CSE140: Components and Design Techniques for Digital Systems

High-Level Finite State Machines: Examples

Tajana Simunic Rosing
Class Status, Updates & Plans

• **What we covered:**
  – Combinational logic
  – Latches, FFs
  – Registers, counters, FSMs & timing
  – HLSM & timing

• **Plan for today:**
  – HLSM examples, memories

• **Upcoming deadlines:**
  – HW#6 due
  – HW#7 assigned, due next Thursday
    • The last HW for the quarter!!!!

• **Piazza posts**
  – Posting anonymous angry messages is not allowed in this class
    • If there are concerns you have, come talk to me or email me
## RTL Design Process

<table>
<thead>
<tr>
<th>Step 1: Capture behavior</th>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Capture a high-level state machine</strong></td>
<td>Describe the system’s desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is “high-level” because the transition conditions and the state actions are more than just Boolean operations on single-bit inputs and outputs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2: Convert to circuit</th>
<th>2A</th>
<th>Create a datapath to carry out the data operations of the high-level state machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2B</td>
<td>Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block.</td>
</tr>
<tr>
<td></td>
<td>2C</td>
<td>Derive the controller’s FSM for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Simple data encryption/decryption device

- B = 1, set offset O = I [0:31]
- B = 0 e = 1: encrypt mode: output J = I + O
- B = 0 e = 0: decrypt mode: get I = J - O

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Reaction timer circuit

- Goal: measure reaction time
  - Inputs: button B, reset rst
  - Outputs: light enable Len, reaction time rtime, slow bit (user took longer than 2000sec)
  - Clock 1kHz -> 1ms period

Is this HLSM:
- a) Control dominant
- b) Data dominant
Sum of 64-bit registers

- Input: 64-bit registers R[0...15]
- Output: sum

Is this HLSM:
- a) Control dominant
- b) Data dominant
Hot Water Detector

Create an alarm system that sets alarm=1 when the average temperature of four consecutive samples CT meets or exceeds a threshold WT. Signal clr=1 disables the alarm.

Initial Condition:
- $t_0 = t_1 = t_2 = t_3 = CT$

Sample:
- $temp_0 = CT$
- $temp_1 = t_0$
- $temp_2 = t_1$
- $temp_3 = t_2$

Average:
- $avg = \frac{\sum_{i=0}^{3} temp_i}{4}$

Comparison:
- $avg < WT \Rightarrow alarm = 1$

$clr = 1$ disables the alarm.
Inputs: byte a, byte b, bit go
Outputs: byte gcd, bit done
GCD:
while(1) {
    while(!go);
    done = 0;
    while ( a != b ) {
        if( a > b ) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    gcd = a;
    done = 1;
}
Finish HLSM Design

- Design an 8-bit counter using RTL:
  - When input $E = 1$, it counts even numbers (0, 2, 4, 6, ..) and when $E = 0$, it counts odd numbers (1, 3, 5, 7, ..).
  - When input $CLR = 1$ and $E=1$, then it clears the output to 0; if $CLR=1$ and $E=0$, it sets output to “00000001”.
  - If you were initially counting even(odd) numbers, and $E$ flips, then the output changes to the nearest greater odd (even) value.
CSE140: Components and Design Techniques for Digital Systems

Memory

Tajana Simunic Rosing
Memory: basic concepts

- Stores large number of bits
  - \( m \times n \): \( m \) words of \( n \) bits each
  - \( k = \log_2(m) \) address input signals
  - or \( m = 2^k \) words
  - e.g., 4,096 x 8 memory:
    - 32,768 bits
    - 12 address input signals
    - 8 input/output data signals

- Memory access
  - \( r/w \): selects read or write
  - enable: read or write only when asserted
  - multiport: multiple accesses to different locations simultaneously
Composing Memory – Wider Words

- Making memory words wider
  - Easy – just place memories side-by-side until desired width obtained
  - Share address/control lines, concatenate data lines
  - Example: Compose 1024x8 ROMs into 1024x32 ROM
Composing Memory – More Words

- Creating memory with more words
  - Combine memories until the number of desired words is achieved
  - Use decoder to select
  - Example: Compose 1024x8 memories into 2048x8 memory
- More words and wider words – first make enough words, then widen

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Write ability / storage permanence

- Traditional ROM/RAM
  - ROM
    - read only, bits stored without power
  - RAM
    - read and write, lose stored bits without power
- Distinctions blurred
  - Advanced ROMs can be written to
    - e.g., EEPROM
  - Advanced RAMs can hold bits without power
    - e.g., NVRAM

Write ability and storage permanence of memories, showing relative degrees along each axis (not to scale).
Comparing RAM

- **Register file**
  - Fastest
  - But biggest size

- **SRAM**
  - Fast (e.g. 10ns)
  - More compact than register file

- **DRAM**
  - Slowest (e.g. 20ns)
  - And refreshing takes time
  - But very compact
  - Different technology for large caps.
Caches and CPUs

- Servers: Level 1 (L1), L2 & L3 cache on chip
- Embedded: L1, L2 on chip

Intel Xeon Server

Raspberry Pi 3

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Raspberry Pi 3 – Memory Architecture

