What Do We Do With 10^{12} Transistors? The Case for Precision Timing

Stephen A. Edwards

Columbia University

What Not To Do

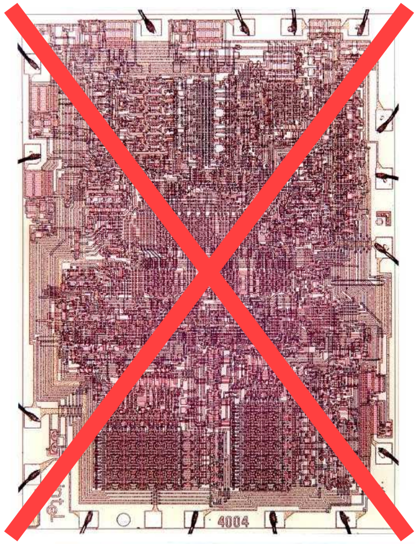
- Not just a single CPU

  Processor architects have already given up trying to figure out how to waste that many transistors

- Not just one big memory

  Von Neumann Bottleneck

  10^{12} bits vs. a 1 GHz clock: minutes



What Not To Do

- Not "Internet-on-a-chip" (TCP/IP over Ethernet)

  On-chip communication more reliable

  No on-chip backhoes to worry about

  We are not good at programming these anyway

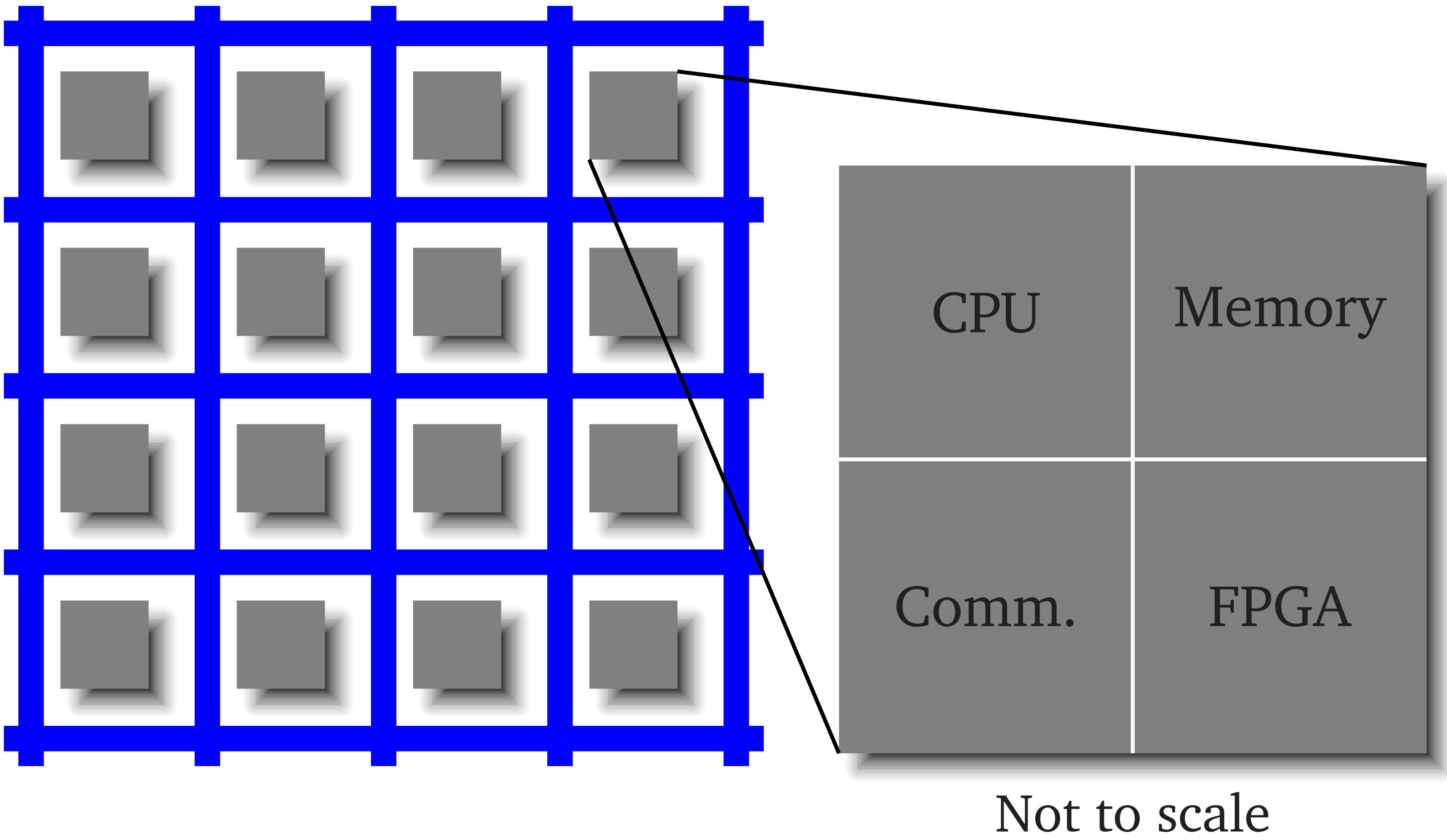
- Not just an FPGA

  Non-software systems disappeared in the early 1980s

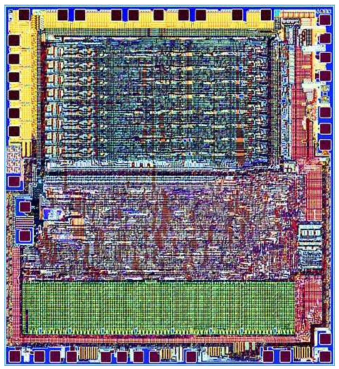
  Every interesting system has lots of software

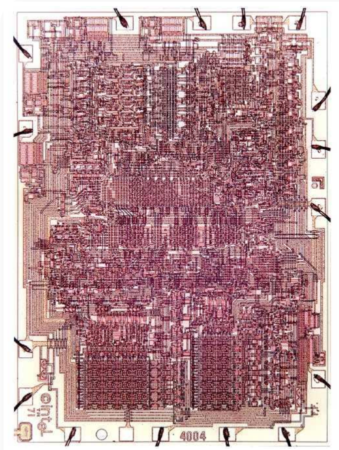
What We Probably Will Do

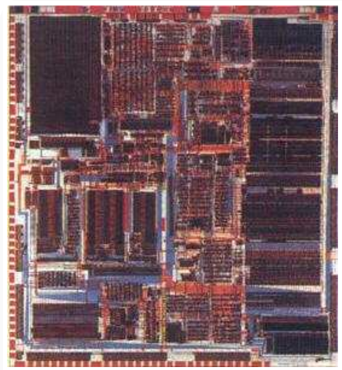
An FPGA-like mesh of computational elements floating in a sea of communication.

