INTRODUCTION TO
THE TMS320C6000
VLIW DSP

Prof. Brian L. Evans

in collaboration with
Niranjan Damera-Venkata and
Magesh Valliappan

Embedded Signal Processing Laboratory
The University of Texas at Austin
Austin, TX 78712-1084

http://signal.ece.utexas.edu/
Outline

- C6000 instruction set architecture review
- Vector dot product example
- Pipelining
- Finite impulse response filtering
- Vector dot product example
- Conclusion
**TI TMS320C6000 DSP Architecture (Review)**

**Simplified Architecture**

- External Memory
  - Sync
  - Async
- Program RAM or Cache
- Data RAM
- CPU
- Internal Buses
- Addr
- Data
- Regs (A0-A15)
  - .D1
  - .M1
  - .L1
  - .S1
- Regs (B0-B15)
  - .D2
  - .M2
  - .L2
  - .S2
- Control Regs
- DMA
- Serial Port
- Host Port
- Boot Load
- Timers
- Pwr Down

C6200 fixed point
C6400 fixed point
C6700 floating point
Address 8/16/32 bit data + 64-bit data on C67x

Load-store RISC architecture with 2 data paths
- 16 32-bit registers per data path (A0-A15 and B0-B15)
- 48 instructions (C6200) and 79 instructions (C6700)

Two parallel data paths with 32-bit RISC units
- Data unit - 32-bit address calculations (modulo, linear)
- Multiplier unit - 16 bit x 16 bit with 32-bit result
- Logical unit - 40-bit (saturation) arithmetic & compares
- Shifter unit - 32-bit integer ALU and 40-bit shifter
- Conditionally executed based on registers A1-2 & B0-2
- Work with two 16-bit halfwords packed into 32 bits
TI TMS320C6000 DSP Architecture (Review)

- **.M multiplication unit**
  - 16 bit x 16 bit signed/unsigned packed/unpacked

- **.L arithmetic logic unit**
  - Comparisons and logic operations (and, or, and xor)
  - Saturation arithmetic and absolute value calculation

- **.S shifter unit**
  - Bit manipulation (set, get, shift, rotate) and branching
  - Addition and packed addition

- **.D data unit**
  - Load/store to memory
  - Addition and pointer arithmetic
C6000 Restrictions on Register Accesses

- Each function unit has read/write ports
  - Data path 1 (2) units read/write A (B) registers
  - Data path 2 (1) can read one A (B) register per cycle
- Two simultaneous memory accesses cannot use registers of same register file as address pointers
- Limit of four 32-bit reads per register per cycle
- 40-bit longs stored in adjacent even/odd registers
  - Extended precision accumulation of 32-bit numbers
  - Only one 40-bit result can be written per cycle
  - 40-bit read cannot occur in same cycle as 40-bit write
  - 4:1 performance penalty using 40-bit mode
Other C6000 Disadvantages

- No ALU acceleration for bit stream manipulation
  - 50% computation in MPEG-2 decoder spent on variable length decoding on C6200 in C
  - C6400 direct memory access controllers shred bit streams (for video conferencing & wireless basestations)

- Branch in pipeline disables interrupts:
  *Avoid branches by using conditional execution*

- No hardware protection against pipeline hazards:
  *Programmer and tools must guard against it*

- Must emulate many conventional DSP features
  - No hardware looping: use register/conditional branch
  - No bit-reversed addressing: use fast algorithm by Elster
  - No status register: only saturation bit given by .L units
FIR Filter

- Difference equation (vector dot product)
  \[ y(n) = 2 \, x(n) + 3 \, x(n - 1) + 4 \, x(n - 2) + 5 \, x(n - 3) \]

- Signal flow graph
  \[ y(n) = \sum_{i=0}^{N-1} a(i) \, x(n-i) \]

- Dot product of inputs vector and coefficient vector
- Store input in circular buffer, coefficients in array
Each tap requires
- Fetching data sample
- Fetching coefficient
- Fetching operand
- Multiplying two numbers
- Accumulating multiplication result
- Shifting one sample in the delay line

