**Project Goals**

- Design a simple, but relatively complete, digital logic system in CMOS using a modern approach to logic cell design
- Estimate complexity, area, speed, power for a variety of possible technologies
  - From original to today
  - Including ND 2 micron
- Compare where possible to real implementations
- Serve as a possible basis for a second semester senior design project

**Project Approach**

- Select some “reference” implementation of some chip
  - Ideally very simple with known characteristics
  - Identify basic library of standard cells
- Estimate how many and where transistors are used in its design
- Project what versions in other technologies might look like: Area, Power, Max clock
  - In variety of technologies (esp. ND 2 micron & today)
- Perform a more detailed design
  - Either Verilog description or standard cell layout
- Analyze that design
- Prepare a report and make a class presentation
Reference Options

- See class website link page for pointers to documentation

<table>
<thead>
<tr>
<th>Microprocessor</th>
<th>Data Width (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6502</td>
<td>8</td>
</tr>
<tr>
<td>1802</td>
<td>8</td>
</tr>
<tr>
<td>8048/8051</td>
<td>8</td>
</tr>
<tr>
<td>8080</td>
<td>8</td>
</tr>
<tr>
<td>PDP-8</td>
<td>12</td>
</tr>
<tr>
<td>Simple 12</td>
<td>12</td>
</tr>
<tr>
<td>JAM-8</td>
<td>8</td>
</tr>
<tr>
<td>NOVA</td>
<td>16</td>
</tr>
<tr>
<td>MIPS</td>
<td>32</td>
</tr>
<tr>
<td>mini TPU</td>
<td>8</td>
</tr>
</tbody>
</table>

Presentation Outline

- Gather reference implementation characteristics
  - Technology, speed, power, area, transistor count, …
- Overview of instruction set (if processor)
- Overview of microarchitecture and how data flows
- What “off-chip” connections needed (include power, clocks, reset)
- Outline major blocks (and estimate area)
- Transistor estimate for each
  - Where appropriate show transistor diagrams of block
- Project ahead to smaller feature sizes
  - Estimate area, power, speed, …
- Do projection in 2 steps
  - Include stop at 2 microns (ND process)
  - Using Dennard scaling (until 2004 technology)
  - Using constant voltage scaling
### Options

- **Design options**
  - Verilog of complete processor (dataflow & control)
  - Electric layout of a dataflow slice
  - Electric layout of control logic using standard cell
- **One person:** one of first two above
- **Two persons:** Verilog and dataflow slice
- **Three persons:** All three
- **Permitted simplifications** (review with instructor)
  - For a microprocessor: Simplified instruction set
  - “Low speed” data flow (i.e. a simple ripple adder)
  - Don’t worry about off-chip pads/drivers

### Standardizing Libraries OK

- **Class development of std cell & bit slice library** encouraged
- **Tuesday Oct. 23rd:**
  - Bring your understanding of your chip
  - Identify common cells
  - Each student selects subset of cells for implementation
- **Later in semester**
  - Agree on
    - standard height (for standard cells)
    - standard width (for bit slices)
    - Standard ports
  - Revise Electric implementation & share
Example: Simple 12

Top Level Simple12

Opcode (4) | Address (8)

256 words of 12 bits each.
Addresses all 8 bits wide.
Opcode selects one of 16 instructions from 4 Classes:
- Jump: change PC
- Load/Store: access memory
- Operate: compute new value
- Reserved: (for your later enjoyment)

Programmer Visible Registers:
A: Accumulator - Holds 12 data bits
PC: Program Counter - Holds 8 bits of current instruction address
Simple12 ISA

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>STOP</td>
<td>Stop execution until reset</td>
</tr>
<tr>
<td>0001</td>
<td>JMP X</td>
<td>PC&lt;X</td>
</tr>
<tr>
<td>0010</td>
<td>JN X</td>
<td>if A&lt;0 then PC&lt;X else PC++</td>
</tr>
<tr>
<td>0011</td>
<td>JZ X</td>
<td>if A=0 then PC&lt;X else PC++</td>
</tr>
<tr>
<td>0100</td>
<td>LOAD X</td>
<td>A&lt;-M(X)</td>
</tr>
<tr>
<td>0101</td>
<td>STORE X</td>
<td>M(X)&lt;-A</td>
</tr>
<tr>
<td>0110</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>AND X</td>
<td>A&lt;- A and M(X)</td>
</tr>
<tr>
<td>1001</td>
<td>OR X</td>
<td>A&lt;- A or M(X)</td>
</tr>
<tr>
<td>1010</td>
<td>ADD X</td>
<td>A&lt;- A + M(X)</td>
</tr>
<tr>
<td>1011</td>
<td>SUB X</td>
<td>A&lt;- A - M(X)</td>
</tr>
<tr>
<td>1100</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>reserved</td>
<td></td>
</tr>
</tbody>
</table>

A Simple Multi-Cycle Data Flow

Diagram showing the data flow and components of the Simple12 ISA.
Dataflow that does AND/OR/SUB

Possible Optimizations:
- Combine right side muxes
- AND and OR from Adder for logic functions
- Remove Left-side AND (we have a 4th input on the ALU Mux)

