ESE250: Digital Audio Basics

Day 8: November 3, 2009
Hardware
ESE250: Week 8

- What hardware is required for an MP3 player?
iPod Shuffle

- Battery
  - Battery regulation
- Headphone jack
  - also control, USB
- Integrated Stack
  - Processor SoC
  - Flash RAM (2GB)
  - DRAM

http://www.ifixit.com/Teardown/iPod-Shuffle-3rd-Generation/673/1
iPod Processor

- Early based on PortalPlayer series
  - Two ARM7TDMI cores
  - 80MHz each
- Guesses are ARM7 or ARM8
Course Map

Numbers correspond to course weeks

Today: hardware
Outline

• Teaser: Shuffle teardown
• What’s required for an MP3 player
• Notional Platform
• Interlude
• “Stored-Program” Computer (Processor)
Analog→Digital Conversion

• **Sound**: pressure waves
• **Microphone**: converts physical displacement to analog voltage
• **Next challenge**: convert analog voltage to digital
Flash Analog $\rightarrow$ Digital Converter

ESE215 knows all about OpAmps.
- ESE 250 “assumes” them
- Future 250: spend 20 min on them?
Store Digital Samples

• Need to collect up set of amplitude samples
  – Convert to frequency coefficients
  – Decide which coefficients to keep
  – …and how to quantize
Random Access Memory

- A Memory:
  - Series of locations
  - Can write values into
  - Read values from
  - Return last value written
Two Pieces of a Memory

1. Element to remember a value
2. Way to address/select that element
Switch $\Rightarrow$ Storage Element

- Multiplexor (mux)
  - switch: $o = (i_1 \cdot l^c) + (i_2 \cdot l)$
  - serve as a storage element?

- Feedback
  - load=0
    - out=out
    - holds value
  - load=1
    - out=in
    - loads in new value
Could build Memory w/ Muxes

\[
\begin{align*}
  w_3 &= \text{and} \ a_0 \ a_1 \ \text{(not read)} \\
  w_2 &= \text{and} \ \text{(not} \ a_0) \ a_1 \ \text{(not read)} \\
  w_1 &= \text{and} \ a_0 \ \text{(not} \ a_1) \ \text{(not read)} \\
  w_0 &= \text{and} \ \text{(not} \ a_0) \ \text{(not} \ a_1) \ \text{(not read)}
\end{align*}
\]
Random Access Memory (RAM) with Capacitor Memories

Decoder

Read

Address

Din

Dout

Learn more: CE Circuits or ESE570
Persistent Storage

• Need to store MP3 for later playback
• Like memory
  – We can read and write to it
  – …. but shouldn’t consume energy to preserve data
• Non-volatile – data doesn’t go away
FLASH Memory (new “EEPROM”)

- Exploit tunneling
- Use high-voltage to reduce barrier
  - Tunnel charge onto floating node
  - Charge trapped on node
- Use field from floating node to modulate conduction

[Image: Flash Programming.png]
Digital → Analog Conversion

- Convert from digital amplitude samples back to voltage
- **Speaker (headphones):** voltage to drive speaker displacement
- Use bits to regenerate analog voltage

\[
\begin{align*}
V/2 & \rightarrow d[2] \\
V/4 & \rightarrow d[1] \\
V/8 & \rightarrow d[0]
\end{align*}
\]
Digital → Analog Conversion

User I/O

• **Input**: User needs to select song, dial, etc.
  – Switch
  – Collection of switches:
    • Keyboard, joystick, touchscreen

• **Outputs**: User may need to “see” what’s happening (stored, where in menus, options)
  – LCD display, LED
  – VoiceOver on shuffle
Audio Platform

- A-to-D
- LCD
- Keyboard
- D-to-A
- Memory
- Non-volatile Memory
Audio Platform

- Still need to:
  - Provide computation
    - (Fourier, Search, Quant, lossless code)
  - Route data
Audio Platform

A-to-D

<table>
<thead>
<tr>
<th>LCD</th>
<th>Keyboard</th>
</tr>
</thead>
</table>

Memory

Processor

Non-volatile Memory

D-to-A
Interlude

Moore’s Law
“Moore’s Law”

- The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue….


“Moore’s Law” 1965

“Moore’s Law” Today

- Geometric growth in Integrated Circuit (IC) capacity
- Driven by a geometric shrink in transistor feature size
MOS Transistor Scaling (1974 to present)

S = 0.7

[0.5x per 2 nodes]

Source: 2001 ITRS - Exec. Summary, ORTC Figure

[from Andrew Kahng]
Half Pitch (= Pitch/2) Definition

Source: 2001 ITRS - Exec. Summary, ORTC Figure

[from Andrew Kahng]
Moore’s Challenge

• Not a law of nature
• But an engineering challenge
  – Can we keep making things smaller?
• ITRS: International Technology Roadmap for Semiconductors
  – Joint industry effort to chart scaling
  – …and identify challenges
Not limited to Silicon?

• Version from Ray Kurzweil

Will This Last Forever?

