Precision-Timed (PRET) Machines

Stephen A. Edwards
Columbia University

Joint work with 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)
Isn’t Real-Time Scheduling Solved?

Fixed-priority (RMA): schedulable if < 69% utilization
Variable-priority (EDF): schedulable if < 100% utilization

Hinges on knowing task execution times
Interrupt Latency and Response

Interrupts may be disabled here

Scheduler runs

Latency

Response time

Need longest interrupt-disabled time + scheduling time

Jitter from Delaying for One Tick

Tick

ISR

Higher-priority tasks

“Delay 10 ms”

ISR time + other task time + our delay

10 ms

6.3 ms 5.5 ms 19.8 ms

Starved
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
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)
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>Best-effort communication</td>
<td>Fixed-latency communication</td>
</tr>
<tr>
<td>Time-sharing</td>
<td>Multiple independent processors</td>
</tr>
</tbody>
</table>
Application Areas

Hard real-time systems

- Avionics
- Automotive
- Multimedia
- Consumer Electronics
- Simple digital hardware
Basic Idea

Q: How do you make software run at a precise speed?

A: Give it access to a clock.
Basic Idea

Q: How do you make software run at a precise speed?

A: Give it access to a clock.
One Usual Way: Timers

Period timer interrupt triggers scheduler

Large period reduces overhead

Linux uses a 10 ms clock

Result: OS provides 10 ms resolution at best

Higher precision requires more overhead

0 ms 10 ms 20 ms 30 ms 40 ms 50 ms 60 ms
Or NOPs/cycle counting

Code from Linux arch/i386/kernel/timers/timer_none.c

delay_none:
0: push %ebp
1: mov %esp,%ebp
3: sub $0x4,%esp
6: mov 0x8(%ebp),%eax
9: jmp 10
10: jmp 20
20: dec %eax
21: jns 20
23: mov %eax,-4(%ebp)
26: leave
27: ret

Tricky
Clock speed + cache behavior + branch behavior + ?
This example worries about cache alignment
Very much an assembly-language trick
1000s of lines of code in Linux needed for busy wait
Related Work: Giotto


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

Precision limited by periodic timer interrupt.

```plaintext
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);
}
```
Related Work: STI

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 PRET processors; 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

MIPS-like processor with 16-bit data path as proof of concept

One additional “deadline” instruction:

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

Wait until \textit{timer} expires, then immediately reload it with \textit{timeout}.

Programmer’s Model

General-purpose Registers

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 (= 0)</td>
<td></td>
</tr>
<tr>
<td>$1</td>
<td></td>
</tr>
<tr>
<td>$2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$13</td>
<td></td>
</tr>
<tr>
<td>$14</td>
<td></td>
</tr>
<tr>
<td>$15</td>
<td></td>
</tr>
</tbody>
</table>

Timers

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t0</td>
<td></td>
</tr>
<tr>
<td>$t1</td>
<td></td>
</tr>
<tr>
<td>$t2</td>
<td></td>
</tr>
<tr>
<td>$t3</td>
<td></td>
</tr>
</tbody>
</table>

Program counter

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>add</td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>and</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>andi</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>be</td>
<td>Rd, Rs, offset</td>
</tr>
<tr>
<td>bne</td>
<td>Rd, Rs, offset</td>
</tr>
<tr>
<td>j</td>
<td>target</td>
</tr>
<tr>
<td>lb</td>
<td>Rd, (Rt + Rs)</td>
</tr>
<tr>
<td>lbi</td>
<td>Rd, (Rs + offset)</td>
</tr>
<tr>
<td>mov</td>
<td>Rd, Rs</td>
</tr>
<tr>
<td>movi</td>
<td>Rd, imm16</td>
</tr>
<tr>
<td>nand</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>nandi</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>nor</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>nori</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>xor</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>xori</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>xnor</td>
<td>Rd, Rs, Rt</td>
</tr>
<tr>
<td>xnori</td>
<td>Rd, Rs, imm16</td>
</tr>
<tr>
<td>deadi</td>
<td>T, imm16</td>
</tr>
<tr>
<td>dead</td>
<td>T, Rs</td>
</tr>
</tbody>
</table>
### Behavior of Dead

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>$t_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>deadi $t_0$, 8</td>
<td>3</td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>add $r_1$, $r_2$, $r_3$</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>deadi $t_0$, 10</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>add $r_1$, $r_2$, $r_3$</td>
<td>9</td>
</tr>
</tbody>
</table>

8 cycles
Idioms: Straightline Code

deadi $t0, 42

: First block will take at least 42 cycles.

deadi $t0, 58

: Second block: at least 58 cycles.

deadi $t0, 100
Idioms: Loops

L1:

: 

deadi $t0, 42

: 

bne $r1, $r2, L1

Put a deadline in a loop:

Each iteration will take at least 42 cycles.
Case Study: Video

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

Need 40 ns precision

Shift register in hardware; everything else in software
Case Study: Video

Two nested loops:

- Active line
- Character

Two timers:

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

78 lines of assembly replaces 450 lines of VHDL (1/5th)
Case Study: Serial Receiver

Sampling rate under software control

Standard algorithm:

1. Find falling edge of start bit
2. Wait half a bit time
3. Sample
4. Wait full bit time
5. Repeat 3. and 4.
Implementation

Synthesized on an Altera Cyclone II FPGA (DE2 board)

Coded in VHDL

Runs at 50 MHz

Unpipelined

Uses on-chip memory
Conclusions

- Embedded applications need timing control
- RTOSes on modern processors too unpredictable
- We need hardware support
- High-performance processors with predictable timing
- Predictable performance our mantra
- A first cut: MIPS-like processor with timers
- 50 MHz on an Altera Cyclone II FPGA
- *Dead* instruction waits for timeout, then reloads
- Video controller 1/5 the size of VHDL
- Serial controller even simpler