SIGNAL AND IMAGE PROCESSING ON THE TMS320C54x DSP

Prof. Brian L. Evans

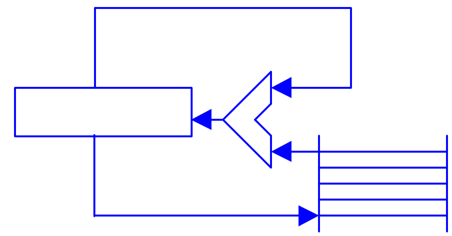
in collaboration with

Niranjan Damera-Venkata and
Wade Schwartzkopf

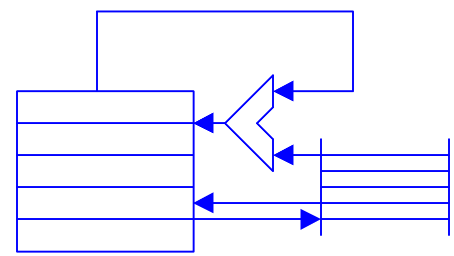
Embedded Signal Processing Laboratory
The University of Texas at Austin
Austin, TX 78712-1084

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

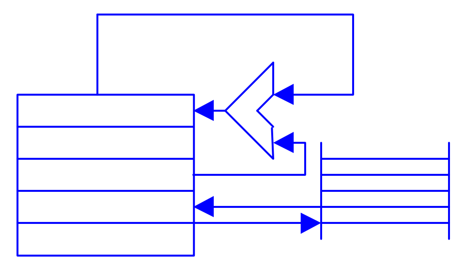
Accumulator architecture



Memory-register architecture



Load-store architecture

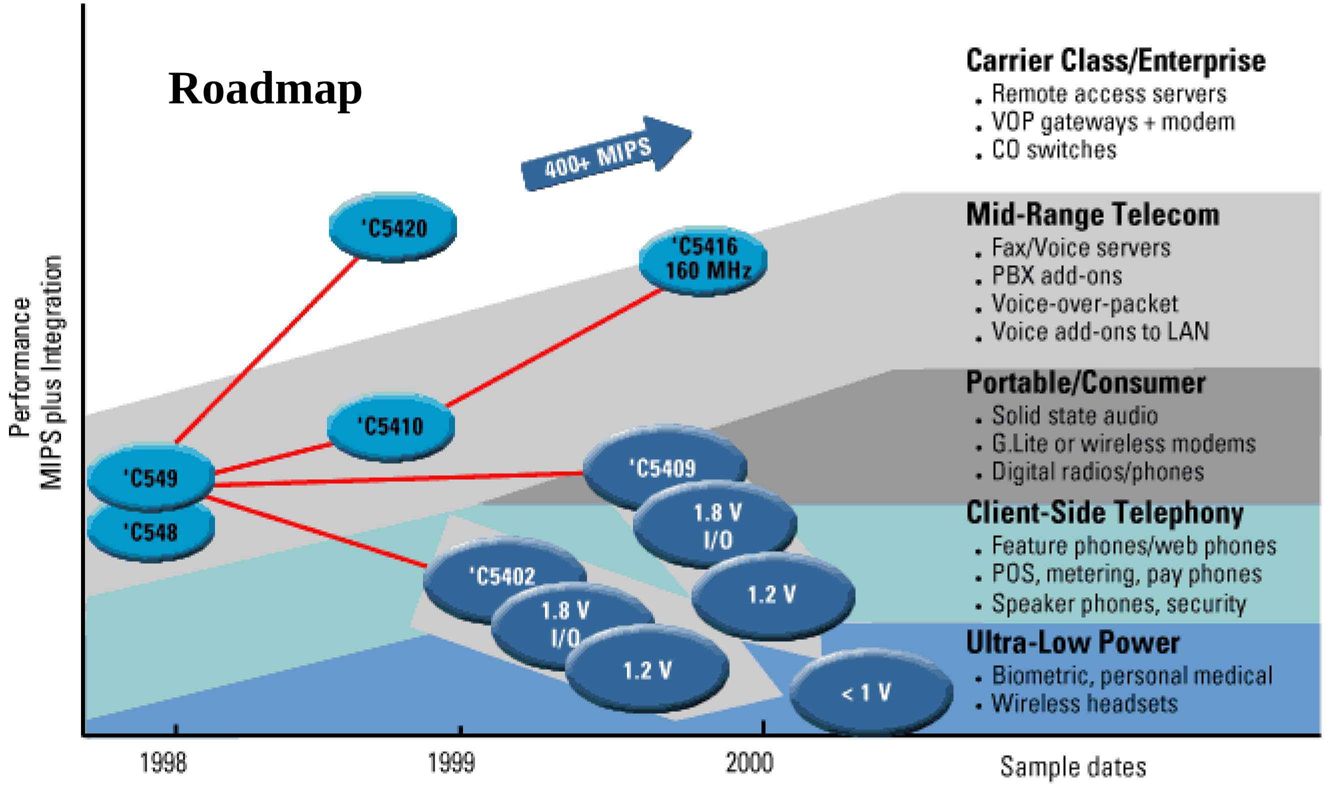


Outline

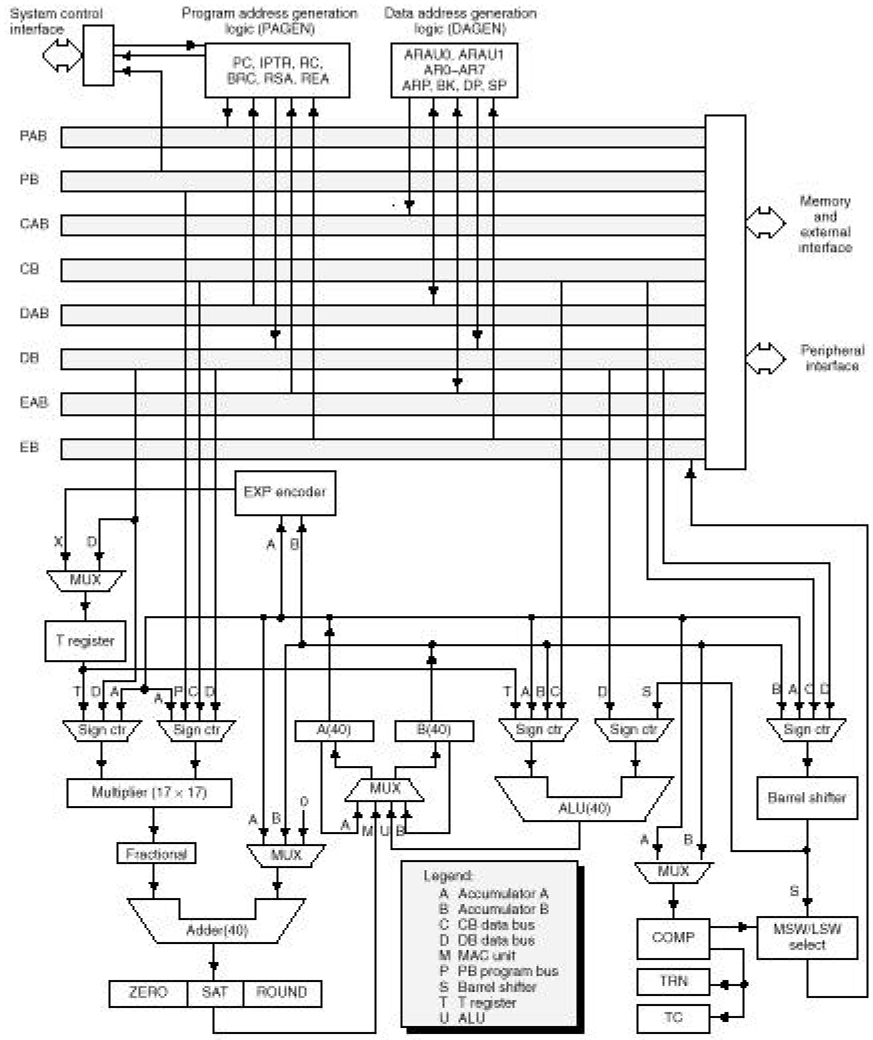
- Introduction
- Instruction set architecture
- Vector dot product example
- Pipelining
- Algorithm acceleration
- C compiler
- Development tools and boards
- Conclusion

Introduction to TMS320C54x

- Lowest DSP in power consumption: 0.54 mW/MIP
- Acceleration for FIR and LMS filtering, code book search, polynomial evaluation, Viterbi decoding



Instruction Set Architecture



Instruction Set Architecture

- Conventional 16-bit fixed-point DSP
  - 8 16-bit auxiliary/address registers (ar0-7)
  - Two 40-bit accumulators (a and b)
  - One 16 bit x 16 bit multiplier
  - Accumulator architecture
- Four busses (may be active each cycle)
  - Three read busses: program, data, coefficient
  - One write bus: writeback
- Memory blocks
  - ROM in 4k blocks
  - Dual-access RAM in 2k blocks
  - Single-access RAM in 8k blocks
- Two clock cycles per instruction cycle

C54x Addressing Modes

- Immediate
  - Operand is part of the instruction

```
ADD #0FFh
```

- Absolute
  - Address of operand is part of the instruction

```
LD *(LABEL), A
```

- Register
  - Operand is specified in a register

```
READA DATA
;(data read
from address in
accumulator A)
```

C54x Addressing Modes

- Direct
  - Address of operand is part of the instruction (added to implied memory page)
- Indirect
  - Address of operand is stored in a register
  - Offset addressing
  - Register offset (ar1+ar0)
  - Autoincrement/decrement
  - Bit reversed addressing
  - Circular addressing

```
ADD 010h,A

ADD *AR1
ADD *AR1(10)
ADD *AR1+0
ADD *AR1+
ADD *AR1+B
ADD *AR1+0B
```

