Texas Instruments

TMS320C240x 1-Day Workshop

Student Guide

DSP24 1-Day Version 2.0
May 2001

Technical Training

Copyright © 1999, 2000, 2001 Texas Instruments Incorporated.
All rights reserved.

Notice

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of Texas Instruments.

Texas Instruments reserves the right to update this Guide to reflect the most current product information for the spectrum of users. If there are any differences between this Guide and a technical reference manual, references should always be made to the most current reference manual. Information contained in this publication is believed to be accurate and reliable. However, responsibility is assumed neither for its use nor any infringement of patents or rights of others that may result from its use. No license is granted by implication or otherwise under any patent or patent right of Texas Instruments or others.

Revision History

August 1999, Version 1.0

January 2000, Version 1.1

May 2000, Version 1.2

May 2001, Version 2.0

TMS320C240x DSP 1-Day Workshop

eZdsp™ LF2407 Starter Kit

Copyright © 1999-2001 Texas Instruments Incorporated eZdsp is a trademark of Spectrum Digital, Inc.

Agenda

- **M1: C240x 1-Day Workshop Introduction (30 min)**
- **M2: eZdsp™ LF2407 Kit (60 min)**
- **M3: TMS320C240x Architecture (90 min)**
- **Lunch (60 min)**
- **M4: Analog-to-Digital Converter (ADC) (75 min)**
- **M5: PWM Generation & Event Manager (75 min)**
- **M6: Filtering Techniques (75 min)**
- **M7: What's Next? (15 min)**

This page is intentionally blank.

Introduction

Learning Objectives

Learning Objectives

- **Learn to use the eZdsp™ LF2407 Starter Kit**
- **Understand the basic architecture and peripherals of the LF2407 and family devices**
- **Understand DSP in control applications**
- **Learn how to take the next development step**

