Asynchronous RISC Processors

This Presentation is co-prepared by Hoon Lee & Simon Singh for Prof. Scott Wills ECE7102 RISC Arch.

Outline
- Asynchronous Approach (Hoon Lee)
- ARM RISC vis-à-vis AMULET (HL)
- Micro-pipeline (Simon Singh)
- Register Lock (SS)
- Self-timed ALU (SS)
- RF for On-chip Communication (HL)

** The oral talk was split as shown by the names above. The research and thinking was done jointly as a team.

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

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

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 Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Name</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMv1</td>
<td>Core</td>
<td>Features inherited from the Berkeley RISC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A load-store architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fixed length 32-bit instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-address instruction format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Features rejected from the Berkeley RISC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Register window</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delayed branches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Single cycle execution of all instructions</td>
</tr>
</tbody>
</table>

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

- **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

- **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

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>MBPS/W</td>
<td>77</td>
<td>285</td>
<td>780</td>
</tr>
</tbody>
</table>

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?

- 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

- 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
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.

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.

Register Lock FIFO Mechanism

- Destination register numbers are stored -- in decoded form-- in a FIFO.
- These are stored until the associated result is returned from the execution pipeline to the register bank.
- Each stage of FIFO holds a "1" in the position corresponding to the destination register.
- FIFO is shown in a state where:
  - 1st result to arrive written into r0
  - 2nd result to arrive written into r2
  - 3rd result to arrive written into r12
  - 4th FIFO stage is EMPTY.

Working of Register Lock FIFO

- When a subsequent instruction requests r12 as a source operand, the FIFO is inspected.
- Looking at column r12, the "1" signifies a pending write to r12 -- so its current contents are invalid.
- Read waits until the "1" is cleared, then it can proceed.
- Inspection is done in hardware using a logical "OR" function across the column for each register.
- Caution: The data in FIFO may move down the FIFO whilst the "OR" output is being used.
- Rescue: Micropipelines automatically copy data to the next stage before deleting it from the current stage.
Working of Register Lock FIFO

- In asynchronous FIFO, the data moves by being duplicated from one stage into the next, only then is it removed from the first stage.
- That is, a propagating "1" will appear alternately in one or two positions and it will never disappear completely.
- "OR" output will therefore be stable even though the data is moving.

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>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Move'</td>
<td>Moves data</td>
</tr>
<tr>
<td>Basic logic</td>
<td>Performs AND, OR, XOR</td>
</tr>
<tr>
<td>Addition</td>
<td>Performs addition</td>
</tr>
<tr>
<td>Complementing</td>
<td>Complements one or both of the inputs</td>
</tr>
<tr>
<td>Forcing-to-zero</td>
<td>Forcing one or both of the inputs to zero</td>
</tr>
</tbody>
</table>

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 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.
## Quick References

- Fuber, VLSI RISC Architecture and Organization, Marcel Dekker, NY, 1989
- Furer, A Monopipeline ALU, Proceedings of VLSI ’93, September 1993, best paper awarded
- Suts, System Timing, 5th Introduction to VLSI Systems, Chap 7, Addison-Wesley, 1986
- Sutherland, Microarchitecture, Communications of the ACM, 32(7):720-738, January 1989
- Furer, Four-Phase Micropipeline Latch Control Circuits, CS Dept at Univ. of Manchester, 1995
- SIC, Processor Architecture, Springer, 1999
- Feller, Wireless Interconnections in a VLSI IC with Integrated Antennas, ISCC 1996 Digest of Technical Papers, pp. 451-452
- 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 wireless interconnections are parts of the slides presented by Feller at ISCC 2000.

---

### Area of ALUs

**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

---