Why formal verification remains on the fringes of commercial development

Arvind
Computer Science & Artificial Intelligence Laboratory
Massachusetts Institute of Technology

WG2.8, Park City, Utah
June 16, 2008

A designer's perspective

- The goal is to design systems that meet some criteria such as cost, performance, power, compatibility, robustness, …
- The design effort and the time-to-market matter ($$$)

Can formal methods help?

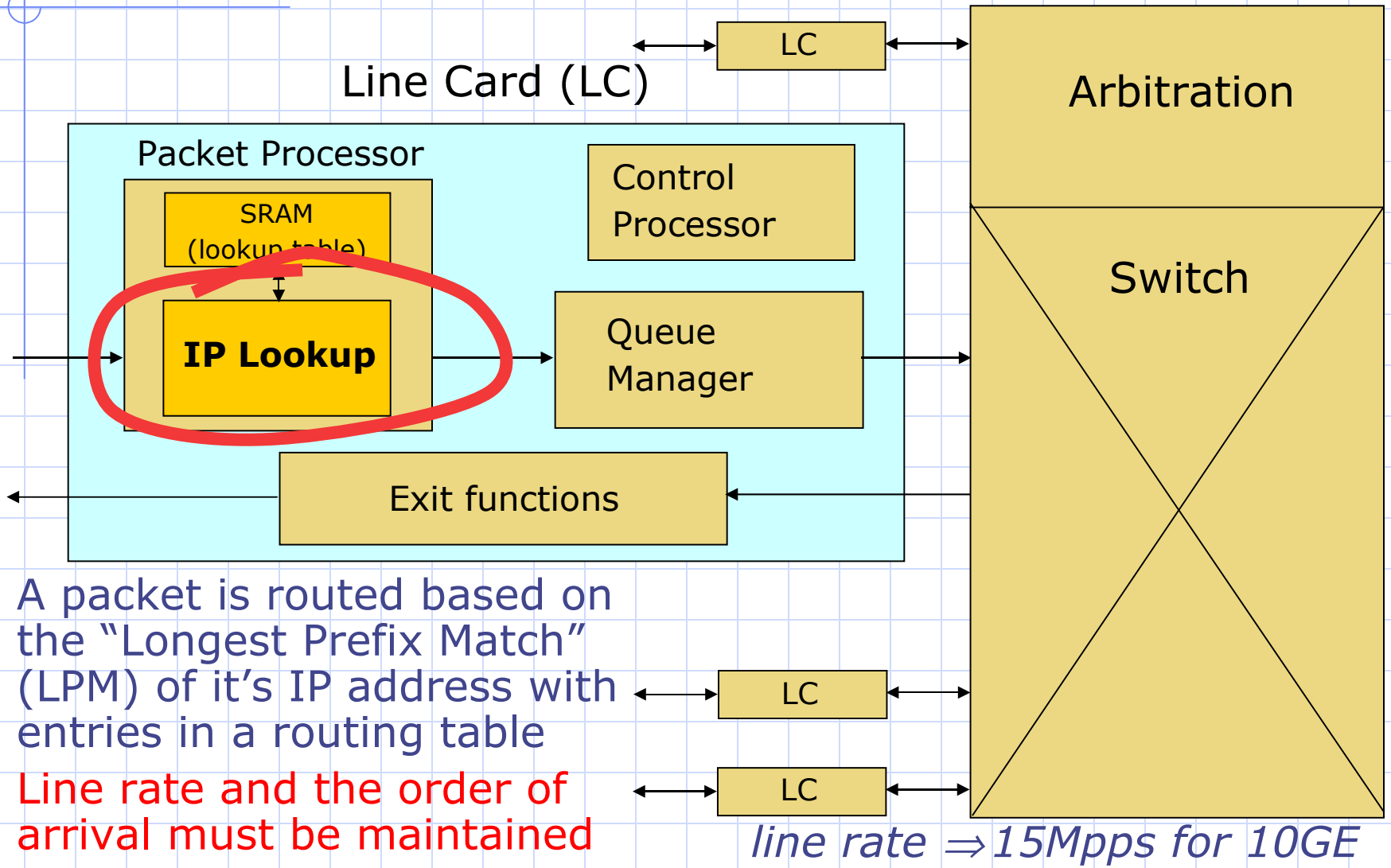
Examples

- IP Lookup in a router
- 802.11a Transmitter
- H.264 Video Codec
- OOO Processors
- Cache Coherence Protocols

