
EE 265
Digital Signal Processing Lab.

**Introduction to DSP
Architecture, Programming, and
the TI TMS320C55x**

Today's Agenda

- ◆ Course Overview
- ◆ Lab Overview
- ◆ TI DSP Architecture
- ◆ Lab1 Introduction
- ◆ Lab Demo

Course Overview

- ◆ This class is about practical DSP:
 - Practice DSP theory through implementing DSP algorithms efficiently on a DSP processor.
 - Understand various tradeoffs in DSP applications.
- ◆ Prerequisites:
 - DSP theory and programming skills.

The DSP Lab

- ◆ Packard 001 in the basement
- ◆ Access to the lab: Please submit the registration form.
- ◆ Computer account: Please subscribe to the mailing list, ee265@lists.stanford.edu .
- ◆ Please take good care of the equipment.

Lab Overview

- ◆ Lab1: lots of reading, know where to find what you need, implement a single convolution operation.
- ◆ Lab2: learn to configure the hardware; implement a real-time waveform generator and an audio recorder; verify using Matlab.
- ◆ Lab3: real-time filtering.
- ◆ Lab4: FFT, frequency-domain filtering.
- ◆ Lab5: up-sampling and down-sampling.
- ◆ Lab6: Viterbi decoding
- ◆ Final Project: OFDM receiver.