- **Broadcom BCM2837 SoC**
  - CPU: Quad-core Cortex-A53: L1 and L2 cache
  - GPU: VideoCore IV® Processor: exclusive memory system
  - Main Memory: 1GB RAM : Shared by CPU and GPU
ARM v8 memory hierarchy (in RPi3)

Tightly coupled memory (TCM) = SRAM organized as main memory
Random Access Memory (RAM)

- RAM – Readable and writable memory
  - Logically the same as register file
    - RAM just one port; register file two or more
  - RAM vs. register file
    - RAM is larger
    - RAM stores bits using a bit storage vs. FFs
    - RAM implemented on a chip in a square – keeps longest wires (hence delay) short
• Similar internal structure as register file
  – Decoder enables appropriate word based on address inputs
  – \( rw \) controls whether cell is written or read
  – Let’s see what’s inside each RAM cell
**Static RAM (SRAM) - writing**

- "Static" RAM cell
  - 6 transistors (recall inverter is 2 transistors)
  - Writing this cell
    - *word enable* input comes from decoder
    - When 0, value *d* loops around inverters
      - That loop is where a bit stays stored
    - When 1, the *data* bit value enters the loop
      - *data* is the bit to be stored in this cell
      - *data’* enters on other side
      - Example shows a “1” being written into cell

---

**SRAM cell**

- Diagram of SRAM cell with data and *data’* inputs, showing how data is written.

**Sources:** TSR, Katz, Boriello, Vahid, Perkowski
Static RAM (SRAM) - reading

- “Static” RAM cell - reading
  - When rw set to read, the RAM logic sets both data and data’ to 1
  - The stored bit d will pull either the left line or the right bit down slightly below 1
  - “Sense amplifiers” detect which side is slightly pulled down
Dynamic RAM (DRAM)

- **“Dynamic” RAM cell**
  - 1 transistor (rather than 6)
  - Relies on *large* capacitor to store bit
  - Write: Transistor conducts, data voltage level gets stored on top plate of capacitor
  - Read: Just look at value of \( d \)
  - Problem: Capacitor discharges over time
    - Must “refresh” regularly, by reading \( d \) and then writing it right back
Comparing Memory Types

- Register file
  - Fastest
  - But biggest size
- SRAM
  - Fast
  - More compact than register file
- DRAM
  - Slowest
  - And refreshing takes time
  - But very compact
  - Different technology for large caps.

Sources: TSR, Katz, Boriello, Vahid, Perkowski
RAM Example: Digital Sound Recorder

- **Behavior**
  - Record: Digitize sound, store as series of 4096 12-bit digital values in RAM
    - We’ll use a 4096x16 RAM (12-bit wide RAM not common)
  - Play back later from RAM
Queues

- **FIFO Queue** (first-in-first-out)
  - Write at the back: \textit{push}, Read at the front: \textit{pop}
  - Treat memory as a circle

- **Common uses**:
  - Computer keyboard
    - Pushes pressed keys onto queue; Meanwhile pop and send to computer
  - Digital video recorder
    - Pushes frames onto queue; Meanwhile pops frames, compresses them, and stores them
  - Routers
    - Pushes incoming packets onto queue; Meanwhile pops packets, processes destination information, and forwards each packet out over appropriate port
Queues

- Two conditions have front= rear need FSM to detect:
  - Full: No pushes until a pop
  - Empty: No pops until a push
- Use Register file for storage
- Implement Rear and front with up counters:
  - rear as RF’s write address, front as read address
HLSM with memory example: Fibonacci Lookup Table

- Design a lookup table 256 x 256 bit that stores Fibonacci #s:
  - $F_n = 0$ if $n = 0$, 1 if $n = 1$, $F_{n-1} + F_{n-2}$ otherwise ($n < 256$)
Non-volatile memory yesterday and today

- **Erasable Programmable ROM (EPROM)**
  - Uses “floating-gate transistor” in each cell
  - Programmer uses higher-than-normal voltage so electrons *tunnel* into the gate
    - Electrons become trapped in the gate
    - Only done for cells that should store 0
    - Other cells will be 1
  - To erase, shine ultraviolet light onto chip
    - Gives trapped electrons energy to escape
    - Requires chip package to have window

- **Electronically-Erasable Programmable ROM (EEPROM)**
  - Programming similar to EPROM
  - Erasing one word at a time *electronically*

- **Flash memory**
  - Like EEPROM, but large blocks of words can be erased *simultaneously*

- **EEPROM & FLASH are in-system programmable**
ROM Example: Digital Telephone Answering Machine

- Record the **outgoing announcement**
  - When \( \text{rec}=1 \), record digitized sound in locations 0 to 4095
  - When \( \text{play}=1 \), play those stored sounds to digital-to-analog converter
ROM Example: Digital Telephone Answering Machine

- High-level state machine
  - Once rec=1, begin erasing flash by setting er=1
  - Wait for flash to finish erasing by waiting for bu=0
  - Execute loop that sets local register a from 0 to 4095, reading analog-to-digital converter and writing to flash for each a
Non-volatile memory going forward