Computing an FIR tap in one instruction cycle
- Two data memory and one program memory accesses
- Auto-increment or auto-decrement addressing modes
- Modulo addressing to implement delay line as circular buffer
Example: Vector Dot Product (Unoptimized)

- A vector dot product is common in filtering

\[ Y = \sum_{n=1}^{N} a(n) \times x(n) \]

- Store \( a(n) \) and \( x(n) \) into an array of \( N \) elements

- C6000 peaks at 8 RISC instructions/cycle
  - For 300-MHz C6000, RISC instructions per sample: 300,000 for speech; 54,421 for audio CD; and 290 for luminance NTSC video
  - Generally requires hand coding for peak performance

- First dot product example will not be optimized
Example: Vector Dot Product (Unoptimized)

- **Prologue**
  - Initialize pointers: A5 for \( a(n) \), A6 for \( x(n) \), and A7 for \( Y \)
  - Move the number of times to loop (\( N \)) into A2
  - Set accumulator (A4) to zero

- **Inner loop**
  - Put \( a(n) \) into A0 and \( x(n) \) into A1
  - Multiply \( a(n) \) and \( x(n) \)
  - Accumulate multiplication result into A4
  - Decrement loop counter (A2)
  - Continue inner loop if counter is not zero

- **Epilogue**
  - Store the result into \( Y \)
Example: Vector Dot Product (Unoptimized)

Coefficients $a(n)$

Data $x(n)$

Using A data path only

; clear A4 and initialize pointers A5, A6, and A7
MVK .S1 40,A2 ; A2 = 40 (loop counter)

loop
LDH .D1 *A5++,A0 ; A0 = $a(n)$
LDH .D1 *A6++,A1 ; A1 = $x(n)$
MPY .M1 A0,A1,A3 ; A3 = $a(n) \times x(n)$
ADD .L1 A3,A4,A4 ; Y = Y + A3
SUB .L1 A2,1,A2 ; decrement loop counter
[A2] B .S1 loop ; if A2 != 0, then branch
STH .D1 A4,*A7 ; *A7 = Y
Example: Vector Dot Product (Unoptimized)

- **MoVeKonstant**
  - MVK .S 40,A2 ; A2 = 40
  - Lower 16 bits of A2 are loaded

- **Conditional branch**
  - [condition] B .S loop
  - [A2] means to execute the instruction if A2 $\neq 0$
  - Only A1, A2, B0, B1, and B2 can be used

- **Loading registers**
  - LDH .D *A5, A0 ; Loads half-word into A0 from memory

- **Registers may be used as pointers (A1++)**

- **Implementation not efficient due to pipeline effects**
Pipelining

- CPU operations
  - Fetch instruction from (on-chip) program memory
  - Decode instruction
  - Execute instruction including reading data values

- Overlap operations to increase performance
  - Pipeline CPU operations to increase clock speed over a sequential implementation
  - Separate parallel functional units
  - Peripheral interfaces for I/O do not burden CPU
Pipelining

Sequential (Motorola 56000)

Pipelined (Most conventional DSP processors)

Superscalar (Pentium, MIPS)

Superpipelined (TMS320C6000)

Managing Pipelines

• compiler or programmer (TMS320C6000)
• pipeline interlocking in processor (TMS320C30)
• hardware instruction scheduling
TMS320C6000 Pipeline

- One instruction cycle every clock cycle
- Deep pipeline
  - 7-11 stages in C62x: fetch 4, decode 2, execute 1-5
  - 7-16 stages in C67x: fetch 4, decode 2, execute 1-10
  - If a branch is in the pipeline, interrupts are disabled
  - Avoid branches by using conditional execution
- No hardware protection against pipeline hazards
  - Compiler and assembler must prevent pipeline hazards
- Dispatches instructions in packets
Program Fetch (F)

- Program fetching consists of 4 phases
  - Generate fetch address (FG)
  - Send address to memory (FS)
  - Wait for data ready (FW)
  - Read opcode (FR)

- Fetch packet consists of 8 32-bit instructions
- Decode stage consists of two phases
  - Dispatch instruction to functional unit (DP)
  - Instruction decoded at functional unit (DC)
## Execute Stage (E)

