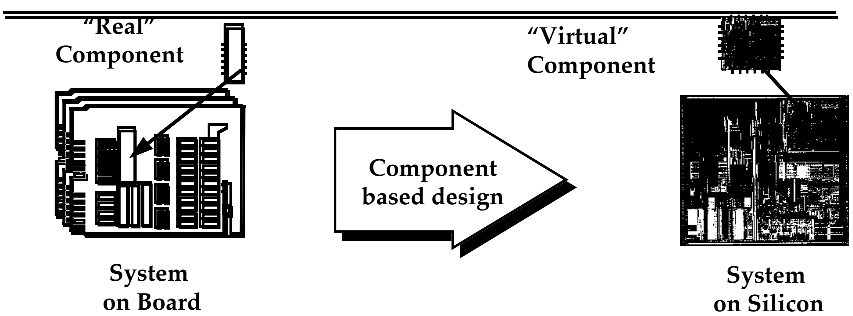
Dataquest
Dec. 18, 1995

**SLI =
>100k gates, >1 Programmable Core, + Memory*

Deep Submicron Paradigm Shift

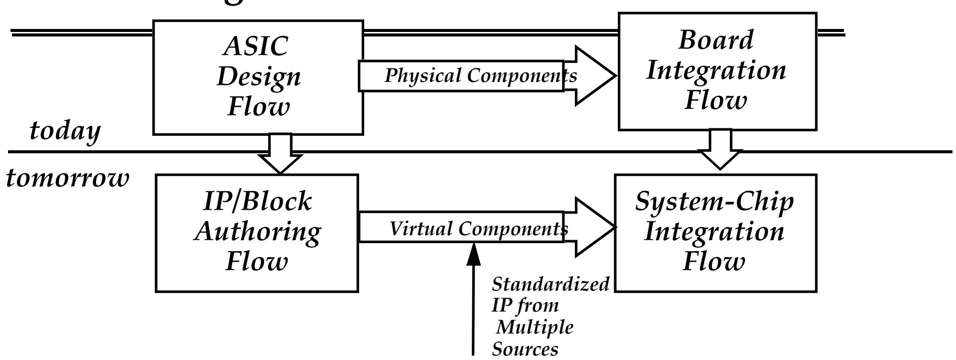


Virtual Components and System Design



- Select the best components from several different vendors
- Integrate the system using standard interfaces
- Validate the design functionality, performance and reliability.

The Challenge



SEMATECH Design Investment Roadmap (spec to tape out)

1995	.35u	5.1MTr	362 staff-years	$51M
1998	.25u	20.2MTr	818 staff-years	$114M
2001	.18u	81.3MTr	1846 staff-years	$256M

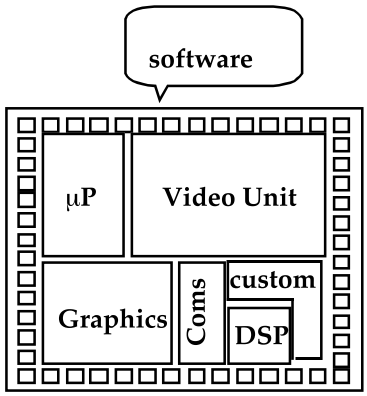
** assumes 21%/yr productivity improvements*

Microprocessors vs. Systems

- *~5 companies*
- Implementation from the ground up required by Process Technology
- Intensive Design & Verification effort (2+ yr)
- Design cost justified by volume

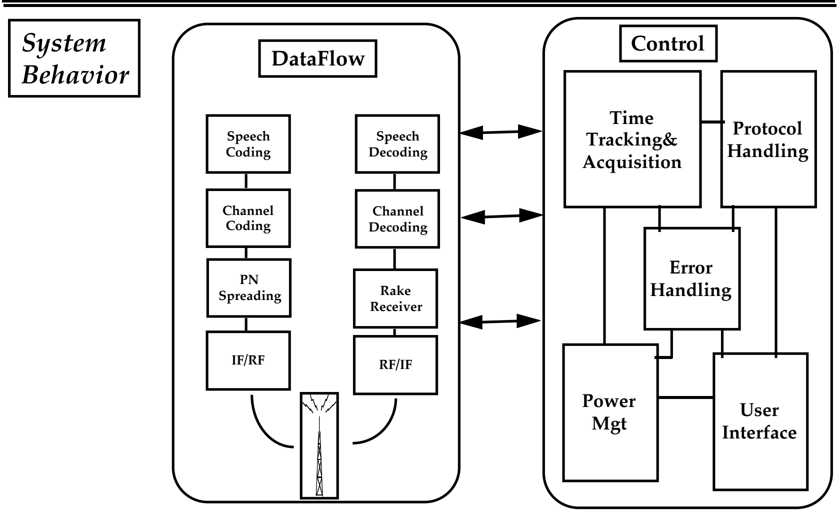
- *~$1,000 billion industry*
  - Digital Communication
  - Video Conferencing
  - Set Top Boxes
  - Automotive Automation
- Implementation from the ground up incompatible with Time to Market (0.5yr)
- Embedded Applications: Software

