Introduction to asynchronous circuit design

Using slides from:
Jordi Cortadella, Universitat Politècnica de Catalunya, Spain
Michael Kishinevsky, Intel Corporation, USA
Alex Kondratyev, Theseus Logic, USA
Luciano Lavagno, Università di Udine, Italy
Hoon Lee & Simon Singh

Motivation

- No Global synchronization
- Better Performance
- Lower Power Consumption
- Lower Cost
Global Synchronization

- Synchronous Approach
  - Global clock distribution and synchronization
  - Advantages
    - Simple way to design
    - Well understood
  - Disadvantages
    - Difficult to maintain the global synchronization for small feature size technologies
    - Clock Skew

Synchronous circuit

Implicit synchronization
Global Synchronization

- Asynchronous Approach
  - No global clock distribution
  - No global synchronization
  - No clock skew

Asynchronous circuit

Explicit synchronization: Req/Ack handshakes
Synchronous communication

- Clock edges determine the time instants where data must be sampled
- Data wires may glitch between clock edges (set-up/hold times must be satisfied)
- Data are transmitted at a fixed rate (clock frequency)

Example: memory read cycle

Transition signaling, 4-phase
**Example: memory read cycle**

- Transition signaling, 2-phase

---

**Performance**

- **Synchronous Approach**
  - Optimize for the worst case conditions

- **Asynchronous Approach**
  - Possible to construct circuit optimized for the typical (average) cases
Power Consumption

- Synchronous Approach
  - Unconditional power dissipation
    - All parts of the circuit are activated and dissipate power whether they are needed or not

- Asynchronous Approach
  - Conditional power dissipation
    - Particular parts of the circuit are activated and dissipate power when they are actually required

Asynchronous Approach

- Advantages
  - No global synchronization required
  - Average-case performance
  - Less power consumption

- Disadvantages
  - Control logic complexity
  - Risk of deadlock
  - Possible loss of implied knowledge
Asynchronous modules

Data IN → DATA PATH → Data OUT

CONTROL

req in → start → done → req out
ack in →

Signaling protocol:

req in+ start+[computation] done+ req out+ ack out+ ack in+
req in- start- [reset] done- req out- ack out- ack in-

Completion detection

Completion detection tree
Bundled-data logic blocks

Conventional logic + matched delay

Micropipelines (Sutherland 89)
Asynchronous Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Group</th>
<th>Note</th>
</tr>
</thead>
</table>
| SCALP     | Univ. of Manchester | * First asynchronous super scalar processor  
                                          * Pipelined, Multiple functional units, Multiple issue, Explicit forwarding |
| CAP       | Caltech          | * First demonstration of asynchronous processor  
                                          * Delay insensitive with dual-rail encoded comm.  
                                          * Fetch-execute pipeline |
| NSR       | Univ. of Utah    | * Pipelined processor with pipeline stages separated by FIFO queues  
                                          * Pipelined, No forwarding, Decoupled branch and load/store |
| Fred      | Univ. of Utah    | * Development of NSR (From 16 to 32 bit data-path and instructions)  
                                          * Pipelined, Multiple functional units, Single issue, No forwarding, Decoupled branch and load/store |
| CFPP      | Sutherland Group | * Contribution on the result forwarding in an asynchronous pipeline  
                                          * Pipelined, Multiple execution stages, Single issue Result pipeline, Forwarding using counter-flow |
| TITAC     | Tokyo Inst. of Tech | * No pipelining and simple accumulation-based instruction set |
| AMULET    | Univ. of Manchester | * First asynchronous implementation of commercially important ISA |
ARM Architecture

- Acorn (Advanced) RISC Machine
  - Originally developed by Acorn Computers Ltd., England, between 1983-1985
  - One of the earliest RISC microprocessor for commercial use
  - Significant differences from subsequent RISC architecture
  - Inherited from the Berkely RISC I & II

ARM Architecture

- Features inherited from the Berkely RISC
  - A load-store architecture
  - Fixed length 32-bit instructions
  - 3-address instruction format
- Features rejected from the Berkely RISC
  - Register window
  - Delayed branches
  - Single cycle execution of all instructions
ARM Architecture

- Simplicity
  - Simple hardware
  - Simple instructions
  - Better code density
  - Better power efficiency
  - Smaller core size