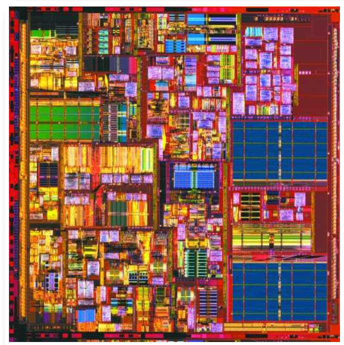


What Sort Of Processor?





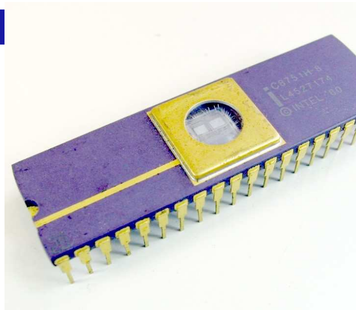




Hypothesis: it should be a precision-timed “PRET” processor

Embedded Systems Dominate

- In 2004, 97% of the 6.5 billion processors shipped went into embedded system.



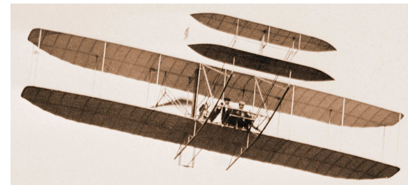
- In 2004, 674 million cell phones sold,
  3.3 billion total subscribers
  2004 world population: 6.4 billion

- 100 processors in a typical automobile

Embedded Application Areas

Hard real-time systems

- Avionics
- Automotive
- Multimedia
- Consumer Electronics



The World as We Know It

We do not consider how fast a processor runs when we evaluate whether it is "correct."

Salvador Dali, *The Persistence of Memory*, 1931. (detail)

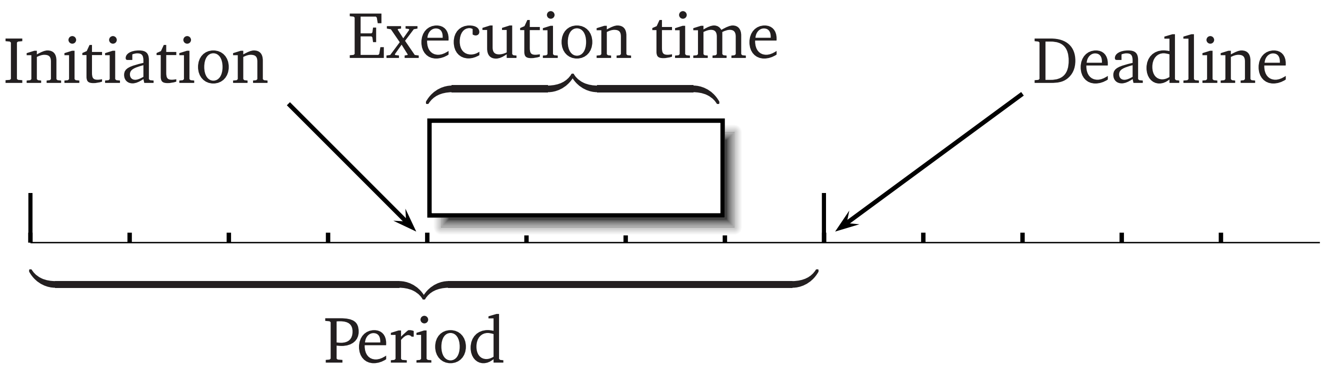
This Is Sometimes Useful For

- 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

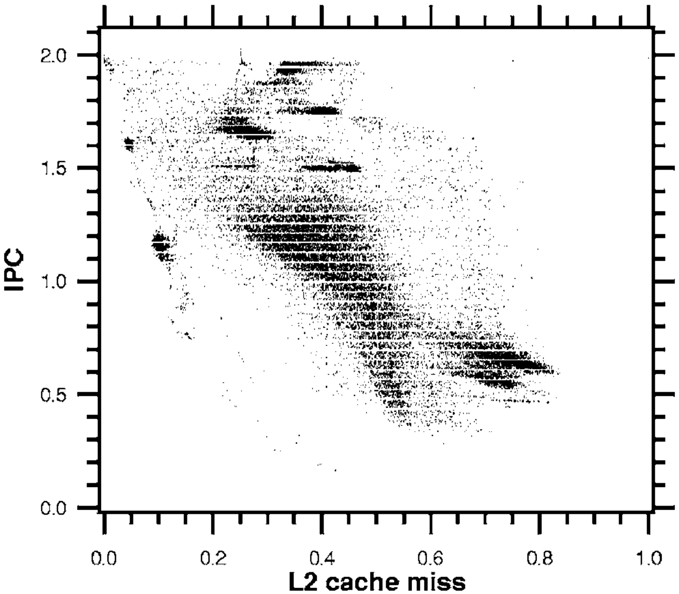
Hinges on knowing task execution times

Worst-Case Execution Time

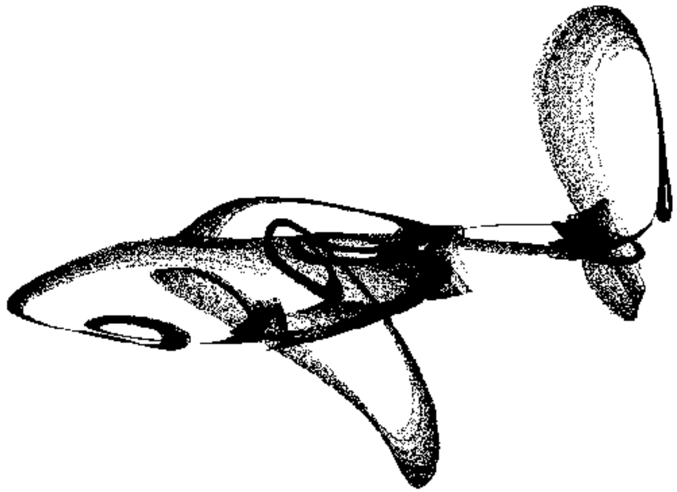
Virtually impossible to compute on modern processors.