### Type

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th># Instr</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISC</td>
<td>Single cycle</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>IMPY</td>
<td>Multiply</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>LDx</td>
<td>Load</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>Branch</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

### Execute Phase

<table>
<thead>
<tr>
<th>Execute Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>ISC instructions completed</td>
</tr>
<tr>
<td>E2</td>
<td>Int. mult. instructions completed</td>
</tr>
<tr>
<td>E3</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>Load memory value into register</td>
</tr>
<tr>
<td>E6</td>
<td>Branch to destination complete</td>
</tr>
</tbody>
</table>
Vector Dot Product with Pipeline Effects

; clear A4 and initialize pointers A5, A6, and A7
MVK .S1 40,A2 ; A2 = 40 (loop counter)
loop
LDH .D1 *A5++,A0 ; A0 = a(n)
LDH .D1 *A6++,A1 ; A1 = x(n)
MPY .M1 A0,A1,A3 ; A3 = a(n) * x(n)
ADD .L1 A3,A4,A4 ; Y = Y + A3
SUB .L1 A2,1,A2 ; decrement loop counter
[A2] B .S1 loop ; if A2 != 0, then branch
STH .D1 A4,*A7 ; *A7 = Y

Multiplication has a delay of 1 cycle
Load has a delay of four cycles
### Fetch packet

<table>
<thead>
<tr>
<th>F</th>
<th>DP</th>
<th>DC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVK</td>
<td>LDH</td>
<td>LDH</td>
<td>LDH</td>
<td>MPY</td>
<td>ADD</td>
<td>SUB</td>
<td>B</td>
<td>STH</td>
</tr>
<tr>
<td>(F_{1,4})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time (t) = 4 clock cycles
<table>
<thead>
<tr>
<th>F</th>
<th>DP</th>
<th>DC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(2-5)</td>
<td>MVK</td>
<td>LDH</td>
<td>LDH</td>
<td>MPY</td>
<td>ADD</td>
<td>SUB</td>
<td>B</td>
<td>STH</td>
</tr>
</tbody>
</table>

Time (t) = 5 clock cycles
### Decode

<table>
<thead>
<tr>
<th>F</th>
<th>DP</th>
<th>DC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(2-5)</td>
<td>LDH</td>
<td>LDH</td>
<td>MPY</td>
<td>ADD</td>
<td>SUB</td>
<td>B</td>
<td>STH</td>
<td>MVK</td>
</tr>
</tbody>
</table>

Time (t) = 6 clock cycles
### Execute (E1)

<table>
<thead>
<tr>
<th>F(2-5)</th>
<th>DP</th>
<th>DC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDH</td>
<td></td>
<td>LDH</td>
<td>MVK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time \( t \) = 7 clock cycles
### Execute (MVK done LDH in E1)

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>DP</th>
<th>DC</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>$F(2,5)$</td>
<td>MPY</td>
<td>ADD</td>
<td>SUB</td>
<td>B</td>
<td>STH</td>
<td>LDH</td>
<td>LDH</td>
<td></td>
</tr>
</tbody>
</table>

- **MPY**
- **ADD**
- **SUB**
- **B**
- **STH**

**MVK Done**

Time ($t$) = 8 clock cycles
Vector Dot Product with Pipeline Effects

; clear A4 and initialize pointers A5, A6, and A7
MVK  .S1  40,A2  ; A2 = 40 (loop counter)

loop
LDH  .D1  *A5++,A0  ; A0 = a(n)
LDH  .D1  *A6++,A1  ; A1 = x(n)
NOP  4
MPY  .M1  A0,A1,A3  ; A3 = a(n) * x(n)
NOP
ADD  .L1  A3,A4,A4  ; Y = Y + A3
SUB  .L1  A2,1,A2  ; decrement loop counter
[B2]
B    .S1  loop  ; if A2 != 0, then branch
NOP  5
STH  .D1  A4,*A7  ; *A7 = Y

Assembler will automatically insert NOP instructions
Assembler can also make sequential code parallel
Optimized Vector Dot Product on the C6000