AMULET Processor

- Asynchronous Microprocessor Using Low Energy Technology
  - Developed at the Univ. of Manchester, England in 1990’s
- AMULET (1,2,3)
  - World’s first commercial asynchronous RISC processor using ARM architecture
  - Adopt Sutherland’s Micropipeline design style
  - Increased throughput with reduced power
AMULET Processor

- Key innovations
  - Asynchronous communication
    - Request-Acknowledge handshake
    - Use of Micropipeline technique
  - Register coherency
    - Register lock without arbiters
  - Self-timed ALU
    - Reduced size
    - Faster operations
    - Reduced parasitics

Processor Performance

<table>
<thead>
<tr>
<th></th>
<th>AMULET1</th>
<th>AMULET2</th>
<th>AMULET3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>1 µm</td>
<td>0.5 µm</td>
<td>0.35 µm</td>
</tr>
<tr>
<td>Core Area</td>
<td>25 mm²</td>
<td>25 mm²</td>
<td>21 mm²</td>
</tr>
<tr>
<td>Transistors</td>
<td>58,374</td>
<td>93,000</td>
<td>113,000</td>
</tr>
<tr>
<td>Power</td>
<td>152 mW</td>
<td>140 mW</td>
<td>155 mW</td>
</tr>
<tr>
<td>MIPS/W</td>
<td>77</td>
<td>285</td>
<td>780</td>
</tr>
</tbody>
</table>
What is Micropipeline?

- Pipelines have both storage elements & processing logic
- This twosome alternate along a pipeline’s length
- Stripped of the processing logic, a pipeline is akin to a series of storage elements through which data can pass

What is Micropipeline?

- INELASTIC pipeline
  - Amount of data in it is fixed
  - Input rate and output rate must match exactly
  - Stripped of processing logic, inelastic pipeline acts like a shift register

- ELASTIC pipeline
  - Amount of data in it may vary
  - Input rate and output rate may differ momentarily because of internal buffering
  - Stripped of processing logic, an elastic pipeline becomes a flow-through first-in-first-out (FIFO) memory
What is Micropipeline?

- FIFOs can be clocked or event-driven, here the imperative property being that FIFOs are elastic
- Sutherland coined the phrase “Micropipeline” for an event-driven elastic pipeline
- The asynchronous FIFO pipeline can be with or without processing logic

![Simple Micropipeline FIFO](image)

Micropipelines with Processing

- The simple Micropipeline FIFO can be extended to include processing functions by the addition of logic interspersed between adjacent latch stages
- This operates in a similar manner to the empty FIFO with events rippling down the Micropipeline

![Processing Micropipeline](image)
Micropipelining of AMULET

- The processor is divided into pipeline stages
- Each gray box represents a pipeline latch

Register Lock without Arbiter

- High performance register bank is a key component of asynchronous RISC microprocessor
- Require register operations that allow concurrent read and write access along with arbitrary timing and dependencies between them
- Avoid metastability, so need a superb arbiter (or synchronizer)
- AMULET has a cool trick !!!
  - No arbiter
  - Novel register locking first-in-first-out queue
  - Enables efficient read operations in the presence of multiple pending writes
Register Bank Operation

- Decoder extracts the addresses of registers to be read (a, b), and the register where result will be written (w) from the op-code.
- The number of registers read by any instruction is limited by two output ports on the register bank.

![](image)

**ARM ALU Functions**

- Generically, the ALU set of functions can be decomposed into:
  - ‘Move’ operations
  - Basic logic functions (AND, OR, XOR)
  - Addition operation
    - (Along with the optional complementing or forcing-to-zero of one or both of the input buses)
- ALU can also perform functions that are implicit in other instructions:
  - Moving data from the ‘A’ bus to its output
  - Providing a zero value output
ARM ALU Data Processing Functions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>( \bar{W} ) Input</th>
<th>( \bar{W} ) Input</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AND</td>
<td>true</td>
<td>true</td>
<td>AND</td>
</tr>
<tr>
<td>1</td>
<td>EOR</td>
<td>true</td>
<td>true complement</td>
<td>EOR</td>
</tr>
<tr>
<td>2</td>
<td>SUB</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>3</td>
<td>RSB</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>4</td>
<td>ADD</td>
<td>true</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>5</td>
<td>ADC</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>6</td>
<td>SRC</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>7</td>
<td>RSC</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>8</td>
<td>AND</td>
<td>true complement</td>
<td>true complement</td>
<td>AND</td>
</tr>
<tr>
<td>9</td>
<td>XOR</td>
<td>true</td>
<td>true complement</td>
<td>XOR</td>
</tr>
<tr>
<td>A</td>
<td>CMP</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>B</td>
<td>CMN</td>
<td>true complement</td>
<td>true complement</td>
<td>ADD</td>
</tr>
<tr>
<td>C</td>
<td>ORR</td>
<td>true complement</td>
<td>true complement</td>
<td>OR</td>
</tr>
<tr>
<td>D</td>
<td>MVN</td>
<td>true complement</td>
<td>true complement</td>
<td>OR</td>
</tr>
<tr>
<td>E</td>
<td>MOV</td>
<td>true complement</td>
<td>true complement</td>
<td>OR</td>
</tr>
<tr>
<td>F</td>
<td>BIC</td>
<td>true complement</td>
<td>true complement</td>
<td>AND</td>
</tr>
<tr>
<td>8</td>
<td>MVN</td>
<td>true complement</td>
<td>true complement</td>
<td>OR</td>
</tr>
</tbody>
</table>