Module Topics

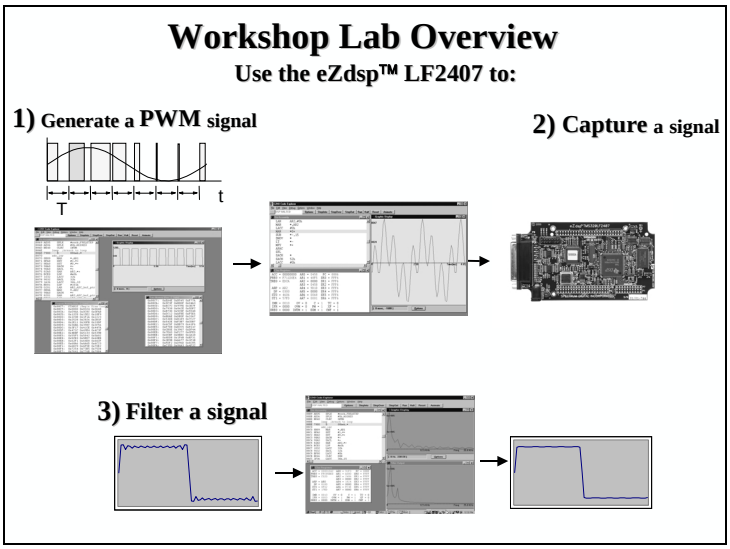
Introduction **1-1**

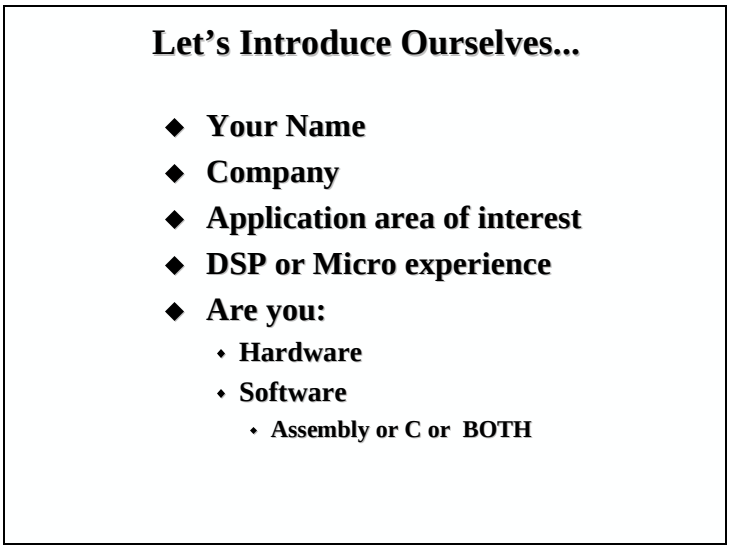
Module Topics *1-2*

Workshop Introduction *1-3*

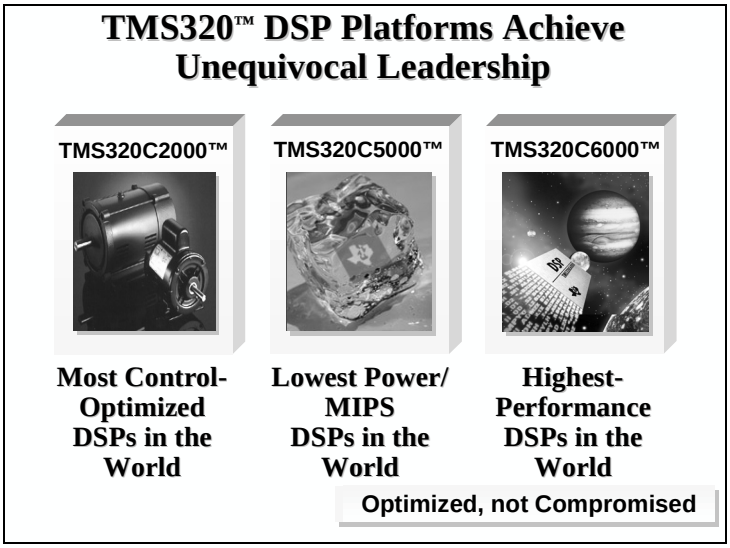
TMS320C24xx Device Family 1-4

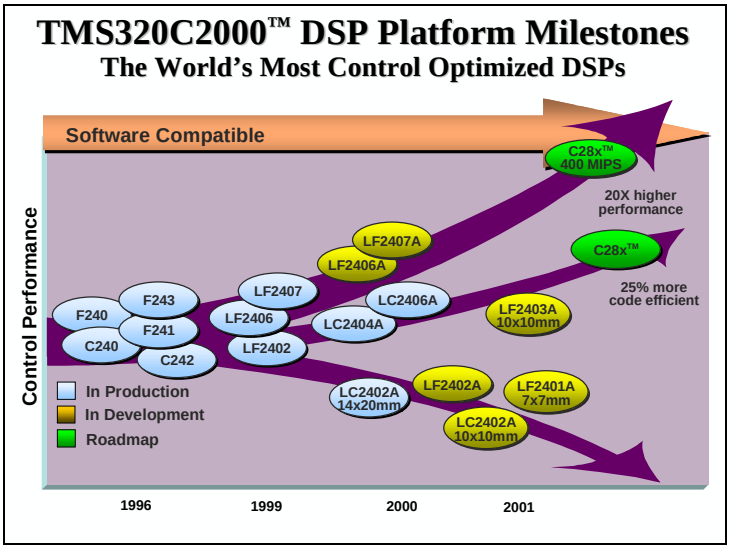
Workshop Introduction



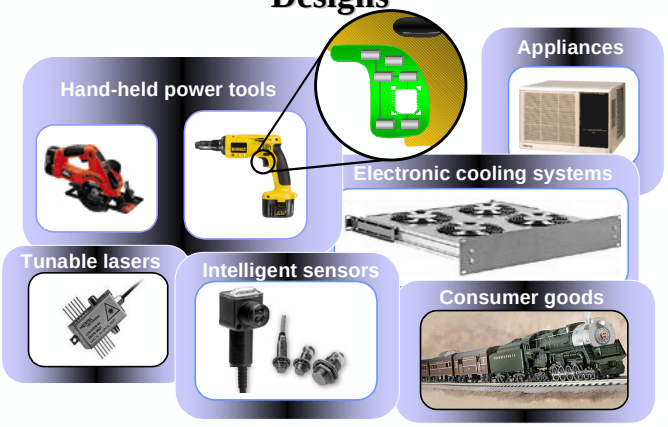


TMS320C24xx Device Family





TI Enables OEMs to Develop More Compact, Intelligent and Robust Digital Control Designs



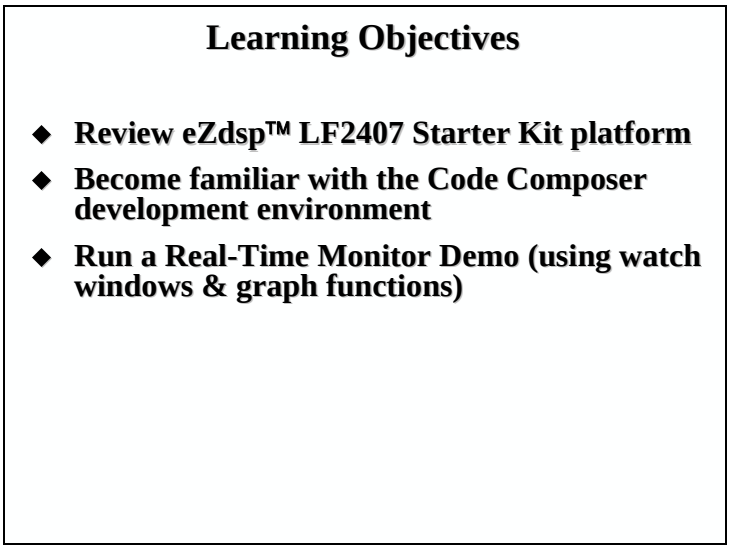
A Broad Portfolio to Fit Every Need

		Memory				Event Manager		10 bit ADC				Comm Ports						2001
Device	MIPS	ROM	Boot ROM	RAM	Flash	Timers	PWM Chan	# of Chans	Conv Time	EMIF	WD Timer	SPI	SCI	CAN	# of I/O	Volts	Package	10K/yr Price
TMS320F240	20			544	16K	3	12	16	6.6µs	Y	Y	Y	Y		28	5	132 PQFP	$14.44
TMS320C240	20	16K		544		3	12	16	6.6µs	Y	Y	Y	Y		28	5	132 PQFP	$9.80
TMS320F241	20			544	8K	2	8	8	850ns		Y	Y	Y	Y	26	5	64 PQFP 68 PLCC	$10.40 $11.49
TMS320C242	20	4K		544		2	8	8	850ns		Y		Y		26	5	64 PQFP 68 PLCC	$3.77 $4.39
TMS320F243	20			544	8K	2	8	8	850ns	Y	Y	Y	Y	Y	32	5	144 TQFP	$11.82
TMS320LC2406A	40	32K		2.5K		4	16	16	500ns		Y	Y	Y	Y	41	3.3	100 TQFP	$5.90
TMS320LC2404A	40	16K		1.5K		4	16	16	500ns		Y	Y	Y		41	3.3	100 TQFP	$5.32
TMS320LC2402A	40	6K		544		2	8	8	500ns		Y		Y		21	3.3	64 TQFP	$2.95
TMS320LF2407A	40		256	2.5K	32K	4	16	16	500ns	Y	Y	Y	Y	Y	41	3.3	144 TQFP	$9.93
TMS320LF2406A	40		256	2.5K	32K	4	16	16	500ns		Y	Y	Y	Y	41	3.3	100 TQFP	$9.45
TMS320LF2402A	40		256	544	8K	2	8	8	500ns		Y		Y		21	3.3	64 PQFP	$7.70
TMS320LF2403A	40		256	1K	16K	2	8	8	500ns		Y	Y	Y	Y	21	3.3	64 TQFP	$8.84
TMS320LF2401A	40		256	1K	8K	2	7	5	500ns		Y		Y		13	3.3	32 LQFP	$6.59
TMS320LC2402A	40	6K		544		2	8	8	500ns		Y		Y		21	3.3	64 TQFP**	$2.95

*** 64 TQFP for new LC2402A DSP is 100 mm^{2}***
C28x Broad Sampling Mid 2001

This page is intentionally blank.

