Precision-Timed (PRET) Machines

Stephen A. Edwards
Columbia University

Edward A. Lee
University of California, Berkeley
In 1980, Patterson and Ditzel did not invent reduced instruction set computers (RISC machines).

Another Major Historical Event

In 2006, Lee and Edwards did not invent reduced precision-timed computers (PRET machines).


http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-149.html
We do not consider how fast a processor runs when we evaluate whether it is “correct.”

This Is Sometimes Useful For

- Programming languages
- Virtual memory
- Caches
- Dynamic dispatch
- Speculative execution
- Power management (voltage scaling)
- Memory management (garbage collection)
- Just-in-time (JIT) compilation
- Multitasking (threads and processes)
- Component technologies (OO design)
- Networking (TCP)
But Time Sometimes Matters

Kevin Harvick winning the Daytona 500 by 20 ms, February 2007. (Source: Reuters)
Certification in Avionics

- Rather expensive
- Software is *not* certified
- Entire system is certified
- Slight change, e.g., in the microprocessor, requires recertification
- Solution: stockpile parts; trust nobody

(Source: NASA)
Worst-Case Execution Time

Virtually impossible to compute on modern processors.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Nearby instructions</th>
<th>Distant instructions</th>
<th>Memory layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelines</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Caches</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
State-of-the-art WCET

- Motorola ColdFire
- Two coupled pipelines (7-stage)
- Shared instruction & data cache
- Artificial example from Airbus
- Twelve independent tasks
- Simple control structures
- Cache/Pipeline interaction leads to large integer linear programming problem

C. Ferdinand et al., “Reliable and precise WCET determination for a real-life processor,” EMSOFT 2001
The Problem

Digital hardware provides extremely precise timing

20.000 MHz (± 100 ppm)

and architectural complexity discards it.
Our Vision: PRET Machines

PREcision-Timed processors: Performance & Predicability

(Image: John Harrison’s H4, first clock to solve longitude problem)
## Our Vision: PRET Machines

Predictable performance, not just good average case

<table>
<thead>
<tr>
<th>Current</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caches</td>
<td>Scratchpads</td>
</tr>
<tr>
<td>Pipelines</td>
<td>Thread-interleaved pipelines</td>
</tr>
<tr>
<td>Function-only ISAs</td>
<td>ISAs with timing</td>
</tr>
<tr>
<td>Function-only languages</td>
<td>Languages with timing</td>
</tr>
<tr>
<td>Data Races</td>
<td>Deterministic concurrency</td>
</tr>
<tr>
<td>Best-effort communication</td>
<td>Fixed-latency communication</td>
</tr>
</tbody>
</table>
Related Work: Giotto


The RTOS style: specify a collection of tasks and modes. Compiler produces schedule (task priorities).

Precision limited by periodic timer interrupt.

mode forward() period 200 {
    actfreq 1 do leftJet(leftMotor);
    actfreq 1 do rightJet(rightMotor);
    exitfreq 1 do point(goPoint);
    exitfreq 1 do idle(goIdle);
    exitfreq 1 do rotate(goRotate);
    taskfreq 2 do errorTask(getPos);
    taskfreq 1 do forwardTask(getErr);
}
Software Thread Integration [Dean, RTSS 1998]

Insert code for a non-real-time thread into a real-time thread.

Pad the rest with NOPs

Often creates code explosion

Requires predictable processor; he uses AVR*
Related Work: VISA

VISA [Meuller et al., ISCA 2003]

Run two processors:

- Slow and predictable
- Fast and unpredictable

Start tasks on both.

If fast completes first, use extra time.

If fast misses a checkpoint, switch over to slow.
A First Attempt

16-bit MIPS-like processor augmented with timers

One additional “deadline” instruction:

\[ \text{dead timer}, \timeout \]

Wait until \( \text{timer} \) expires, then reload it with \( \timeout \).

Case Study: Video

80 × 30 text-mode display, 25 MHz pixel clock

Shift register in hardware; everything else in software

![Diagram of video signal flow](image)
Case Study: Video

Two nested loops:

- Active line
- Character

Two timers:

- $t1$ for line timing
- $t0$ for character output

78 lines of assembly replaces 450 lines of VHDL (1/5th)
Conclusions

- Embedded applications need timing control
- We need hardware support
- High-performance processors with predictable timing
- Predictable performance our mantra
- A first cut: MIPS-like processor with timers
- *Dead* instruction waits for timeout, then reloads
- Video controller 1/5 the size of VHDL