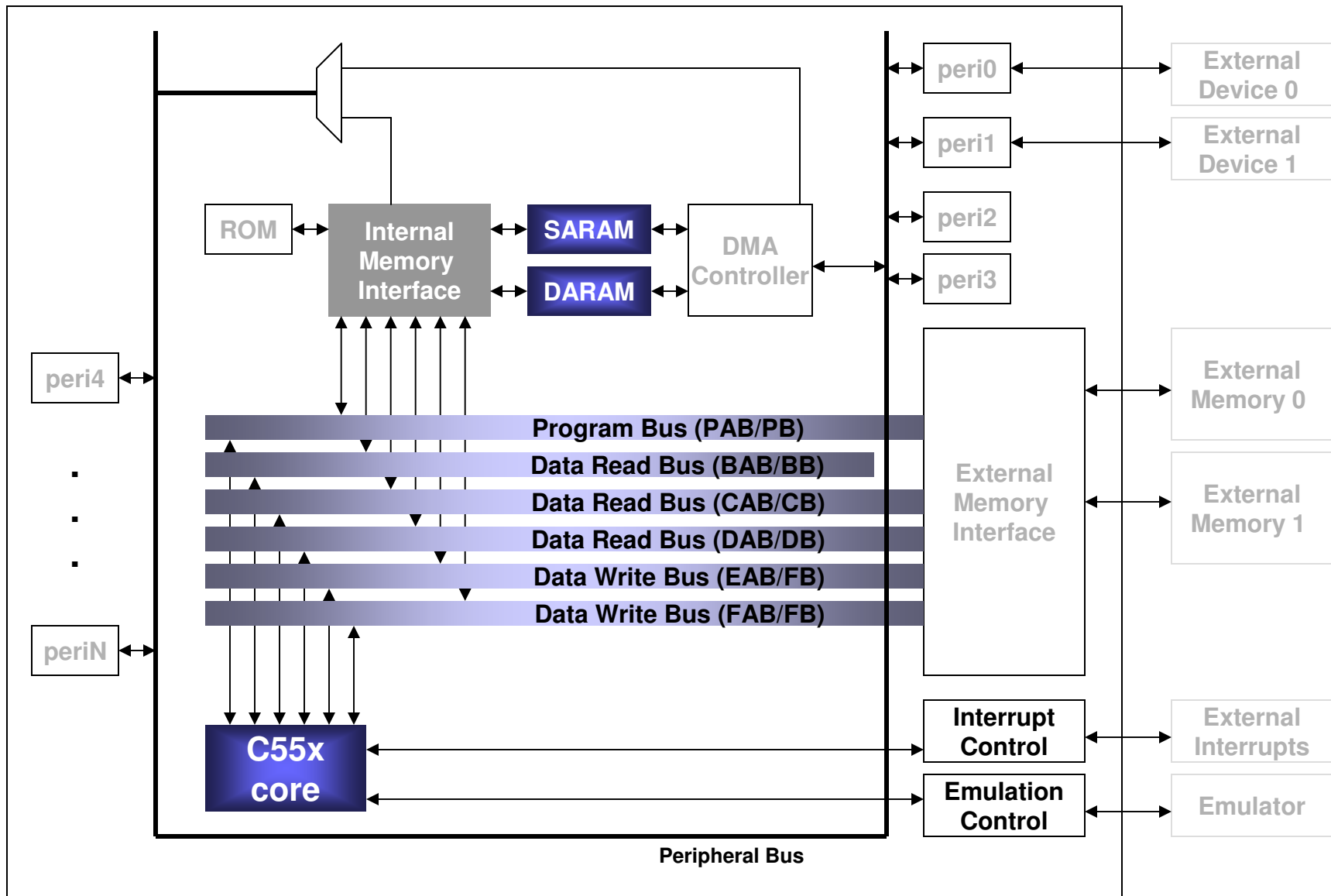
In the Lab

- ◆ Please help and get help from each other, but DO NOT COPY CODE.
- ◆ Lab demonstration
- ◆ Grading:
 - Functionality
 - Performance/code efficiency:
 - ◆ Cycle count; data size; code size.
 - Report

TI C5509A DSP

- ◆ TI C55x core
 - up to 200MHz clock speed
 - one/two instruction(s) executed per cycle
 - Variable instruction length; 1 to 6 bytes
 - Single cycle 17x17 dual MAC hardware (up to 400MMACS)
 - 40-bit ALU, can be split into 2x “16bit-ALU”
 - three data read buses and two data write buses
- ◆ Internal Memory
 - SARAM 192kB, DARAM 64 kB, 64kB on-chip ROM
- ◆ On-chip Peripherals
 - Timers, DMA Controllers, Serial Ports, ...
- ◆ What's the maximum number of FIR filter taps to process 48KHz stereo audio ?
 - $N = 200 \times 10^6 / 48000 = 41666.67$ Taps