The new IC: "Microsystems"

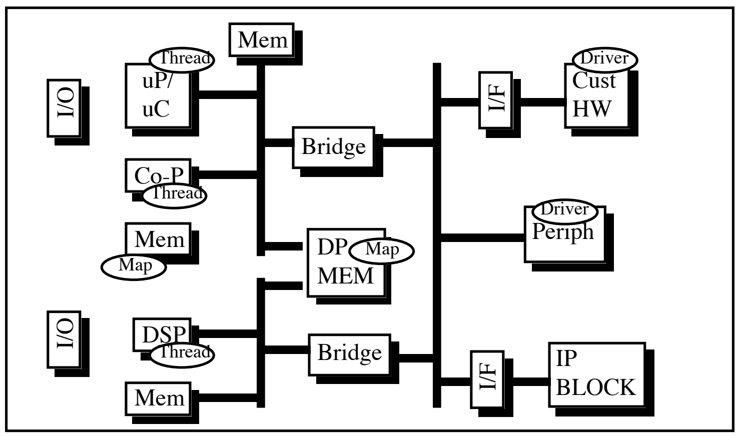


- Assembly of "prefabricated component" often purchased from external vendors ("IP")
  - "black box" hierarchy
- Design & Verification at the System level
  - rather than the logic level
  - Interface and communication