Program Control

- **Conditional execution**
  - **XC** *n*, *cond* [, *cond* [, *cond*]] ; 23 possible conditions
  - Executes next *n* (1 or 2) words if conditions (*cond*) are met
  - Takes one cycle to execute

```
xc      1,ALEQ              ; test for accumulator a ≤ 0
mac     *ar1+,*ar2+,a       ; perform MAC only if a ≤ 0
add     #12,a,a             ; always perform add
```

- **Repeat single instruction or block**
  - Overhead: 1 cycle for RPT/RPTZ and 4 cycles for RPTB
  - Hardware loop counters count down

```
rptz    a,#39       ; zero accumulator a
                    ; repeat next instruction 40 times
mac     *ar2+,*ar3+,a ; a += a(n) * x(n)
```

Special Arithmetic Functions

- Scalar arithmetic
  - ABS Absolute value
  - SQUR Square
  - POLY Polynomial evaluation
- Vector arithmetic acceleration
  - Each instruction operates on one element at at time
  - ABDIST Absolute difference of vectors
  - SQDIST Squared distance between vectors
  - SQURA Sum of squares of vector elements
  - SQURS Difference of squares of vector elements

```
rptz   a,#39          ; zero accumulator a, repeat next
                      ; instruction over 40 elements
squra  *ar2+,a        ; a += x(n)^2
```

C54X Instructions Set by Category

Arithmetic	***Logical***	***Program Control***	***Application Specific***
ADD	AND	B	ABS
MAC	BIT	BC	ABDST
MAS	BITF	CALL	DELAY
MPY	CMPL	CC	EXP
NEG	CMPM	IDLE	FIRS
SUB	OR	INTR	LMS
ZERO	ROL	NOP	MAX
	ROR	RC	MIN
	SFTA	RET	NORM
	SFTC	RPT	POLY
	SFTL	RPTB	RND
	XOR	RPTZ	SAT
		TRAP	SQDST
		XC	SQUR
			SQURA
			SQURS

Data Management
LD
MAR
MV(D,K,M,P)
ST

Notes

CMPL *complement* MAR *modify address reg.*

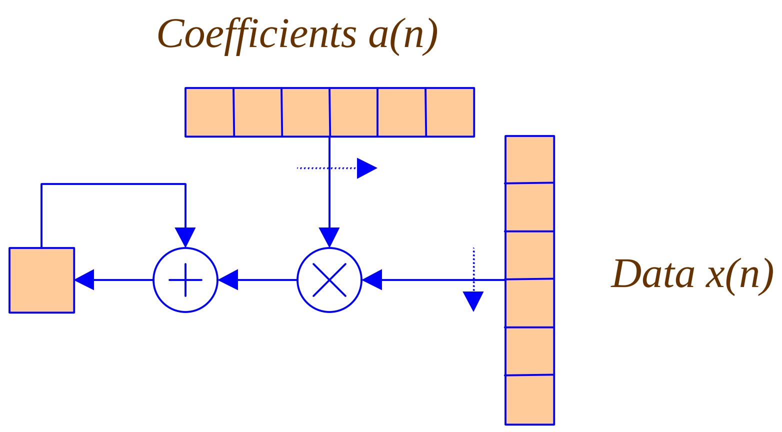
CMPM *compare memory* MAS *multiply and subtract*

Example: Vector Dot Product

- A vector dot product is common in filtering

$$Y = \sum_{n=0}^{N-1} a(n)\ x(n)$$

- Store $a(n)$ and $x(n)$ into an array of N elements
- C54x performance: N cycles



Example: Vector Dot Product

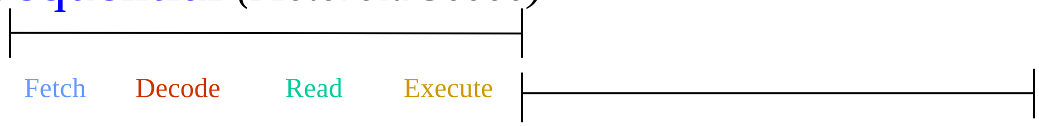
- Prologue
  - Initialize pointers: ar2 for $a(n)$ and ar3 for $x(n)$
  - Set accumulator (A) to zero
- Inner loop
  - Multiply and accumulate $a(n)$ and $x(n)$
- Epilogue
  - Store the result into Y

Reg	Meaning
AR2	$\&a(n)$
AR3	$\&x(n)$
A	Y

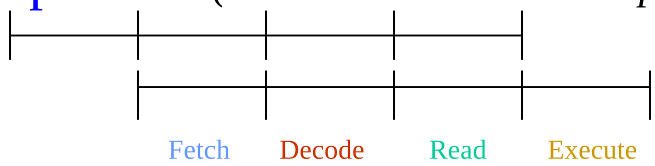
```
; Initialize pointers ar2 and ar3 (not shown)
rptz   a,#39             ; zero accumulator a
                         ; repeat next instruction 40 times
mac    *ar2+,*ar3+,a     ; a += a(n) * x(n)
sth    a,#Y              ; store result in Y
```