Learning Objectives



Module Topics

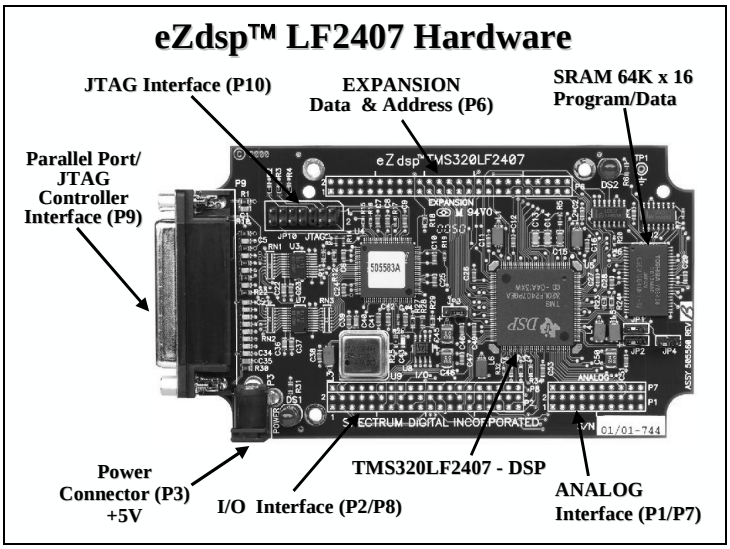
eZdsp™ LF2407 Starter Kit 2-1

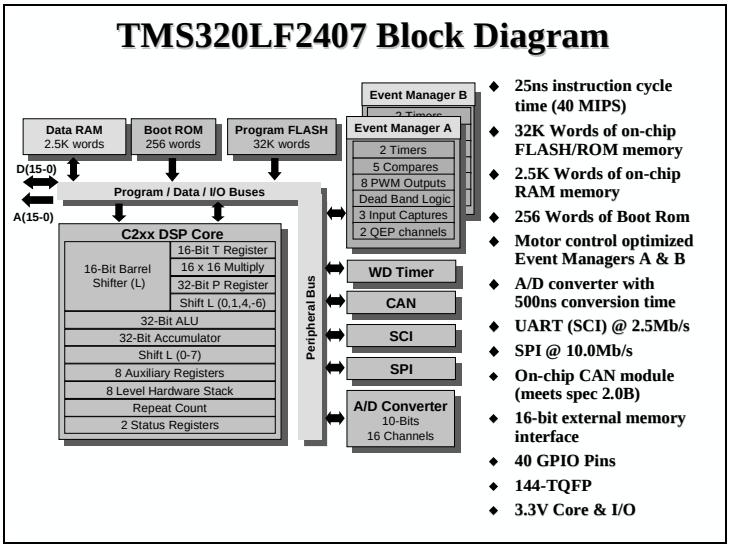
Module Topics 2-2

eZdsp™ LF2407 2-3

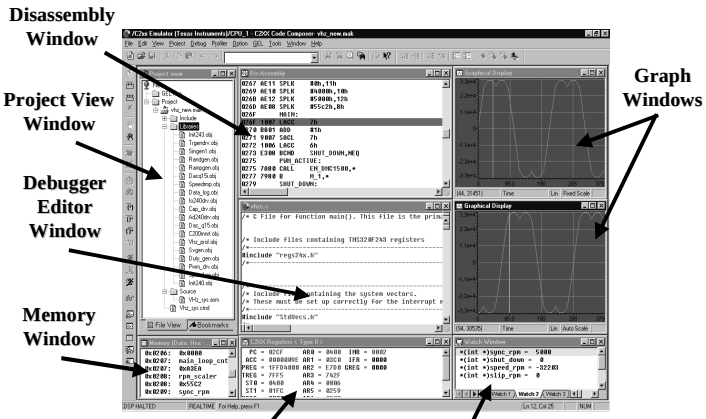
Module 2 Lab: Getting Started with the eZdsp™ LF2407 Starter Kit 2-6

EZdsp™ LF2407





Code Composer



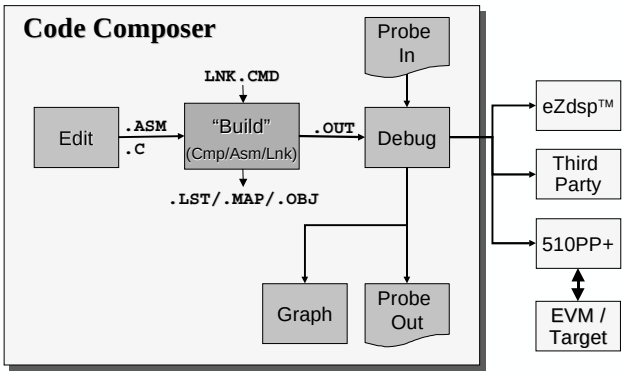
Assembler, Mnemonics, Directives, Sections

- **Assembler**
  - **Instructions & directives cannot be in first column, and comments defined by a ";" or "*" (asterisk located in column one only)**
- **Mnemonics**
  - **Lines of TMS320 code ("processor code") become components of program memory**
- **Directives / Sections**
  - **Begin with a period (.) and are lower case - and can create sections, constants and variables; and may occupy no memory space when used to control assembly and link process**

Assembler Directive	Definition
.text	Code to follow
.data	Constants to follow
.bss	Allocate space for variables
.int	Creates 16-bit integer constant
.set	Define assembly-time constant

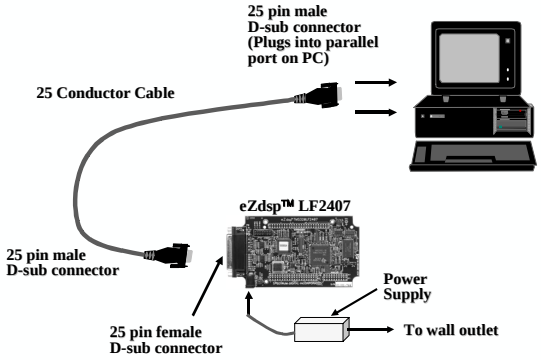
```
        .bss    z,2
        .data
x       .int    2
y       .int    7
        .text
start   LACC    x
        ADD     y
        SACL    z
        SACH    z+1
        B       start
```

Software Development Tool Flow



- Code Generation Tools / Debug Tools

Connecting the eZdsp™ to your PC



Code Composer – eZdsp™ LF2407 Configuration

Module 2 Lab: Getting Started with the eZdsp™ LF2407 Starter Kit

Module 2 Lab:
Getting Started with the eZdsp™ LF2407

Objective:

1) Build, load & run a project on eZdsp™

2) Learning to use Code Composer with Real-Time Mode
- **Build project**
- **Load program**
- **Enable Real-Time Mode**
- **Run program**
- **Set-up Watch, Memory and Graph Windows**
- **Control waveforms by changing operational variables**

Introduction

The objective of this lab is to become familiar with the basic operation of the Code Composer debugger, which is provided with the eZdsp™ LF2407. The REALTIME.MAK project used in this lab has a sine waveform generation routine that creates two sine waves of controllable and variable frequency and phase difference. These waveforms are buffered in memory so that it can be viewed with Code Composer graph windows without aliasing.

In this lab you will:

- Open the REAL24XX.MAK project
- Build the project and download the example program to the eZdsp™ board
- Set-up Code Composer in Real time mode and run the program
- View the operation of the program through watch windows, graphs and memory windows in real-time
- Use the watch windows to make real-time changes to variables and waveforms

No external signals or connections are used in this lab.

➢ Procedure

Part #1: Loading the Project and Running the Program

1. Connect the eZdsp™ to the PC parallel port using cable provided.

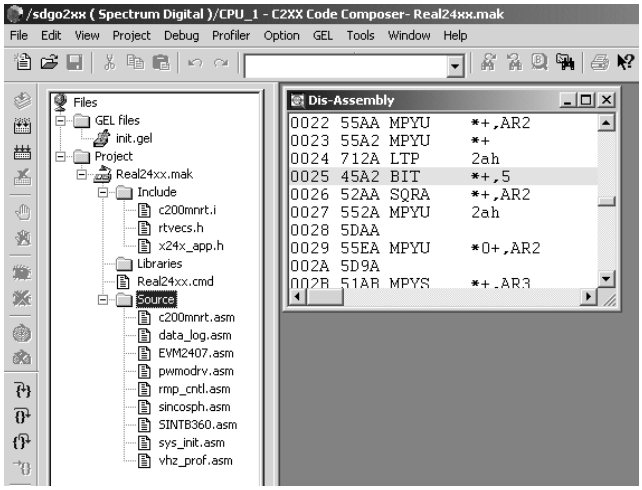
2. Connect the power supply to the eZdsp™. Verify the red power lights are on.