Feature	Nearby instructions	Distant instructions	Memory layout
Pipelines	√		
Branch Prediction	√	√	
Caches	√	√	√

Processors are Actually Chaotic



Berry et al., *Chaos in computer performance*, Chaos 16:013110, 2006.

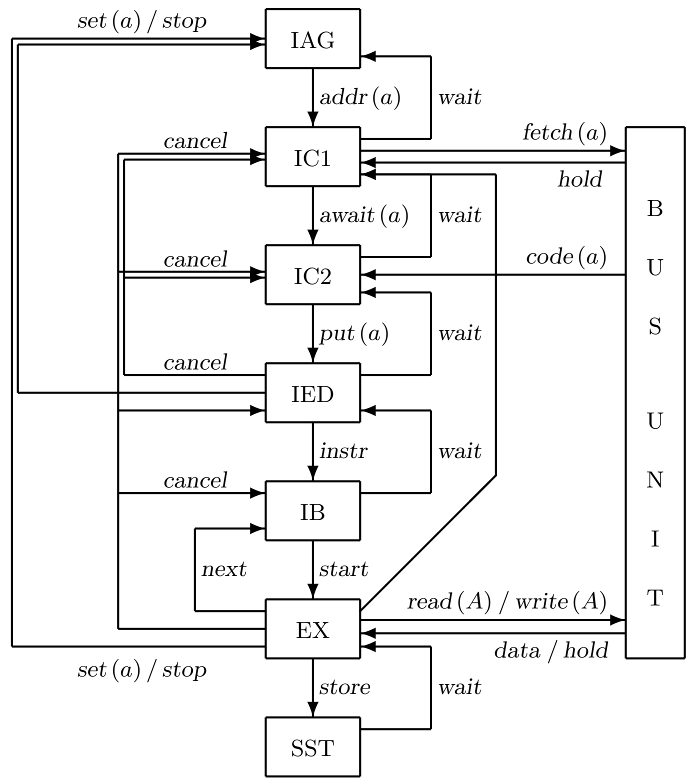


Sprott, *Strange Attractors*, Figure 5–13.

Herring

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

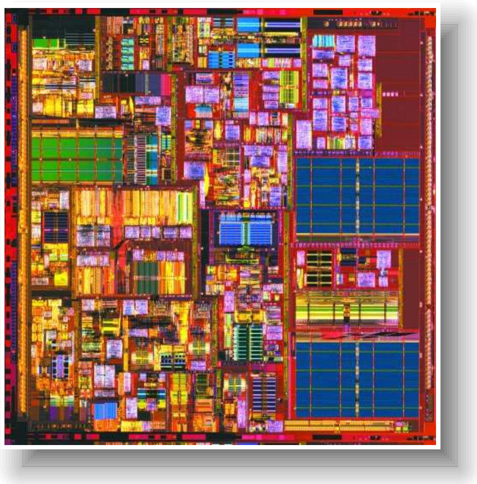
Digital hardware provides extremely precise timing

20.000 MHz ($\pm$ 100 ppm)

and modern architectural complexity discards it.

Our Vision: PRET Machines

PREcision-Timed processors: Performance & Predicability



+

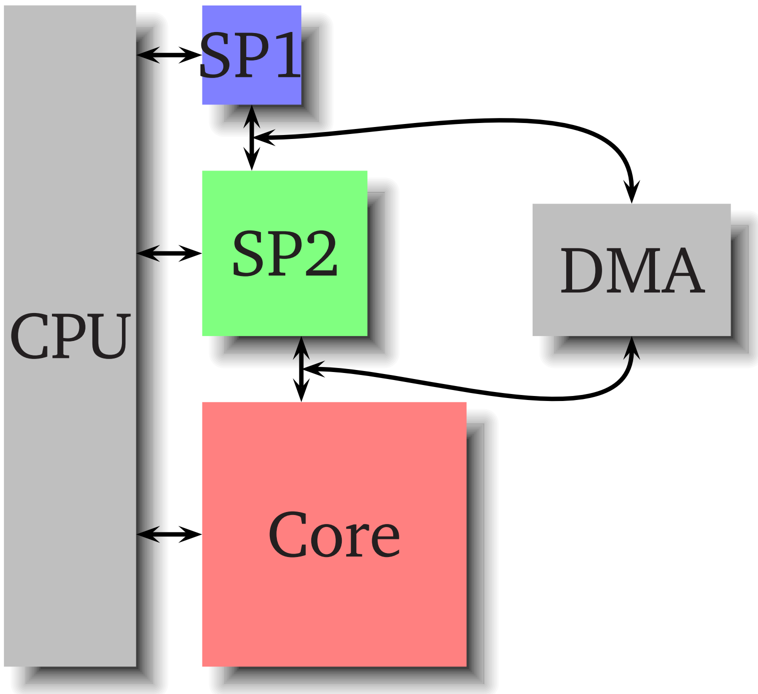
= PRET

(Image: John Harrison's H4, first clock to solve longitude problem)

Caches and Memory Hierarchy?

Our goal: a predictable memory hierarchy

Use software-managed scratchpads with compiler support



Well-studied:

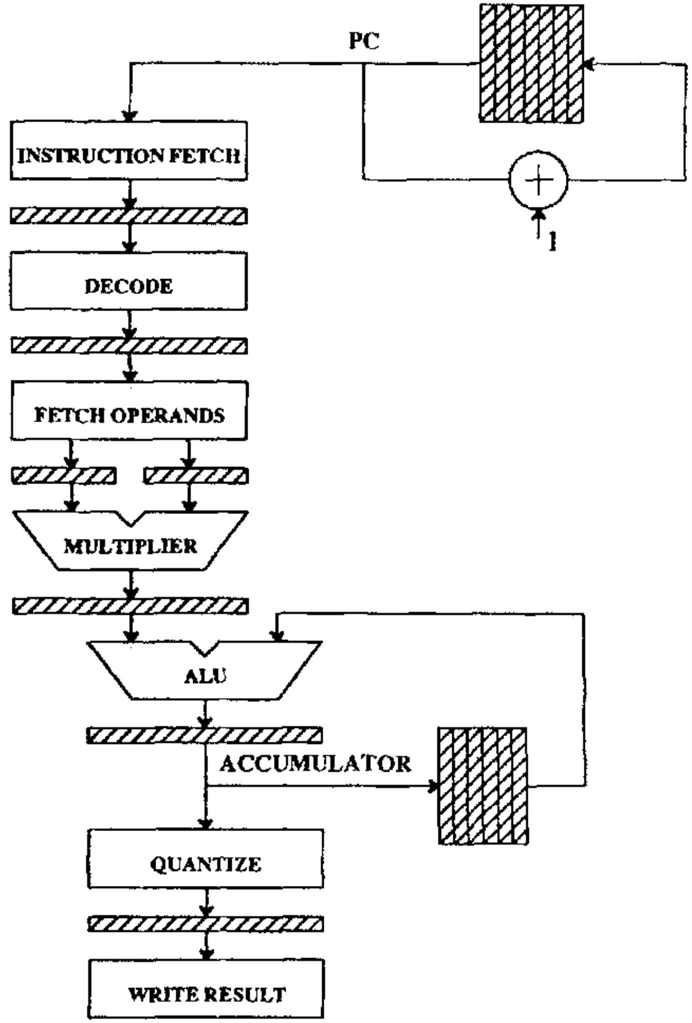
Panda et al. [EDAC 1997],
Kandemir et al. [DAC 2001, 2002],
Banakar et al. [CODES 2002],
Angiolini et al. [CASES 2003, 2004],
Udaykumaran et al. [CASES 2003],
Verma et al [DATE 2004],
Francesco et al. [DAC 2004],
Dominguez et al. [JES 2005],
Li et al. [PACT 2005],
Egger et al. [Emsoft 2006],
Janapsatya et al. [ASPDAC 2006].

Pipelines?

Use thread-interleaved pipelines to avoid hazards

An old idea (60s): one thread per pipeline stage

Like Simultaneous Multi-threading, but it works



Lee and Messerschmitt, *Pipeline Interleaved Programmable DSP's: Architecture*, ASSP-35(9) 1987.

Interrupts?

One processor per interrupt source

Use polling; more predictable

I/O processors have a long history anyway

Really a way to share the processor resource across I/O sources

Isn't this wickedly inefficient?

Go Ahead: Leave Processors Idle

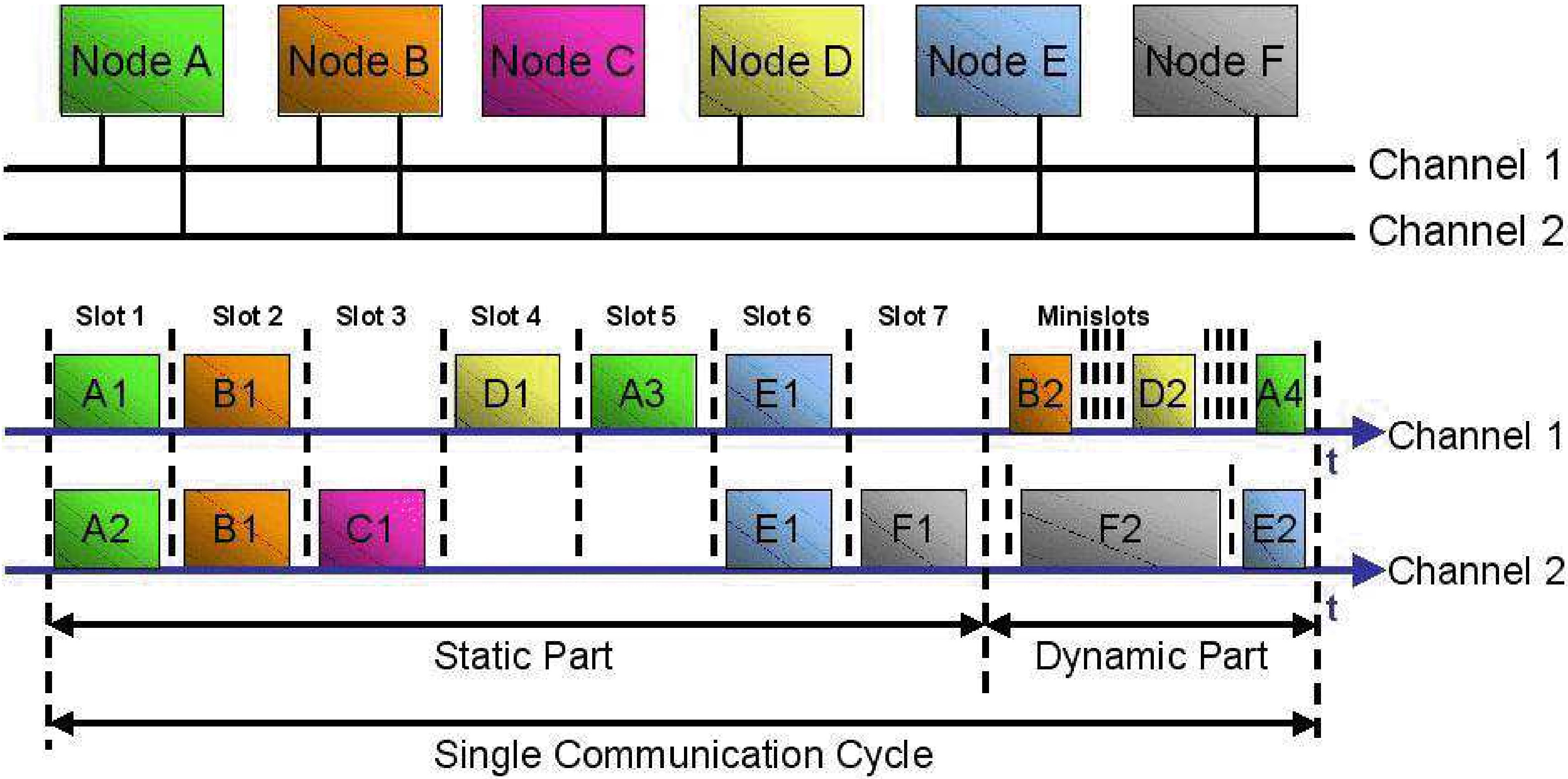
Modern processors do this at the functional unit level.
Schuette and Shen (MICRO 1991) found for their VLIW,

Unit	**Utilization**
Integer Fetch Unit	12–44%
Floating-point Fetch Unit	7–23%
Integer Registers	4–37%
Floating-point Registers	8–25%
Shared Registers	1–65%
Integer Bus	1–22%
Floating-point Bus	4–25%
Shared Bus	2–5%
Address Bus	2–37%

This is actually a good thing for power

Communication?