Increasingly challenging

Example 1: Simple deterministic functionality

Internet router

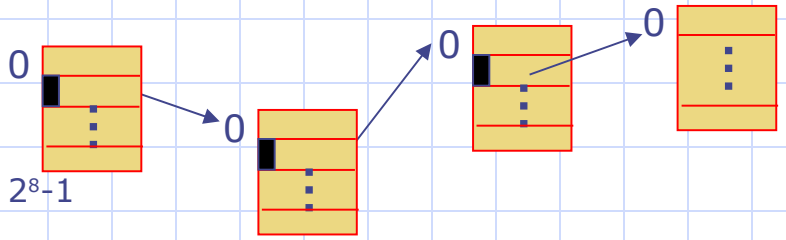


- A packet is routed based on the "Longest Prefix Match" (LPM) of it's IP address with entries in a routing table
- Line rate and the order of arrival must be maintained

line rate ⇒15Mpps for 10GE

"C" version of LPM

```
int
lpm (IPA ipa)
/*  3 memory lookups */
{  int p;
     /*  Level 0: 8 bits  */
   p = RAM [ipa[31:24]];
   if (isLeaf(p)) return value(p);
     /*  Level 1: 8 bits  */
   p = RAM [ipa[23:16]];
   if (isLeaf(p)) return value(p);
     /*  Level 2: 8 bits  */
   p = RAM [ptr(p) + ipa [15:8]];
   if (isLeaf(p)) return value(p);
     /*  Level 3:  8 bits  */
   p = RAM [ptr(p) + ipa [7:0]];
     return value(p);
    /* must be a leaf */
}
```



Not obvious from the C code how to deal with
- memory latency
- pipelining

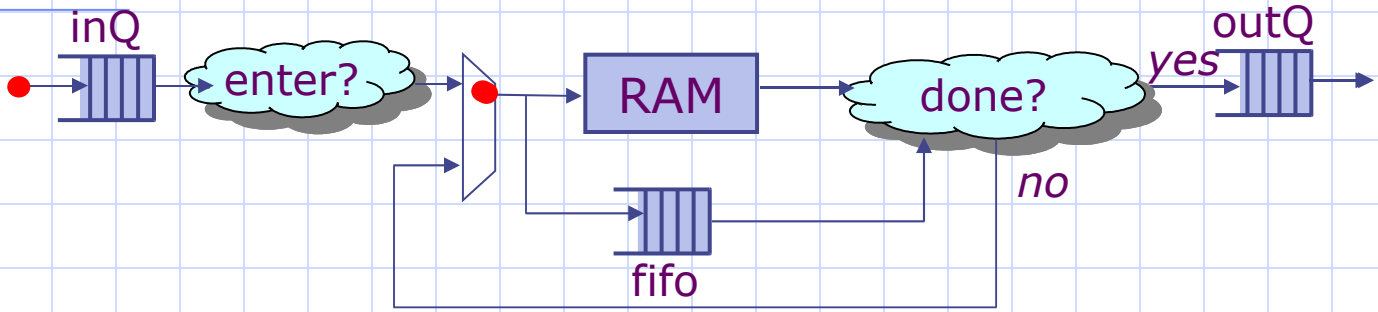
Must process a packet every 1/15 μs or 67 ns

