TITAC–2:

A 32-bit Scalable-Delay-Insensitive Microprocessor

Takashi Nanya[1),2)]
Akihiro Takamura[2)], Masashi Kuwako[1)], Masashi Imai[1)]
Taro Fujii[2)], Motokazu Ozawa[2)], Izumi Fukasaku[2)], Yoichiro Ueno[2)]
Fuyuki Okamoto[3)], Hiroki Fujimoto[3)], Osamu Fujita[3)]
Masakazu Yamashina[3)], Masao Fukuma[3)]

1) Research Center for Advanced Science and Technology, University of Tokyo
2) Department of Computer Science, Tokyo Institute of Technology
3) Microelectronics Research Laboratories, NEC Corporation


PRESENTATION OVERVIEW

Why Asynchronous ?

Scalable-Delay-Insensitive Model

TITAC–2 Architecture

Design Methodology

Chip Implementation

Conclusions


Why Asynchronous ?

CMOS Technology in 2007 (suggested at ISSCC97)

Minimum Tr. size : $0.1\mu m$
Number of logic Trs. : $50M/cm^2$
Gate delay: $10ps$
Chip area: $10cm^2$

Fundamental Limitation:

"No signal can run $3mm$, or probably even $1mm$, in $10ps$."

Wire delays moving into dominance on chip, getting harder and harder to control

Synchronous systems cannot enjoy the ultra-high speed of "picosecond devices" !!


Why Asynchronous ?

Without any global clock

1. **Potentially fast operation:**
   Achieve "average-case" performance, i.e. optimize frequent operations and allow rare operations to spend more time
   Future technology scales up system performance
2. **Timing-fault-tolerance:**
   Robust and resilient with possible delay variation caused in fabrication process and operating environment
3. **Low power consumption:**
   Signal transitions occur only when and where needed
4. **Ease of modular composition:**
   No clock alignment needed at interfaces
   Can overcome design complexity


Delay Models

Assumptions on gate and wire delays.

Play an essential role for dependable system design.

Must be carefully examined and validated for design process, device technology and operating environment.

Synchronous: Fixed or bounded delays that are known a priori

Asynchronous: Variable delays that are unknown

Fundamental-Mode: Bounded gate and wire delays

Speed-Independent: Unbounded gate delays with no wire delays

Delay-Insensitive: Unbounded gate delays and wire delays

Quasi-Delay-Insensitive: Delay-Insensitive + Isochronic Forks
(All branches of a fork have the same wire delay)

Observations

Synchronous or Bounded-Delay model:

Too optimistic of uncontrollable wire delays

Can make design expensive to guarantee reliable operations

Delay-Insensitive or Quasi-Delay-Insensitive model:

Too cautious against unlikely variations of delay

Can cause excessive hardware overhead and performance penalty

What is likely in future VLSI technology ?

Component delays are liable to uncontrollable variation through design phase, fabrication process, operating environment, aging, etc

However,

It is unlikely that some delays decrease (increase) while others increase (decrease)

$\rightarrow$ Scalable-Delay-Insensitive (SDI) Model

Scalable-Delay-Insensitive (SDI) Model

Unbounded delays with bounded relative variation ratio

[Definition]

Consider any two components C_1 and C_2

d_1, d_2**:** Delays for C_1, C_2

$D = d_1/d_2$**:** Relative delay D of C_1 to C_2

D_e**:** Estimated relative delay(at design phase)

D_a**:** Actual relative delay (through system's lifetime)

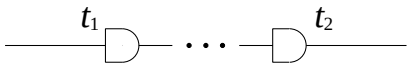
$R = D_a/D_e$**:** Relative variation ratio

Then, SDI model assumes that $1/K < R < K$, where K is a constant (maximum variation ratio)

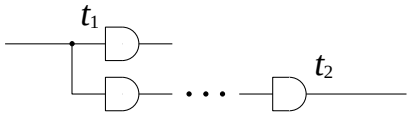
How SDI model works

Specification: Transition t_1 precedes transition t_2

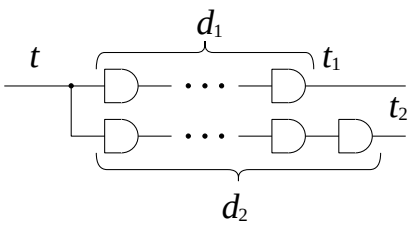
DI implementation : t_2 must be caused by t_1