- **Prologue**
  - Retime dot product to compute two terms per cycle
  - Initialize pointers: A5 for \(a(n)\), B6 for \(x(n)\), A7 for \(y(n)\)
  - Move number of times to loop (\(N\)) divided by 2 into A2

- **Inner loop**
  - Put \(a(n)\) and \(a(n+1)\) in A0 and \(x(n)\) and \(x(n+1)\) in A1 (**packed data**)
  - Multiply \(a(n)\) \(x(n)\) and \(a(n+1)\) \(x(n+1)\)
  - Accumulate even (odd) indexed terms in A4 (B4)
  - Decrement loop counter (A2)

- **Store result**

<table>
<thead>
<tr>
<th>Reg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>(a(n) || a(n+1))</td>
</tr>
<tr>
<td>B1</td>
<td>(x(n) || x(n+1))</td>
</tr>
<tr>
<td>A2</td>
<td>((N - n)/2)</td>
</tr>
<tr>
<td>A3</td>
<td>(a(n) \times x(n))</td>
</tr>
<tr>
<td>B3</td>
<td>(a(n+1) \times x(n+1))</td>
</tr>
<tr>
<td>A4</td>
<td>(y_{\text{even}}(n))</td>
</tr>
<tr>
<td>B4</td>
<td>(y_{\text{odd}}(n))</td>
</tr>
<tr>
<td>A5</td>
<td>&amp;a</td>
</tr>
<tr>
<td>B6</td>
<td>&amp;x</td>
</tr>
<tr>
<td>A7</td>
<td>&amp;y</td>
</tr>
</tbody>
</table>
FIR Filter Implementation on the C6000

MVK .S1 0x0001,AMR ; modulo block size $2^2$
MVKH .S1 0x4000,AMR ; modulo addr register B6
MVK .S2 2,A2 ; A2 = 2 (four-tap filter)
ZERO .L1 A4 ; initialize accumulators
ZERO .L2 B4

; initialize pointers A5, B6, and A7
fir LDW .D1 *A5++,A0 ; load $a(n)$ and $a(n+1)$
LDW .D2 *B6++,B1 ; load $x(n)$ and $x(n+1)$
MPY .M1X A0,B1,A3 ; A3 = $a(n) * x(n)$
MPYH .M2X A0,B1,B3 ; B3 = $a(n+1) * x(n+1)$
ADD .L1 A3,A4,A4 ; yeven(n) += A3
ADD .L2 B3,B4,B4 ; yodd(n) += B3

[A2] SUB .S1 A2,1,A2 ; decrement loop counter
[A2] B .S2 fir ; if A2 != 0, then branch
ADD .L1 A4,B4,A4 ; $Y = Yodd + Yeven$
STH .D1 A4,*A7 ; *A7 = Y

Throughput of two multiply-accumulates per instruction cycle
### Selected TMS320C6000 Fixed-Point DSPs