Must sustain 4 memory dependent lookups in 67 ns

Memory latency ~30ns to 40ns

Real LPM algorithms are more complex

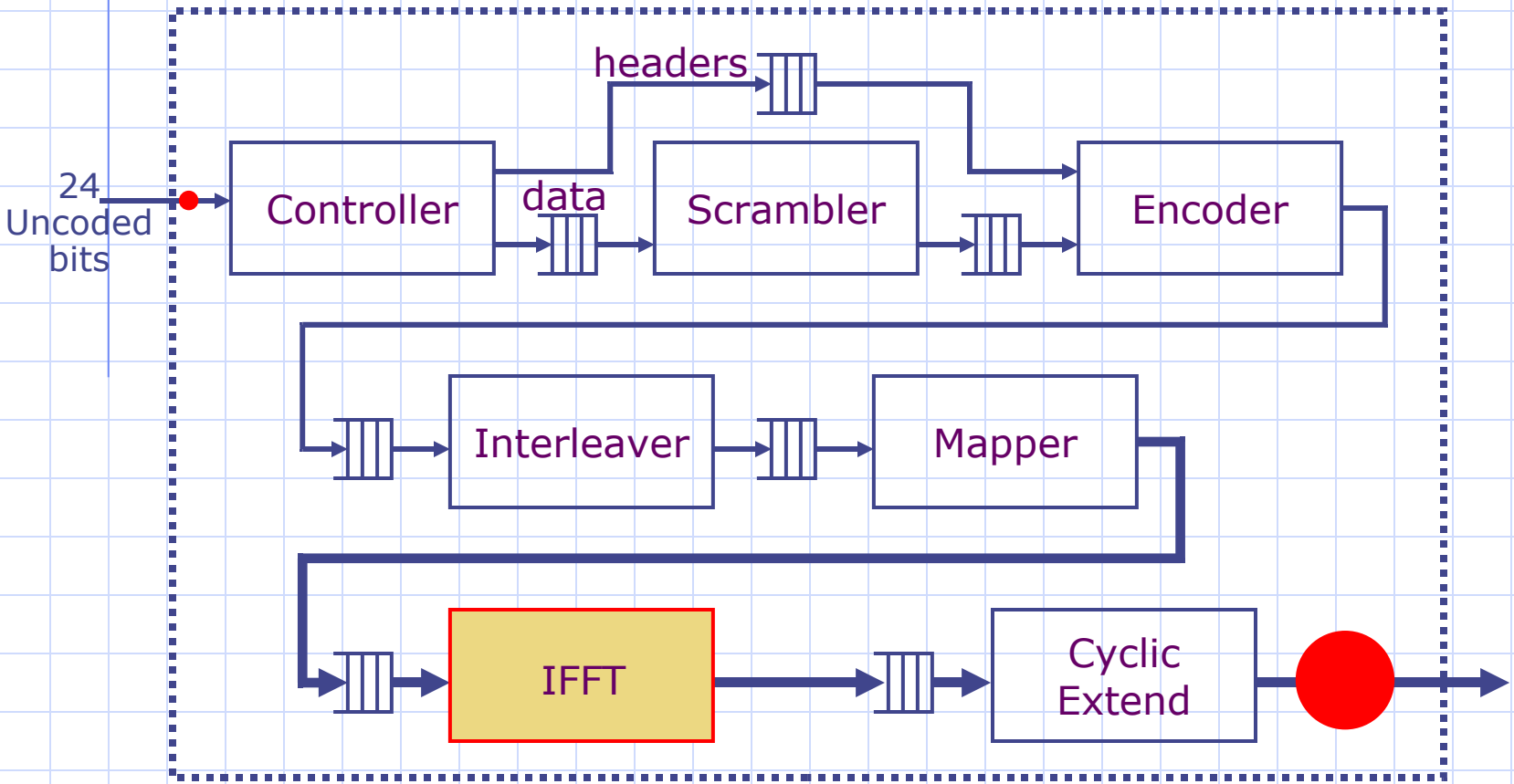
An implementation:
Circular pipeline



- Does the look up produce the right answer?
  - Easy: check it against the C program
- Performance concern: Are there any "dead cycles"?
  - Has direct impact on memory cost
- Do answers come out in the right order?
  - Is it even possible to express in a given logic?
  - Alternative: The designer tags input messages and checks that the tags are produced in order

Example 2: Dealing with Noise

802.11a Transmitter



accounts for 85% area

Must produce one OFDM symbol (64 Complex Numbers) every 4 μsec

Verification Issues

Control is straightforward

- Small amounts of testing against the C code is sufficient, provided the arithmetic is implemented correctly
  - C code may have to be instrumented to capture the intermediate values in the FIFOs
- No corner cases in the computation in various blocks
  - High-confidence with a few correct packets

Still may be worthwhile proving that the (non standard) arithmetic library is implemented correctly