Pipelining

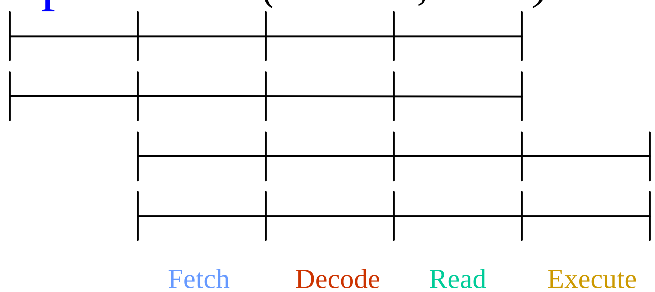
Sequential *(Motorola 56000)*



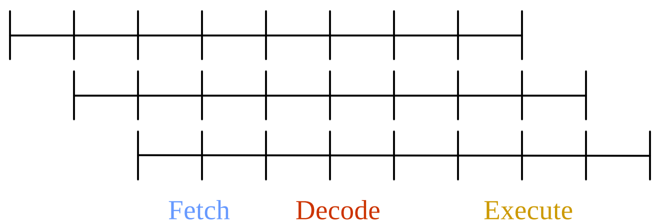
Pipelined *(Most conventional DSP processors)*



Superscalar *(Pentium, MIPS)*



Superpipelined *(TMS320C6x)*



Managing Pipelines

- compiler or programmer (TMS320C6x and C54x)
- pipeline interlocking in processor (TMS320C3x)
- hardware instruction scheduling

TMS320C54x Pipeline

- Six-stage pipeline
  - *Prefetch*: load address of next instruction onto bus
  - *Fetch*: get next instruction
  - *Decode*: decode next instruction to determine type of memory access for operands
  - *Access*: read operands address
  - *Read*: read data operand(s)
  - *Execute*: write data to bus
- Instructions
  - 1-3 words long (most are one word long)
  - 1-6 cycles to execute (most take one cycle) not counting external (off-chip) memory access penalty

TMS320C54x Pipeline

- Instructions affecting pipeline behavior
  - Delayed branches (BD), calls (CALLD), and returns (RETD)
  - Conditional branches (BC), execution (XC), and returns (RC)
- No hardware protection against pipeline hazards
  - Compiler and assembler must prevent pipeline hazards
  - Assembler/linker issues warnings about potential pipeline hazards

Block FIR Filtering

- $y[n] = h_0\, x[n] + h_1\, x[n-1] + \ldots + h_{N-1}\, x[n-(N-1)]$
  - h stored as linear array of N elements (in prog. mem.)
  - x stored as circular array of N elements (in data mem.)

```
; Addresses: a4 h, a5 N samples of x, a6 input buffer, a7 output buffer
; Modulo addressing prevents need to reinitialize regs each sample
; Moving filter coefficients from program to data memory is not shown
firtask:  ld     #firDP,dp  ; initialize data page pointer
          stm    #frameSize-1,brc     ; compute 256 outputs
          rptbd  firloop-1
          stm    #N,bk                ; FIR circular buffer size
          ld     *ar6+,a              ; load input value to accumulator b
          stl    a,*ar4+%  ; replace oldest sample with newest
          rptz   a,#(N-1)             ; zero accumulator a, do N taps
          mac    *ar4+0%,*ar5+0%,a    ; one tap, accumulate in a
          sth    a,*ar7+              ; store y[n]
firloop:  ret
```

Accelerating Symmetric FIR Filtering

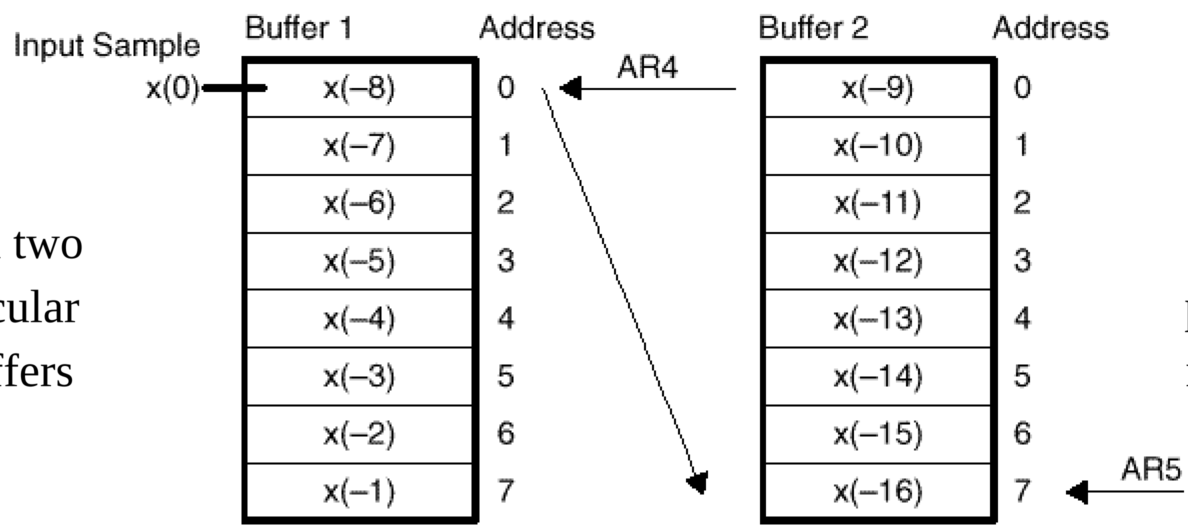
- Coefficients in linear phase filters are either symmetric or anti-symmetric
- Symmetric coefficients