TI C5509A DSP: 10km Overview

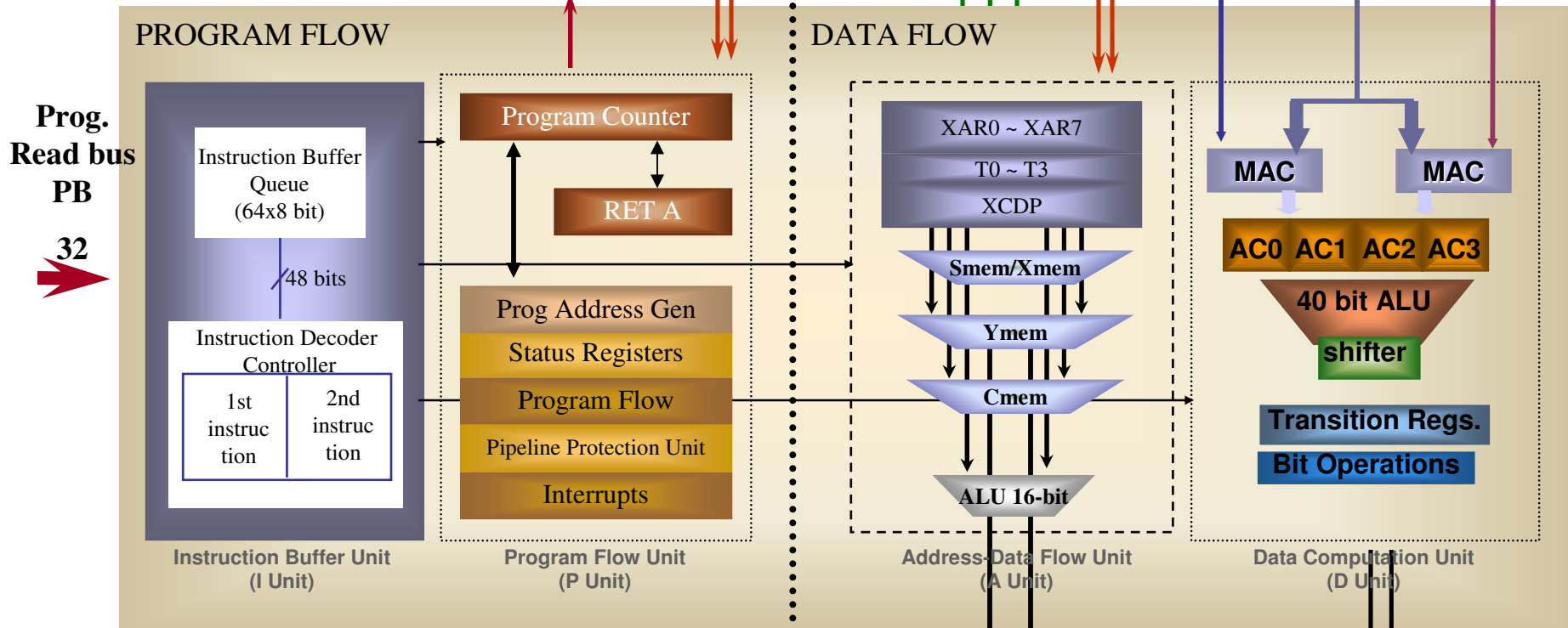


C55x Core Architectural Overview

Data Read buss: BB, CB, DB (3x16)

Data Read Address bus: BAB, CAB, DAB (3x23)

Program Address bus: PAB (24)



Data Write Address bus: EAB, FAB (2x23)

Data Write busses EB, FB (2x16)

Useful Registers

- ◆ 40 bit ACcumulator
 - AC0, AC1, AC2, AC3
 - Used for MACs and 40-bit ALU
- ◆ 23bit eXtended Auxiliary Registers
 - XAR0 – XAR7, XCDP
 - Used as pointers
 - ◆ Address generation deals with only lower 16bits
- ◆ 16bit Temporary registers
 - T0, T1, T2, T3
 - Used as index or temporary data store
- ◆ Others
 - XSP, XSSP, XDP, PDP
 - ◆ Stack pointers, direct addressing, I/O space access
 - TRN0, TRN1
 - ◆ For Viterbi specific instructions

Memory Model

- ◆ Three memory spaces – Program, Data, and I/O
- ◆ Program memory and Data memory are logically separated, but physically shared
 - Program memory space – 24bits, byte addressable
 - Data memory space – 23bits, word (2B) addressable
- ◆ No Virtual memory, No memory protection
- ◆ Not uniform memory access
 - Fast internal memory access – DARAM, SARAM
 - Slow external memory access
- ➔ Needs explicit memory mapping via linker command file

Memory Space

◆ Program / Data space

	Data-space addresses (Hexadecimal ranges)	Data/program memory	Program-space addresses (Hexadecimal ranges)
Main data page 0	MMRs 00 0000–00 005F		00 0000–00 00BF
	00 0060–00 FFFF		00 00C0–01 FFFF
Main data page 1	01 0000–01 FFFF		02 0000–03 FFFF
Main data page 2	02 0000–02 FFFF		04 0000–05 FFFF
⋮	⋮		⋮
Main data page 127	7F 0000–7F FFFF		FE 0000–FF FFFF

- ◆ I/O space – control registers of DSP peripheral
 - 16bit address, word addressable → total 64k words

C5509A Memory Map

Byte Addr	Program / Data Memory	Word Addr	
00_0000	MMR (192B)	00_0000	Memory Mapped Registers
00_00C0	DARAM (64kB-192B)	00_0060	Dual Accesses per cycle on each block 2k x 32bit DARAM block 8 blocks
01_0000	SARAM (192kB)	00_8000	Single Access per cycle on each block 2k x 32bit SARAM block 24 blocks
04_0000	External	02_0000	Very slow Asynchronous SRAM, Flash Memory, or Synchronous DRAM
FF_0000	ROM (64kB)	7F_0000	Boot code, some tables, ...
FF_FFFF		7F_FFFF	

Lab1 Introduction

- ◆ There are a lot of readings; so start early.
- ◆ Know where things are; so you can find what you need.
- ◆ Convolution: 80 FIR filter taps.
- ◆ Explore; get to know all the parts of a CCS project.
- ◆ Due on the next Thursday