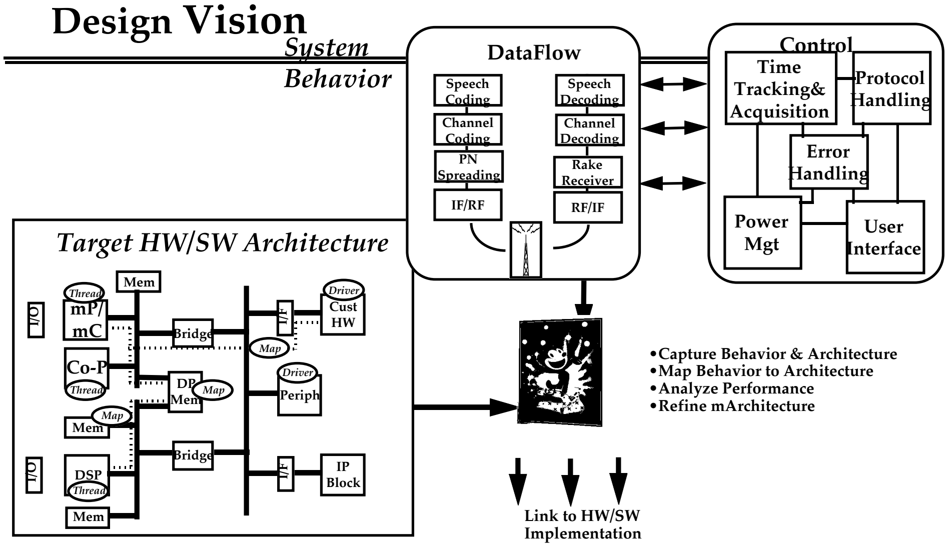
System Behavior



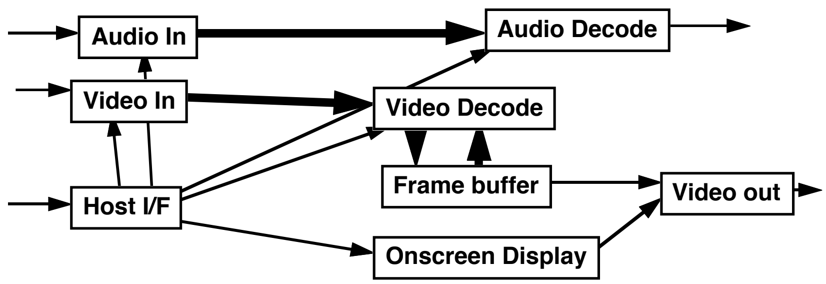
Target HW/SW Architecture



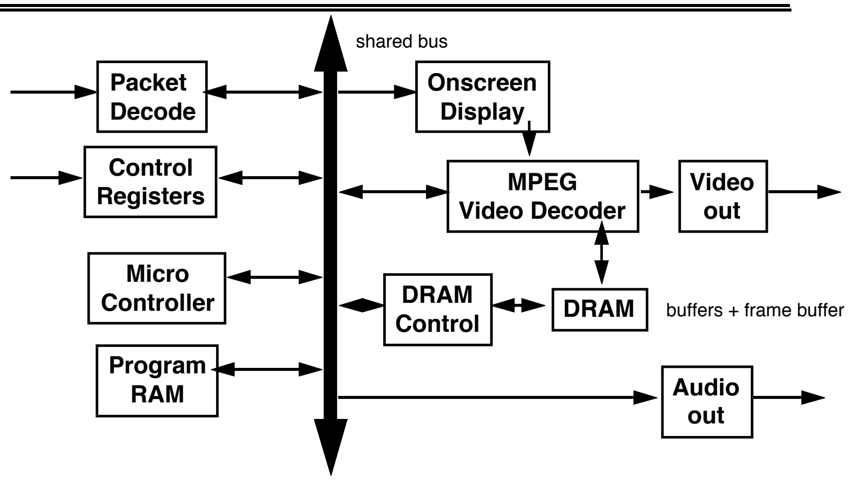
Design Vision



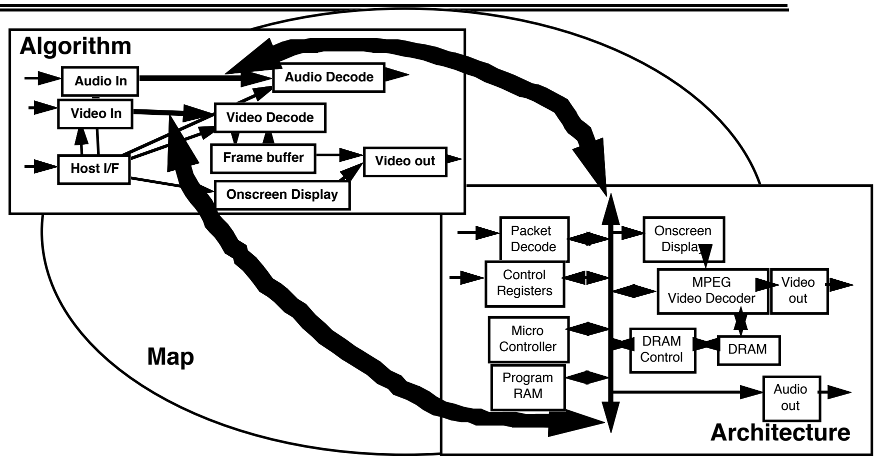
MPEG Behavior



MPEG Implementation Architecture



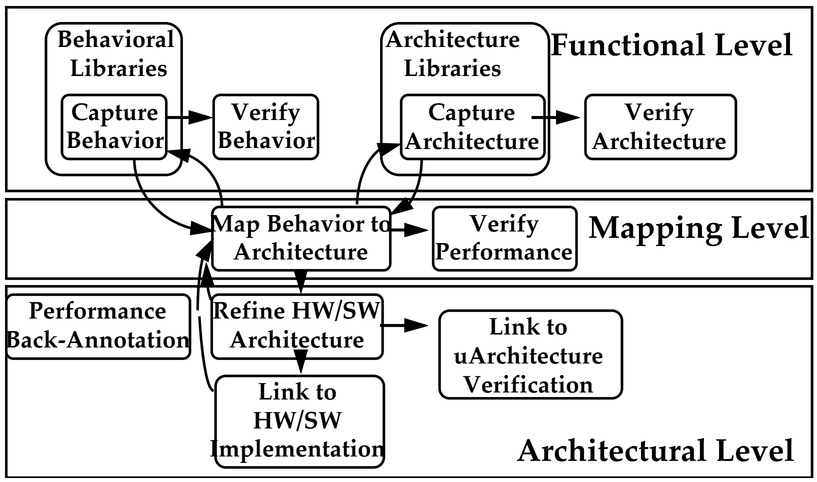
Mapping Algorithm to Architecture



The Team

- *UC Berkeley*: **H. Hsieh, B. Tabbara, M. Sgroi, A. Ferrari**
- *Magneti-Marelli Elettronica*: **P. Giusto, A. Jurecska, A. Damiano**
- *Cadence Berkeley Labs*: **L. Lavagno, F. Balarin, E. Sentovich, C. Passerone, P. Scaglia**
- *Cadence Alta System*: **M. Chiodo, Jim Rowson, W. Baker**
- *Daimler-Benz*: **R. van Haxleden**
- *Hitachi*: **K. Suzuki**