3. From the desktop, launch Code Composer. You should see the debugger window open.

4. Open project **Real24xx.mak** : From the pull-down menu bar, go to Project → Open

5. Browse to the project file directory: *C:\DSP24_1DAY\LABS\Lab2\Real24xx.mak* and click Open.

6. Under the "Files" window, expand the project tree. You should see:



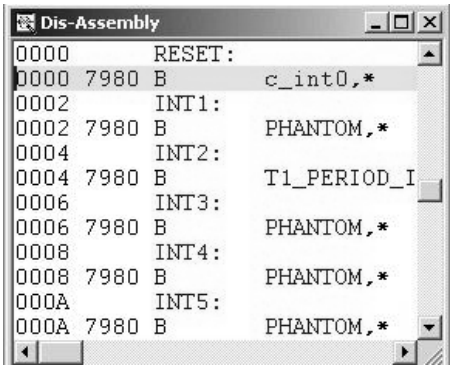
7. Edit the x24x_app.h header file to select the appropriate DSP. Double-click on the x24x.h file and view the text editor window. Since this lab is working with the LF2407 DSP, the section of text should read:

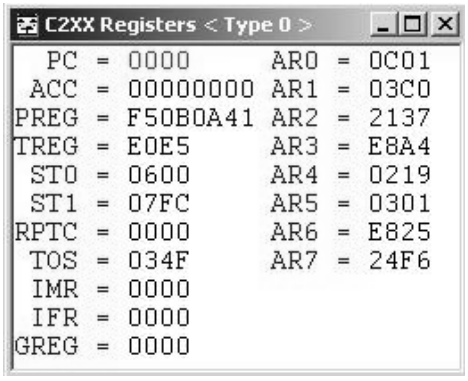
```
x240        .set    0       ; C/F240
x241        .set    0       ; F241
x242        .set    0       ; C242
x243        .set    0       ; C/F243
x2402       .set    0       ; C/F2402
x2404       .set    0       ; C2404
x2406       .set    0       ; C/F2406
x2407       .set    1       ; F2407
```

8. Build the project by going to the pull-down menu: Project → Rebuild All. The output window status should read: "Build complete, 0 Errors, 0 Warnings" when complete.

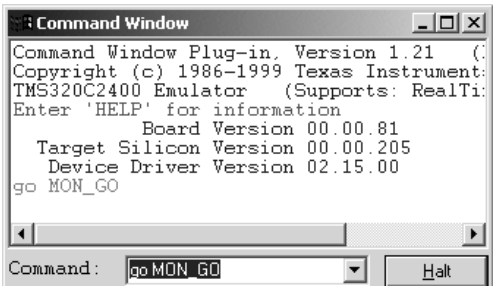
9. Download tutorial example to eZdsp™ board. From the pull-down menu go to File → Load Program. Select *Real24xx.out* from the dialog box in directory *C:\DSP24_1DAY\LABS\Lab2* and click on Open. *Note: If you receive a loading error, verify you have the eZdsp™ target jumpers set for disabled watchdog (JP3 = 2-3) and microprocessor mode (JP4 = 1-2).*

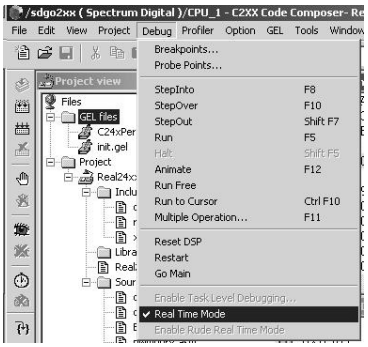
10. Reset the DSP by going to pull-down menu item Debug → Reset DSP. This resets the Program Counter (PC) to 0. Verify this by checking the Disassembly Window. The yellow bar shows the current memory location of the PC. You can also view the PC along with other CPU registers by going to pull-down menu View → CPU Registers → CPU Register.





11. Initialize real-time mode. Go to Tools → Command Window. The Command Window will appear. Type go MON_GO in the Command line and execute. The program will begin executing until encountering the label "MON_GO." (View the Disassembly Window and check the location of the PC for reference.) Y*ou may see the file C200mnrt window appear... close this window as it will not be reviewed in this lab.* Finally, go to Debug → Real Time Mode.





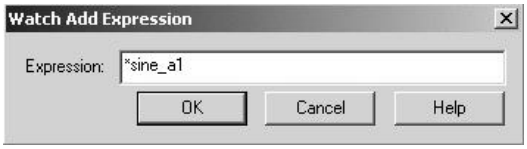
12. After selecting Real Time mode, run the software. Go to menu bar Debug → Run. The DSP should be running in Real Time Mode. The bottom left of the screen should be as seen below:

Part #2: Adding Watch Windows.

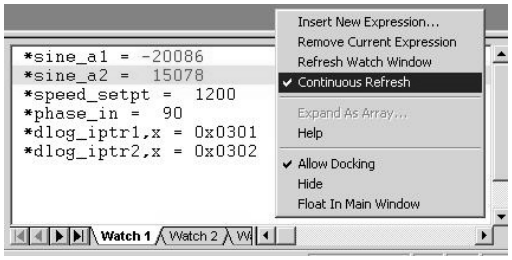
1. Add watch windows by going to the pull down menu, go to View → Watch Window.

2. Declare variables or peripheral registers to watch. Place cursor on watch window and right – click, select "Insert New Expression" and then type in the watch Add Expression box:
   *sine_a1 and then enter. Repeat for these other variables:
   *sine_a2
   *speed_setpt
   *phase_in

*dlog_iptr1,x
*dlog_iptr2,x

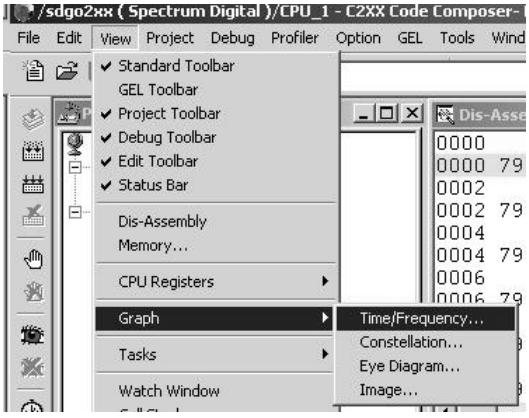


3. At this point you will notice that even though the program is running, the variables in the watch window are not changing. Enable the continuous refresh of watch variables by placing the cursor on the watch window and right-mouse clicking on Continuous Refresh (see next graphic.) If the variables are not changing, halt the program (Debug → Run) and carefully repeat Steps 10-12 in Part 1 above.