Assembly Instructions

- ◆ Arithmetic
 - add, subtract, multiply
- ◆ Logic
 - bit-wise and/or/xor/test, shift
- ◆ Program Control
 - loop, branch, call/return, interrupt
- ◆ load, store and move between registers and memory

Simple C55x Assembly Exercise

```
x      .usect "vars",2  ; allocate 2 words
y0     .usect "vars",1  ; allocate 1 word

      .sect "tabl"
aa     .word 10h, 4     ; initialize aa array

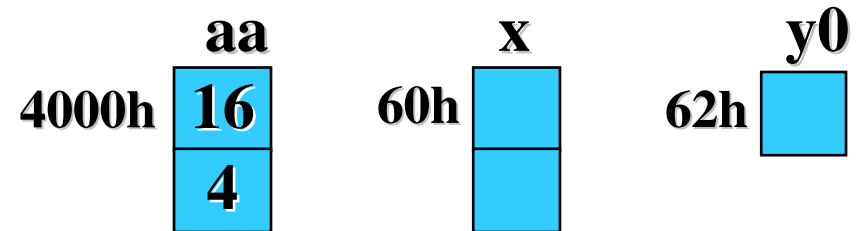
      .text

start
      AMOV #x, XAR2     ; init AR2 to 60h
      AMOV #aa, XAR3    ; init AR3 to 4000h

      MPY *AR2+, *AR3+, AC0 ;AC0 =aa[0]*x[0]
      MAC *AR2, *AR3, AC0  ;AC0 +=aa[1]*x[1]

      MOV AC0, *(y0)
      ; write the result to y0 (62h)
```

$$y0 = (aa0 * x0) + (aa1 * x1)$$



Assembler Directives

- ◆ **Column 1: comments or labels**
- ◆ **Comments: “;” in any column**

◆ text (Program Code Section)

```
.text
```

◆ Constants (Initialized Section)

```
.sect "?"
```

```
label .word 1,2,3,4
```

◆ Variables (Un-initialized Section)

```
label .usect "?", #words
```

How do we map these sections into memory?

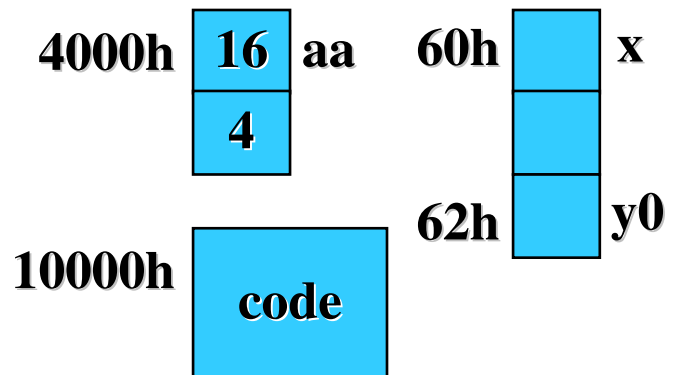
Linker Command File

```
-e start /* entry point */

/* define memory segment */
MEMORY
{
    PAGE 0:
    MMR      (R) :    o=0x000000 l=0x000c0
    DARAM1   (RWX) : o=0x0000c0 l=0x07F40
    DARAM2   (RWX) : o=0x008000 l=0x08000
    SARAM1   (RWX) : o=0x010000 l=0x08000
}

/* map each section to memory segment */
SECTIONS
{
    .text    :> SARAM1 PAGE 0
    .tbl     :> DARAM2 PAGE 0
    .vars    :> DARAM1 PAGE 0
}
```

- Map s/w section to appropriate physical memory
- Note that the addresses in linker command file are byte addresses
- Beware of DARAM, SARAM for the best performance



Note: it's byte address

Lab 1 assignment

- ◆ Implement 80-tap FIR filter in C55x assembly
 - Read Lab 1 web document
 - Start with the given template files
- ◆ Optimize it as fast as possible

