



---

# **vFaad: von Neumann Formal Analysis and Annotation Tool**

**David Greve  
Dr. Matthew Wilding**

**Rockwell Collins  
Advanced Computing Systems  
Cedar Rapids, IA**

**dagreve@rockwellcollins.com  
mmwildin@rockwellcollins.com**

**April 2003**



## ● Rockwell Collins History

- Have 10+ years experience in applying Formal Methods
  - Have embraced and extended a variety of techniques

## ● Observations

- Advantages to Accurate Low-Level Models
  - Abstraction can be tedious
- Domain knowledge
  - Missing from generic theorem provers
  - Can be codified
- Techniques generalize to
  - Different Verification Targets
  - Different Theorem Provers

**vFaot: A collection of tools and techniques to help simplify reasoning about complex systems”**

## ● Conclusion

- Judicious use of automation simplifies verification task
  - Improve Productivity, Extend Scope



- **Motivating History**
- **Observations**
- **Future Directions**



# RC Formal Methods Projects

---

- **AAMP5/AAMP-FV**
- **JEM1: Symbolic Simulation**
- **JEM2: Executable Formal Models**
- **CAPS: High-Assurance Processor**
- **AAMP7: Intrinsic Partitioning**



**“Formal Verification of the AAMP5 Microprocessor”, NASA Report 1995**  
**“Formal Verification of the AAMP-FV Microcode”, NASA Report 1999**

## ● Goal

- Demonstrate use of Formal Methods in Industrial Setting

## ● Project

- Sponsored by NASA Langley
- Used PVS and teamed with SRI
- Abstract representation of instruction execution created
- Functionally described Microarchitecture
- Standard Commuting Diagram Proof



- **Inspections connected model to implementation**
  - Found variety of errors in formal specification
- **Was there a better way to validate the model?**
  - Better Integration with design process?
- **Verified a number of instructions**
  - Even found a few bugs
- **Brute Force Formalization**
  - Microcode specified by hand
  - “Clock functions” crafted by hand
- **Automated microcode specification an obvious next step**



# JEM1: Symbolic Simulation

**“Symbolic Simulation of the JEM1 Microprocessor”, FMCAD-98**

## ● Goals

- Integration of Formal Models into Design Process
- Leverage Automated Analysis
- Detect Microcode Errors (Bug Finding)

## ● Project

- Specified JEM1 Microarchitecture in PVS
- Used PVS to execute symbolically the model
  - Generated Symbolic Results for Microcode Basic Blocks
- Analyzed Symbolic Results as part of Microcode Inspections



# JEM1: Symbolic Simulation

---

- **Symbolic Results Difficult to Read**
  - Good for detecting data-flow errors (Definition/Use)
    - Unexpected Side-Effects
    - Unexpected Data-Dependencies
- **Demonstrated Effective Use of Automation**
  - Codified knowledge of problem domain
    - Control Flow Analysis Dictated Proof Architecture
  - Employed Automatic Generation of:
    - Function Definitions (uCode)
    - Theorems and Theory Structure (Proof Architecture)
    - Symbolic Results (batch mode PVS)





# JEM2: Executable Formal Models

**“Efficient Simulation of Formal Processor Models”, FMSD 2002**

## ● Goals

- Integration of Formal Models into Design Process
- Improve Model Validation Technique
  - Replace Microcode Simulator with Executable Formal Model

## ● Project

- Modeled JEM2 in logic of ACL2
  - Subset of Common Lisp
- Compiled model to C and linked into GUI development environment



# JEM2: Executable Formal Models

---

- **Impressive Results**

- Final simulator ran as fast as original
  - No penalty to developers for using formal model
- Successfully executed regression tests
  - Guaranteed validity of formal model

- **Exposed ACL2 tool limitations**

- Model was large and complex
- Is it possible to reason about such models?



# CAPS: High-Assurance Processor

**“Evaluatable, High-Assurance Microprocessors”, HCSS-02**

## ● Goals

- Reason about an Executable Formal Model
- Perform Instruction Level Proofs of CAPS processor

## ● Project

- Modeled Microarchitecture of CAPS in ACL2
- Executed Standard Regression Tests to Validate Model
- Formalized a set of CAPS instructions
- Constructed Instruction Level Proofs



The ACL2 CAPS uarch model replaces the C model in the CAPS microcode simulator. The replacement is not observable to users.