Note: Functions 8 to 11 do not produce a result and are only used to set the flags.

Bottleneck of Addition in Synchronous ALU

- For synchronous architecture, the overall performance of the ALU is pretty much limited by the addition operation -- it is the most time consuming.
- Speed of addition operation is related to how quickly the carry signals can propagate the across the word.
- WORST CASE addition paradox
  - The carry propagates across all bits in the word.
  - For synchronous design, the clock period is chosen to allow time for this worst case operation (usually result ready much sooner).
  - To reduce time for the worst case and in turn reduce the clock period, in synchronous design a lot of effort is expended in schemes like carry look-ahead and carry select.
  - These schemes require a large amount of circuitry to deal with the few pathological cases.
Addition in Asynchronous ALU

- Asynchronous ALU (without the constraint of external clock) allows design which is quick for “typical” operands and slower for “worst case” operands.
- The design strategy for asynchronous being:
  - Make the average case operation fast
  - Allow more time for calculation in the worst case
- Synchronous design is not able to allow more time for the worst case because the clock period is fixed.
- The AMULET designer found that the mean carry propagation distance for 32-bit addition is only 4.4 bits. This is clearly less than 32-bit worst case situation.

AMULET ALU Adder

- Addition function is performed by 32 full adders w/o any special acceleration logic to speed up carry propagation.
- The carry signal is encoded in dual-rail format:
  - A completion detection circuit signals to the environment when the carry propagation is complete.
- The asynchronous ALU of AMULET1 is about 40% of the area of its synchronous counterpart.
- The sparse, area-inefficient, carry-select logic can clearly be seen on the left-hand-side of the ARM6 ALU.
Area of ALUs

System Performance (Measured)

- Peak Native MIPS 79 MIPS (96 in Thumb code)
- 149 kDhrystones/s – 85 Dhrystone MIPS (ARM)
- 108 kDhrystones/s – 62 Dhrystone MIPS (Thumb) (-30%)
- AMULET3i power average 130 mW
  - 60% is within the processor core (simulation result)
- 660 MIPS/W for the system
  - 1100 MIPS/W for the processor core
About 15% slower than expected – awaiting silicon process information

For comparison:
- 0.35μm ARM9 ⇒ 120 MHz, (133 Dhrystone MIPS)
  800 MIPS/W

1. Dhrystone 2.1 benchmark (normalised to VAX MIPS)
Conclusion

- Asynchronous: Potential candidate for the next generation RISC microprocessors
  - Number of advantages over the synchronous RISC microprocessors
  - Need to get over many of existing problems
- Synchronous: Higher degree of performance innovation is required
  - Intensive on-going research for the next generation synchronous RISC processors

Quick References

- Fuber, VLSI RISC Architecture and Organization. Marcel Dekker, NY, 1999
- Fuber, A Micropipelined ARM. Proceedings of VLSI ’93, September 1993, best paper awarded
- Fuber, Four-Phase Micropipeline Latch Control Circuits. CS Dept at Univ. of Manchester, 1995
- Yun, High-Performance Asynchronous Pipeline Circuits, 1996
- Silc, Processor Architecture. Springer, 1999
- Info for Amulet processors: visit “www.cs.man.ac.uk/amulet”

These are informal references for students in ECE7102(Fall 2000) at Georgia Tech
The slides for the wireless interconnections are parts of the slides presented by Floyd at ISSCC 2000