<table>
<thead>
<tr>
<th>DSP</th>
<th>MHz</th>
<th>MIPS</th>
<th>Data (kbits)</th>
<th>Program (kbits)</th>
<th>Level 2 (kbits)</th>
<th>Price</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6202</td>
<td>250</td>
<td>2000</td>
<td>1000</td>
<td>2000</td>
<td></td>
<td>$59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>2400</td>
<td></td>
<td></td>
<td></td>
<td>$70</td>
<td></td>
</tr>
<tr>
<td>C6203</td>
<td>250</td>
<td>2000</td>
<td>4000</td>
<td>3000</td>
<td></td>
<td>$63</td>
<td>modems banks;</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>2400</td>
<td></td>
<td></td>
<td></td>
<td>$75</td>
<td>ADSL1 modems</td>
</tr>
<tr>
<td>C6204</td>
<td>200</td>
<td>1600</td>
<td>512</td>
<td>512</td>
<td></td>
<td>$10</td>
<td></td>
</tr>
<tr>
<td>C6416</td>
<td>500</td>
<td>4000</td>
<td>128</td>
<td>128</td>
<td>8000</td>
<td>$95</td>
<td>ADSL2 modems</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>8000</td>
<td>128</td>
<td>128</td>
<td>8000</td>
<td>$250</td>
<td>3G basestations</td>
</tr>
<tr>
<td>C6418</td>
<td>500</td>
<td>4000</td>
<td>128</td>
<td>128</td>
<td>5000</td>
<td>$57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>4800</td>
<td>128</td>
<td>128</td>
<td>5000</td>
<td>$56</td>
<td></td>
</tr>
<tr>
<td>DM640</td>
<td>400</td>
<td>3200</td>
<td>128</td>
<td>128</td>
<td>1000</td>
<td>$23</td>
<td>Video conferencing</td>
</tr>
<tr>
<td>DM641</td>
<td>500</td>
<td>4000</td>
<td>128</td>
<td>128</td>
<td>1000</td>
<td>$35</td>
<td>Video conferencing</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>4800</td>
<td>128</td>
<td>128</td>
<td>1000</td>
<td>$38</td>
<td></td>
</tr>
<tr>
<td>DM642</td>
<td>500</td>
<td>4000</td>
<td>128</td>
<td>128</td>
<td>2000</td>
<td>$43</td>
<td>Video conferencing</td>
</tr>
<tr>
<td></td>
<td>720</td>
<td>5760</td>
<td>128</td>
<td>128</td>
<td>2000</td>
<td>$68</td>
<td></td>
</tr>
</tbody>
</table>

C6416 has Viterbi and Turbo decoder coprocessors.

Unit price is for 1,000 units. Prices effective June 3, 2005.

For more information: [http://www.ti.com](http://www.ti.com)
Conventional digital signal processors
- High performance vs. power consumption/cost/volume
- Excel at one-dimensional processing
- Have instructions tailored to specific applications

TMS320C6000 VLIW DSP
- High performance vs. cost/volume
- Excel at multidimensional signal processing
- Maximum of 8 RISC instructions per cycle
Conclusion

Web resources
- comp.dsp news group: FAQ www.bdti.com/faq/dsp_faq.html
- embedded processors and systems: www.eg3.com
- on-line courses and DSP boards: www.techonline.com

References
- B. L. Evans, “EE345S Real-Time DSP Laboratory,” UT Austin. [http://www.ece.utexas.edu/~bevans/courses/realtime/](http://www.ece.utexas.edu/~bevans/courses/realtime/)
- B. L. Evans, “EE382C Embedded Software Systems,” UT Austin. [http://www.ece.utexas.edu/~bevans/courses/ee382c/](http://www.ece.utexas.edu/~bevans/courses/ee382c/)
COEFFP .set 02000h ; Program mem address
X .set 037Fh ; Newest data sample
LASTAP .set 037FH ; Oldest data sample

...  
LAR AR3, #LASTAP ; Point to oldest sample  
RPT #127 ; Repeat next inst. 126 times  
MACD COEFFP, *- ; Compute one tap of FIR  
APAC  
SACH Y,1 ; Store result -- note shift
### TMS320C6200 vs. StarCore S140

<table>
<thead>
<tr>
<th>Feature</th>
<th>C6200</th>
<th>S140</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Units</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>multipliers</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>adders</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>other</td>
<td>--</td>
<td>8</td>
</tr>
<tr>
<td>Instructions/cycle</td>
<td>8</td>
<td>6 + branch</td>
</tr>
<tr>
<td>RISC instructions *</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>conditionals</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Instruction width (bits)</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>Total instructions</td>
<td>48</td>
<td>180</td>
</tr>
<tr>
<td>Number of registers</td>
<td>32</td>
<td>51</td>
</tr>
<tr>
<td>Register size (bits)</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>Accumulation precision (bits) **</td>
<td>32 or 40</td>
<td>40</td>
</tr>
<tr>
<td>Pipeline depth (cycle)</td>
<td>7-11</td>
<td>5</td>
</tr>
</tbody>
</table>

* Does not count equivalent RISC operations for modulo addressing

** On the C6200, there is a performance penalty for 40-bit accumulation