802.11a transceiver:
Higher-level correctness

- Does the receiver actually recover the full class of corrupted packets as defined in the standard?
  - Designers totally ignore this issue
  - This incorrectness is likely to have no impact on sales

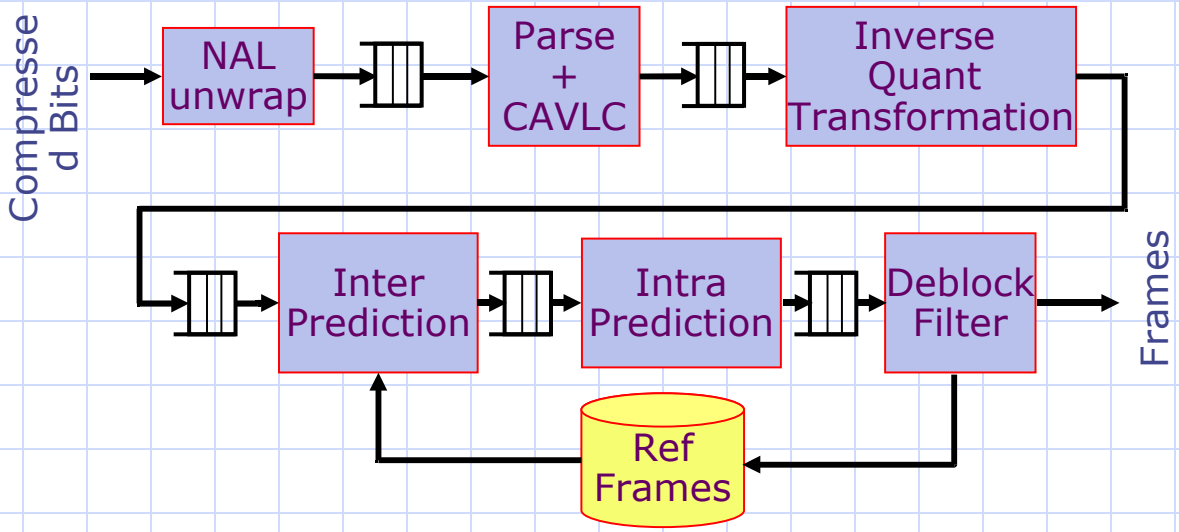
Who would know?

- If we really wanted to test for this, we could do it by generating the maximally-correctable corrupted traffic

All these are purely academic questions!

Example 3: Lossy encodings

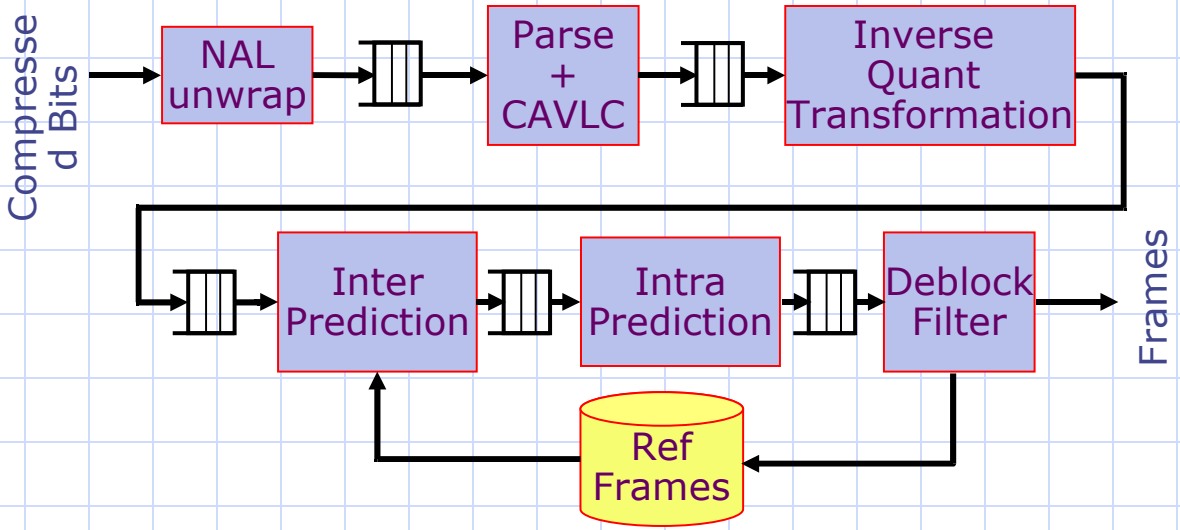
H.264 Video Decoder



Errors don't matter much

- The standard is 400+ pages of English; the standard implementation is 80K lines of convoluted C. Each is incomplete!
- Only viable correctness criterion is bit-level matching against the reference implementation on sample videos
- Parallelization is more complicated than what one may guess based on the dataflow diagram because of data-dependencies and feedback