$$y[n] = h_0\, x[n] + h_1\, x[n-1] + h_1\, x[n-2] + h_0\, x[n-3]$$

$$y[n] = h_0\, (x[n] + x[n-3]) + h_1\, (x[n-1] + x[n-2])$$

- Accelerated by FIRS (FIR Symmetric) instruction

x in two circular buffers

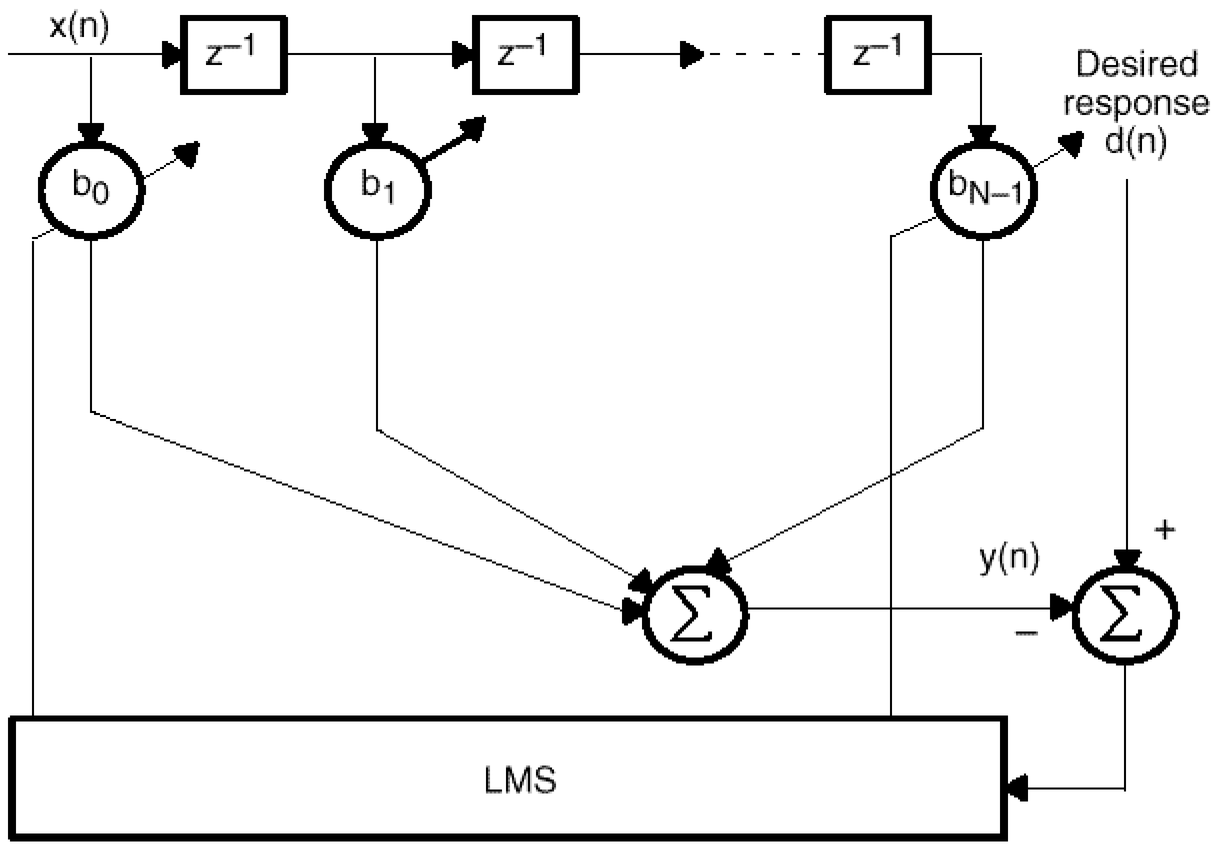


h in program memory

Accelerating Symmetric FIR Filtering

```
; Addresses:  a6 input buffer, a7 output buffer
;   a4 array with x[n-4], x[n-3], x[n-2], x[n-1] for N = 8
;   a5 array with x[n-5], x[n-6], x[n-7], x[n-8] for N = 8
; Modulo addressing prevents need to reinitialize regs each sample
firtask:  ld     #firDP,dp  ; initialize data page pointer
          stm    #frameSize-1,brc     ; compute 256 outputs
          rptbd  firloop-1
          stm    #N/2,bk              ; FIR circular buffer size
          ld     *ar6+,b              ; load input value to accumulator b
          mvdd   *ar4,*a5+0%          ; move old x[n-N/2] to new x[n-N/2-1]
          stl    b,*ar4%              ; replace oldest sample with newest
          add    *a4+0%,*a5+0%,a      ; a = x[n] + x[n-N/2-1]
          rptz   b,#(N/2-1) ; zero accumulator b, do N/2-1 taps
          firs   *ar4+0%,*ar5+0%,coeffs         ; b += a * h[i], do next a
          mar    *+a4(2)%  ; to load the next newest sample
          mar    *ar5+%               ; position for x[n-N/2] sample
          sth    b,*ar7+
firloop:  ret
```