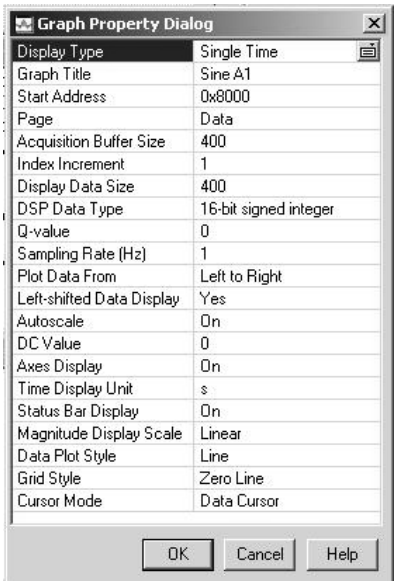
Part #3: Adding Graph View

1. In order to add a graph, go to the pull-down menu and select View → Graph → Time/Frequency in order to open a Graph Property Dialog box.

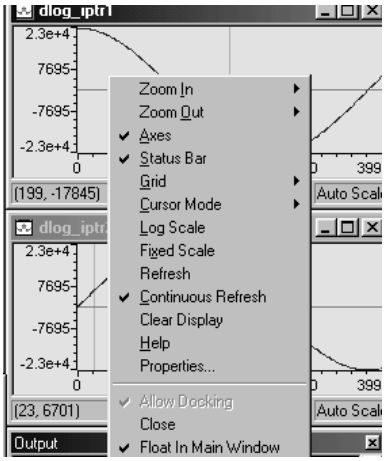


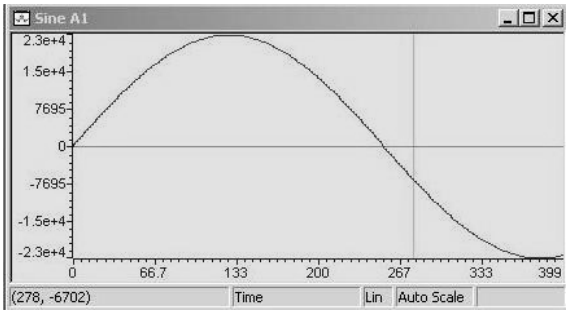
2. Modify the following in Graph Property Dialog box (shown below) and click OK.
   - Graph Title Sine_A1
   - Start Address 0x8000
   - Acquisition Buffer Size 400
   - Display Data Size 400

- DSP Data Type 16-bit signed integer

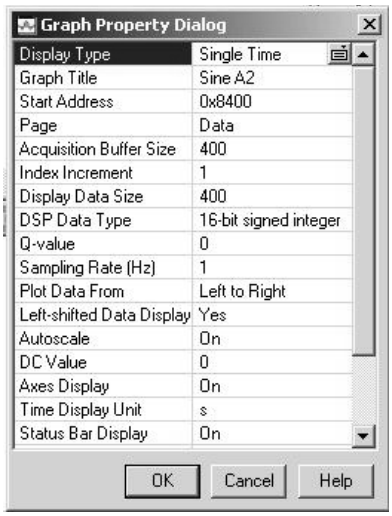


- Right Click on Graph Window and click on > Continuous refresh
- Graph Window should be displaying a Sine Wave at this time – as shown below.

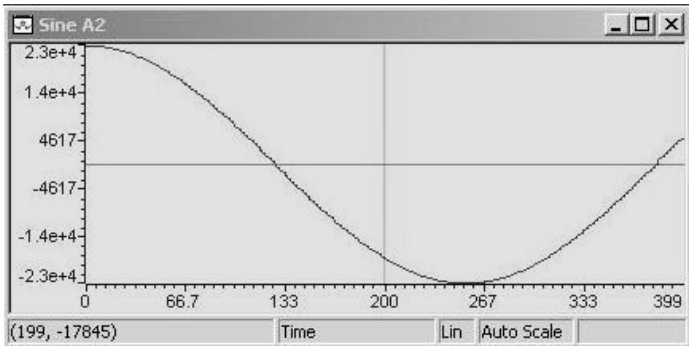




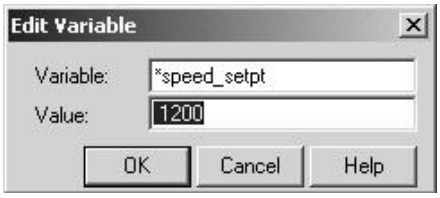
3. Add a second Graph Window similar to the first window. This second Graph Window will mimic the previous one. The only change is modifying the Graph Title to Sine A2 and the Starting Address to 0x8400.



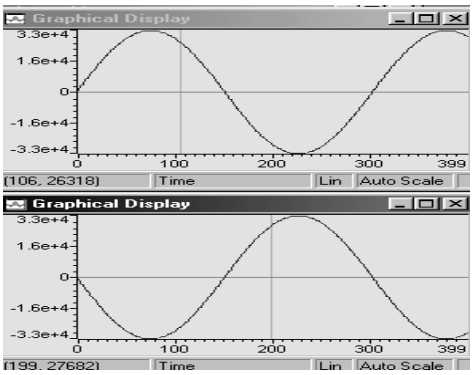
4. Click OK on Graph Property Dialog box for Second Graph Window. Right-click on this Second Graph Window and click on >Continuous Refresh, as before. This Second Graph Window should now be displaying a Cosine wave as shown below.

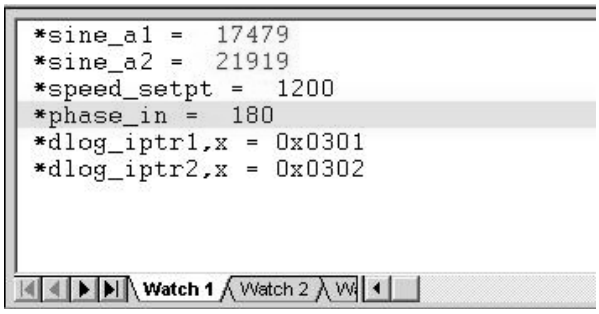


5. The frequency of these waveforms can be changed by putting a different value for the variable *speed_setpt* in the Watch Window. Variables in the Watch Window can be changed by double-clicking on the variable. Double-click on *speed_setpt* in the Watch Window and edit the variable in the following dialog box:



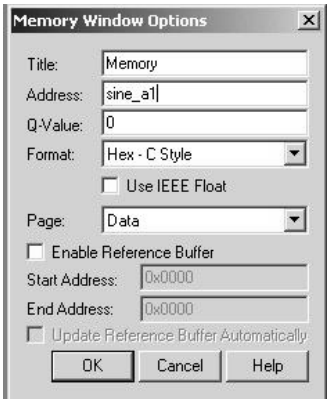
6. Change *speed_setpt* to 2000 and watch the graph windows change accordingly. The phase angle between these waveforms can be changed by modifying the variable *phase_in* in the Watch Window. The following graph shows waveforms at *speed_setpt* = 2000 and *phase_in* = 180. The corresponding watch window is also shown. Experiment with other values in *speed_setpt* and *phase_in.*