Proposed Design Methodology



Essence of Design Method

- Integrate HW & SW IP with FSM glue
- Algorithm composition in SDF and DE
- Separate capture of architecture
- Map algorithm onto architecture
- Analyze performance of algorithm mapped onto architecture
- Refine architecture into implementation

Important Technologies

- Map algorithm onto architecture
  - Incremental refinement of communication
  - Explicit and separate architecture
  - Many software blocks can share processor
  - Hardware IP mapped 1 to 1
- High speed performance analysis of algorithm mapped onto architecture
  - Use Architectural Delay Equations
  - Simulate algorithm with annotated architectural delays
  - Estimation for control software

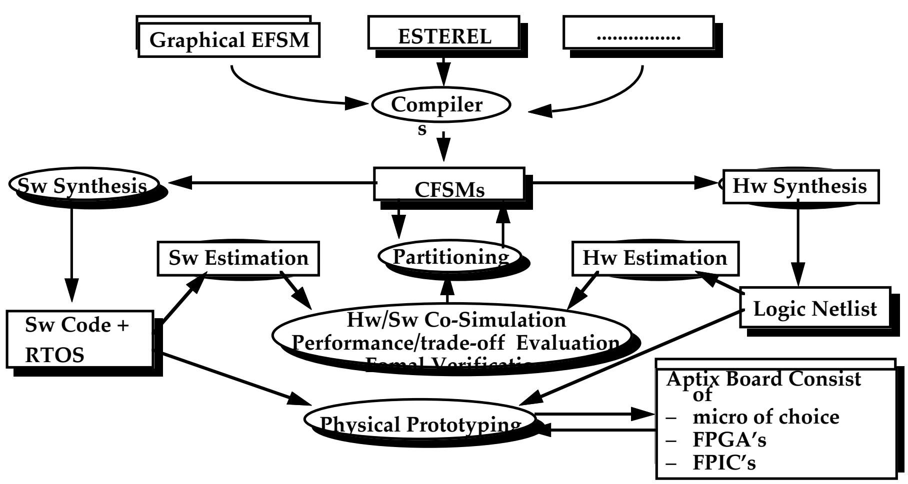
Important Technologies

- Manually created glue, synthesized implementation
  - As part of refinement, user creates glue FSMs
  - Software synthesis, HDL generation for implementation

Computer Assisted Refinement

- Design Tool does not synthesize architecture
- User does trade-offs, mapping, refinements
- User programs protocol, refines high level communication
- Tool helps
  - Estimate performance
  - Guide methodology by separating descriptions
  - Synthesize implementation of user defined refinements

Polis Methodology

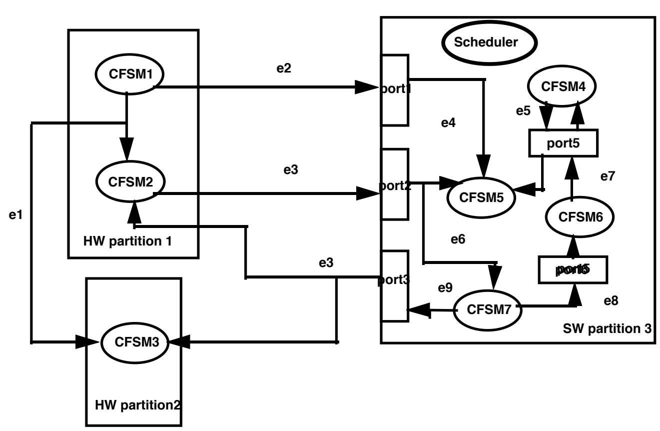


Codesign Finite State Machines

- We have chosen an FSM model for
  - uncommitted
  - synthesizable
  - verifiable

  HW/SW specification
- Translators from state diagrams, Esterel, HDLs into a single FSM-based language
- Need efficient hw/sw communication primitive:
  - Event broadcasting
- Software response could take a long time:
  - Unbounded delay assumption