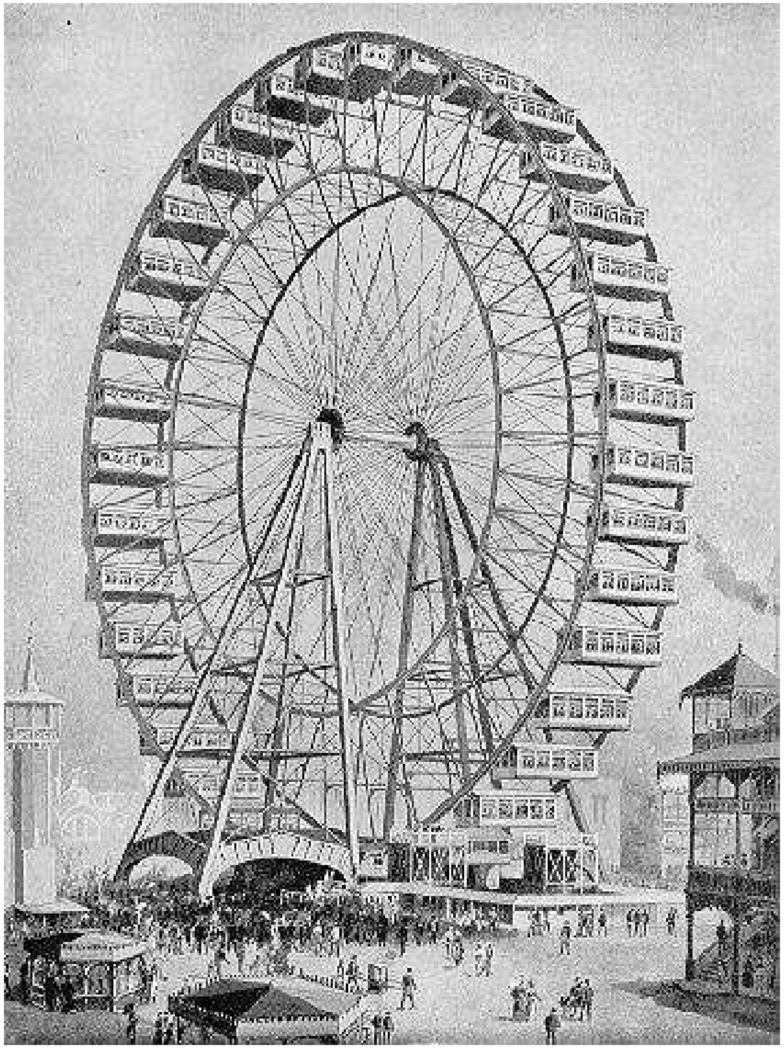
Use time-triggered busses (statically scheduled, periodic)
Examples: FlexRay, TTP, ATM



Source: TZM

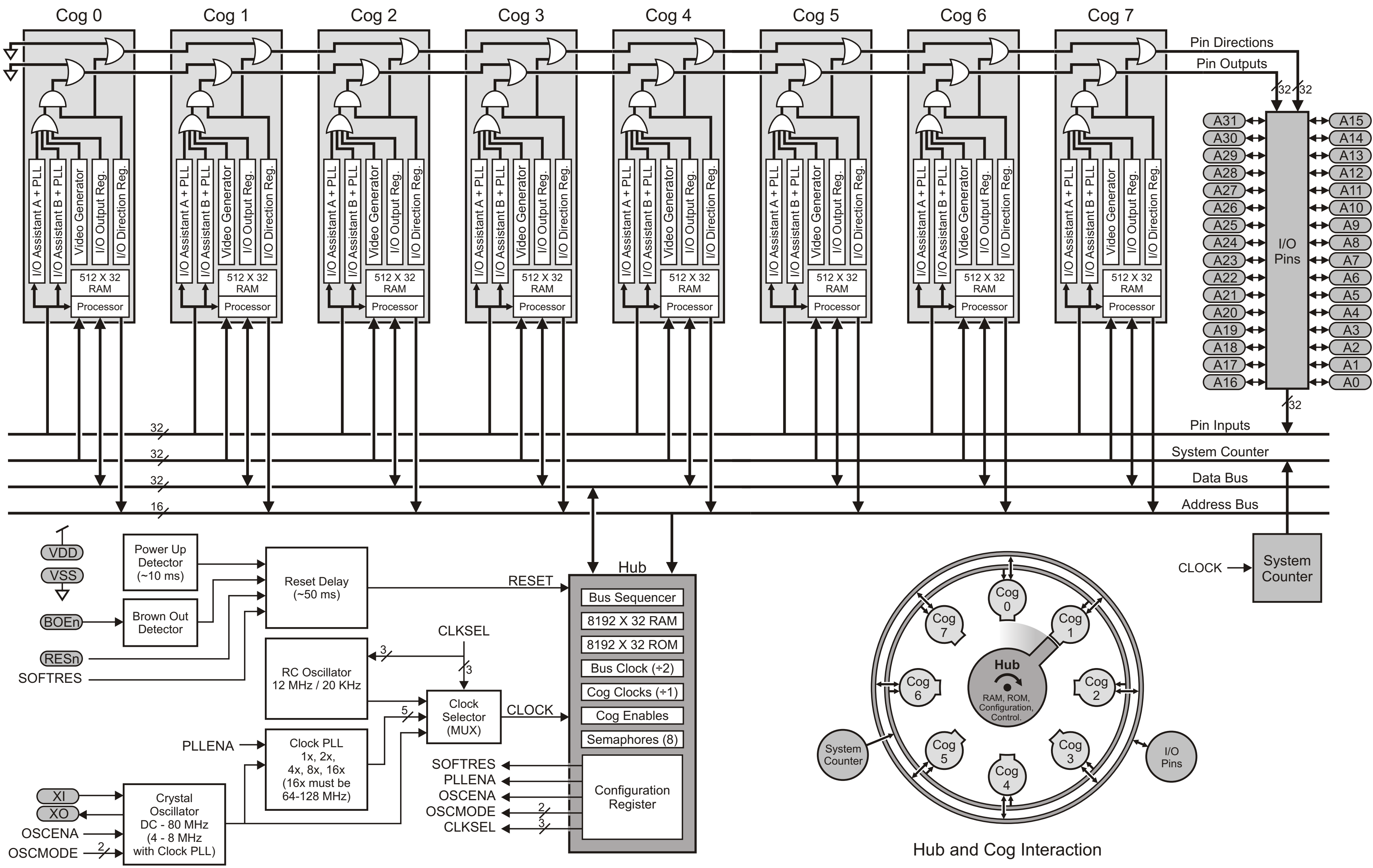
Shared Resources?