QDI implementation: t_2 may be caused by other fanout branch that shares its stem with t_1

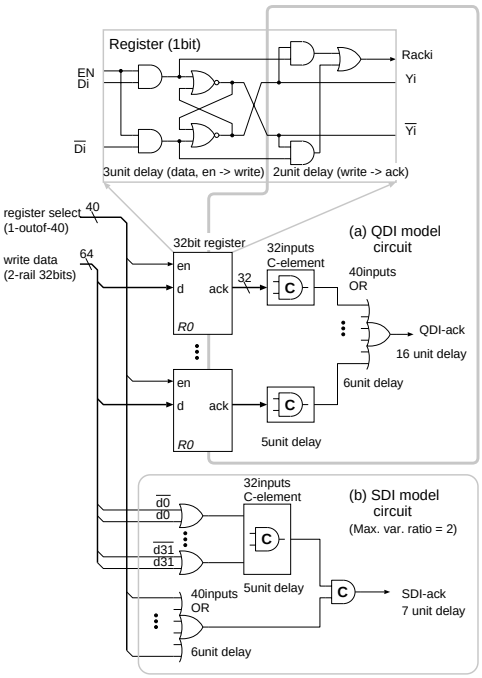


SDI implementation: t_2 may be caused by common stem t that also causes t_1, if $K \cdot d_1 < d_2$



Example

Completion signal generation with QDI and SDI models




SDI Design Methodology

Step 1: Divide the entire system into functional blocks

Step 2: Design each block as well as interconnections with QDI model

Step 3: Determine K considering process technology and area size of each block

Step 4: Apply SDI implementation to each QDI block whenever $K\,d_1 < d_2$

TITAC–2 design: $K = 2$ validated by limiting each functional block to be within a maximum of $1.93mm \times 1.93mm$ based on 0.5μ CMOS technology used


TITAC–2 Architecture

Asynchronous (clock-free) version of MIPS–R2000

Why MIPS–R2000 ?

- Nice target to demonstrate asynchronous design
- Easy to compare with synchronous counterpart
- Reasonably simple to design
- C Compiler available

Instruction set ; almost same

Some instructions modified;

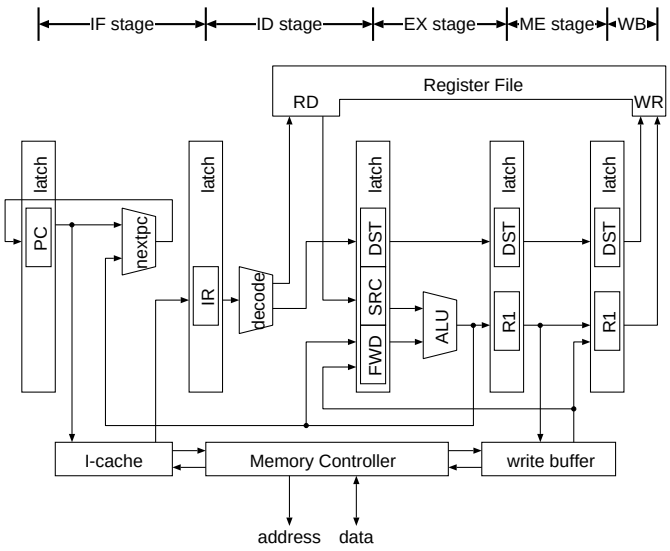
- multiply/divide resulting in least significant 32 bits
- 2 delay slots for branch instructions
- privileged instructions

Object code: not compatible due to different instruction encoding


TITAC–2 Instruction Set

Logical	AND, ANDI, OR, ORI XOR, XORI, LU, LUI
Arithmetic	ADD, ADDI, SUB, SUBI
Multiply	MUL, MULU
Divide	DIV, DIVU, MOD, MODU
Compare	SLT, SLTI, SLTU, SLTIU
Shift	SLL, SRA, SRL SLLV, SRAV, SRLV
Load	LB, LBU, LH, LHU, LW
Store	SB, SH, SW
Branch	J, JR, JAL, JALR BEQ, BNE, BGTZ, BLEZ BGEZ, BGEZAL, BLTZ, BLTZAL
Privileged Instructions	MOVSR, MOVRS, RFE, SYSCALL


