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

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%

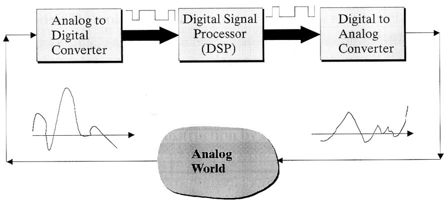
Deliverables

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

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

DSP vs Microprocessor

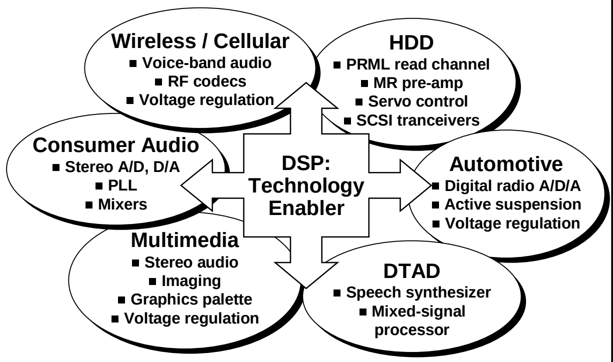
◆ DSPs can do several memory accesses in a single instruction

◆ DSPs are optimized to handle repitition / looping operations common in signal processing algorithms

◆ DSPs allow specialized addressing modes (e.g., circular, indirect) to implement SP algorithms

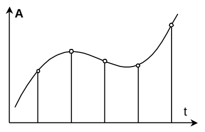
◆ DSPs possess peripherals that allow for efficient I/O interfacing

Present Day Applications



What Problem Are We Trying To Solve?

Digital sampling of an analog signal:

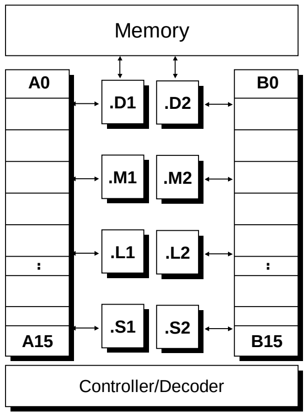


Most DSP algorithms can be expressed as:

$$Y = \sum_{i=1}^{count} a_i * x_i$$

```
for (i = 1; i < count; i++){
  sum += m[i] * n[i]; }
```

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

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

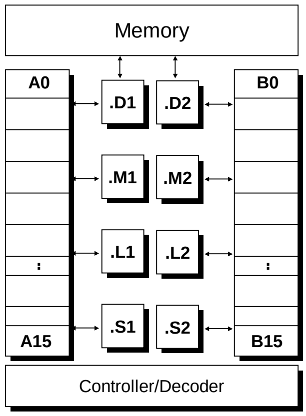
Device	MFLOPS	MHz	KBytes	Pins	W	$	Periphs
6701	1000	167	128	352	1.4	110 - 170	D2H
6711	900	150	72	256	1.1	25 - 50	E2H
6712	600	100	72	256	0.7	10 - 22	E2

Peripherals Legend:

D,E	DMA (4), EDMA (16)
2,3	# of McBSP Serial Ports
H,X,P	HPI (Host Port), XBUS, PCI

Pin-for-Pin Compatibility:
6201 & 6701
6211 & 6711 & 6712
6202 & 6203 & 6204

Fast MAC using Natural C



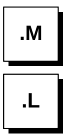

```
float mac(float *m, float *n, int count)
{ int i, float sum = 0;

  for (i=0; i < count; i++)  {
    sum += m[i] * n[i]; }   ...
```

```
;** -----------------------------------------------*
LOOP:     ; PIPED LOOP KERNEL
          LDDW    .D1     A4++,A7:A6
||        LDDW    .D2     B4++,B7:B6
||        MPYSP   .M1X    A6,B6,A5
||        MPYSP   .M2X    A7,B7,B5
||        ADDSP   .L1     A5,A8,A8
||        ADDSP   .L2     B5,B8,B8
|| [A1]   B       .S2     LOOP
|| [A1]   SUB     .S1     A1,1,A1
;** -----------------------------------------------*
```

The Core of DSP : Sum of Products

The 'C6000

Designed to handle DSP's math-intensive calculations



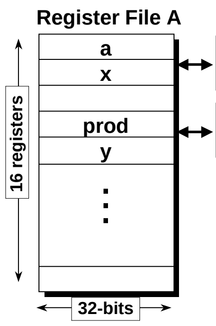
$$y = \sum_{n=1}^{40} a_n * x_n$$

MPY	.M	a, x, prod
ADD	.L	y, prod, y

Note:
You don't have to specify functional units (.M or .L)

Where are the variables?

