Lecture 9

DSP MicroProcessor for Communication

Fall, 2004
Two kinds of Microprocessor

- **General Purpose Microprocessor**
  - Example: PC/Workstation processor: Pentium, Itanium, Athelon, SPARC; Server processor (UltraSPARC, Power4, Pentium)
  - Data Manipulation, word processing, database management
    (Ex: A -> B, If A == B, then ....)
  - Off line processing
  - Usually bursty

- **Special Purpose Microprocessor**
  - Digital Signal Processor
    - Speech signal processing, video/image processing, spread spectrum processing, modulation/demod, coding/decoding, convolution
    - Stream signal based computation, math calculation, scientific and engineering simulation (Ex: A+B=C, AxB=C, digital filters, etc)
    - Real Time constraint
    - No advantage for higher speed when real-time constraint is satisfied
    - Usually continuous, correlated data stream, and parallel processing on data stream required
DSP uP and Embedded is increasing used everywhere

- Dedicated ASIC not always the best choice
- Flexibility
- Portability
- Maintainability
- Reusability
Von Neumann Architecture

- Single memory space for both instruction and data
- Single data bus to transfer data in-and-out of the processor core
- Inefficient for memory intensive operations
- Example: ARM7

Note: John Von Neumann (1903-1957) developed the concepts of stored program computer, formalizing the mathematics of quantum mechanics and work on the atomic bomb.
Harvard Architecture

- Developed at Harvard Univ. in 1940's, led by Howard Aiken (1900-1973)
- Separate program and data memory
- Example: ARM9
- Dual buses for independent instruction and data fetch
- Potential problem: single instruction multiple data stream (SIMD) applications.
  Ex: When 2 numbers are multiplied, 2 data to pass from data memory and only 1 data (the instruction) to pass from program memory
Instruction cache in processor core
- Very often, the same set of program instructions are used repeatedly.
- May store coefficient in program memory (as secondary data)
- I/O controller: direct memory access (DMA)
TI Modified Harvard Architecture

- Separate program and data memory
- Enables parallel memory access (improves w/ DARAM)
- May store coefficients in program memory (ROM)
Typical DSP Architecture
TMS320 Digital Signal Processor Family
TMS322C54x DSP processor block diagram

Specifications
- 16-bit fixed-point DSPs
- Power dissipation as low as 60 mW for 100 MIPS
- Single-/multi-core, delivering 30–532 MIPS performance
- Multi-channel buffered serial port
- Host port interface
- 6-channel DMA controller per core

Features
- Integrated Viterbi accelerator
- 40-bit adder and two 40-bit accumulators to support parallel instructions
- 40-bit ALU with a dual 16-bit configuration capability for dual one-cycle operations
- $17 \times 17$ multiplier allowing 16-bit signed or unsigned multiplication
- Four internal buses and dual address generators enable multiple program and data fetches and reduce memory bottleneck
- Eight auxiliary registers and a software stack enable advanced fixed-point DSP C compiler

National Tsing Hua University, COM 5195: 通訊系統晶片設計
TMS322C54x DSP processor architecture

- Memory Access
  - 4 Internal Bus Pairs
  - 8 Auxiliary Registers (AR0-AR7)
- Address Generation
  - Circ. Buffers
  - Inc/Dec.
- Number Crunching
  - 40 bit Acc. (A and B)
  - 40 bit Barrel Shifter
  - Temporary Register
  - Dedicated support
    - CSSU (Viterbi)
    - Bit reverse (FFT)
TMS320C54x DSP processor

- **Technology**
  - 1.0 - 5.0 V Core
  - 0.25μ CMOS
  - 30-160 MHz
  - 0.21-0.52 mW/MIP
  - 4.0 mW standby

- **Architecture** (From 10,000 ft.)
  - 16 bit fixed instructions
  - 64K/64K Data/Program
  - 1 MAC, 1 ALU, 2 Accumulators
  - 8 Auxiliary Registers (ARs)
  - DARAM
  - Compare Select and Store (CSSU) for Viterbi

Source: www.ti.com and ISLPED 2000 Tutorial
ADI TigerSHARC DSP Processor

National Tsing Hua University, COM 5195: 通訊系統晶片設計

source: http://www.analog.com/processors
Programming DSP uP

- **Characteristics of DSP routines**
  - short
  - repeat very often
  - time/performance critical

- **High Level Programming (Ex: C)**
  - Productivity: Speed in development
  - Maintainability: easy to maintain/reuse
  - Easy to develop (lower development cost)
  - Portability

- **Low Level Programming (Ex: Assembly)**
  - higher performance, performance critical DSP software often written and optimized in assembly
  - more difficult to develop and maintain
Compilers

- Increasing important for success
- Sophisticated compilers are becoming increasing vital
  - DSP software is becoming larger and complex
  - Processor architectures are becoming more complicated
  - Optimization is needed
Example Assembly Program: FIR Filter

20 tap FIR filter implementation
Goal: compute one output \( y_0 \)

\[ y_0 = a_0 x_0 + a_1 x_1 + a_2 x_2 + \ldots + a_{19} x_{19} \]

\[ y_0 = \sum_{n=0}^{19} a_n \ast x_n \]
Circular Buffering

- Circular buffers are used to store the most recent values of a continually updated signal as new samples are acquired.
- This illustration shows how an eight sample circular buffer might appear at some instant in time (a), and how it would appear one sample later (b).
**FIR Reloaded**

\[
y[n] = \sum_{i=0}^{7} c_i x[n-i]
\]