Like communication, scheduled, periodic access sharing



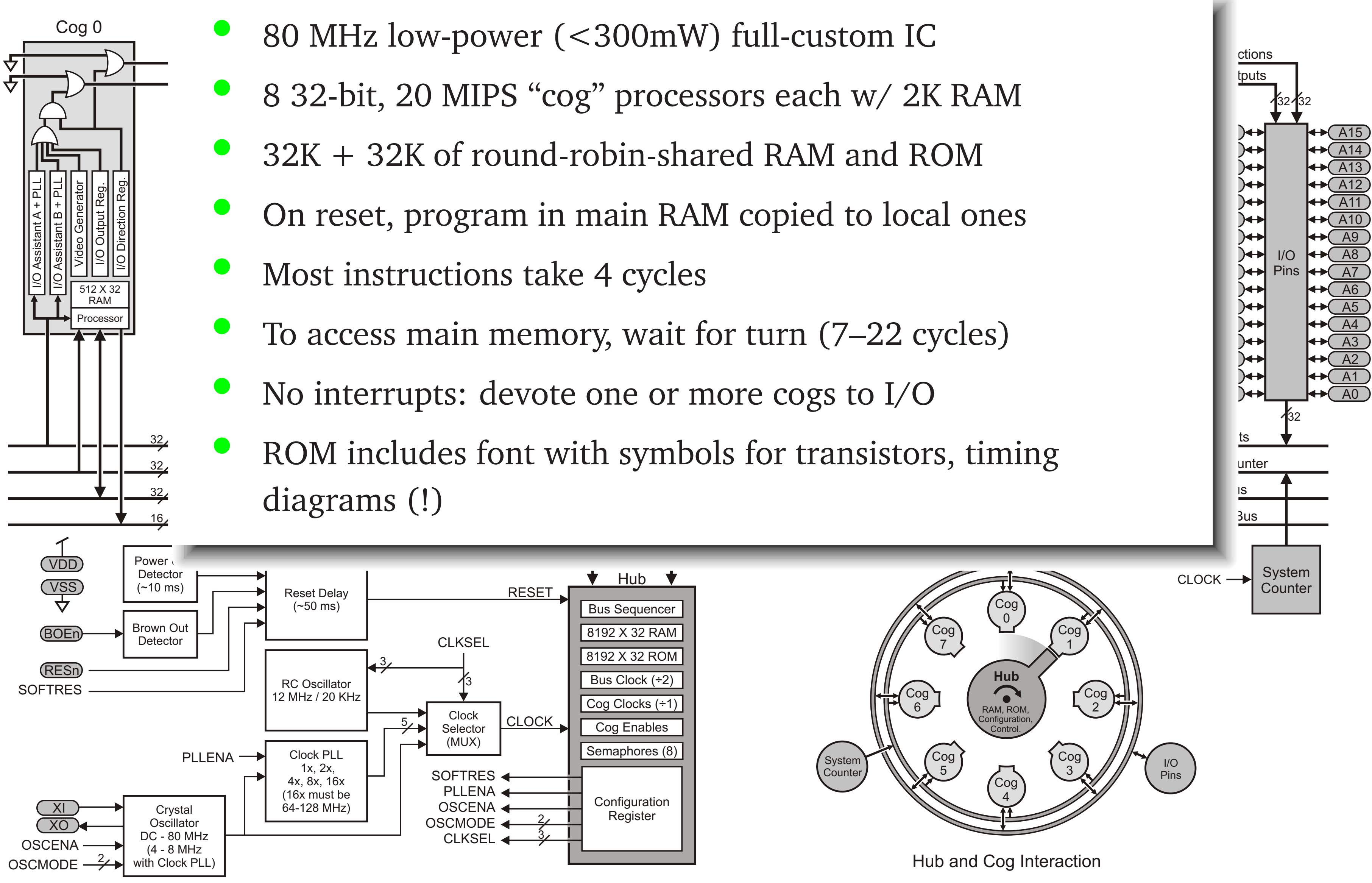
First Ferris Wheel, 1893 World's Columbian Exposition, Chicago

The Parallax Propeller Chip



The Parallax Propeller Chip

- 80 MHz low-power (<300mW) full-custom IC
- 8 32-bit, 20 MIPS “cog” processors each w/ 2K RAM
- 32K + 32K of round-robin-shared RAM and ROM
- On reset, program in main RAM copied to local ones
- Most instructions take 4 cycles
- To access main memory, wait for turn (7–22 cycles)
- No interrupts: devote one or more cogs to I/O
- ROM includes font with symbols for transistors, timing diagrams (!)



Operating System?

Process scheduling not necessary

Resource allocation largely static

Hardware abstraction layer (device drivers, etc.) useful

An Example: An ISA with Timing

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

One additional “deadline” instruction:

dead *timer*, *timeout*

Wait until *timer* expires, then immediately reload it with *timeout*.

Nicholas Ip and Stephen A. Edwards, “A Processor Extension for Cycle-Accurate Real-Time Software,” Proceedings of EUC, Seoul, Korea, August 2006.