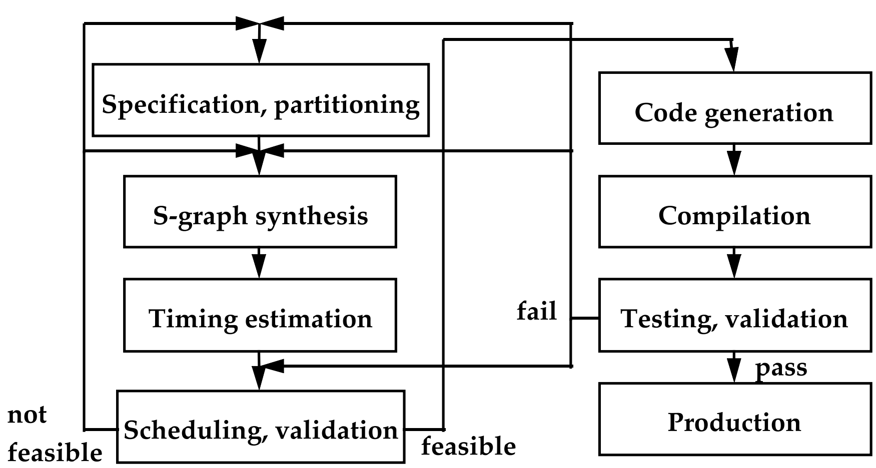
System Partitioning



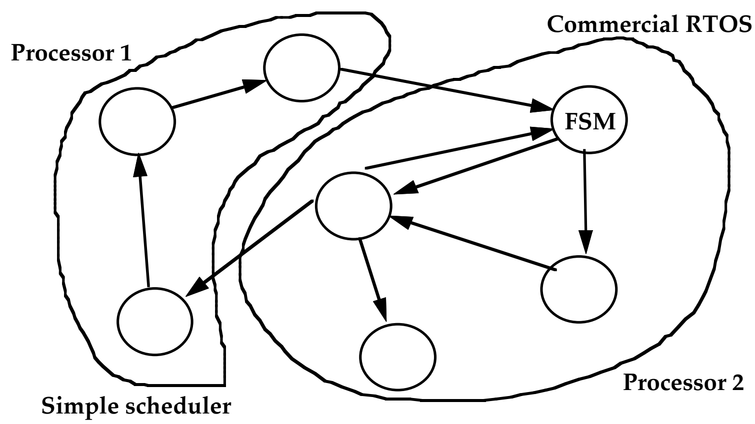
Software Implementation

- How to do it ?
- Traditional approach:
  - hand-coding of procedures
  - hand-estimation of timing input to scheduling algorithms
- Long and error-prone
- Our approach: three-step *automated* procedure:
  - synthesize each task separately
  - extract (estimated) timing
  - schedule the tasks
- Customized RT-OS (scheduler + drivers)

Software synthesis procedure

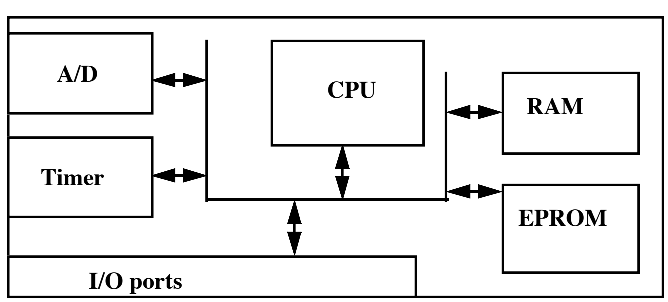


Software Partitioning



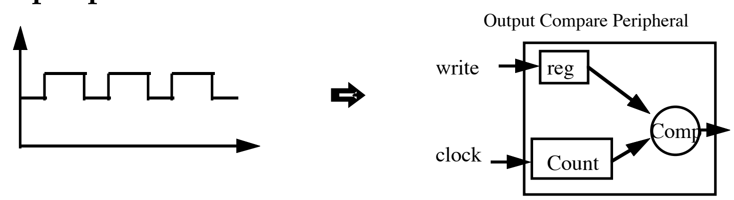
Microcontroller Peripherals

- Many peripheral devices exist on microcontroller



Use of Peripheral Devices

- Implementing function with peripherals are often
  - Cheaper than HW implementation (ASIC...)
  - Faster than SW implementation
- Example of functions often implemented with peripherals: Pulse-Width Modulation



Design Methodology

- General methodology:
  - Efficiently simulate and evaluate a particular implementation choice
  - Automatic downstream synthesis

- Must be able to perform fast system co-simulation
  - Based on synthesized software timing estimates
  - Synthesized C code annotated with clock cycles