- **Steps needed to perform FIR in microprocessor**
  1. Obtain a sample with the ADC; generate an interrupt
  2. Detect and manage the interrupt
  3. Move the sample into the input signal's circular buffer
  4. Update the pointer for the input signal's circular buffer
  5. Zero the accumulator
  6. Control the loop through each of the coefficients
  7. Fetch the coefficient from the coefficient's circular buffer
  8. Update the pointer for the coefficient's circular buffer
  9. Fetch the sample from the input signal's circular buffer
  10. Update the pointer for the input signal's circular buffer
  11. Multiply the coefficient by the sample
  12. Add the product to the accumulator
  13. Move the output sample (accumulator) to a holding buffer
  14. Move the output sample from the holding buffer to the DAC
Instruction Pipeline Processing

Pipeline Phases
- P - generate program address
- F - get opcode
- D - decode instruction
- A - generate read address
- R - read operands
- X - execute

Full Pipeline
Coding Environment

lab1.obj
-o lab1.out
-m lab1.map

MEMORY {
  PAGE 1: /* Data Memory */
    SPRAM: org=00060h len=0020h
    InRAM: org=00400h len=0400h
    OutRAM: org=00800h len=0400h
  PAGE 0: /* Program Memory */
    ROM: org=0F000h len=0F80h
}

SECTIONS {
  code -> ROM PAGE 0
  init -> ROM PAGE 0
  input -> InRAM PAGE 1
  output -> OutRAM PAGE 1
  coeff -> SPRAM PAGE 1
}

Link.cmd

Overview

C54x
InRAM
x[20]
OutRAM
y[1]
SPRAM
a[20]

ROM
code
init_a[20]

x .usect "input",20
a .usect "coeff",20
y .usect "output",1
.sect "init"
init_a .int 1,2,3,4,5

.int 1,2,3,4,5
.int 1,2,3,4,5
.int 1,2,3,4,5
.mmregs
.sect "code" FIR.asm

National Tsing Hua University, COM 5195: 通訊系統晶片設計
Initializing Pointers

FIR.asm

fir:

STM #a, AR2
STM #x, AR3

math: MAC *AR2+, *AR3+, A

done:

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Input Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR2</td>
<td>AR3</td>
</tr>
<tr>
<td>a0</td>
<td>x0</td>
</tr>
<tr>
<td>a1</td>
<td>x1</td>
</tr>
<tr>
<td>a2</td>
<td>x2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

STM
- Stores #value to the MMR early in the pipeline to avoid latencies
- 2 words, 2 cycles
Load Accumulators

- We must initialize accumulator A using a load instruction

```
fir:
STM   #a, AR2
STM   #x, AR3
LD    #0, A

math: MAC  *AR2+, *AR3+, A

done:
```

Accumulator A

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>39-32</td>
<td>31-16</td>
<td>15-0</td>
</tr>
</tbody>
</table>
Processing loop

Two methods may be used to find $y_0$:

1. Multiply, then add
   
   $$\text{MPY }^*\text{AR2+}, \ *\text{AR3+}, \ B$$
   $$\text{ADD } B, A$$

2. Multiply/Accumulate
   
   $$\text{MAC }^*\text{AR2+}, \ *\text{AR3+}, \ A$$
Store Result

Store the result back to memory location

source A: accumulator
destination y: memory location address

FIR.asm

fir:

STM   #a, AR2
STM   #x, AR3
LD    #0, A

math:
MAC   *AR2+, *AR3+, A
STL   A, *(y)

done:

Accumulator A

<table>
<thead>
<tr>
<th>G</th>
<th>H</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>39-32</td>
<td>31-16</td>
<td>15-0</td>
</tr>
</tbody>
</table>
Overall FIR.asm Program Flow

- RPT #n: repeat the next block for n+1 times
- MVPD copy values from one memory location to another

```
FIR.asm

fir:
STM   #a, AR2
RPT   #(20-1)
MVPD  #init_a,*AR2+
STM   #a,AR2
STM   #x,AR3
LD    #0,A
RPT   #(20-1)

math:
MAC   *AR2+,*AR3+,A
STL   A, *(y)

done:  RET
```

How many cycle does this program take?

National Tsing Hua University, COM 5195: 通訊系統晶片設計
<table>
<thead>
<tr>
<th>ARM family</th>
<th>Embedded Cores</th>
<th>Platform Cores</th>
<th>Intel ARM-based Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM family</td>
<td>Cache Size (Inst/Data)</td>
<td>Tightly Coupled Memory</td>
<td>Memory Management</td>
</tr>
<tr>
<td>ARM7TDMI</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ARM7TDMI-S</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ARM7EJ-S</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ARM966E-S</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ARM940T</td>
<td>4K/4K</td>
<td>No</td>
<td>MPU</td>
</tr>
<tr>
<td>ARM946E-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MPU</td>
</tr>
<tr>
<td>ARM1026EJ-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU+MPU</td>
</tr>
<tr>
<td>ARM1020T</td>
<td>8K unified</td>
<td>No</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM920T</td>
<td>16K/16K</td>
<td>No</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM922T</td>
<td>8K/8K</td>
<td>No</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM926EJ-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM1020E</td>
<td>32K/32K</td>
<td>No</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM1022E</td>
<td>16K/16K</td>
<td>No</td>
<td>MMU</td>
</tr>
<tr>
<td>ARM1026EJ-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU+MPU</td>
</tr>
<tr>
<td>ARM1136J(F)-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>StrongARM</th>
<th>Intel XScale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size (Inst/Data)</td>
<td>16K/8K</td>
<td>32K/32K</td>
</tr>
<tr>
<td>MMU</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AHB Bus Interface</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Thumb</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSP</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Jazelle</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Clock (MHz)</td>
<td>206</td>
<td>400</td>
</tr>
</tbody>
</table>