Introduction to

Computing Without Clocks

Simon Moore

URL: http://www.cl.cam.ac.uk/users/swm11

Overview

- why use self-timed circuits?
- flavours of self-timing
  - matched delays
  - completion detection
- self-timed processors
  - micropipelining and counterflow pipelines
  - Amulet
  - rotary pipelines

Self-Timed Advantages — Often Cited

Al Davis, Async'94

1. archive average case performance
2. power consumed only where needed
3. ease of modular composition
4. no clock alignment at the interfaces
5. metastability has time to end
6. avoid clock distribution costs
7. easier to exploit concurrency
8. intellectual challenge
9. intrinsic elegance
10. global synchrony does not exist anyway

Self-Timed Advantages — NOT Often Cited

Al Davis, Async'94

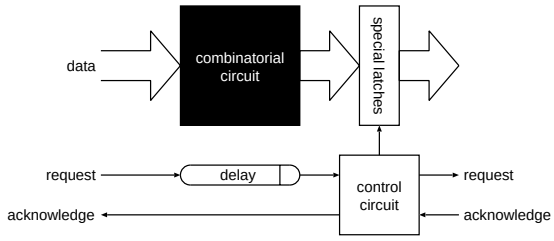
1. it really pisses my boss off
2. I like reinventing wheels
3. I like to be different
4. gee — I really don't know
5. people and circuits need to play by same rules
6. I don't understand synchronous circuits
7. world problems stem from glitches
8. synchronous design gives me gas
9. clock radiation causes hair loss
10. it's none of your business

Historical Perspective

- early digital circuit design explored various synchronisation mechanisms
- clocked circuits won because lower device (valve!) count and faster (e.g. Ordvac, Illinois, 1952)
- Muller instigated a great deal of research into self-timed circuits in the 50's and 60's
- as we reach the CMOS end point:
  - transistors are cheap and plentiful
  - long wires and global synchronisation are a problem
  - low power is important (don't switch the transistors too often)

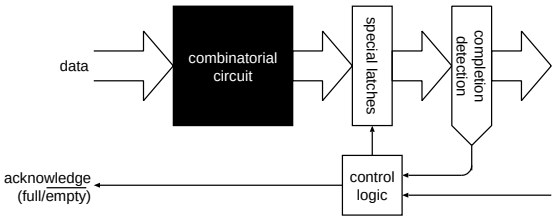
Self-timed Circuit Approaches

Delay based



Self-timed Circuit Approaches cont...

Completion based



☞ completion signal (request) encoded with the data...more later

Encoding Completion Signals

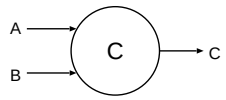
code	meaning
00	clear
01	logical 0
10	logical 1

(a) 1 of 2 encoding

code	meaning
0000	clear
0001	logical 0
0010	logical 1
0100	logical 2
1000	logical 3

(b) 1 of 4 encoding

Muller C-element



(a) symbol

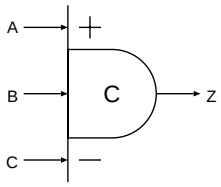
A	B	C
0	0	0
0	1	C
1	0	C
1	1	1

(b truth table

☞ think of it as a majority flip-flop

☞ little known and yet immensely useful

Asymmetric C-elements — Furber Style

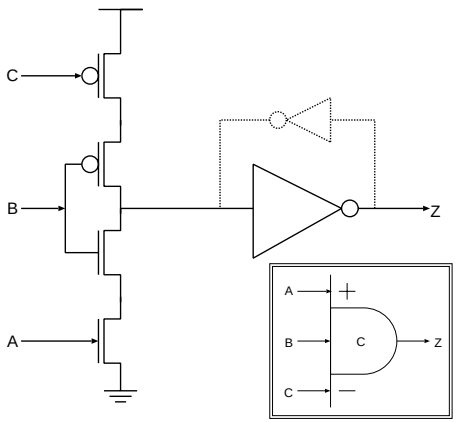


(a) symbol

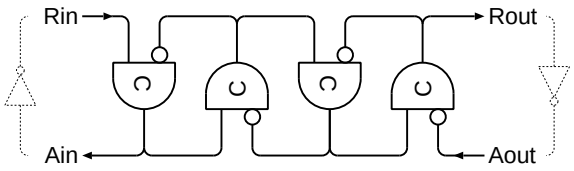
A	B	C	Z
X	0	0	0
X	0	1	C
0	1	X	C
1	1	X	1

(b) truth table

Implementing Asymmetric C-elements



Event FIFO



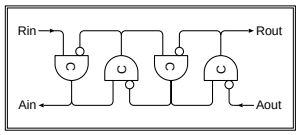
N.B. events are edges

☞ works with level information but you use twice as many C-elements

Human Event FIFO

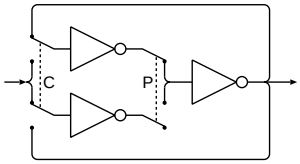
IF predecessor AND successor differ
THEN copy predecessor
ELSE hold your state