[Moore, ISSCC2003]
“Stored Program” Processor
“Stored Program”
Computer/Processor

• Can build physical substrates which can be *programmed* to perform any computation.
• Can be built with limited hardware that is *reused* in time.
• **Historically:** this was a key contribution from Penn’s Moore School
  – Computer Engineers: Eckert and Mauchly
  – ENIAC $\rightarrow$ EDVAC
  – (often credited to Von Neumann)
Basic Idea

- Express computation in terms of a few primitives
  - E.g. Add, Multiply, OR, AND, NAND
- Provide one of each hardware primitive
- Store intermediates in memory
- Sequence operations on hardware to perform larger computation
- Store *description* of operation sequence in memory as well – hence “Stored Program”
- By filling in memory, can program to perform *any* computation
Express Computation

- Familiar with expressing in terms of a set of primitives
  - E.g. Java operators
  - Add, subtract, multiply, divide, shift

```java
k4 = k2 << 1;
fi = fz;

while (fi < fn)
{
  FLOAT f0, f1, f2, f3;
  f1 = fi[0] - fi[k1];
  f0 = fi[0] + fi[k1];
  f3 = fi[k2] - fi[k3];
  f2 = fi[k2] + fi[k3];

  fi[k2] = f0 - f2;
  fi[0] = f0 + f2;
  fi[k3] = f1 - f3;
  fi[k1] = f1 + f3;

  gi += k4;
  fi += k4;
}
```
Decompose Universal Primitives

• All of these operators can actually be decomposed into smaller primitives
  – Multiply $\rightarrow$ sequence of Adds, ANDs
  – Add $\rightarrow$ collection of XORS, ANDs, inverts
  – XOR, AND, invert $\rightarrow$ can be expressed in terms of 2-input NANDS

• NAND universality
  – Can compute any boolean function out of a suitable collection of nand gates (ESE170)

Query: who’s had 170?
Arithmetic and Logic Unit (ALU)

• A particularly primitive logic is the ALU
  – Can perform any of a number of operations on a series of words (strings of bits)
  – **Operations**: Add, subtract, shift-left, shift-right, xor, and, or, invert, ….
  – Operates on “words”

• Identify a set of control bits that select the operation it forms
  – Makes it “programmable”
Computer Word

• Computer **word** is a sequence of bits treated together
  – Typically power of two
    • 8, 16, 32, 64…
  – Addressable unit on a computer often in **words**
ALU Ops (on 8b words)

• ADD  00011000 00010100  = 00101100
  – Add 0x18 to 0x14   result is: 0x2c
  – Add 24 to 20      result is: 44

• SUB  00011000 00010100  = 00000100
  – Subtract 0x14 from 0x18  …result is: 0x4

• INV   00011000 XXXXXXXXXX  = 11100111
  – Invert the bits in 0x18   …gives us 0xE7

• SLL  00011000 XXXXXXXXXX  = 00110000
  – Shift left 0x18   … gives us 0x30
ALU
ALU Encoding

- Each operation has some bit sequence
  - ADD 0000
  - SUB 0010
  - INV 0001
  - SLL 1110
  - SLR 1100
  - AND 1000
Programming

• If I can give this piece of logic (ALU) data, including the operation
  – It will perform various operations

What operation is this performing?
Subtract 0x24 0x20
Only Need One

• By reusing this “universal” operator in time, can perform any computation

• **Extreme:** “universal” operator could be a 2-input NAND gate
A Simple Programmable Device
Add Value in slot 0 to Value in slot 1 and put in slot 2
(N.B. store to both memories)
Programming an Operation

• Consider:
  ▪ $C = (A+2B) \& 00001111$

• Cannot do this all at once
  • But can do it in pieces
Programming an Operation

• Consider: $C = (A+2B) \& 00001111$

  ▪ Find a place for $A$, $B$, $C$
    • $A$ – slot 0
    • $B$ – slot 1
    • $C$ – slot 7
    • $00001111$ – slot 4
Programming an Operation

- Consider: $C = (A+2B) \& 00001111$
- Decompose into pieces
  - Compute $2B$
  - Add $A$ and $2B$
  - AND sum with mask
    - $(00001111$ is mask)
Programming an Operation

- Decompose into pieces
  - Compute 2B: 0000 1 001 001 010
  - Add A and 2B: 0000 1 000 010 011
  - AND sum with mask: 1000 1 011 100 111
Controlling the Operation

• Need a sequence of steps to perform this task

• How do we feed controls to this assembly to make it perform the operations we want?
“Stored Program” Control

• Add another Memory to hold “operation”
  – Call them Instructions
  – Call the memory Instruction Memory
Instruction Control

• How know which Instruction to use?
  – Add a counter to sequence through operations
“Stored Program” Processor
iPod Processor

- Compare ARM7
Basic Idea

- Express computation in terms of a few primitives
  - E.g. Add, Multiply, OR, AND
- Provide one of each hardware primitive
- Store intermediates in memory
- Sequence operations on hardware to perform larger computation
- Store *description* of operation sequence in memory as well – “Stored Program”
- By filling in memory, can program to perform *any* computation
Frequency Challenge

• Can reuse in time
  – Modest hardware can solve “Large” problem
  – Large = many operations to perform
  – Large problems $\rightarrow$ more cycles $\rightarrow$ more time

• To solve in fixed/bounded time
  – *E.g.* decode music producing 44K samples/s
  – Must run fast enough $\rightarrow$ high enough cycle frequency

• Lab
  – Explore how many ops needed for MP3 encode/decode
  – How that relates to Moore’s Law scaling of processor performance
You Were Here
Learn More

• Online reading/pointers
  – Teardowns, datasheets, scaling

• Courses
  – CIS240 – introduction to computer org.
  – CIS371 – computer architecture
  – ESE534 – computer organization
  – ESE200/201 – digital logic
  – ESE205/215 – analog circuits
Big Ideas

• Can build machine to store and process audio
  – Requirements are quite modest
• Bits represent computation
  – Compare: bits represent sound
• Universal “Stored Program” Machines
  – Can build a physical machine
  – Perform any computation
  – By interpreting the bits