**CAPS ACL2 uarch model passes 3-hr standard CAPS regression test!**

High-speed, formal models provide for evaluatability (looks like C, passes regression tests, integrated into dev process, proofs checked)

uLoad	Reset	clkCyc	nicCyc	mapCyc	backstep	Refresh
Current		Previous		uControl		uInstruction Decode
S4:S5 0000 7009	S4:S5 0000 7009	uADR 05a	ZERO 0	CONT		uADR: 05a
S2:S3 0000 0915	S2:S3 0000 0915	uPC 05a	STGN 1	ZERO		
S0:S1 beef 01c9	S0:S1 beef 01c9	SAVE 001	GARRY 0	NOV		
R2 0000 0022	R2 0000 0022	CONST 000	V16 0	R<-A		
R1 0000 0010	R1 0000 0010	NIBL 0	V32 1	S<-0		
RO 0000 0000	RO 0000 0000		SVX 0	F<-S or R		
Q 0000 7061	Q 0000 7061	026b 0 J	UM 0	B<-F		
PAGE 0000 449e	PAGE 0000 449e	010a 0 m	INTR 0	A<-STK2		
TOS 0000 7061	TOS 0000 7061	0203 0 J	SKMT 0	B<-STK1		
LENV 0000 7068	LENV 0000 7068	0109 0 m	PC23 0	data adr = 0		
PC 0000 049c	PC 0000 049c	010c 0 J	MODE 0	NOB LNKO		
Bout beef 01c9	Bout beef 01c9	010a 0 m	LOCK 0	EXCLUDE uCycle		
Aout 01c9 0000	Aout 01c9 0000	0006 0 t	DVR 0			
Sin 0000 0000	Sin 0000 0000	0007 0	Z 1	OPC 0a	ODC 4	
Rin 01c9 0000	Rin 01c9 0000	0058 0 J	CC 0	Int Ctl	Flt Mon	
Fout 01c9 0000	Fout 01c9 0000	0059 0	INTx 00	DAP 00	FM 10	
Bin 01c9 0000	Bin 01c9 0000	005a 0	MASK ff	IN 0	CNT 0	
			SV 7	INTR 0	MCB 0	

Control Word: 0000 0000 301f 3980    Bus: DEN:0 XRQ:0 XAK:0 IR:150a IHI:00 OP:0a DR:01c9 DBW:01c9  
DEN:0 DIF:0 CEF:0 CEN:1 uCyc:1 map:0 MD:0000 RD:150a

show listing

```
intfc::cycle = 0
fcp_ext::set_reset(1)
stb::FCP2K_UENG_RESET
useq::useq_do_reset()
stb::FCP2K_CSTORE_LOAD_CS
stb::FCP2K_PPR0M_LOAD
fcp_ext::set_reset(1)
stb::FCP2K_UENG_RESET
useq::useq_do_reset()
fcp_ext::set_reset(1)
stb::FCP2K_UENG_RESET
useq::useq_do_reset()
fcp_ext::set_reset(1)
stb::FCP2K_UENG_RESET
useq::useq_do_reset()
stb::FCP2K_CSTORE_LOAD_CS
stb::FCP2K_PPR0M_LOAD
fcp_ext::set_reset(0)
intfc::cycle = 0
cycle = 0
cycle = 1000
cycle = 2000
intfc::cycle = 10000
cycle = 3000
cycle = 4000
```

Filter: ACL/fcp2k2/sim/\*.sod

Directories: /sim/, /sim/dbg, /sim/gui, /sim/hen, /sim/prs, /sim/sym, /sim/tests

Files: bist.sod, clean.sod, cputest.sod, diag.sod, diag\_uw.sod, f.sod, fixrs.sod

Select command file to execute: /fcp2k2/sim/diag.sod

File Edit Options Window Setup uSim Help

FAAMP: Halted Bus: Halted T5sim: USIM Instr / Sec 20.0 Mhz

Go Halt Reset Run Previous Up User 0 User 2  
IStep IStep 0v BackTrace History Next Down User 1 User 3