Peripheral Modeling Approach

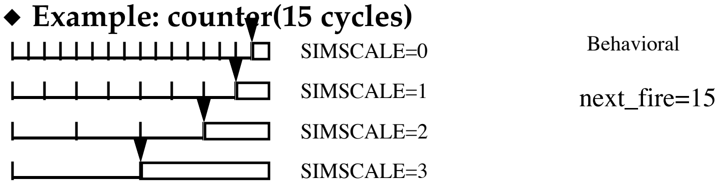
- Three different models may be used
  - CFSM model: co-simulation and rapid prototyping
  - C model: programming and interfacing peripherals
  - Behavioral model: specific to simulators (Ptolemy)
    - Speed-up by calling simulator routine directly
    - Not cycle accurate
- Parameters customize models simultaneously
- Synthesiable CFSM model key to re-targetability

Designing with Peripherals

- The user must
  - Pick functions that may be implemented on peripheral
  - Choose the best fitting model from a library
  - Co-simulate to decide implementation
- The co-design environment takes care of
  - Synthesis in SW or HW
  - Extracting peripheral programming SW from library (may be micro-controller independent)

Speed-up simulation

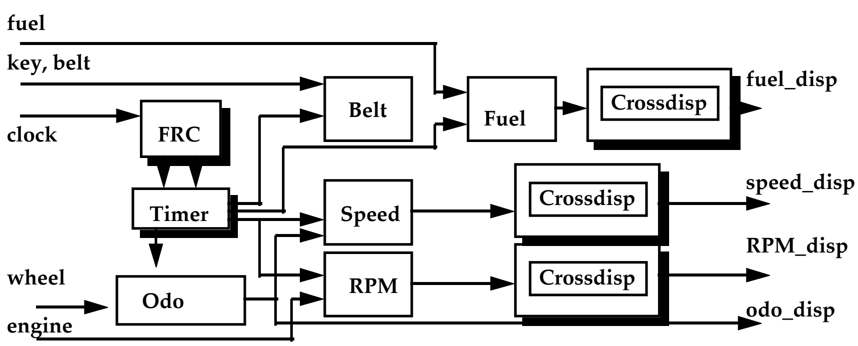
- **RTL model**
  - **Trade-off between speed and precision by dividing clock (simulation scale)**
  - **Higher scale = less accurate = faster simulation**
- **Behavioral model: Use Ptolemy function directly**
- **Example: counter(15 cycles)**



Car Dashboard Example

☐ Can be implemented with peripheral

☐ Implemented in SW



Result

- Trade-off between speed and accuracy:

	clock cycle per second
RTL (simscale=0)	2000
RTL (simscale=5)	50,000
Behavioral	260,000

60MHz ULTRASparc

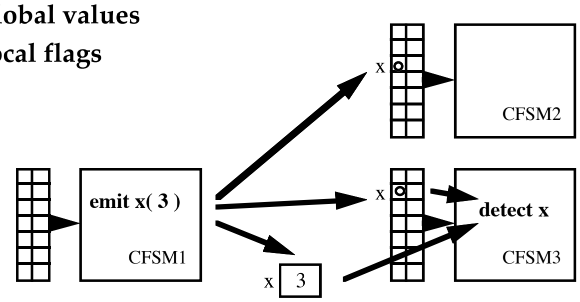
- Flexibility in implementation is partially retained.
  - Peripherals can be re-mapped into HW (or SW):
    - 374 CLB, 60 I/O pads: An extra Xilinx 4010

Generated RT-OS

- Function:
  - Enable communication between SW-CFSMs and SW-CFSMs, HW-CFSMs, or MP-CFSMs
    - Define *emits* and *detects* using memory, I/O ports and interrupts
  - Coordinate SW-CFSMs
    - Keep track which SW-CFSMs are ready to execute and decide which one to execute
- Advantages
  - Communication known in generation time
  - Small, only necessary features are included

Events: SW to SW

- **For every event, RTOS maintains**
  - **Global values**
  - **Local flags**



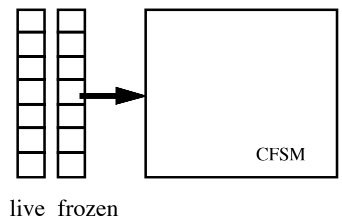
Events: atomicity problems

TASK 1	TASK 2	TASK 3
detect x		
	emit x	
		if detect x then emit y
detect y		

- **TASK 1 detects y AND NOT x, which is never true**
- **To avoid, need atomic detects**