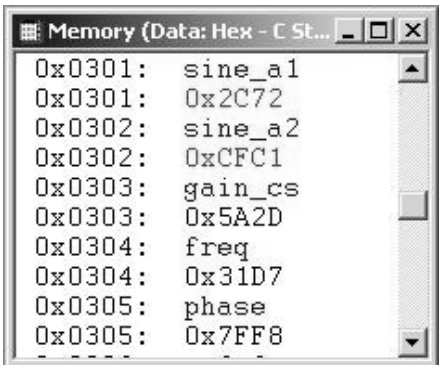




Part #4: Adding Memory View Windows

1. In order to view the memory contents, go to the pull-down menu and select View → Memory. A dialog box of the Memory Window Options will appear. In order to view a specific memory location, the absolute hex address (e.g. 0x0301h) or a variable name (e.g. sine_a1) can be entered in the Address line. (Don't forget the Continuous Refresh.) *FYI Only - The Realtime refresh interval is default set at 100msecs, which is the shortest inveterval, but can be changed by going to pull-down menu View → Realtime Refresh Options.*





2. In order to change the addresses viewed, either scroll the window up or down, or right-click on the table, select "Properties" and the Memory Window Options dialog box will be opened. Data memory space is currently being viewed. *FYI Only - The Program and IO memory spaces can also be viewed by changing the "Page:" selection.*

3. Halt the Program and deactivate Realtime mode – First, go to pull-down menu Debug → Halt. Next, go to Debug → Real Time Mode and uncheck the Real Time Mode selection.

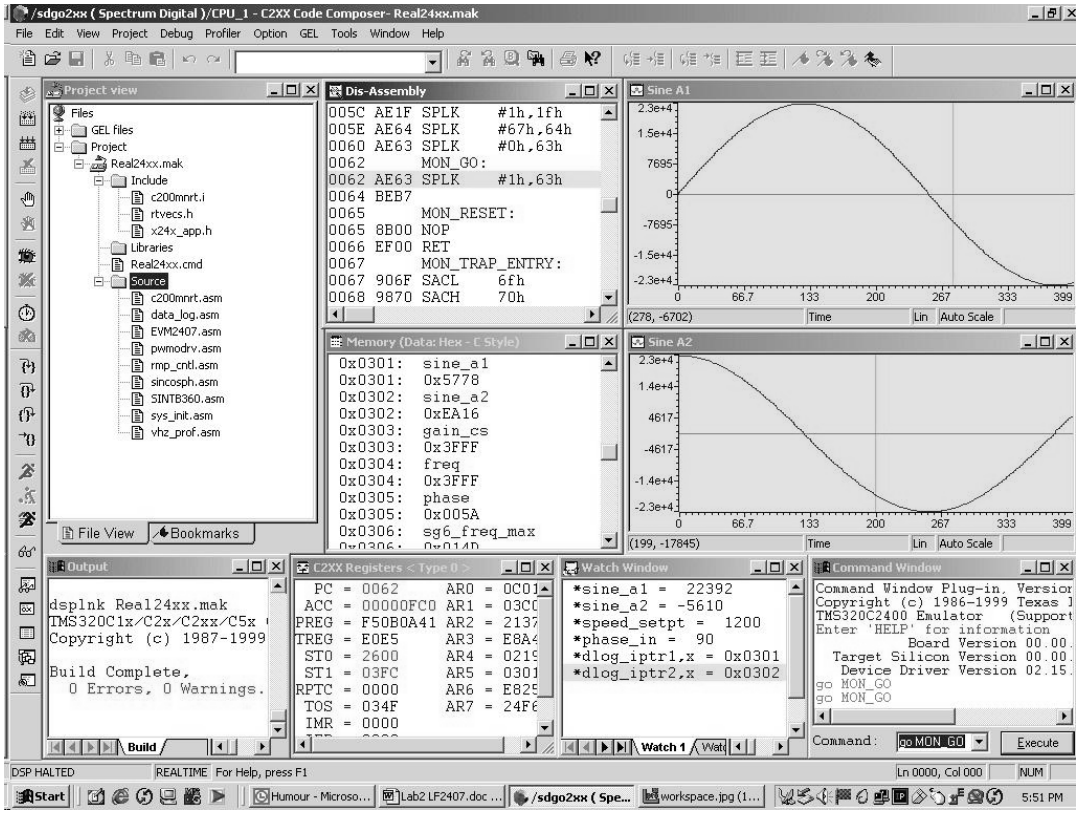
Part #5: Using Workspaces

1. Save your workspace by going to pull-down menu File → Workspace → Save Workspace and using a new file name (like Lab2.wks)

2. This lab tutorial also provides a workspace example. Open the example workspace by going to the Menu bar and select File → Workspace → Load Workspace and browse to / open *C:\DSP24_1DAY\LABS\Lab2\real24xx.wks* Next, reactivate Debug → Real Time Mode.

3. The resulting workspace is shown below. The placement, sizing and properties of each view window are preserved in the workspace file and can be restored in subsequent sessions. You can also switch between different working environments in the same debugging session. One property that is not saved though is continuous refresh, which must be re-enabled after setting up Real-Time mode as described in Part 1.

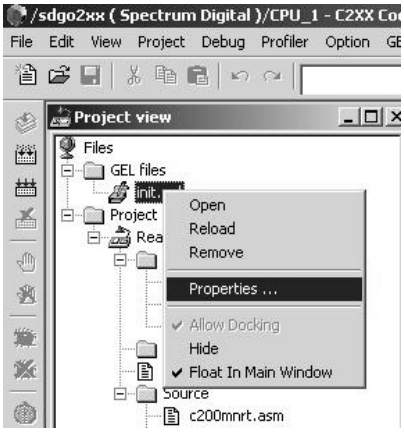
4. The project file is contained within the workspace as well. From a fresh start of Code Composer, loading the workspace also loads the project. Project files (.mak files) do not contain workspace information so loading a project will not load a workspace.

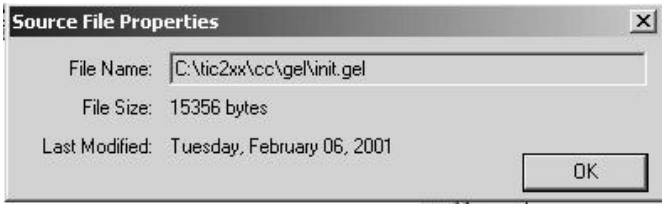


Part 6 – Using General Extension Language (GEL) capabilities

You can use GO DSP Extension Language (GEL) files to add custom features to the GEL menu on the Code Composer menu bar. Code Composer will automatically load the GEL file "init.gel" at the start of each session. This GEL file can be customized with specialty features, and other GEL files can be created and added to the project.

1. In the Project View window of your workspace, right-click on the "init.gel" file under the folder "GEL Folders" and select the properties. This will show the source path of the file.





