Real-time Digital Signal Processing with the TMS320C6000

Objectives:
- Learn the architecture of a typical DSP
- Learn how to perform basic real-time DSP tasks on ‘real’ hardware
- Learn about interrupts and I/O devices

Digital Signal Processing
- Digital signal processing involves the manipulation of digital signals to extract useful information from them.

DSP vs VLSI
- DSPs have more application flexibility
- DSPs are more cost effective; VLSI are normally built for a single application / customer
- Higher sampling rates can typically be achieved using VLSI
- New features (bug fixes) can be added to DSP with software upgrade

Present Day Applications
- Wireless / Cellular
  - Voice-band audio
  - RF codecs
  - Voltage regulation
- Consumer Audio
  - Stereo A/D, D/A
  - PLL
  - Mixers
- Multimedia
  - Stere audio
  - Imaging
  - Graphics palette
  - Voltage regulation
- Automotive
  - Digital radio A/D/A
  - Active suspension
  - Voltage regulation
- HDD
  - PRML read channel
  - MR pre-amp
  - Servo control
  - SCSI transceivers
- Multimedia
  - Speech synthesizer
  - Mixed-signal processor
- DTAD
  - Servo control
  - SCSI transceivers

Course Information
- Lecturers: Mike Brookes, Peter Cheung, Darren Ward
- Web page: http://www.ee.ic.ac.uk/pcheung/teaching/ee3_Study_Project/index.html
- Assessment:
  - Test (14 June) 25%.
  - Labs (22 June) 30%, Project (22 June) 45%
- Deliverables:
  1. Labs 3-4,6-8: Program listings (with comments) with evidence that programs work
  2. Project: Full listing of your program with a short description (2-9) pages and assessment of performance
- You will be working in pairs, each pair submits a single set of reports
What Problem Are We Trying To Solve?

Digital sampling of an analog signal:

Most DSP algorithms can be expressed as:

\[ Y = \sum_{i=1}^{\text{count}} a_i \cdot x_i \]

for \( i = 1 \) to \( \text{count} \), \( i++ \) \{ \text{sum} += m[\text{index}] \cdot n[\text{index}]; \}

C6000 Architecture

8 functional units:
- D (data): 32-bit operands
- L (ALU): 32- or 40-bit operands
- S (shift): 32- or 40-bit operands
- M (multiply): 16x16 bit integer or 32x32 bit floating point

2 register files (32-bit)

All 8 units can be used in parallel each cycle

6000 Series Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>MIPS</th>
<th>MHz</th>
<th>Kbps</th>
<th>Pins</th>
<th>W</th>
<th>$</th>
<th>Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>6201</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>352</td>
<td>1.3</td>
<td>80-100</td>
<td>D2H</td>
</tr>
<tr>
<td>6202</td>
<td>2000</td>
<td>250</td>
<td>384</td>
<td>384/352</td>
<td>2.1</td>
<td>110-170</td>
<td>D3X</td>
</tr>
<tr>
<td>6203</td>
<td>2400</td>
<td>300</td>
<td>896</td>
<td>384</td>
<td>1.3</td>
<td>150-190</td>
<td>D3X</td>
</tr>
<tr>
<td>6204</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>384</td>
<td>0.8</td>
<td>30-60</td>
<td>D2X</td>
</tr>
<tr>
<td>6205</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>306</td>
<td>0.8</td>
<td>40-70</td>
<td>D2P</td>
</tr>
<tr>
<td>6211</td>
<td>1200</td>
<td>150</td>
<td>72</td>
<td>256</td>
<td>0.9</td>
<td>25-40</td>
<td>E2H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>MFLOPS</th>
<th>MHz</th>
<th>Kbps</th>
<th>Pins</th>
<th>W</th>
<th>$</th>
<th>Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>6701</td>
<td>1000</td>
<td>167</td>
<td>128</td>
<td>352</td>
<td>1.4</td>
<td>110-170</td>
<td>D2H</td>
</tr>
<tr>
<td>6711</td>
<td>900</td>
<td>150</td>
<td>72</td>
<td>256</td>
<td>1.1</td>
<td>25-50</td>
<td>E2H</td>
</tr>
<tr>
<td>6712</td>
<td>600</td>
<td>100</td>
<td>72</td>
<td>256</td>
<td>0.7</td>
<td>10-22</td>
<td>E2H</td>
</tr>
</tbody>
</table>