Events: SW to SW

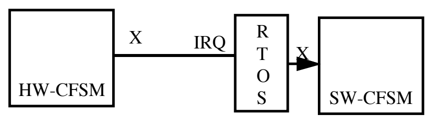
- **For atomicity:**
  - **Always read from frozen**
  - **Others always write to live**
  - **At the beginning of execution, switch**



Events: HW to SW

- **Event can be polled or driving an interrupt**
- **For polled events:**
  - **Allocate I/O port bits for value, occurrence and acknowledge flags**
  - **Generate the polling task that acknowledges and emits all polled events that have occurred**
- **For events driving an interrupt:**
  - **Allocate I/O port bits for value**
  - **Allocate an interrupt vector**
  - **Create an interrupt service routine that emits an event**

Events: interrupts



- **Interrupt service routine:**

```
{
  emit x
}
```

- **Optional interrupt service routine**

```
{
  emit x
  execute SW-CFSM
}
```

Events

- **SW to HW**
  - **Allocate I/O port bits for value and occurrence flag**
  - **Write value to I/O port**
  - **Create a pulse on occurrence flag**
- **SW to/from MP**
  - **Use *emit*, *detect* functions from the library**

Coordination

- Consider SW-CFSM ready to run whenever it has some not consumed input events
- Choose the next ready SW-CFSM to run according to:
  - Round-robin scheduling, or
  - Static priority scheduling

Experiments

- A subset of the dashboard controller (seatbelt)
- Performed on a rapid prototyping environment
- Typical commercially available RTOS
  - Offer over 60 functions for multi-tasking

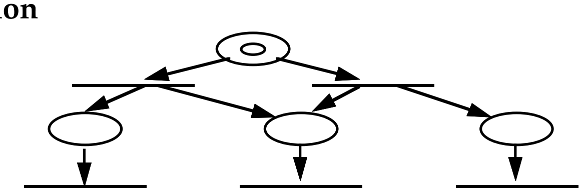
	Commcial	Polis
Speed of polling	969us	230us
Speed of emission	150us	15us
Size	4.7kb	1.5kb

Conclusions

- RTOS provides:
  - Coordination of SW tasks
  - Communication mechanisms among SW tasks and between SW tasks and the rest of the system
- In a methodology based on a formal model, it is possible to generate RTOS automatically
- In size and efficiency, it competes with hand-crafted RTOSs

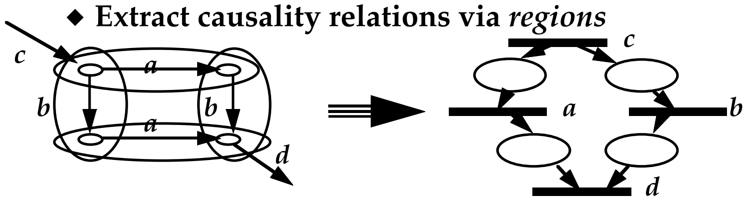
Petri Nets

- Abstract model for concurrent systems
- Represents explicitly:
  - Causality (sequencing)
  - Concurrency
  - Conflict (choice)
- Sets of Transitions and Places, joined by a Flow Relation



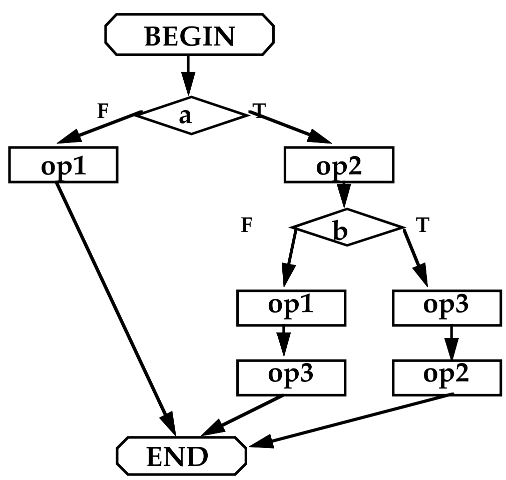
Petri Net Synthesis

- **Start from State Graph (FSM)**
- **Extract causality relations via *regions***