A Matching Execution Cycle

- CLEARPC: PC := 0, MAR := 0
- IFETCH: IR := Mem[MAR], PC := PC + 1
- DECODE: if JMP or (JZ&A==0) or (JN&A[11]) then PC := MAR[7:0] else MAR := MDR[7:0]
- LOAD1: MDR := Mem[MAR]
- ADD1: MDR := Mem[MAR]
- STORE1: Mem[MAR] := A

Similar for AND, OR, SUB
Simple 12

- Standard cells
  - state machine & timing
  - common height
  - common power
  - ports to wiring bays below

- Bit Slices
  - data flow
  - common width
  - common power
  - wiring up & down in slice
  - wiring horizontal for cross slice

- Decoder logic
  - drive row in dataflow
  - “random” logic

- I/O Cells
  - standard

Simple12 Bit Slice

- A Reg[i]
- Left in AND[i]
- ZeroDetect[i]
- INV[i]
- AND[i]
- OR[i]
- FullAdd[i]
- Mux2[i]
- Mux4[i]
- ALU
- 4 way 4 zeroes
- 2 way 6 zeroes
- 6 way 2 zeroes
- 17 way 17 zeroes
- write_data[i]
- memdata[i]
- memdata[i]
- memdata[i]
Goal: represent large numeric range in small # of bits

- Typical scientific number: +/-1.xxx \times 2^e
  - Exponent e has some limited range
  - Mantissa has some fixed precision (# of bits)

- Today’s floating point formats

<table>
<thead>
<tr>
<th>Sign bits</th>
<th>Exponent bits</th>
<th>Mantissa bits</th>
<th>Numeric Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>32b</td>
<td>1</td>
<td>8</td>
<td>10^{-38} to 10^{+38}</td>
</tr>
<tr>
<td>64b</td>
<td>1</td>
<td>11</td>
<td>10^{-307} to 10^{+307}</td>
</tr>
<tr>
<td>128b</td>
<td>1</td>
<td>15</td>
<td>10^{-4914} to 10^{+4914}</td>
</tr>
</tbody>
</table>

- New apps (ML, AI) need much less precision
  - but small ints have insufficient range
- New ML, AI processors moving to 8 & 16b floating point
**Key ML/AI Operation:**

Vector Inner Product

- Stream of observations
- Stream of Weights

![Diagram](image)

For same stream of observation data, find set of weights that maximizes the inner product.

---

**AI Accelerators**

First Generation Google Tensor Processing Unit Chip:
- H/W Dense Matrix-Vector Product
- Peak 92,000 G flops/s (8 bit floats)

![TPU Chip](image)

4 TPU2’s per card

---

[Additional content related to AI accelerators and TPU chip is included in the image.]

---

9
Possible Project:
one of these but in 8-bit floating point

Floorplan of TPU Die

- The Unified Buffer is almost a third of the die
- Matrix Multiply Unit is a quarter
- Control is just 2%
F8: An 8-bit Float Format

- **S** Sign: 0=>"Positive": 1=>"Negative"
- **E3:E0** Exponent: a 4-bit uint "e"
- **M2:M0** Mantissa: 3 bit “fraction"
- Special Cases
  - X00000000 = “Zero”
  - 01111000 = Positive infinity
  - 11111000 = Negative infinity
  - X1111xyz (xyz!=000) = NaN i.e. “Not a Number”
- “Denormalized” case: S0000xyz: +/-0.xyz*2^b
- Normal Case: Sefghxyz: +/-1.xyz*2^e+b-1
  - “b” is exponent “bias”
  - Note normalized has a leading “1” added

### Assuming Bias = 3

<table>
<thead>
<tr>
<th>S</th>
<th>E3</th>
<th>E2</th>
<th>E1</th>
<th>E0</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
<th>Value (All values are Ints)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+0.001*2^3 = 1 (smallest + non-zero)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-0.001*2^3 = -1 (smallest - non-zero)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+0.111*2^3 = 7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+/- 1.000*2^1+3-1 = 8 (least normalized)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+/- 1.000*2^1+3-1 = 9</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+/- 1.111<em>2^1+3+3-1 = 1512</em>2^13 = 122280 (largest normalized)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NaN</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Other biases provide rational numbers at reduced range
E.g. if bias = -14, range is from 0.000053 to 1.875
**Advantage of This Bias**

Stream of 8b float observations → Accumulator

Stream of 8b float Weights → Multiplier

Multiplier

Need at most 4b x 4b multiplier Left shift only

**Notional F8 MAC**

S1 E1 M1

S2 E2 M2

4b x 4b Multiplier

5b Adder

XOR

Shift Left up to 6 places

11b XOR

If sign is negative, make carry in to adder a 1

2’s complement int to adder
Possible Group Project

- Extend ISA of simple core with F8 MAC instructions
- Develop sample code to do simple AI/ML operations
- Develop Verilog design of:
  - Core
  - F8 Accelerator
  - and combine
- Develop layout (in Electric) of
  - Core
  - F8 Accelerator
  - and combine
- Size and project