2. Code Composer references the init.gel file at the C:\tic2xx\cc\gel directory at the start of each session. Modifying or replacing that file can add special or custom GEL functions, which will be available at the start of each Code Composer session.

3. Go to the Menu bar and note the available options under the GEL pull-down. These were defined by the current init.gel file.

4. Open up Microsoft File Explorer. Browse to *C:\DSP24_1DAY\LABS\Lab2* and find the copy of init.gel located in that directory. Copy the file. Browse to *C:\tic2xx\cc\gel* directory and replace the init.gel file there with the one located in the *Lab2* directory.

5. Shutdown Code Composer and restart it. Now go to the Menu bar and note the new additional functions under the GEL pull-down. These will now be available at the start of each session of Code Composer, and for each previously created workspace that uses the init.gel file.

First init.gel

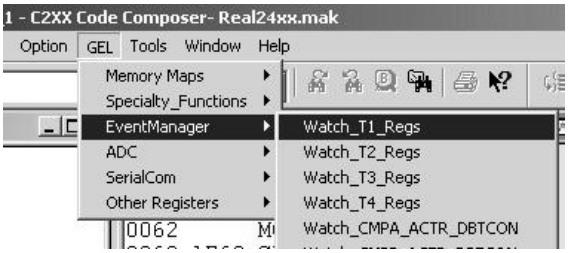
After Step 4 update of init.gel

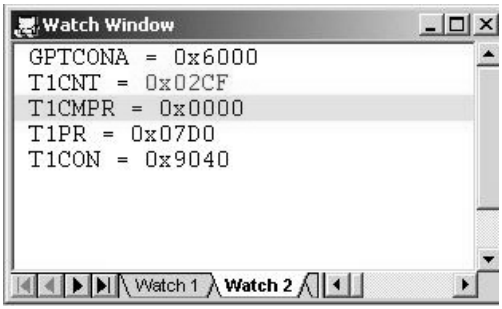
6. Loading and Removing GEL files can also be done by right clicking on the GEL files themselves in the CC Window "Project View." This can only be done, however, if Code Composer is not in Real-time mode. Other GEL files with different file names can be loaded into the project as well.

7. The new init.gel provided Specialty Functions, like go MON_GO. Using this GEL command replaces the repetitive steps of resetting the DSP and executing the go MON_GO in the Command Window. You will still need to complete the final step of going to the pull-down menu item Debug → Real Time Mode.

8. The peripheral registers can be added to the Watch Window. Currently your watch window may be displaying the memory locations / variables listed in Part 2. Those variables were

displayed in the tab labeled “Watch 1.” Click on the tab “Watch 2” for a different Watch window. Go to pull-down menu GEL → EventManager → Watch_T1_Regs and the Timer 1 Registers will be displayed in the active Watch Window (in tab Watch2).

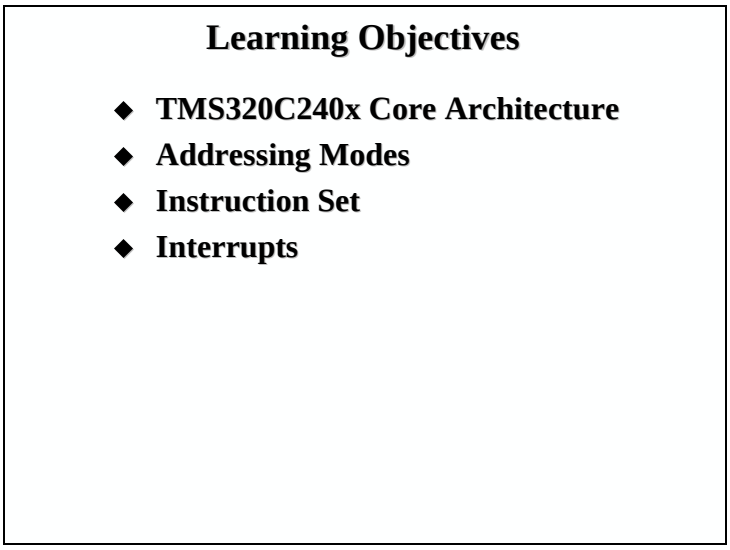




9. Let’s restart the program. If power to the eZdsp™ was lost, then the program Real24xx.out will have to be reloaded to the eZdsp™ target processor. (Repeat the action in Part 1, Step 9). Real-time monitor mode will have to be activated, repeat the action of Part 6, Step 7, (two steps prior.) RUN the program and observe the registers in Watch 2. If they are not updating, right-click on the Watch Window and make sure Continuous Refresh is checked on. (Note that this only affects the active Watch Window. Enabling Continuous Refresh on Watch 1 does not enable it for the other Watch Windows.) T1CNT, T2CNT, and T3CNT are the read/write counter registers in each of the three GP timers. The program is using timer 1 so this Watch operation shows the changing value of the Timer 1 counter register.

This page is intentionally blank.

Learning Objectives



Module Topics

TMS320C240x Architecture 3-1

Module Topics 3-2

TMS320C240x Core Architecture 3-3

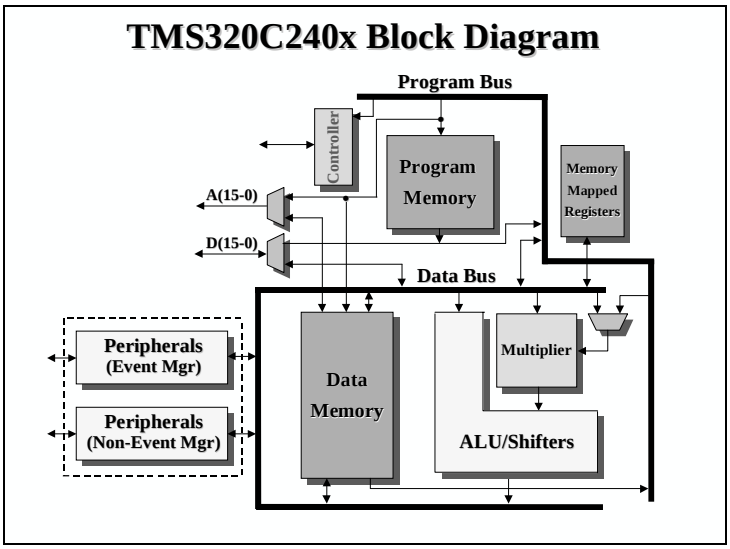
Addressing Modes 3-7
Immediate Addressing 3-7
Direct Addressing 3-8
Indirect Addressing 3-10
Addressing Summary 3-11