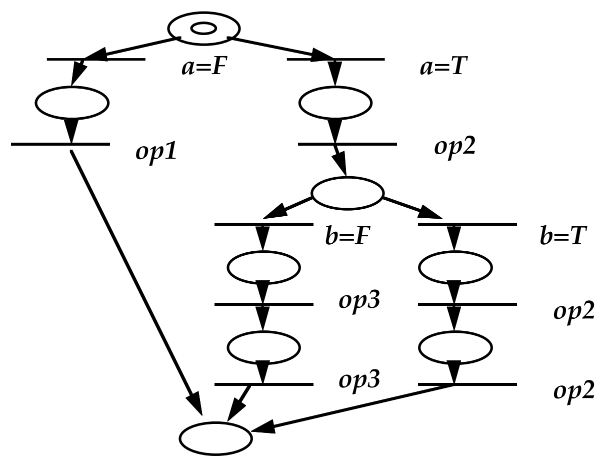


- **Cost function can minimize first transitions, then places**
- **Code cost minimization:**
  transitions = code fragments
  (s-graph nodes)

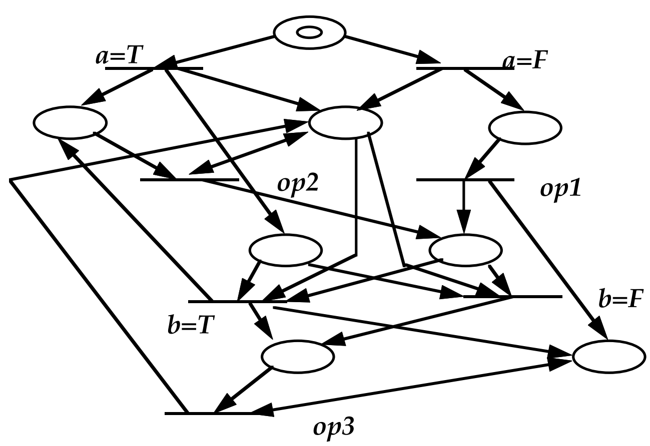
Example S-graph



From S-graph to PN



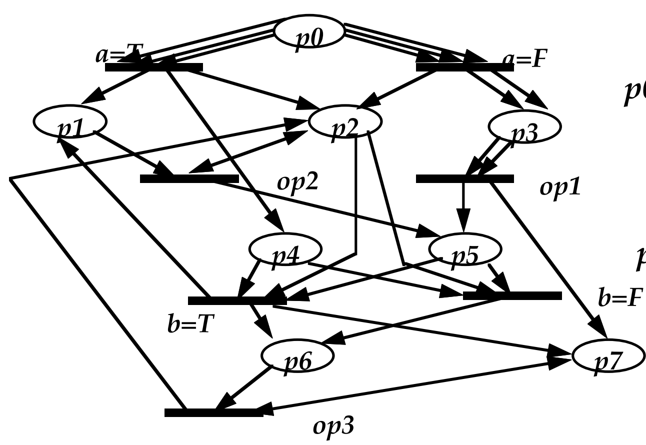
Minimized PN



From PN to S-graph

- How to reconstruct an s-graph ?
- Key notion:
  - Sub-network with single-predecessor, single-successor transition
  - Choice, no concurrency
  - Easy to implement in software...
- Observation: the original PN was a State Machine
- Theorem: any PN can be made *SM-coverable*:
  - Each place, transition and edge belongs to at least one SM
  - All predecessors and successors of a place belong to same SM
  - May need redundant places

Finding an SM Cover



sm0:
p0,p3,p4,p7
sm1:
p0,p2,p6
sm2:
p0,p1,p3,p5

From SM Cover to C Code

- Largest SM implemented by the program counter
- Other SM implemented by integer variables
- Encoding of SM components aimed at
  - Distinguishing every pair of markings
  - Minimizing the number of assignments
    (relation to switching power minimization)
- Currently: greedy heuristics

Minimized C Code

- Encode *sm1*: p0,p2=0, p6=1
- Encode *sm2*: p0,p1=0, p3,p5=1

```
begin: sm1 = sm2 = 0
p0:     if (a) { goto p4; }
        else { sm2 = 1; goto p3 }
p4:     op2;
        sm1 = 1;
        if (b) { sm2 = 0; goto p7; }
        else { sm2 = 1; goto p3; }
p3:     op1;
        goto p7;
p7:     if (sm1) { op3; }
        if (!sm2) { op2; }
```

- Need to duplicate op2 because not covered by *sm0*

Conclusions and Future Work

- **Petri net minimization can be used to minimize code size**
- **Minimization useful only if action code is large**
- **Dashboard example: significant improvement only in one case (computing sine and cosine)**
- **Can generate *single-appearance schedule* for control-dominated code**
- **Future work:**
  - **explore better encoding techniques**
  - **trade off between added and merged TESTs and ASSIGNs**