Working Variables : The Register File



$$y = \sum_{n=1}^{40} a_n * x_n$$

MPY	.M	a, x, prod
ADD	.L	y, prod, y

How are the number of iterations specified?

Loops: Coding on a RISC Processor

1. **Program flow:** the branch instruction

B	**loop**

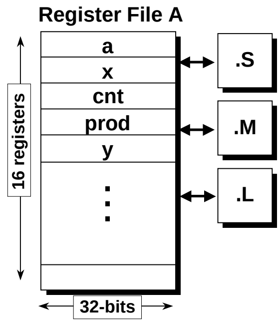
2. **Initialization:** setting the loop count

MVK	**40, cnt**

3. **Decrement:** subtract 1 from the loop counter

SUB	**cnt, 1, cnt**

The "S" Unit : For Standard Operations



$$y = \sum_{n=1}^{40} a_n * x_n$$

```
         MVK   .S    40, cnt
loop:
         MPY   .M    a, x, prod
         ADD   .L    y, prod, y
         SUB   .L    cnt, 1, cnt
         B     .S    loop
```

How is the loop terminated?

Conditional Instruction Execution

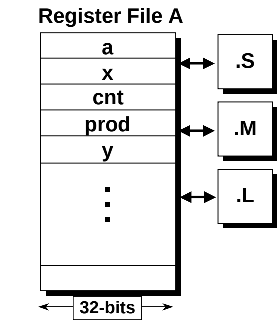
To minimize branching, **all** instructions are conditional

[condition]   B   loop

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

Code Syntax	Execute if:
[cnt]	**cnt ≠ 0**
[!cnt]	**cnt = 0**

Loop Control via Conditional Branch

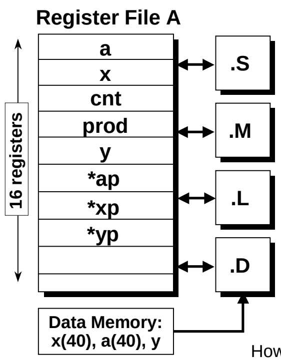


$$y = \sum_{n=1}^{40} a_n * x_n$$

```
         MVK   .S    40, cnt
loop:
         MPY   .M    a, x, prod
         ADD   .L    y, prod, y
         SUB   .L    cnt, 1, cnt
  [cnt]  B     .S    loop
```

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

Memory Access via ".D" Unit

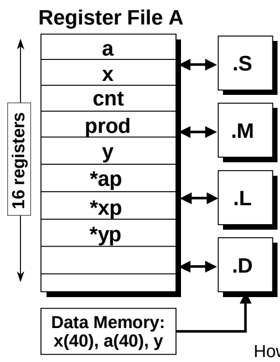


$$y = \sum_{n=1}^{40} a_n * x_n$$

```
         MVK   .S    40, cnt
loop:
         LDH   .D    *ap   , a
         LDH   .D    *xp   , x
         MPY   .M    a, x, prod
         ADD   .L    y, prod, y
         SUB   .L    cnt, 1, cnt
  [cnt]  B     .S    loop
```

How do we increment through the arrays?

Auto-Increment of Pointers

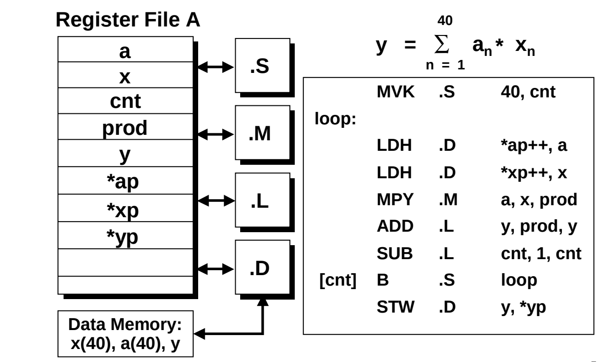


$$y = \sum_{n=1}^{40} a_n * x_n$$

```
         MVK   .S    40, cnt
loop:
         LDH   .D    *ap++, a
         LDH   .D    *xp++, x
         MPY   .M    a, x, prod
         ADD   .L    y, prod, y
         SUB   .L    cnt, 1, cnt
  [cnt]  B     .S    loop
```

How do we store results back to memory?