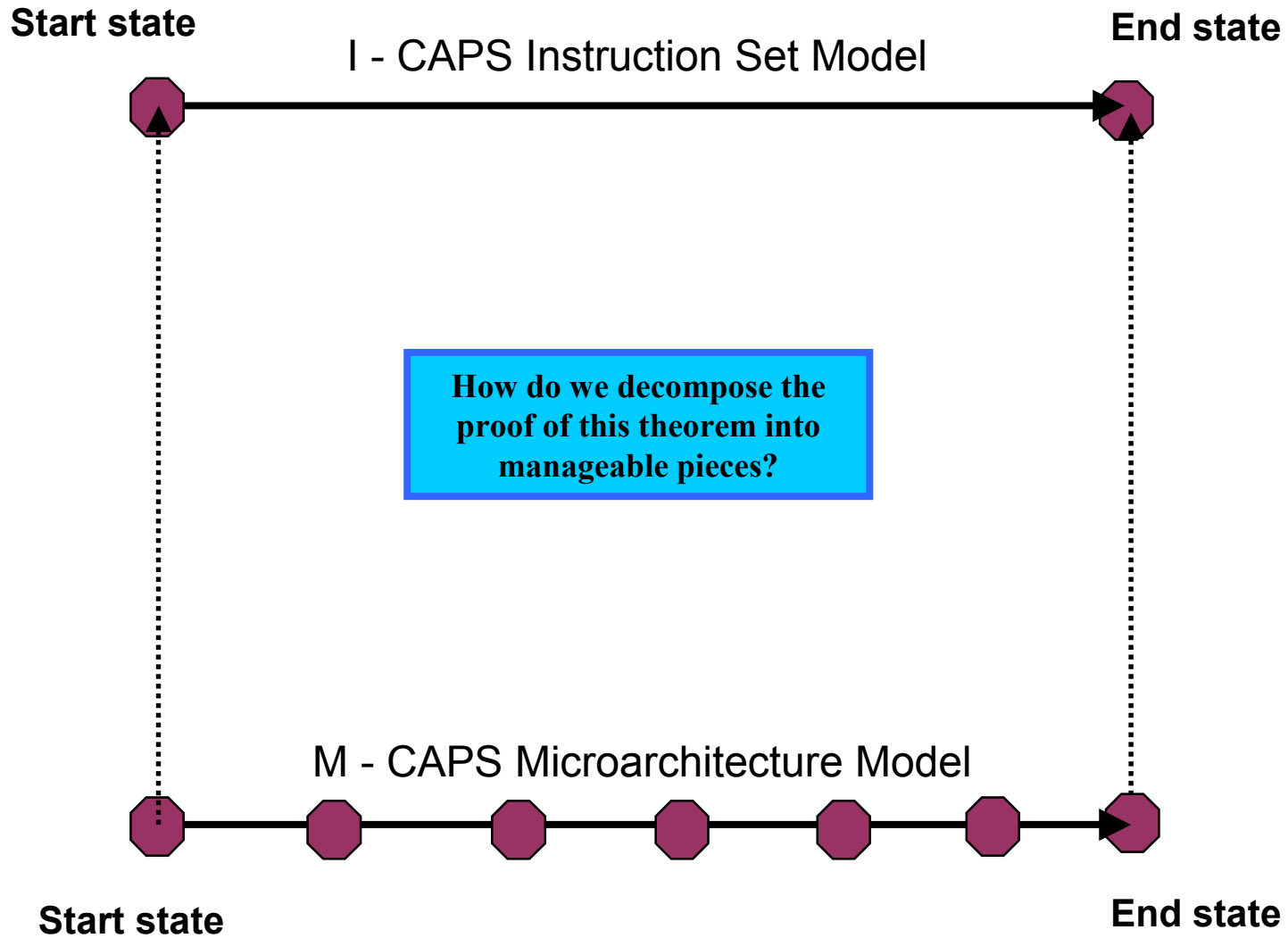
No Source File PC = 00024e.L

```
[diag]-> -- patch MACH_ROM
[diag]-> set 900 32
**** BREAKPOINT OCCURRED ****
ESCAPE key pressed
```

Time	Event	Duration
0	E,000249=5160	[00] 200 ns (4t)
0	E,007009=0000 R	[00] 300 ns (6t)
0	E,00024A=0A60	[00] 200 ns (4t)
0	E,00090D=01C9 R	[00] 300 ns (6t)
0	E,00024B=0915	[00] 200 ns (4t)
0	E,00024C=0900	[00] 200 ns (4t)
0	E,00024D=0000	[00] 200 ns (4t)
0	E,00024E=150A	[00] 200 ns (4t)
0	E,007061=BEEF W	[00] 500 ns (10t)

Command:

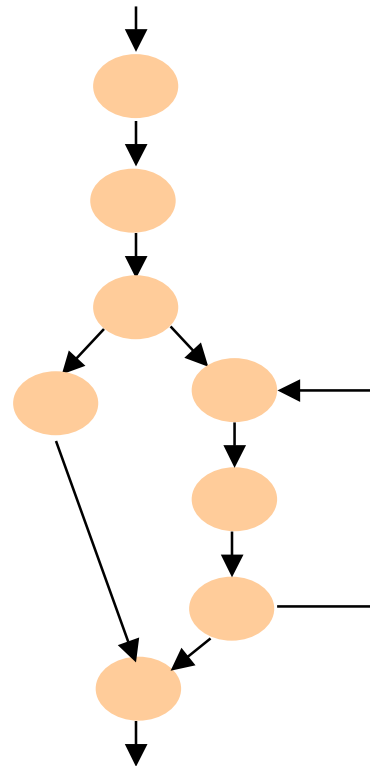






# Microcode sequences can be specified and verified in steps.

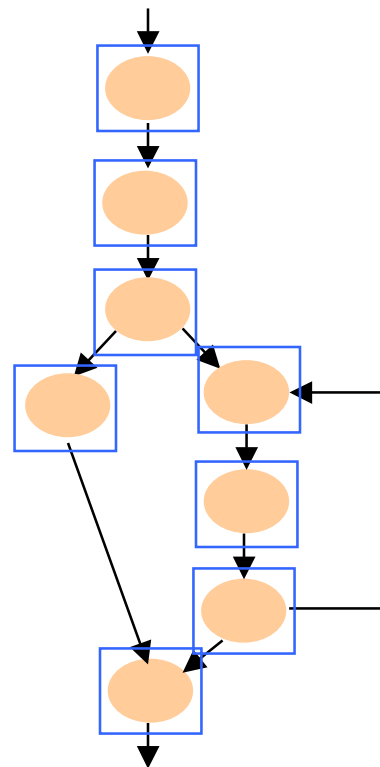
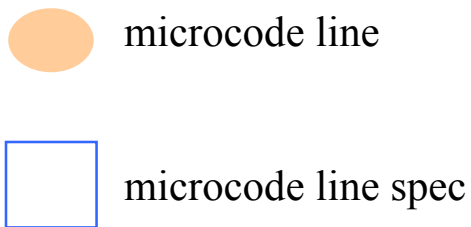
 microcode line



**Instruction microcode implementation**



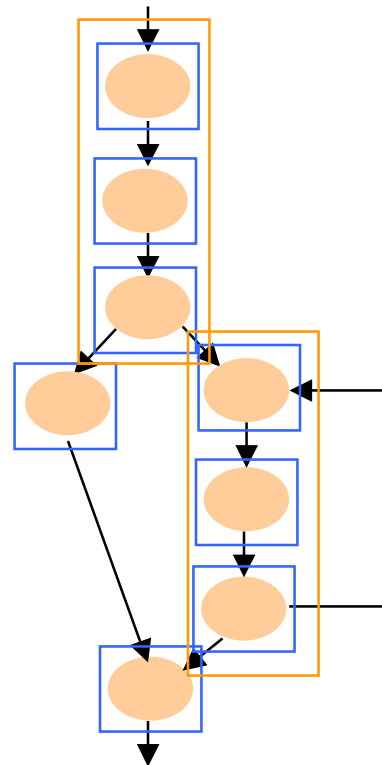
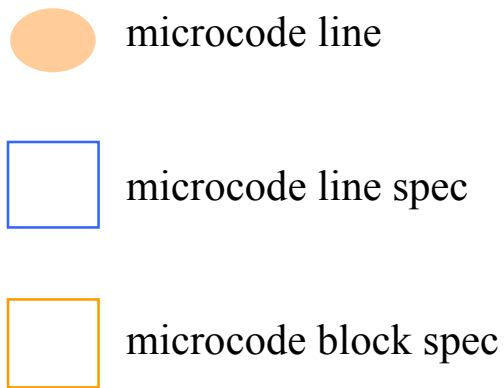
## Microcode sequences can be specified and verified in steps.



**Instruction microcode  
implementation**



## Microcode sequences can be specified and verified in steps.

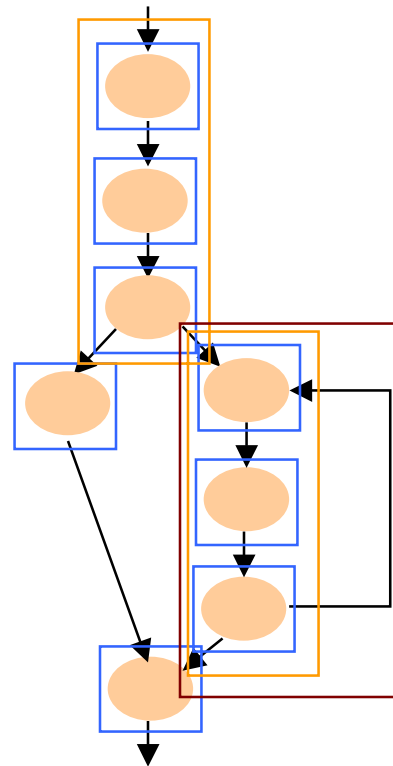
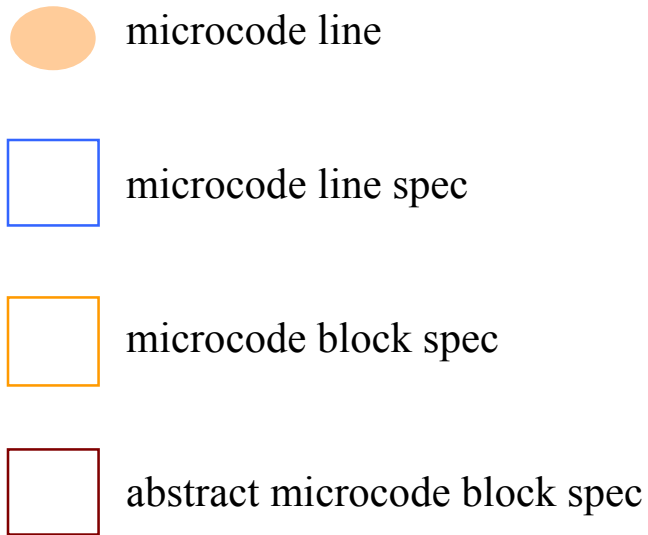