Programmer's Model

General-purpose Registers

15	0
$0 (= 0)	
$1	
$2	
⋮	
$13	
$14	
$15	

Timers

15	0
$t0	
$t1	
$t2	
$t3	

Program counter

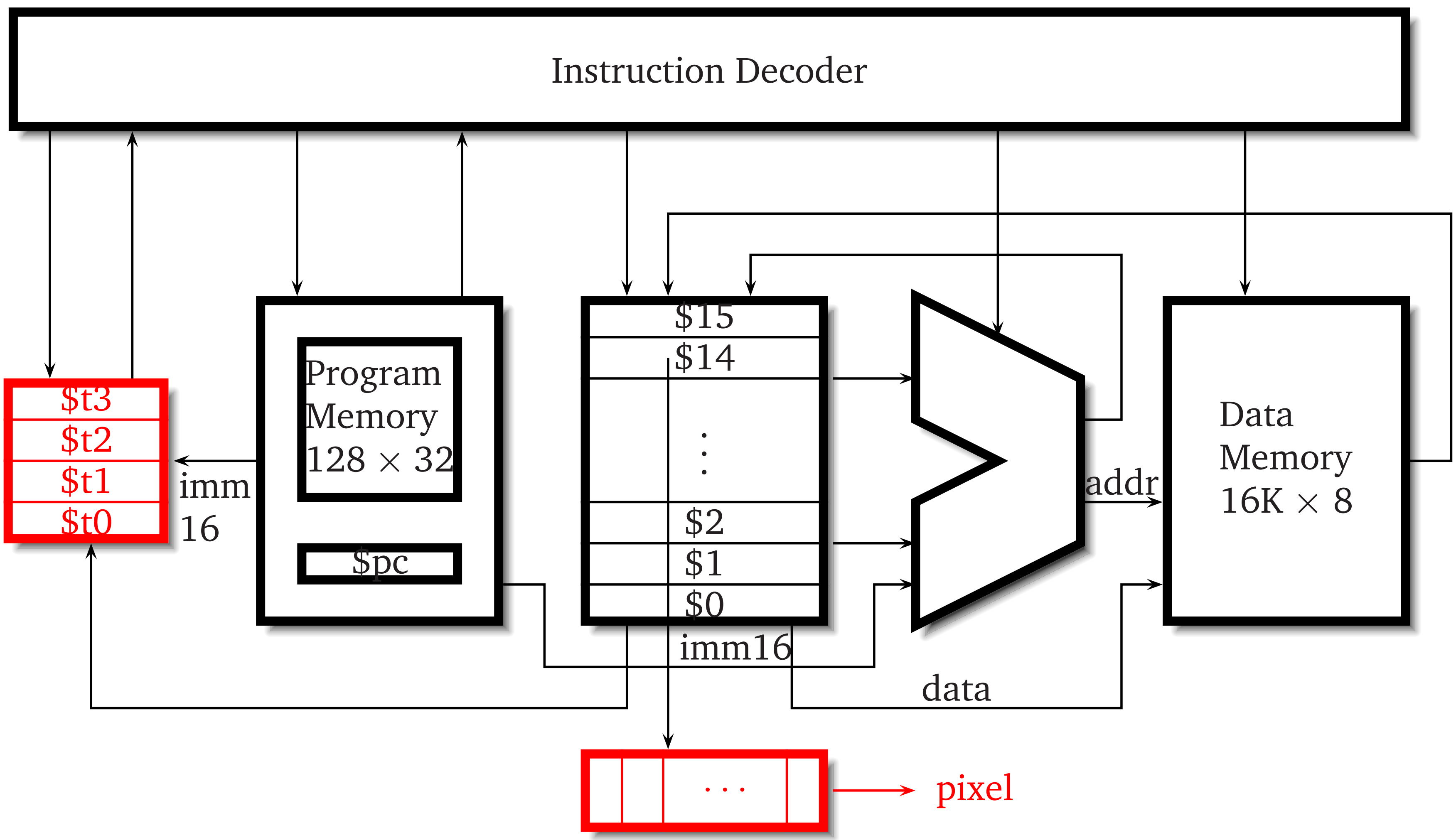
15	0
$pc	

Instructions

add	Rd, Rs, Rt
addi	Rd, Rs, imm16
and	Rd, Rs, Rt
andi	Rd, Rs, imm16
be	Rd, Rs, offset
bne	Rd, Rs, offset
j	target
lb	Rd, (Rt + Rs)
lbi	Rd, (Rs + offset)
mov	Rd, Rs
movi	Rd, imm16
nand	Rd, Rs, Rt
nandi	Rd, Rs, imm16
nop	
nor	Rd, Rs, Rt
nori	Rd, Rs, imm16

or	Rd, Rs, Rt
ori	Rd, Rs, imm16
sb	Rd, (Rt + Rs)
sbi	Rd, (Rs + offset)
sll	Rd, Rs, Rt
slli	Rd, Rs, imm16
srl	Rd, Rs, Rt
srli	Rd, Rs, imm16
sub	Rd, Rs, Rt
subi	Rd, Rs, imm16
dead	T, Rs
deadi	T, imm16
xnor	Rd, Rs, Rt
xnori	Rd, Rs, imm16
xor	Rd, Rs, Rt
xori	Rd, Rs, imm16

Architecture



Behavior of *Dead*

```
deadi   $t0, 8
add     $r1, $r2, $r3
deadi   $t0, 10
add     $r1, $r2, $r3
```

cycle	**instruction**	**$t0**
−4	deadi $t0, 8	3
−3	"	2
−2	"	1
−1	"	0
0	add $r1, $r2, $r3	7
1	deadi $t0, 10	6
2	"	5
⋮	⋮	⋮
7	"	0
8	add $r1, $r2, $r3	9