Accelerating LMS Filtering



- Adapt weights: $b_k(i+1) = b_k(i) + 2\,\beta\, e(i)\, x(i-k)$
- Accelerated by the LMS instruction (2 cycles/tap)

Accelerating LMS Filtering

```
adapt_task:
    STM     #H_FILT_SIZE,BK              ; first circular buffer size
    STM     #hcoff,H_COFF_P              ; H_COFF_P --> last of sys coeff
    ADDM    #1,d_adapt_count
    LD      *INBUF_P+, A                 ; load the input sample
    STM     #wcoff,W_COFF_P              ; reset coeff buffer
    STL     A,d_new_x                    ; read in new data
    LD      d_new_x,A                    ;
    STL     A,*XH_DATA_P+0%              ; store in the buffer
    RPTZ    A,#H_FILT_SIZE-1             ; Repeat 128 times
    MAC     *H_COFF_P+0%,*XH_DATA_P+0%,A  ; mult & acc:a = a + (h * x)
    STH     A,d_primary                  ; primary signal
;   start simultaneous filtering and updating the adaptive filter here.
    LD      d_mu_e,T                     ; T = step_size*error
    SUB     B,B                          ; zero acc B
    STM     #(ADPT_FILT_SIZE-2),BRC      ; set block repeat counter
    RPTBD   lms_end-1
    MPY     *XW_DATA_P+0%, A             ; error * oldest sample
    LMS     *W_COFF_P, *XW_DATA_P        ; B = filtered output (y)
                                         ; Update filter coeff
    ST      A, *W_COFF_P+                ; save updated filter coeff
    || MPY *XW_DATA_P+0%,A               ; error *x[n-(N-1)]

    LMS *W_COFF_P, *XW_DATA_P            ; B = accum filtered output y
                                         ; Update filter coeff
lms_end
```

Accelerating Polynomial Evaluation

- Function approximation and spline interpolation
- Fast polynomial evaluation (*N* coefficients)
  - $y(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$ *Expanded form*
  - $y(x) = c_0 + x (c_1 + x (c_2 + x (c_3)))$ *Horner's form*
  - POLY reduces 2 *N* cycles using MAC+ADD to *N* cycles

```
; ar2 contains address of array [c3 c2 c1 c0]
; poly uses temporary register t for multiplicand x
; first two times poly instruction executes gives
;   1. a = c(3) + x * 0 = c(3); b = c2
;   2. a = c(2) + x * c(3);     b = c1
       ld   *ar2+,16,b      ; b = c3 << 16
       ld   *ar3,t    ; t = x (ar3 contains addr of x)
       rptz a,#3      ; a = 0, repeat next inst. 4 times
       poly *ar2+     ; a = b + x*a || b = c(i-1) << 16
       sth  a,*ar4    ; store result (ar4 is addr of y)
```

C54x optimizing C compiler

- ANSI C compiler
  - Instrinsics, in-line assembly and functions, pragmas

Selected Pragmas	CODE_SECTION	code section
	DATA_SECTION	data section
	FUNC_IS_PURE	no side effects
	INTERRUPT	specifies interrupt routine
	NO_INTERRUPT	cannot be interrupted

- Cl500 shell program contains
  - C Compiler: parser, optimizer, and code generator
  - Assembler: generates a relocatable (COFF) object file
  - Linker: creates executable object file

Optimizing C Code

- Level 0 optimization: -o0 flag
  - Performs control-flowgraph simplifications
  - Allocates variables to registers
  - Eliminates unused code
  - Simplifies expressions and statements
  - Expands inline function calls
- Level 1 optimization: -o1 flag
  - Performs local copy/constant propagation
  - Removes unused assignments
  - Eliminates local common expressions

Optimizing C Code

- Level 2 optimization: -o2 flag
  - Performs loop optimizations
  - Eliminates global common sub-expressions
  - Eliminates global unused assignments
  - Performs loop unrolling
- Level 3 optimization: -o3 flag
  - Removes all functions that are never called
  - Performs file-level optimization
  - Simplifies functions with unused return values
- Program-level optimization: -pm flag

Compiler Optimizations

- Cost-based register allocation
- Alias disambiguation
  - Aliasing memory prevents compiler from keeping values in registers
  - Determines when 2 pointers cannot point to the same location, allowing compiler to optimize expressions
- Branch optimizations
  - Analyzes branching behavior and rearranges code to remove branches or remove redundant conditions

Compiler Optimizations

- Copy propagation
  - Following an assignment compiler replaces references to a variable with its value
- Common sub-expression elimination
  - When 2 or more expressions produce the same value, the compiler computes the value once and reuses it
- Redundant assignment elimination
  - Redundant assignment occur mainly due to the above two optimizations and are completely eliminated