6000 Series Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>MIPS</th>
<th>MHz</th>
<th>Kbps</th>
<th>Pins</th>
<th>W</th>
<th>$</th>
<th>Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>6201</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>352</td>
<td>1.3</td>
<td>80-100</td>
<td>D2H</td>
</tr>
<tr>
<td>6202</td>
<td>2000</td>
<td>250</td>
<td>384</td>
<td>384/352</td>
<td>2.1</td>
<td>110-170</td>
<td>D3X</td>
</tr>
<tr>
<td>6203</td>
<td>2400</td>
<td>300</td>
<td>896</td>
<td>384</td>
<td>1.3</td>
<td>150-190</td>
<td>D3X</td>
</tr>
<tr>
<td>6204</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>384</td>
<td>0.8</td>
<td>30-60</td>
<td>D2X</td>
</tr>
<tr>
<td>6205</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>306</td>
<td>0.8</td>
<td>40-70</td>
<td>D2P</td>
</tr>
<tr>
<td>6211</td>
<td>1200</td>
<td>150</td>
<td>72</td>
<td>256</td>
<td>0.9</td>
<td>25-40</td>
<td>E2H</td>
</tr>
</tbody>
</table>

6000 Series Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>MIPS</th>
<th>MHz</th>
<th>Kbps</th>
<th>Pins</th>
<th>W</th>
<th>$</th>
<th>Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>6201</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>352</td>
<td>1.3</td>
<td>80-100</td>
<td>D2H</td>
</tr>
<tr>
<td>6202</td>
<td>2000</td>
<td>250</td>
<td>384</td>
<td>384/352</td>
<td>2.1</td>
<td>110-170</td>
<td>D3X</td>
</tr>
<tr>
<td>6203</td>
<td>2400</td>
<td>300</td>
<td>896</td>
<td>384</td>
<td>1.3</td>
<td>150-190</td>
<td>D3X</td>
</tr>
<tr>
<td>6204</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>384</td>
<td>0.8</td>
<td>30-60</td>
<td>D2X</td>
</tr>
<tr>
<td>6205</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>306</td>
<td>0.8</td>
<td>40-70</td>
<td>D2P</td>
</tr>
<tr>
<td>6211</td>
<td>1200</td>
<td>150</td>
<td>72</td>
<td>256</td>
<td>0.9</td>
<td>25-40</td>
<td>E2H</td>
</tr>
</tbody>
</table>

6000 Series Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>MIPS</th>
<th>MHz</th>
<th>Kbps</th>
<th>Pins</th>
<th>W</th>
<th>$</th>
<th>Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>6201</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>352</td>
<td>1.3</td>
<td>80-100</td>
<td>D2H</td>
</tr>
<tr>
<td>6202</td>
<td>2000</td>
<td>250</td>
<td>384</td>
<td>384/352</td>
<td>2.1</td>
<td>110-170</td>
<td>D3X</td>
</tr>
<tr>
<td>6203</td>
<td>2400</td>
<td>300</td>
<td>896</td>
<td>384</td>
<td>1.3</td>
<td>150-190</td>
<td>D3X</td>
</tr>
<tr>
<td>6204</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>384</td>
<td>0.8</td>
<td>30-60</td>
<td>D2X</td>
</tr>
<tr>
<td>6205</td>
<td>1600</td>
<td>200</td>
<td>128</td>
<td>306</td>
<td>0.8</td>
<td>40-70</td>
<td>D2P</td>
</tr>
<tr>
<td>6211</td>
<td>1200</td>
<td>150</td>
<td>72</td>
<td>256</td>
<td>0.9</td>
<td>25-40</td>
<td>E2H</td>
</tr>
</tbody>
</table>

Peripherals Legend:
- D, E: DMA (4), EDMA (16)
- L, P: H. V, P: HPI (Host Port), XBUS, PCI

Pin-for-Pin Compatible with:
- 6211: 6711, 6712
- 6203: 6711, 6712
- 6202: 6203, 6204

The Core of DSP: Sum of Products

\[ y = \sum_{n=1}^{40} a_n \cdot x_n \]

Working Variables: The Register File

<table>
<thead>
<tr>
<th>Register File A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>y</td>
<td>prod</td>
</tr>
<tr>
<td>c</td>
<td>y</td>
<td>prod</td>
</tr>
</tbody>
</table>