Storing Results Back to Memory



‘C6700 : RISC-like instruction set

.S .L .D .M

.S Unit

ADD	NEG	ABSSP
ADDK	NOT	ABSDP
ADD2	OR	CMPGTSP
AND	SET	CMPEQSP
B	SHL	CMPLTSP
CLR	SHR	CMPGTDP
EXT	SSHL	CMPEQDP
MV	SUB	CMPLTDP
MVC	SUB2	RCPSP
MVK	XOR	RCPDP
MVKH	ZERO	RSQRSP
		RSQRDP
		SPDP

.L Unit

ABS	NOT	ADDSP
ADD	OR	ADDDP
AND	SADD	SUBSP
CMPEQ	SAT	SUBDP
CMPGT	SSUB	INTSP
CMPLT	SUB	INTDP
LMBD	SUBC	SPINT
MV	XOR	DPINT
NEG	ZERO	SPRTUNC
NORM		DPTRUNC
		DPSP

.D Unit

ADD	NEG
ADDAB (B/H/W)	STB (B/H/W)
LDB (B/H/W)	SUB
LDDW	SUBAB (B/H/W)
MV	ZERO

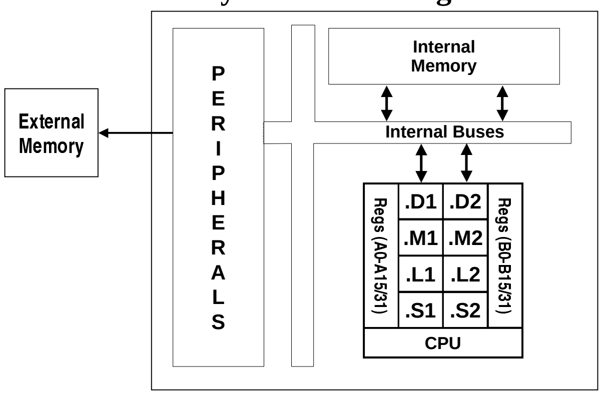
.M Unit

MPY	SMPY	MPYSP
MPYH	SMPYH	MPYDP
MPYLH		MPYI
MPYHL		MPYID