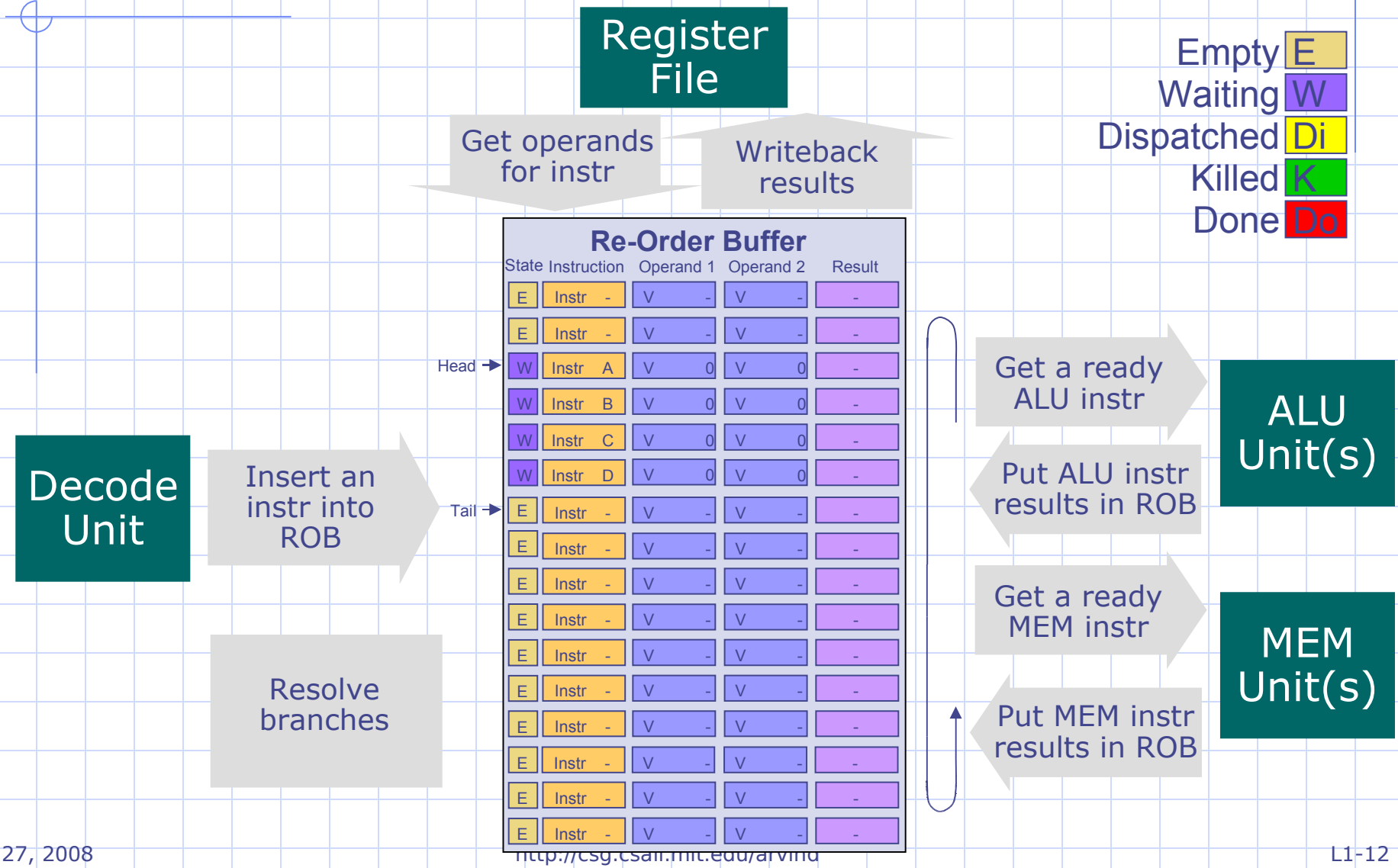
H.264 Decoder: Implementation



- Different requirements for different environments
  - QVGA 320x240p (30 fps)
  - DVD 720x480p
  - HD DVD 1280x720p (60-75 fps)
- Each context requires a different amount of parallelism in different blocks
  - Modular refinement is necessary
  - Verifying the correctness of refinements requires traditional formal techniques (pipeline abstraction, etc.)