Fast MAC using Natural C

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    float sum = 0;
    for (i = 0; i < count; i++)
        sum += m[\text{index}] \cdot n[\text{index}];
    return 0;
}
```

Note: You don't have to specify functional units (M or L)
Loops: Coding on a RISC Processor

1. **Program flow**: the branch instruction
   
   ![Branch Instruction Diagram]

2. **Initialization**: setting the loop count
   
   ![Initialization Diagram]

3. **Decrement**: subtract 1 from the loop counter
   
   ![Decrement Diagram]

The “S” Unit: For Standard Operations

![Register File A](image)

How is the loop terminated?

Conditional Instruction Execution

To minimize branching, all instructions are conditional

![Conditional Instruction Diagram]

Execution based on [zero/non-zero] value of specified variable

<table>
<thead>
<tr>
<th>Code Syntax</th>
<th>Execute if:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![cnt]</td>
<td>cnt = 0</td>
</tr>
<tr>
<td>![cnt]</td>
<td>cnt = 0</td>
</tr>
</tbody>
</table>

Loop Control via Conditional Branch

![Register File A](image)

How are the a and x array values brought in from memory?

Memory Access via “.D” Unit

![Register File A](image)

How do we increment through the arrays?

Data Memory:

x(40), a(40), y

Auto-Increment of Pointers

![Register File A](image)

How do we store results back to memory?

Data Memory:

x(40), a(40), y
Register File A

\[ y = \sum_{n=1}^{40} a_n \cdot x_n \]

Data Memory: x(40), a(40), y

---

\[ \text{MVK:} \quad .S \quad 40, \text{cnt} \]

\[ \text{LDH:} \quad .D \quad \text{ap}++, \text{a} \]

\[ \text{MPY:} \quad .M \quad \text{a, x, prod} \]

\[ \text{ADD:} \quad .L \quad \text{y, prod, y} \]

\[ \text{SUB:} \quad .L \quad \text{cnt, 1, cnt} \]

\[ \text{STW:} \quad .D \quad \text{y, *yp} \]

---

1. **Storing Results Back to Memory**

2. **C6000 System Block Diagram**

3. **C6700: RISC-like instruction set**

4. **C6000 Internal Buses**

5. **C6000 Peripherals**

6. **C6000 Peripherals (EMIF)**

---

- **C6000 Peripherals**
  - Host-Port Interface
  - Internal Buses
  - External Memory
  - EMIF
  - MCBSPs
  - DMA
  - EDMA
  - Boot Loader
  - Time Counter

- **C6000 Peripherals (EMIF)**
  - Async
  - Internal Memory
  - SDRAM
  - SBSRAM
  - EMIF
    - Glueless access to async/sync memory
    - Works with PC100 SDRAM (cheap, fast, and easy!)
    - 8/16/32-bit data or 32-bit program access
**DSP Starter Kit (DSK)**

- **Hardware**
  - 150 MHz 'C6711 DSP
  - TI 16-bit A/D Converter ('AD535)
  - External Memory
    - 16M Bytes SDRAM
    - 128K Bytes Flash ROM
  - LED's
  - Daughter card expansion
  - Power Supply & Parallel Port Cable

- **Software**
  - Code Generation Tools
    - (C Compiler, Assembler & Linker)
  - Code Composer Debugger
    - (256K program limitation)
  - Example Programs & SW Utilities
  - Power-on Self Test
  - Flash Utility Program
  - Board Confidence Test
  - Host access via DLL
  - Sample Program(s)

---

**'C6711 DSK Overview**

- 1.8V Power Supply
- 1MB SDRAM
- 128K FLASH
- Daughter Card IF
- Parallel Port IF
- Power JDSK
- Reset Emulator
- JTAG Header
- 16-bit codecs: AD & DA
- Line Level Input (micromoment)
- Line Level Output (speaker)
Win32 API for Host

dsk6x_open() - Open a connection to the DSK
dsk6x_close() - Close a connection to the DSK
dsk6x_reset_board() - Reset the entire DSK board
dsk6x_reset_dsp() - Reset only the DSP on the DSK
dsk6x_coff_load() - Load a COFF image to DSP memory
dsk6x_hpi_open() - Open the HPI for the DSP
dsk6x_hpi_close() - Close the HPI for the DSP
dsk6x_hpi_read() - Read DSP memory via the HPI
dsk6x_hpi_write() - Write to DSP memory via the HPI
dsk6x_generate_int() - Generate a DSP interrupt