• A new class of data storage/memory devices
• Emerging NVMs have exciting features:
  – Non-volatile like Flash (~ 10 years)
  – Fast access times (~ SRAM)
  – High density (~ DRAM)
• NVM *blurs the distinction* between
  – Memory (*fast, expensive, volatile*) &
  – Storage (*slow, cheap, non-volatile*)
• Key issues:
  – Slow writes, low endurance, costly and complex manufacturing

Sources: TSR, Katz, Boriello, Vahid, Perkowski
1T-1C FeRAM

- Similar in construction to DRAM
  - Both cell types include one capacitor and one access transistor
  - DRAM cell capacitor use a linear dielectric; FeRAM includes ferroelectric material, typically lead zirconate titanate (PZT)
  - Writing is accomplished by applying a field across the ferroelectric layer by charging the plates on either side of it, forcing the atoms inside into the "up"/logic “1” or "down"/logic "0"

- Advantage:
  - No need for refresh – 99% lower power than DRAM
  - Similar performance to DRAM

- Disadvantage:
  - It is unclear how it will scale as materials stop being ferroelectric at small sizes (now produced in 130nm)
  - Reads are destructive – data has to be rewritten

Sources: TSR, Katz, Boriello, Vahid, Perkowski
STT-RAM: Spin-Transfer Torque RAM

- The spin torque direction of electrons to flip a bit in a magnetic tunneling junction (MTJ)

- **Advantages:**
  - High endurance & fast reads

- **Disadvantages:**
  - *Write energy:* Large current needed to reorient the magnetization for most commercial applications.
  - *Asymmetric write:* Writing a “1” needs much more time and energy than writing a “0”
Domain Wall Memory (DWM)

- Similar to STT-RAM structure
- **Advantage:**
  - needs only one tunneling barrier and fixed layer → area savings
- **Disadvantages:**
  - complexity of design, read/write delay due to sequential access

![Domain Wall Memory Diagram]
Shift-based DWM

- Writes by shifting data of one of the two fixed layers with the desirable direction compass
- **Advantage**: faster writes than a traditional DWM
- **Disadvantage**: cost and manufacturing complexity

1-bit DWM is fast

Multi-bit DWM is area efficient, but needs extra latency for shifting
PCM: Phase Change Memory

Phase Change Memory (PCM)
- Flips a bit by changing the state of material
- Crystalline (SET) and amorphous (RESET) phase

Advantages:
- better scalability than other emerging technologies

Disadvantages:
- Slow writes
- Low endurance (10^7 writes)

Candidate for DRAM replacement
ReRAM: Resistive RAM

- Two types: Access-based (1T-1R) and crossbar ReRAM (1T-nR)

- **Access-based ReRAM (1T-1R)**
  - A dielectric, which is normally an insulator, can conduct with sufficiently high voltage

- **Advantage:**
  - Very fast reads and writes ~ 20ns
  - Very high density

- **Disadvantage:**
  - Limited endurance ($10^5$ writes)
Crossbar ReRAM

- Crossbar ReRAM (1T-nR)
- **Advantage:**
  - Highly scalable
  - Can be implemented at the top of the chip with in 3D architecture
  - Very low energy consumption
  - Low cost (possible replacement for Flash)
- **Disadvantage:**
  - Much slower than 1T-1R ~us
Crossbar RRAM Applications

**Crossbar 1T-nR**

Data Center – Block Oriented Data Storage

- Low latency ($10^{-5}$ s), high bandwidth, low energy
- Native 3D compatible array architecture
- Alternatives: SSD, DRAM (disk cache), hard disk

**1T-1R**

Nodes – XiP Code and Random Access Data

- Low latency ($10^{-8}$ s), high bandwidth, low energy
- Alternatives: embedded NOR, discrete NOR

NVRAM Comparison

Sources: TSR, Katz, Boriello, Vahid, Perkowski
## NVMs Comparison

- **STT-RAM**: SRAM *cache* replacement
- **PCRAM**: DRAM *main memory and storage*
- **ReRAM**: **NAND** flash, embedded **NOR** flash

<table>
<thead>
<tr>
<th>Features</th>
<th>SRAM</th>
<th>eDRAM</th>
<th>STT-RAM</th>
<th>PCRAM</th>
<th>ReRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Very high</td>
<td>Very high</td>
</tr>
<tr>
<td>Speed</td>
<td>Very Fast</td>
<td>Fast</td>
<td>Fast for read; slow for write</td>
<td>Slow for read; very slow for write</td>
<td>Slow for read/write</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>Low</td>
<td>Medium</td>
<td>Low for read; very high for write</td>
<td>Medium for read; high for write</td>
<td>Medium for read; high for write</td>
</tr>
<tr>
<td>Leakage Power</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Non-volatility</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Summary

• Memory hierarchy
  – Needs: speed, low power, predictable
• Cache design
  – Mapping, replacement & write policies
• Memory types
  – ROM vs RAM vs NVM
• NVM
  – Many new technologies that are still maturing
  – Excellent target for big data and energy-efficient applications