☞ take your time — this is delay insensitive

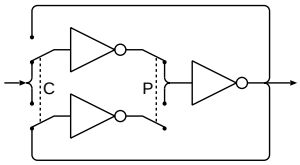


Capture/Pass Latches

Capture Mode



Pass Mode

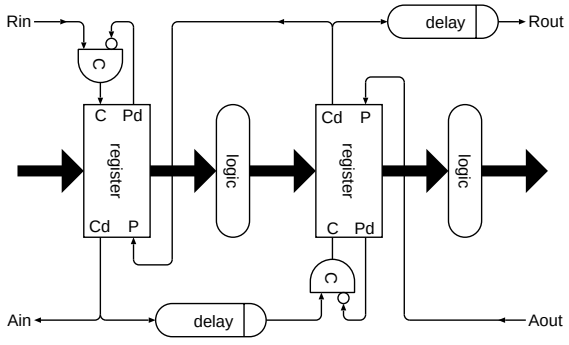


☞ each edge on the control signals has a meaning

☞ just double buffering

Micropipelines

from Ivan Sutherland's Turing award lecture



Self-timed Processor Design Issues

- high end — multiple instruction issue
  - superscalar designs have complex (irregular) feed-forward paths
  - global synchronisation mechanism (clock) simplifies design
  - exception handling is a killer
- low end — low power
  - keep it simple
  - minimise latches
  - minimise transitions

Amulet

Steve Furber's group at Manchester

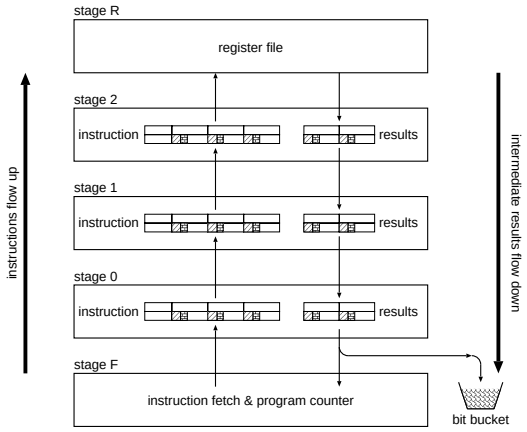
- ARM processor based on micropipelines
- Amulet 1 based on Sutherland's micropipelines
- Amulet 2 — taping out soon
  - — dynamic transparent D-type latches
    - – faster and smaller than capture/pass latches
  - — 4-phase (level sensitive) handshaking

Assessment of Amulet

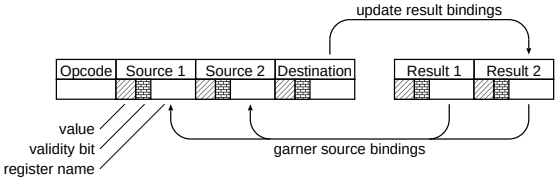
- good points:
  - – low power
  - – small (thus cheap)
- problems:
  - – could not easily be made superscalar (but probably wouldn't want to for low power)
  - – delays require large margin for error (Philips guys say 100%)
  - – margins in delays are cumulative down the pipeline