**Instruction microcode implementation**





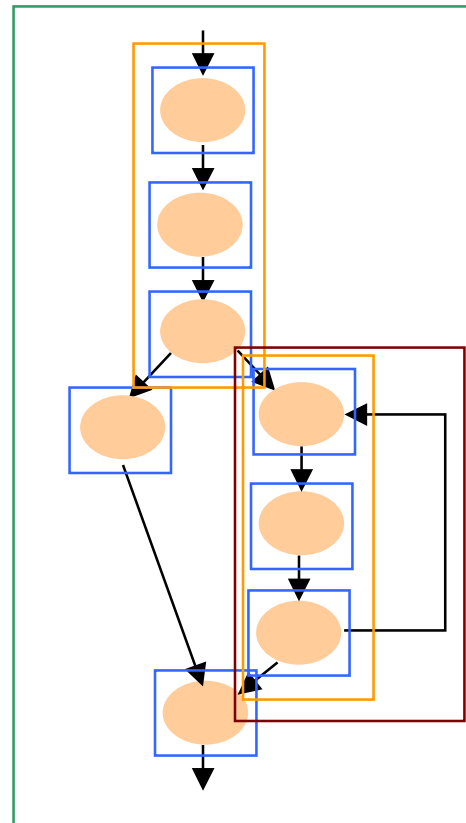
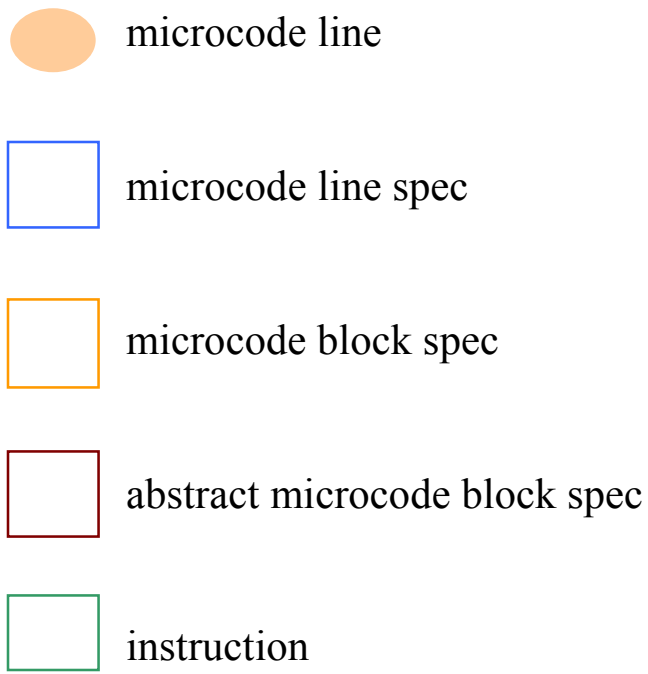
## Microcode sequences can be specified and verified in steps.