Example 4: Absolute Correctness is required

Microprocessor design



"Automated" Processor Verification

- Models are abstracted from (real) designs
  - UCLID – Bryant (CMU) : OOO Processor hand translated into CLU logic (synthetic)
  - Cadence SMV - McMillian : Tomasulo Algorithm (hand written model. synthetic)
  - ACL – Jay Moore: (Translate into Lisp)
  - …
- Some property of the manually abstracted model is verified
  - Great emphasis (and progress) on automated decision procedures

Since abstraction is not automated it is not clear what is being verified!

BAT[Manolios et al] is a move in the right direction

Automatic extraction of abstract models from designs expressed in Verilog or C or SystemC is a lost cause

Example 5: nondeterministic specifications

Cache Coherence

Proofs of Correctness of Cache-Coherence Protocols

Joseph Stoy[1], Xiaowei Shen[2], and Arvind[2]

[1] Oxford University Computing Laboratory
Oxford OX1 3QD, England
Joe.Stoy@comlab.ox.ac.uk
[2] Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge MA 02139, USA
xwshen, arvind@lcs.mit.edu

Abstract. We describe two proofs of correctness for Cachet, an adaptive cache-coherence protocol. Each proof demonstrates soundness (conformance to an abstract cache memory model CRF) and liveness. One proof is manual, based on a term-rewriting system definition; the other is machine-assisted, based on a TLA formulation and using PVS. A two-stage presentation of the protocol simplifies the treatment of soundness, in the design and in the proofs, by separating all liveness concerns. The TLA formulation demands precision about what aspects of the system's behavior are observable, bringing complication to some parts which were trivial in the manual proof. Handing a completed design over for independent verification is unlikely to be successful: the prover requires detailed insight into the design, and the designer must keep correctness concerns at the forefront of the design process.

1 Introduction: Memory Models and Protocols

Shared memory multiprocessor systems provide a global memory image so that processors running parallel programs can exchange information and synchronize with one another by accessing shared variables. In large-scale systems the physical memory is usually distributed across different sites to achieve better performance. Distributed Shared Memory (DSM) systems implement the shared memory abstraction with a large number of processors connected by a network, combining the scalability of network-based architectures with the convenience of shared memory programming. The technique known as caching allows shared variables to be replicated in multiple sites simultaneously to reduce memory access latency. DSM systems rely on cache-coherence protocols to ensure that each processor can observe the semantic effect of memory access operations performed by another processor.

A shared memory system implements a *memory model*, which defines the semantics of memory access instructions. An ideal memory model should allow efficient and scalable implementations while still having simple semantics

- It took Joe Stoy more than 6 months to learn PVS and show that some of the proofs in Xiaowei Shen's thesis were correct

  This technology is not ready for design engineers

Model Checking

- CC is one of the most popular applications of model checking
- The abstract protocol needs to be abstracted more to avoid state explosion
  - For example, only 3 CPUs, 2 addresses
- There is a separate burden of proof why the abstraction is correct
- Nevertheless model checking is a very useful debugging aid for the verification of abstract CC protocols

Implementation

- Design is expressed in some notation which is NOT used directly to generate an implementation
  - The problem of verification of the actual protocol remains formidable
  - Testing cannot uncover all bugs because of the huge non-deterministic space

Proving the correctness of cache coherence protocol implementations remains a challenging problem

Summary

- The degree of correctness required depends upon the application
  - D… …rmal a…
- For … …evel des…
- For … debugging aids during the design process
  - A designer is unlikely to do any thing for the sake of helping the post design verification

The real success of a formal technique is when it is used ubiquitously without the designer being aware of it

e.g., type systems