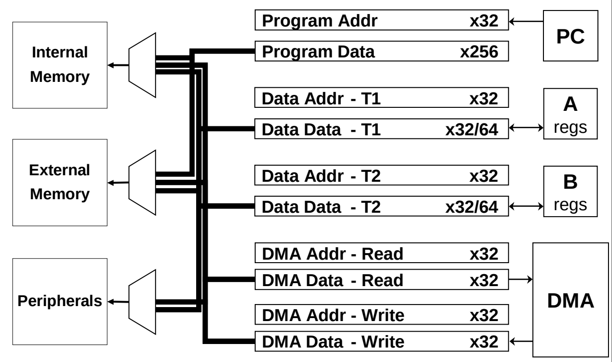
No Unit Used

NOP	IDLE

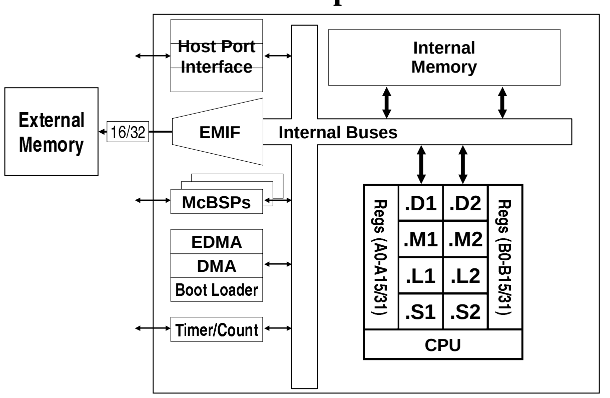
'C6000 System Block Diagram



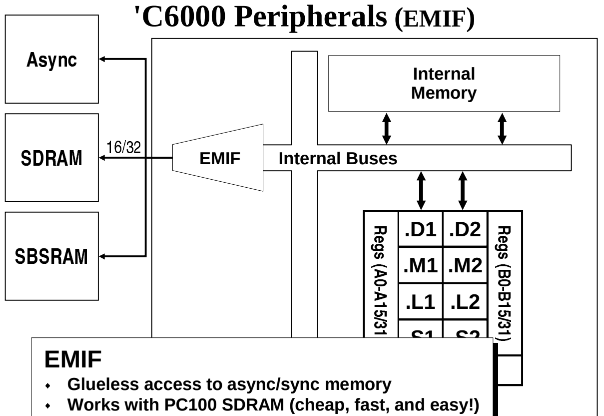
‘C6000 Internal Buses



'C6000 Peripherals



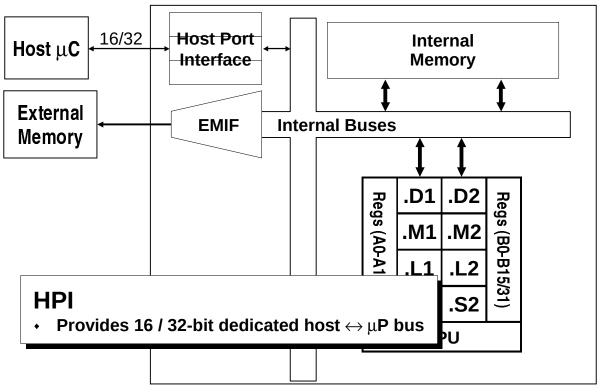
'C6000 Peripherals (EMIF)



EMIF

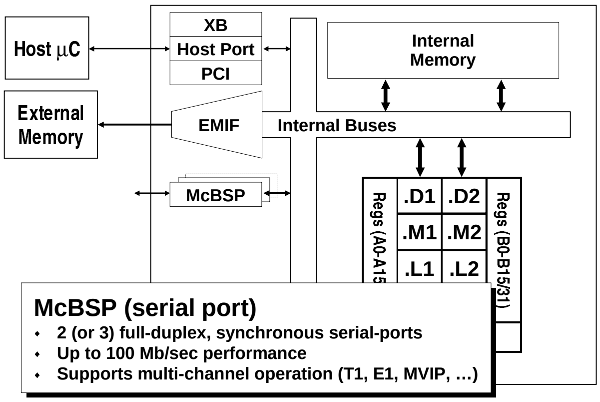
- Glueless access to async/sync memory
- Works with PC100 SDRAM (cheap, fast, and easy!)
- 8/16/32-bit data or 32-bit program access

’C6000 Peripherals (HPI/XB/PCI)



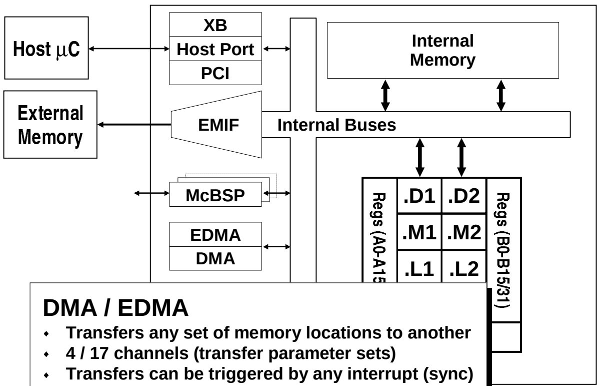
HPI
- Provides 16 / 32-bit dedicated host ↔ μP bus

’C6000 Peripherals (McBSP)