**Instruction microcode implementation**



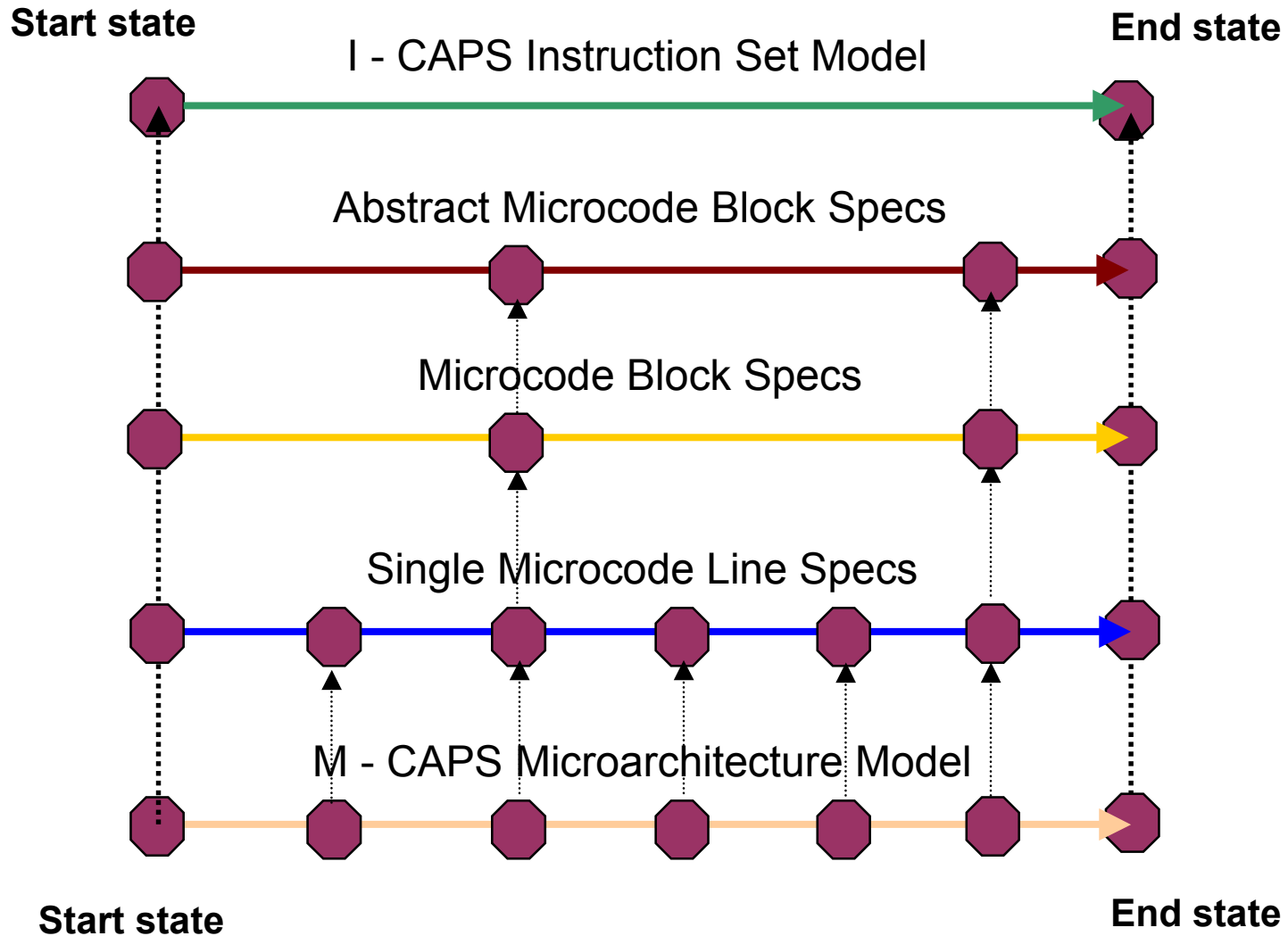
## Microcode sequences can be specified and verified in steps.



**Instruction microcode implementation**



# Proving the CAPS Correctness Theorem





# CAPS: High-Assurance Processor

---

- **Considerable effort expended on EFM reasoning**
  - ACL2 enhanced to deal efficiently with single threaded expressions
- **Techniques to manage complexity**
  - Proof libraries
    - Bit vectors
  - Using low level model to help define abstract model
    - Simplifies abstract specification and proof process
  - Proof-generating Macros
    - Similar to techniques constructed for JEM1 symbolic simulation