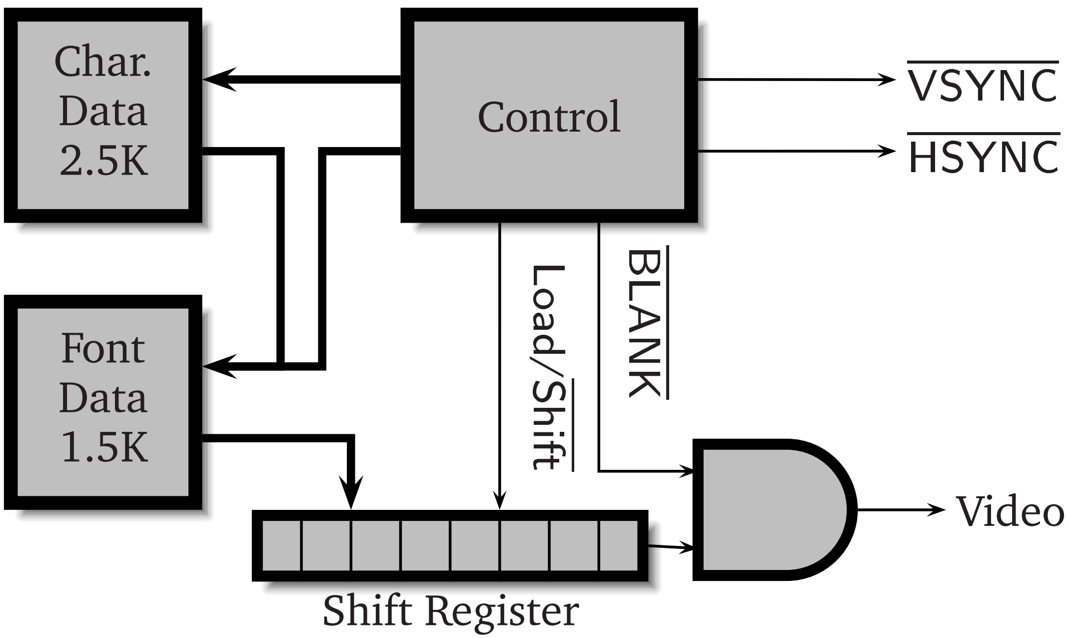
8 cycles

Case Study: Video

80 $\times$ 30 text-mode display, 25 MHz pixel clock

Need 40 ns precision

Shift register in hardware; everything else in software



Case Study: Video

```
    movi    $2, 0              ; reset line address
row:
    movi    $7, 0              ; reset line in char
line:
    deadi   $t1, 96            ; h. sync period
    movi    $14, HS+HB
    ori     $3, $7, FONT       ; font base address
    deadi   $t1, 48            ; back porch period
    movi    $14, HB
    deadi   $t1, 640           ; active video period
    mov     $1, 0              ; column number
char:
    lb      $5, ($2+$1)        ; load character
    shli    $5, $5, 4          ; *16 = lines/char
    deadi   $t0, 8             ; wait for next character
    lb      $14, ($5+$3)       ; fetch and emit pixels
    addi    $1, $1, 1          ; next column
    bne     $1, $11, char
    deadi   $t1, 16            ; front porch period
    movi    $14, HB
    addi    $7, $7, 1          ; next row in char
    bne     $7, $13, line      ; repeat until bottom
    addi    $2, $2, 80         ; next line
    bne     $2, $12, row       ; until at end
```

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

```
  movi $3, 0x0400       ; final bit mask (10 bits)
  movi $5, 651          ; half bit time for 9600 baud
  shli  $6, $5, 1       ; calculate full bit time

wait_for_start:
  bne   $15, $0, wait_for_start
got_start:
  wait  $t1, $5         ; sample at center of bit
  movi $14, 0           ; clear received byte
  movi $2, 1            ; received bit mask
  movi $4, 0            ; clear parity
  dead $t1, $6          ; skip start bit
receive_bit:
  dead $t1, $6          ; wait until center of next bit
  mov  $1, $15          ; sample
  xor   $4, $4, $1      ; update parity
  and   $1, $1, $2      ; mask the received bit
  or     $14, $14, $1   ; accumulate result
  shli  $2, $2, 1       ; advance to next bit
  bne   $2, $3, receive_bit
check_parity:
  be     $4, $0, detect_baud_rate
  andi  $14, $14, 0xff  ; discard parity and stop bits
```

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

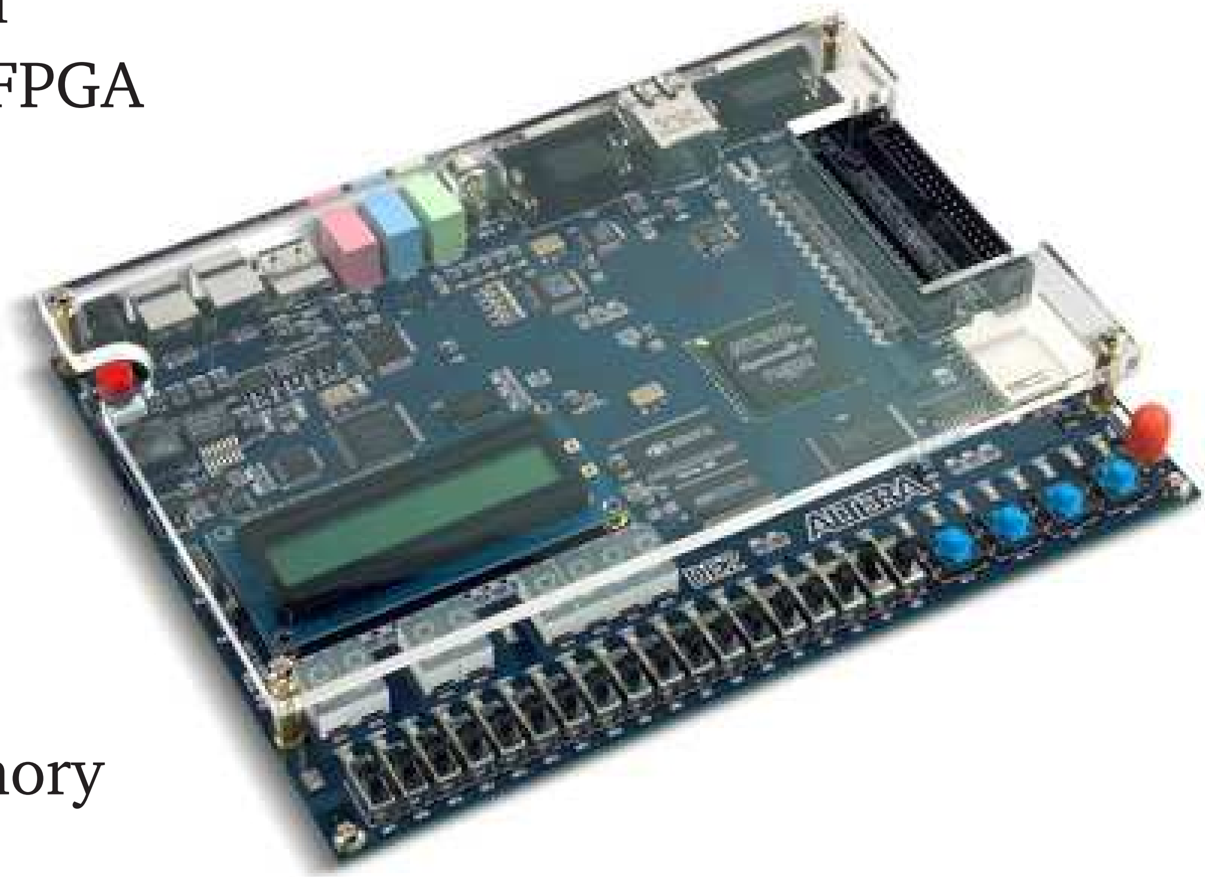
Synthesized on an
Altera Cyclone II FPGA
(DE2 board)

Coded in VHDL

Runs at 50 MHz

Unpipelined

Uses on-chip memory



Our Vision: PRET Machines

Predictable performance, not just good average case

Current	Alternative
Caches	Scratchpads
Pipelines	Thread-interleaved pipelines
Function-only ISAs	ISAs with timing
Function-only languages	Languages with timing
Best-effort communication	Fixed-latency communication
Time-sharing	Multiple independent processors

Final Provocative Hypothesis

PRET will help parallel general-purpose applications by making their behavior reproducible.

Data races, non-atomic updates still a danger, but at least they can be reproduced.