Compiler Optimizations

- Expression simplification
  - Compiler simplifies expressions to equivalent forms requiring fewer instructions/registers

```
/* Expression Simplification*/
g = (a + b) - (c + d);      /* unoptimized */
g = ((a + b) - c) - d;      /* optimized */
```

- Inline expansion
  - Replaces calls to small run-time support functions with inline code, saving function call overhead

Compiler Optimizations

- Induction variables
  - Loop variables whose value directly depends on the number of times a loop executes
- Strength reduction
  - Loops controlled by counter increments are replaced by repeat blocks
  - Efficient expressions are substituted for inefficient use of induction variables (e.g., code that indexes into an array is replaced with code that increments pointers)

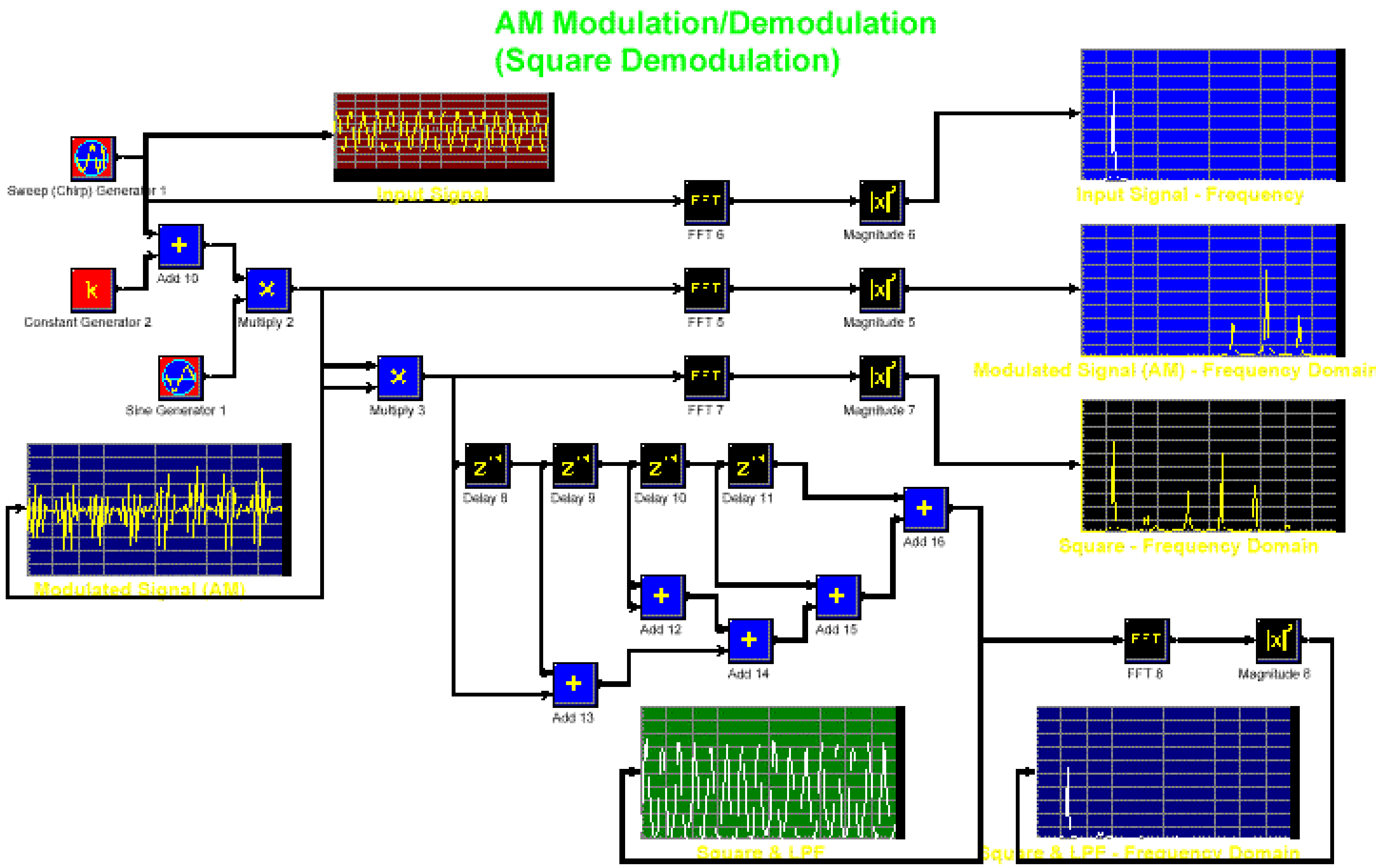
Compiler Optimizations

- Loop-invariant code motion
  - Identifies expressions within lops that always compute the same value, and the computation is moved to the front of the loop as a precomputed expression
- Loop rotation
  - Evaluates loop conditionals at the bottom of loop
- Auto-increment addressing
  - Converts C-increments into efficient address-register indirect access