TITAC–2 Architecture



5-stage pipeline structure

8-KByte Instruction Cache on chip
(Data cache not implemented)

40 32-bit registers
(R32 – R39 for kernel-mode use only)

Exception Handling
(for user-mode only)

External Interrupt

Memory Protection


TITAC–2 Design Features

2-level Delay Assumption
SDI model for local functional blocks
DI/QDI model for global interconnection

Elastic asynchronous pipeline with FIFO buffer

2-rail 2-phase register transfer

Idle-phase acceleration in 2-rail 2-phase data-path

Adjustable delay elements for bundled-data path
(cache, external bus)


Data-Path Encoding

2-rail 2-phase (return-to-zero) for register transfer

Working phase:

$(0,0) \rightarrow (0,1)$: logic value "0" has arrived

$(0,0) \rightarrow (1,0)$: logic value "1" has arrived

Idle phase:

$(0,1) \rightarrow (0,0)$: logic value "0" has cleared away

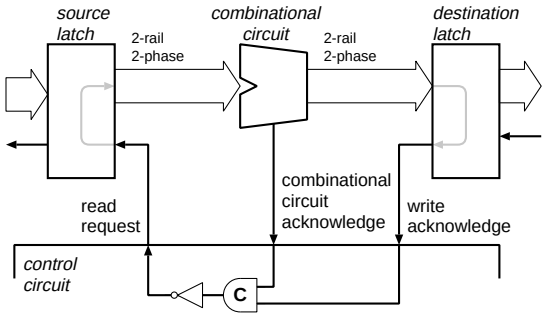
$(1,0) \rightarrow (0,0)$: logic value "1" has cleared away

Bundled-Data for on-chip cache and external interface


Asynchronous Pipeline Stage Model

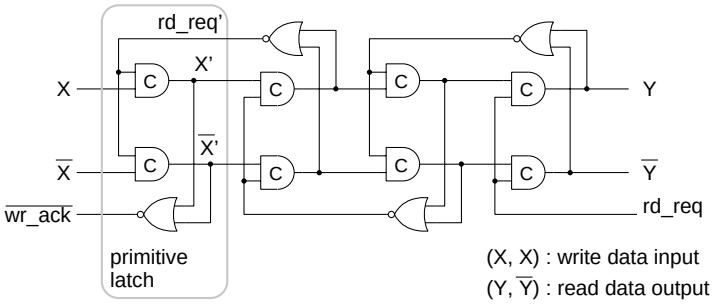
Request/Acknowledge Handshake

Decentralized Control




Pipeline Latch

Asynchronous FIFO: 4 primitive latches in cascade
7% faster than 3-latch FIFO




Combinational Circuit Design

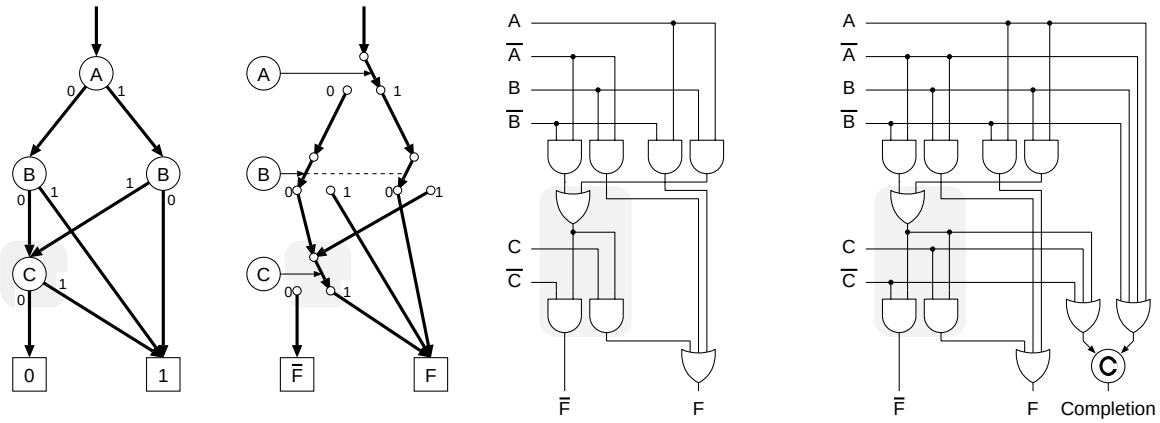
2-rail logic implementation

- **Direct mapping from BDD representation**
- **Easy to generate completion signals**
- **Hazard-free due to monotonic signal transitions**