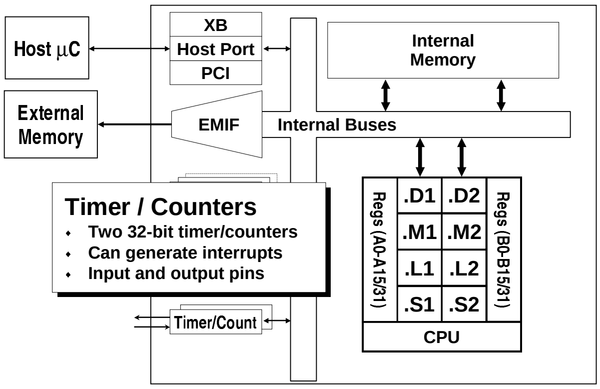
McBSP (serial port)
- 2 (or 3) full-duplex, synchronous serial-ports
- Up to 100 Mb/sec performance
- Supports multi-channel operation (T1, E1, MVIP, …)

’C6000 Peripherals (DMA/EDMA)



DMA / EDMA
- Transfers any set of memory locations to another
- 4 / 17 channels (transfer parameter sets)
- Transfers can be triggered by any interrupt (sync)
- Operates independent of CPU

’C6000 Peripherals (Timer/Counter)

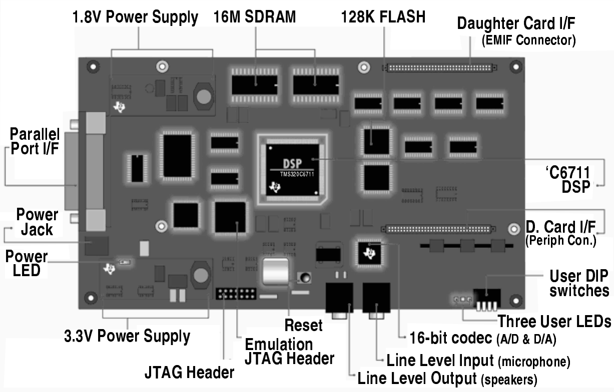


Timer / Counters
- Two 32-bit timer/counters
- Can generate interrupts
- Input and output pins

DSP Starter Kit (DSK)

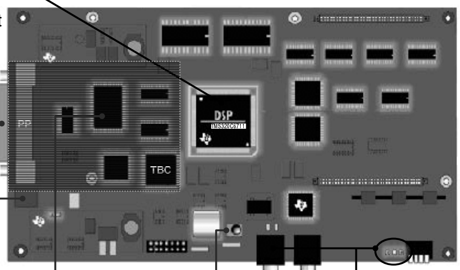
- **Hardware**
  - **150 MHz ‘C6711 DSP**
  - **TI 16-bit A/D Converter (‘AD535)**
  - **External Memory**
    - **16M Bytes SDRAM**
    - **128K Bytes Flash ROM**
  - **LED’s**
  - **Daughter card expansion**
  - **Power Supply & Parallel Port Cable**
- **Software**
  - **Code Generation Tools**
    (C Compiler, Assembler & Linker)
  - **Code Composer Debugger**
    (256K program limitation)
  - **Example Programs & S/W Utilities**
    - **Power-on Self Test**
    - **Flash Utility Program**
    - **Board Confidence Test**
    - **Host access via DLL**
    - **Sample Program(s)**

‘C6711 DSK Overview



DSK Resets

CCS Reset
- Menu selection Debug→DSP Reset
- Resets 'C6711 DSP
- Note: 'C6x11 reset causes bootload from FLASH



Apply Power
- POST runs

Absolute Reset
- Pull power jack
- On rare occasion also parallel port
- POST runs
- Resets full board (incl. emulation)

Emulation Reset
- ResetA.bat
- DSK Reset (Start menu)
- Resets emulation interface (PP, TBC)

Reset Pushbutton
- POST runs
- Doesn't reset emulation

POST
- Counts 1 - 7
- 4: ○○● mic input → spkr out
- 5: ●○● sinewave → spkr out
- Don't start CCS until all 3 LEDs flash

Memory Maps

Address	TMS320C6x11	DSK
0000_0000	64KB Internal (Progam or Data)	16MB SDRAM
0180_0000	On-chip Periph	128K byte FLASH; 4 byte I/O Port
8000_0000	256MB External	
9000_0000	256MB External	Available via Daughter Card Connector
A000_0000	256MB External	
B000_0000	256MB External	
FFFF_FFFF		

9008_0000
- LED's
- Switches
- DSK status
- DSK rev#
- Daughter Card

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

DSK Help