FIR filtering in C

$$y_0 = \sum_{n=0}^{N-1} a_n \times x_n$$

```
short a[N]; // coefficients
short x[N]; // input
long y0;    // result
y0 = 0;
for (n=0; n<N; n++) {
    y0 += ((long)a[n]) * ((long)x[n]);
}
// Note on type cast before multiplication
```

Hints for Lab 1

Useful Assembly:

1. MOV
2. AMOV
3. MPYM
4. MACM
5. RPT
6. RPTB
7. RPTBLOCAL
8. ADD
9. dual-MAC instruction
10. FIRSADD
11. dbl()
12. Parallel execution | |

Useful Assembler Directives:

1. .usect
2. .sect
3. .word
4. .set
5. .global
6. .mmregs
7. .text

FIR filtering in assembly

```
; 1. Initialize XAR registers
```

```
AMOV #coefDATA, XAR1 ; XAR1 = #coefDATA
```

```
AMOV #dataHead, XAR2 ; XAR2 = #dataHead
```

```
AMOV #output, XAR3 ; XAR3 = #output
```

```
; 2. Initialize accumulator
```

```
MOV #0, AC0 ; AC0 = 0
```

```
; 3, Perform 80 times MAC operations
```

```
MACM *AR1+, *AR2+, AC0 ;AC0 += (*AR1++) * (*AR2++)
```

```
MACM *AR1+, *AR2+, AC0 ;AC0 += (*AR1++) * (*AR2++)
```

```
MACM *AR1+, *AR2+, AC0 ;AC0 += (*AR1++) * (*AR2++)
```

```
.
```

```
.
```

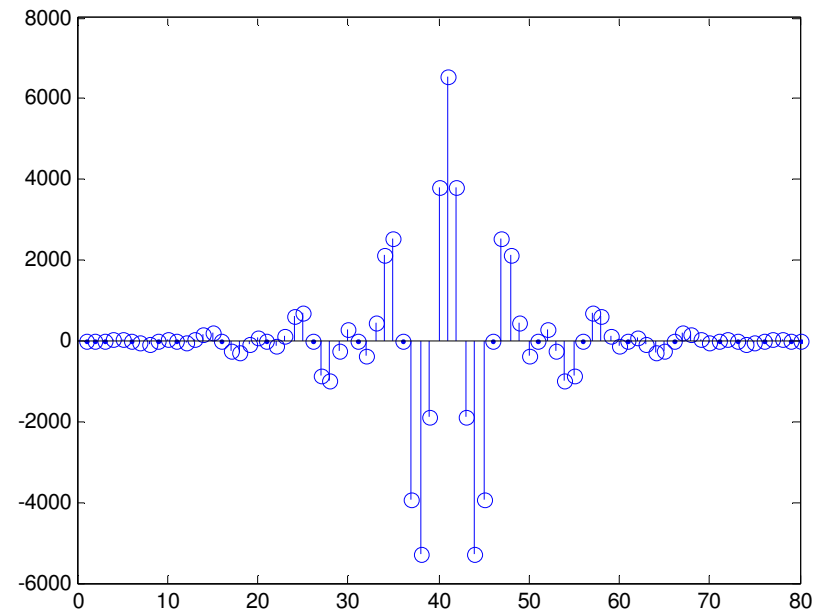
```
< 80 MACM instructions >
```

```
; 4. Store the result
```

```
MOV AC0, dbl(*AR3) ; *AR3 = AC0, 32bit store
```

More optimizations

- ◆ Use repeat instruction instead of writing MACM 80 times
- ◆ Use Parallel (||) to execute two instructions simultaneously
- ◆ MPY doesn't need accumulator initialization
- ◆ Look at the coefficients
 - zeros
 - symmetry



More readings

- ◆ TMS320 VC5509A Fixed-Point Digital Signal Processor Data Manual
 - Chapter 1: Feature
 - Chapter 3.1: Memory → Memory map of C5509A processor
- ◆ TMS320 C55x DSP CPU Reference Guide
- ◆ TMS320 C55x DSP Mnemonic Instruction Set Reference Guide
 - references of individual instruction
- ◆ TMS320 C55x Assembly Language Tools User's Guide
 - Chapter 4: Assembler directive
 - Chapter 8.6: Linker Command File