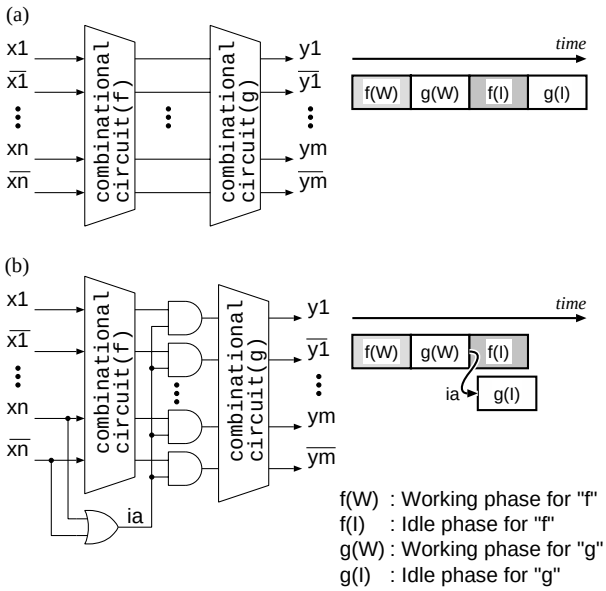
DCVSL implementation for frequently used modules
(e.g. adders in multiplier)

Gate-level implementation for other cases


2-rail Implementation from BDD

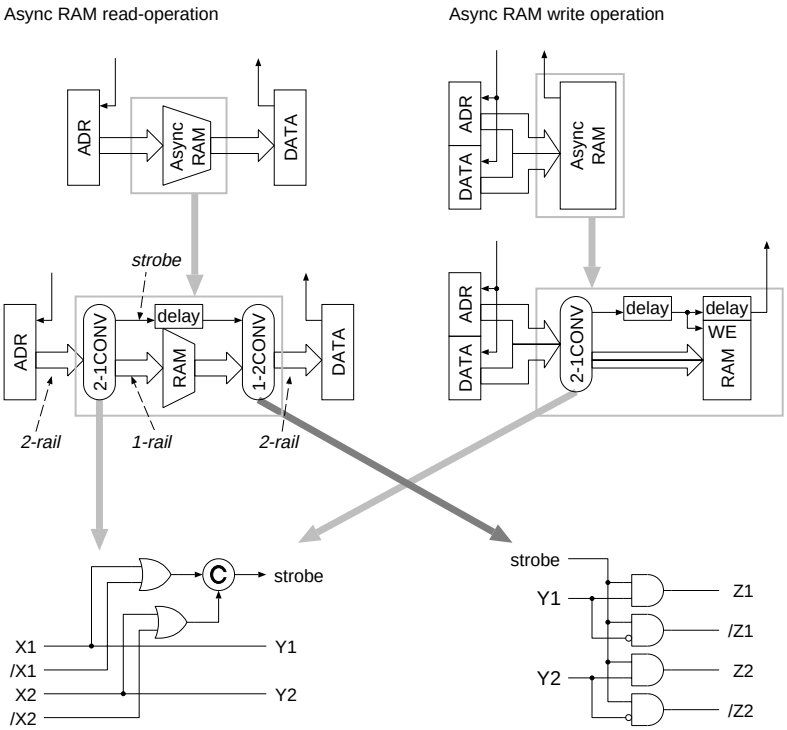


Idle Phase Acceleration



f(W) : Working phase for "f"
f(I) : Idle phase for "f"
g(W) : Working phase for "g"
g(I) : Idle phase for "g"