Hypersignal Block Diagram Environments

- Hierarchical block diagrams (dataflow modeling)
  - Block is defined by dynamically linked library function
  - Create new blocks by using a design assistant GUI
- RIDE for graphical real-time debugging/display
  - 1-D, multirate, and m-D signal processing
  - ANSI C source code generator
  - C54x boards: support planned for 4Q99
  - C6x boards: DNA McEVM, Innovative Integration, MicroLAB TORNADO, and TI EVM
- OORVL DSP Graphical Compiler
  - Generates DSP assembly code (C3x and C54x)

Hypersignal RIDE Environment



Download demonstration software from http://www.hyperception.com

Hypersignal RIDE Image Processing Library

Category	Blocks
Image arithmetic	Add, subtract, multiply, exponentiate
Image generation	Grayscale, noise, sprite
Image I/O	AVI, bitmaps, raw images, video capture
Image display	Bitmaps, RGB
Edge detection	Isotropic, Laplace, Prewitt, Roberts, Sobel
Line detection	Horizontal, 45°, vertical, 135°
1-D filtering	Convolution, DFT, FFT, FIR, IIR,
2-D filtering	DFT, FFT, FIR
Nonlinear filtering	Max, median, min, rank order, threshold
Histograms	Histograms, histogram equalization
Manipulation	Contrast, flip, negate, resize, rotate, zoom
Object-based	Object count, object tracking
Networking	Internet transmit, Internet receive

Same as ImageDSP and Advanced Image Processing Library

TI C54x Evaluation Module (EVM) Board

- Offered through TI and Spectrum Digital
  - 100 MHz C549 & 100 MHz C5410 for under $1,000
  - Memory: 192 kwords program, 64 kwords data
  - Single/multi-channel audio data acquisition interfaces
  - Standard JTAG interface (used by debugger)
  - Spectrum sells 100 MHz C5402 & 66 MHz C548 EVMs
- Software features
  - Compatible with TI Code Composer Studio
  - Supports TI C debugger, compiler, assembler, linker

http://www.ti.com/sc/docs/tools/dsp/c5000developmentboards.html

Sampling of Other C54 Boards

Vendor	Board	RAM	ROM	Processor	I/O
Kane Computing	KC542/ PC	256 kb	256 kb	40-MIP C5402	16-bit stereo
Innovative Integration	SBC54			100-MIP C549	Modular
DSP Research	Tiger 549/PC	256 kb	256 kb	100-MIP C549	
DSP Research	Tiger 5410/PC	256 kb	256 kb	100-MIP C5410	
Odin Telesystems	VIDAR 5x4PCI	2 Mb	0 kb	four 80-MIP C548	
DSP Research	Viper-12 549/PC	12 Mb	0 kb	12 100-MIP C549	

http://www.ti.com/sc/docs/tools/dsp/c5000developmentboards.html

Binary-to-Binary Translation

- Many of today's DSP systems are implemented using the TI C5x DSP (e.g. voiceband modems)
  - TI is no longer developing members of C5x family in favor of the C54x family
  - 3Com has shipped over 35 million modems with C5x
- C5x binaries are incompatible with C54x
  - Significant architectural differences between them
  - Need for automatic translator of binary C5x code to binary C54x code
- Solutions for binary-to-binary translation
  - Translation Assistance Program 5000 from TI
  - C50-to-C54 translator from UT Austin
  - Both provide assistance for cases they cannot handle

TI Translation Assistant Program 5000

- Assists in translating C5x code to C54x code
  - Makes many assumptions about code being translated
  - Requires a significant amount of user interaction
  - Free evaluation for 60 days from TI Web site
- Static assembler to assembler translation
  - Generates automatic translation when possible
  - Twenty situations are not automatically translated: user must intervene
  - Many other situation result in inefficient code
  - Warns user when translation difficulty is encountered
  - Analyzes prior translations

http://www.ti.com/sc/docs/tools/dsp/tap5000freetool.html

Conclusion

- C54x is a conventional digital signal processor
  - Separate data/program busses (3 reads & 1 write/cycle)
  - Extended precision accumulators
  - Single-cycle multiply-accumulate
  - Saturation and wraparound arithmetic
  - Bit-reversed and circular addressing modes
  - Highest performance vs. power consumption/cost/vol.
- C54x has instructions to accelerate algorithms
  - Communications: FIR & LMS filtering, Viterbi decoding
  - Speech coding: vector distances for code book search
  - Interpolation: polynomial evaluation

Conclusion

- C54x reference set
  - *Mnemonic Instruction Set*, vol. II, Doc. SPRU172B
  - *Applications Guide*, vol. IV, Doc. SPRU173. Algorithm acceleration examples (filtering, Viterbi decoding, etc.)
- C54x application notes
  http://www.ti.com/sc/docs/apps/dsp/tms320c5000app.html
- C54x source code for applications and kernels
  http://www.ti.com/sc/docs/dsps/hotline/wizsup5xx.htm
- Other resources
  - comp.dsp newsgroup: FAQ www.bdti.com/faq/dsp_faq.html
  - embedded processors and systems: www.eg3.com
  - on-line courses and DSP boards: www.techonline.com
  - DSP course: http://www.ece.utexas.edu/~bevans/courses/realtime/