# AAMP7: Intrinsic Partitioning

**“High-Assurance Intrinsic Partitioning”, HCSS-03**

## ● Goals

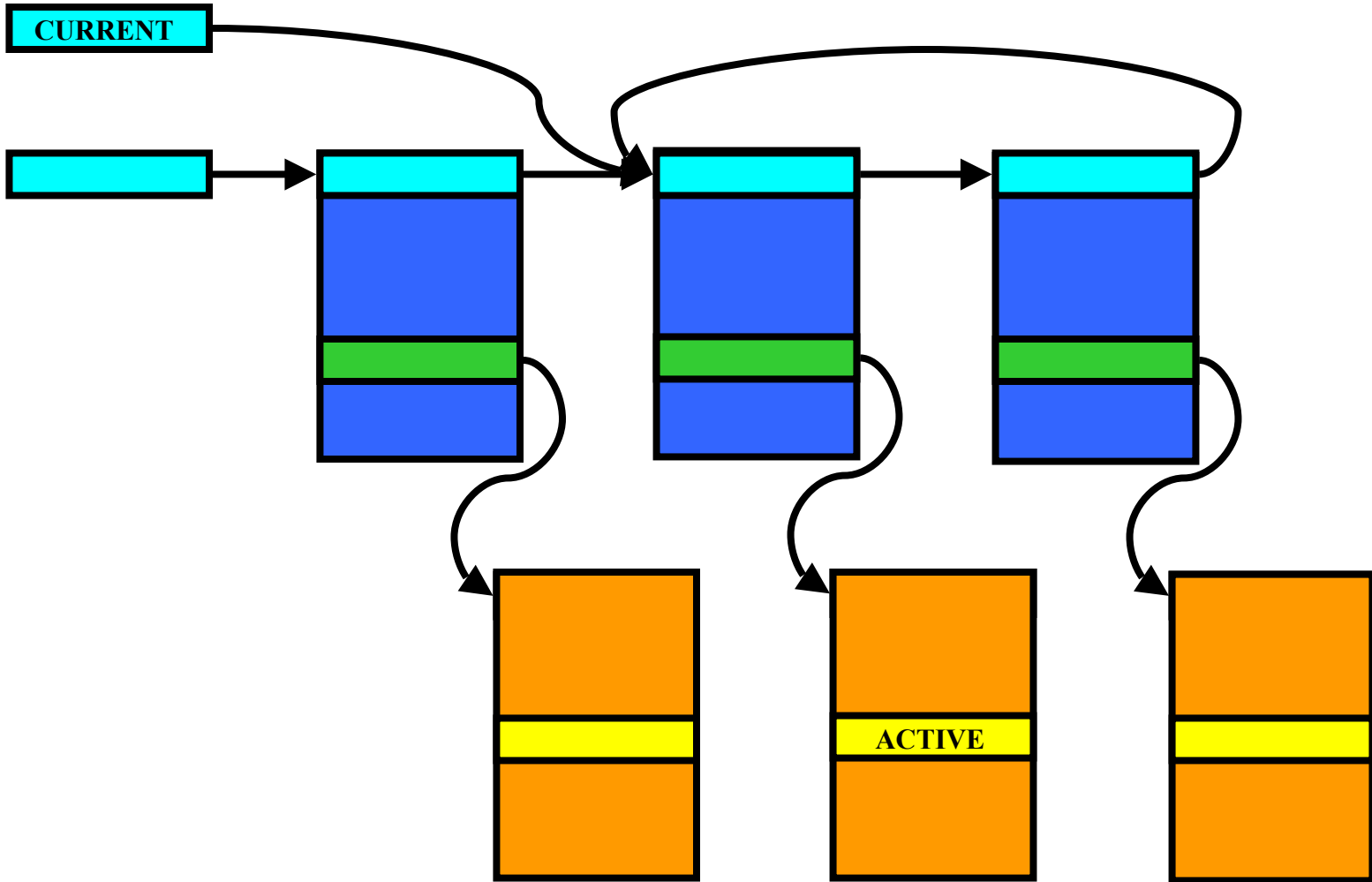
- Verify Security Properties of AAMP Intrinsic Partitioning Mechanism

## ● Project

- Formalize Security Property in ACL2
- Formalize Intrinsic Partitioning Functionality
  - “Instruction Level” Model
  - Linear Address Space
- Prove that Intrinsic Partitioning satisfies Security Property