Conversion between 2-railed data and Bundled-data



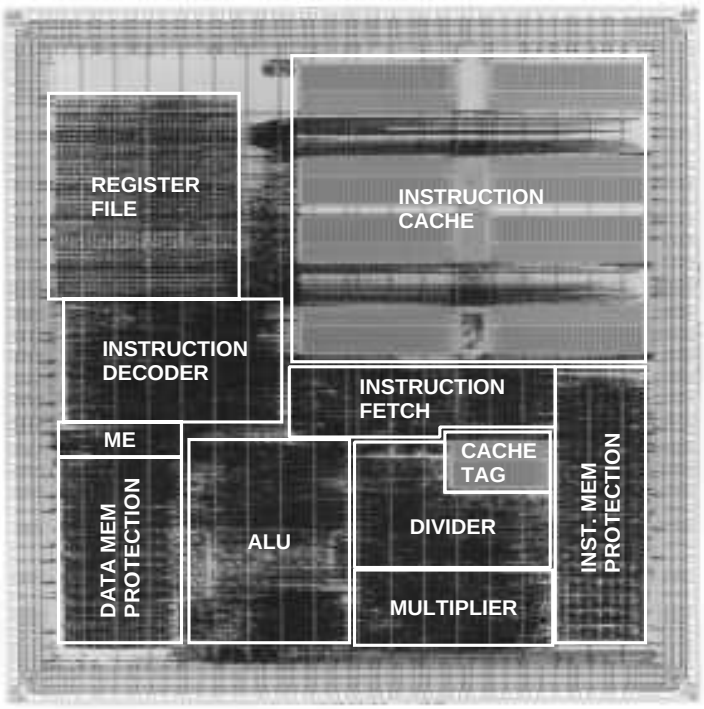
Chip Implementation

Fabricated in NEC's 0.5 μm, 3.3 V CMOS Process with 3 metal layers

Die size: $12.15mm \times 12.15mm$
496,367 Logic Transistors + 8.6K Byte Memory macro

Number of I/O pins: 164 (excluding power pins)

Die Photo



Performance

Dhrystone V2.1 benchmark

52.3 VAX MIPS consuming 2.1 W at 3.3 V in room temperature

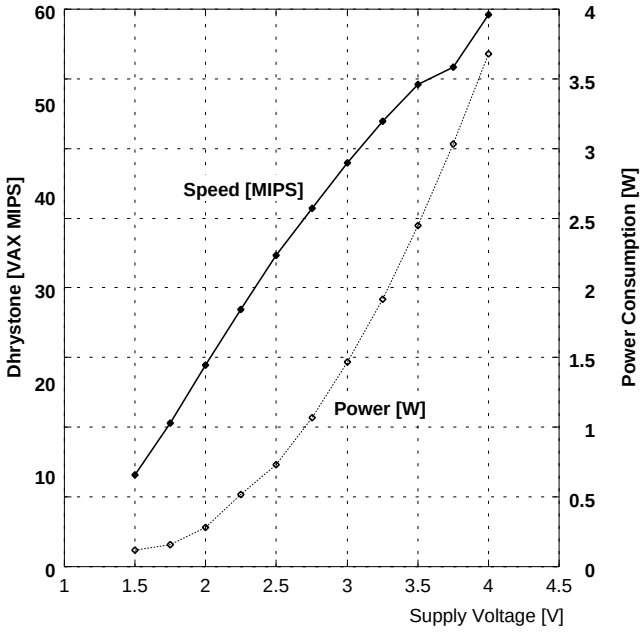
Delay Insensitivity

Power Supply Voltage: from 1.5 V to 6.0 V(variable)

Chip Surface Temperature: from $-196°C$ to $+100°C$

Speed and Power Consumption vs. Supply Voltage

(at room temperature)



Tools Used

Commercial: synchronous

- Verilog for RT-level and logic level simulation
- Cell-3 Ensemble for layout and design rule check
- Saber for analog simulation of original macros

Home-made: asynchronous

- Asynchronous synthesis from STG
- Speed-independent logic verification
- Delay evaluation
- Test generation

Conclusions

Asynchronous design methodology is now available

Performance is already comparable with synchronous counterpart

SDI model ensures both performance and dependability

Easiness of modular design helps much

More suitable architecture can fully exploit concurrency

Testing is a major challenge