Counter-flow Pipelines

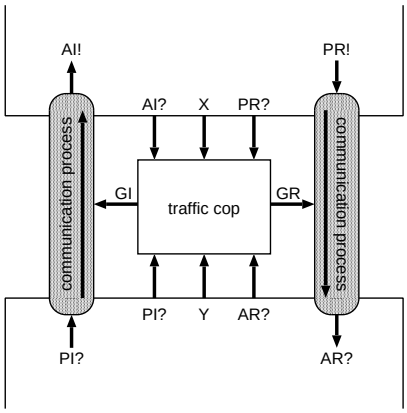
Sutherland and Sproul



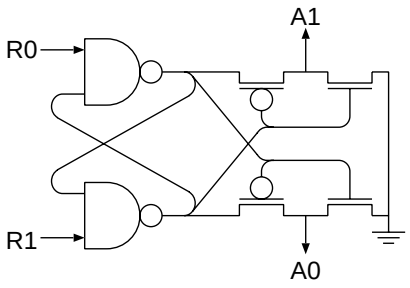
Counter-flow Pipelines — Garnering



Counter-flow Pipelines — Traffic Cops



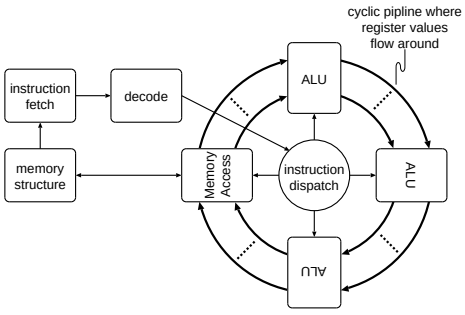
Seitz Arbiter



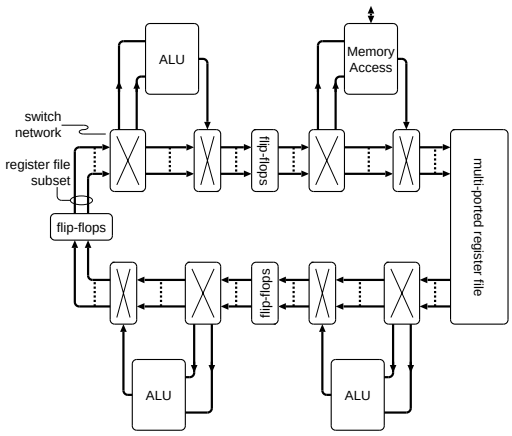
Assessment of Counterflow Pipelines

- good points:
  - register feedforwarding — regular structure
  - speculative execution and exceptions easy
    - register file not written to until instruction completes correctly
    - “poison pills” to remove erroneous values
- problems:
  - too much arbitration
  - multiple instruction issue tricky (faster pipeline and more sidings?)
  - usual problems with micropipelines

Rotary Pipelines



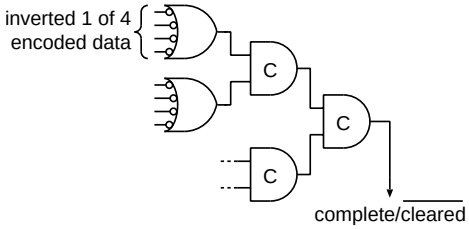
Rotary Pipeline with Register File



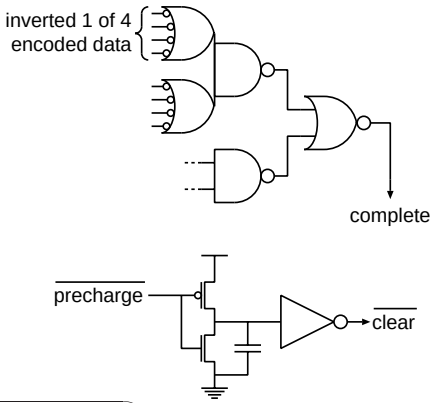
Rotary Pipeline Characteristics

- multiple instruction issue easy
- branches
  - non-conditional branches taken by decode stage
  - conditional branches predicted — confirmation comes from a stage in the ring
- speculative execution is easy
  - hold old values in flip-flops until you know which of the new values are valid
- exception handling easy
  - use speculative execution mechanism
- but circuits structures are a bit big...

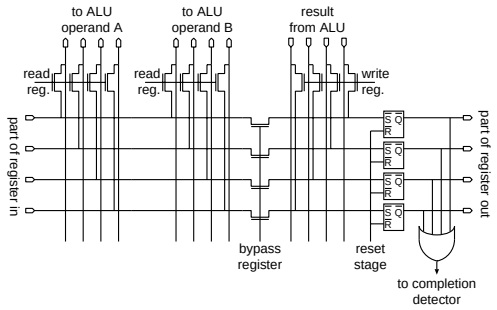
Determining Completion



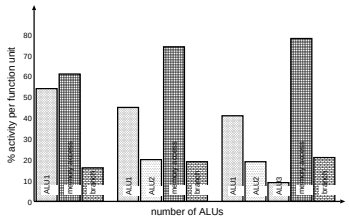
Completion on Data + Matched Delays on Clear



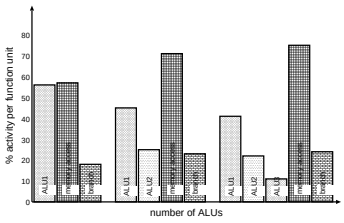
Pipeline Stage



Function Unit Utilisation for Compress



Function Unit Utilisation for Dhrystone



Concluding Remarks

- Will self-timed circuits subsume clocked design?
  - jury out on that one
- Are matched delays better than completion detection?
  - both similar power dissipation
  - matched delays tend to be smaller
  - matched delays = local clocking so work well for clocked designs
  - completion detection does not have cumulative overheads

☞ Self-timed circuits require new architectures

  - the architectural possibilities may be the key to the success of self-timed circuits