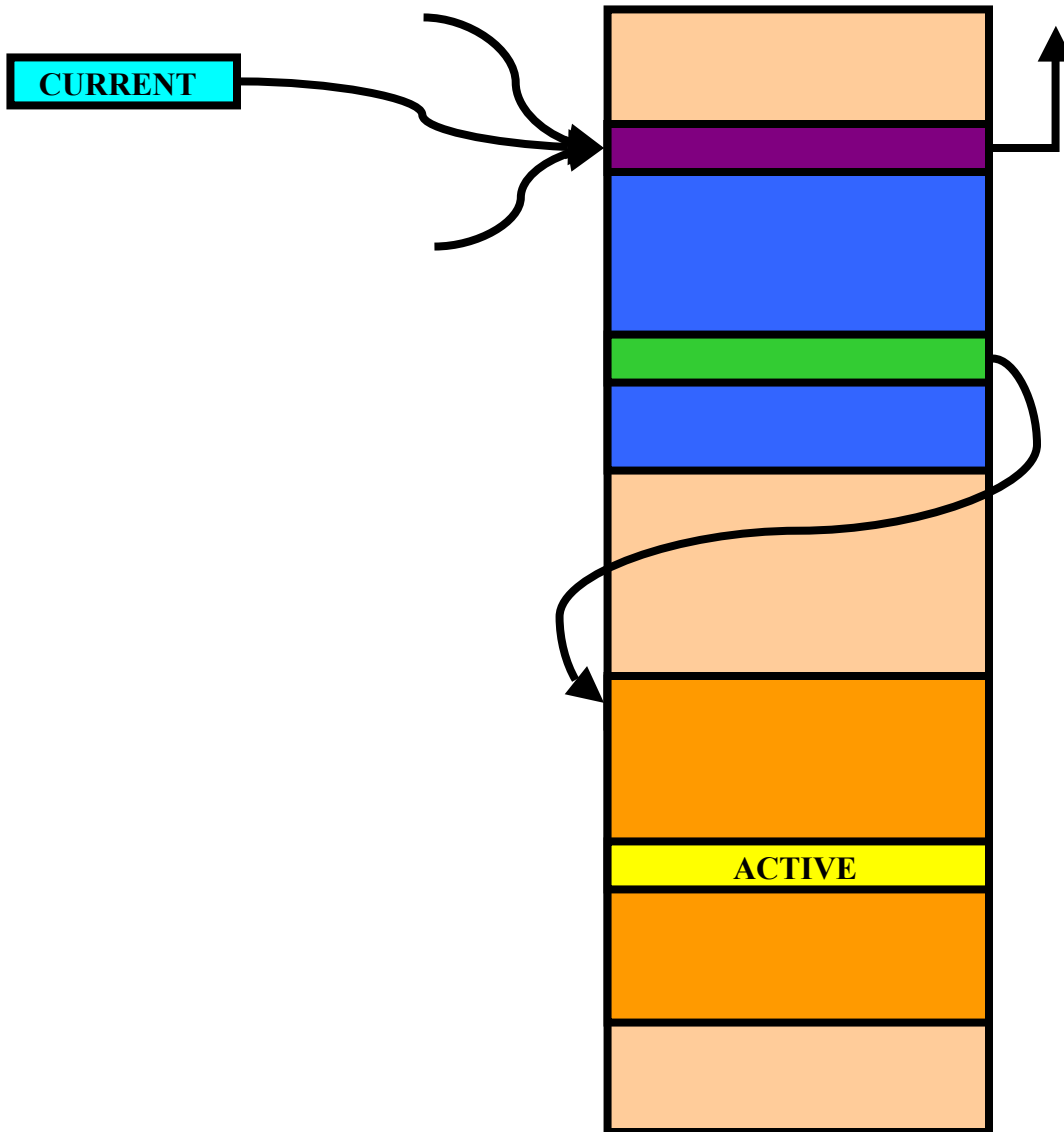


# Linear Address Space Reasoning





# Linear Address Space Reasoning





# AAMP7: Intrinsic Partitioning

---

- **Reasoning about Linear Address Spaces**
  - Identify orthogonal functionality
    - Techniques that scale
    - Make explicit for theorem prover
  - Could Leverage Data Flow (Definition/Use) Analysis
- **Proof Architecture**
  - Block structured decomposition
    - Similar to CAPS work
  - Over function boundaries
    - Not Microcode blocks





- **Motivating History**
- **Observations**
- **Future Directions**



- **Advantages to Accurate Low-Level Models**

- Model Validation
- Tie to Design Process via Simulation

- **Domain knowledge**

- Control Flow, Data Flow Analysis
- Can be codified in 3<sup>rd</sup> party tools
  - Results represented in language of theorem prover

- **Techniques generalize to**

- Different theorem provers (PVS,ACL2)
- Many different processor models
- Different levels of abstraction



- **Motivating History**
- **Observations**
- **Future Directions**



# Future Directions

---

- **Extend techniques to larger, more complex problems**
  - Additional microprocessors in the AAMP/JEM/CAPS family
  - Operating System Kernels
  - Mathematical and Cryptographic Libraries
  - Virtual Machines
  
- **Enable the use of low-level models**
  - Model validation
  - Avoid trusting or reasoning about compilers/abstractions
  - Assembly/micro code is not uncommon in these domains
    - Performance
    - Access to features unavailable in high-level languages
  
- **Encourage continued industrialization of theorem proving technology**
  - More powerful
  - More capable



**“A collection of tools and techniques to help simplify reasoning about complex systems”**

## ● Input

- Domain specific: object files, microcode
- Produces device and theorem prover independent representation

## ● Annotation

- Pre/Post conditions
- Proof Composition

## ● Analysis

- 3<sup>rd</sup> party tools automate capture of domain specific knowledge
- Stores results as annotations

## ● Output

- Generates input for theorem prover
- Isolates 3<sup>rd</sup> party tools from correctness argument



- **Rockwell Collins History**

- Have 10+ years experience in applying Formal Methods

- **Observations**

- Low Level Models are useful in a variety of domains
- Domain knowledge can be codified and used in theorem provers
- Effective techniques generalize over domains and theorem provers

- **Conclusion**

- Combining automated analysis and theorem proving
  - Improves productivity
  - Continues to provide high-assurance results
  - Extends the scope of what is currently feasible