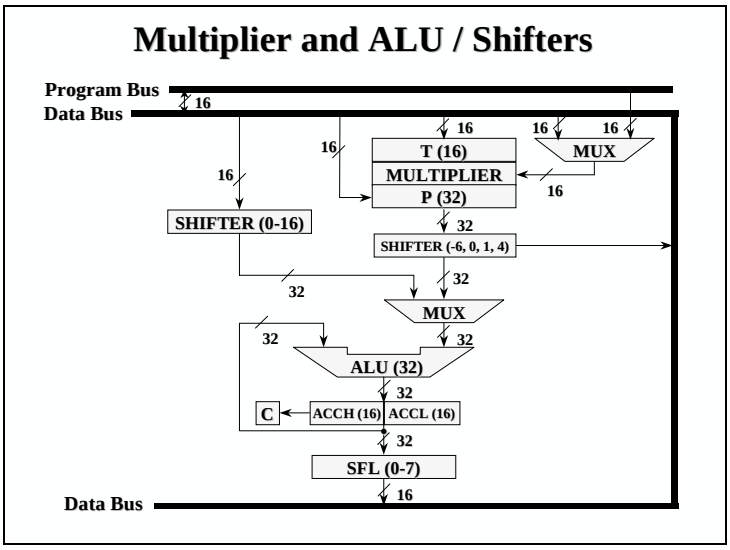
Instruction Set 3-12

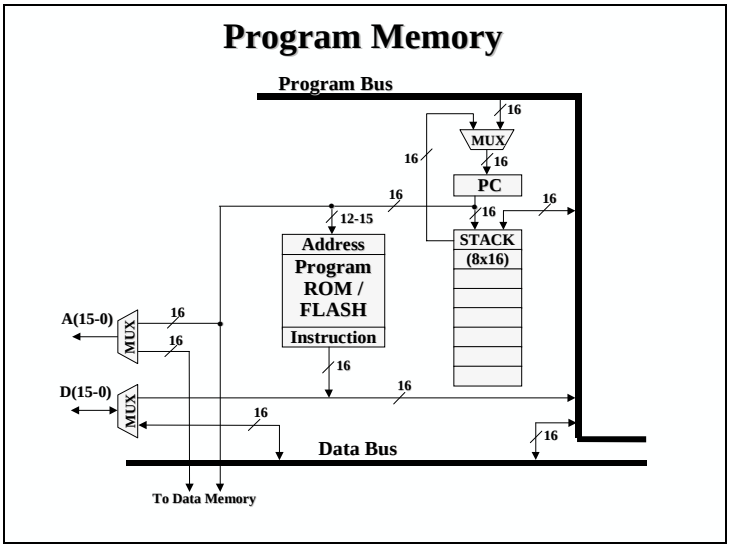
Interrupts 3-21

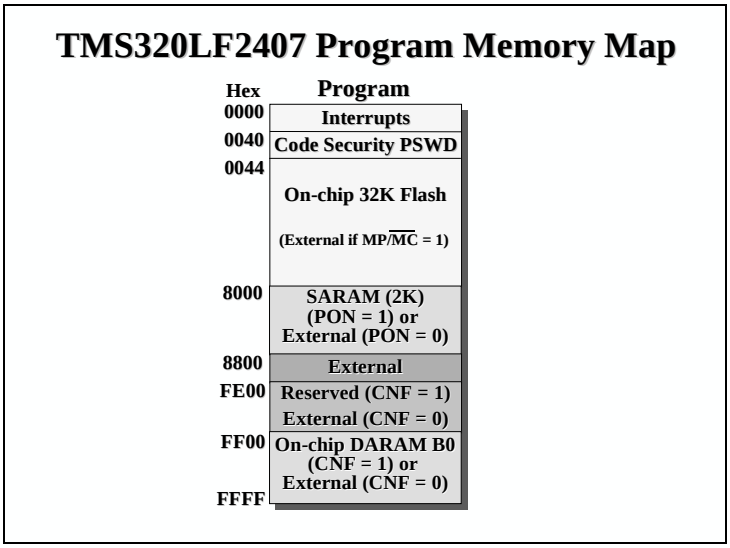
Module 3 Lab: Enhanced Sum of Products 3-24
MOD3LAB.ASM 3-28

TMS320C240x Core Architecture

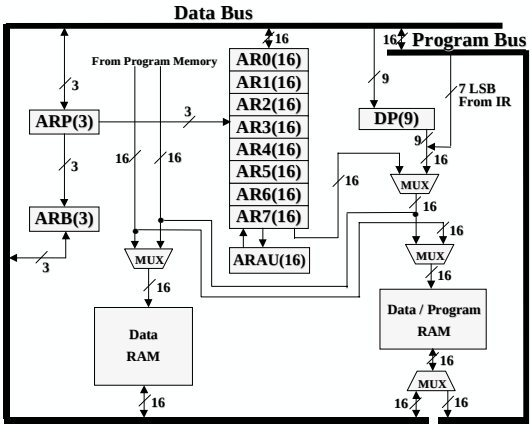





Program Memory
Program Bus
16
MUX
16
PC
16
16
16
12-15
STACK
(8x16)
Address
Program
ROM /
FLASH
Instruction
A(15-0)
MUX
16
16
16
D(15-0)
MUX
16
16
16
Data Bus
To Data Memory

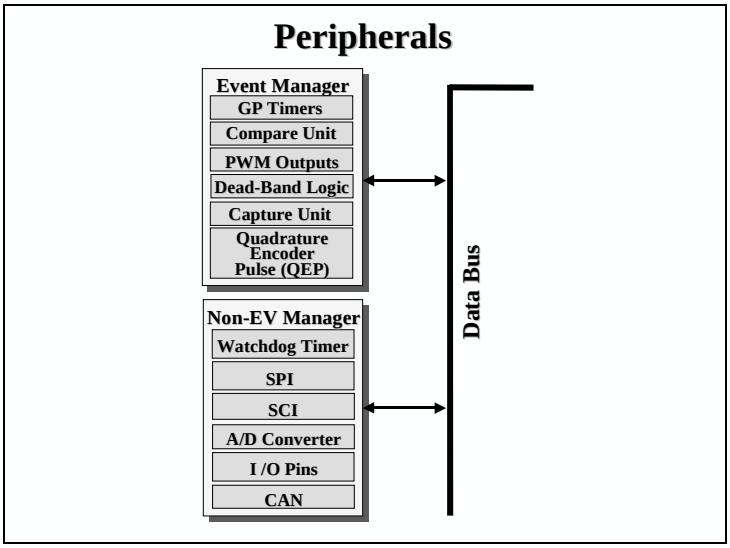

TMS320LF2407 Program Memory Map
Hex
Program
0000
Interrupts
0040
Code Security PSWD
0044
On-chip 32K Flash
(External if MP/MC = 1)
8000
SARAM (2K)
(PON = 1) or
External (PON = 0)
8800
External
FE00
Reserved (CNF = 1)
External (CNF = 0)
FF00
On-chip DARAM B0
(CNF = 1) or
External (CNF = 0)
FFFF

Data Memory



TMS320LF2407 Data Memory Map

Hex	Data
0000	Memory-Mapped Registers
0060	On-chip DARAM B2
0080	Reserved
0200	On-chip DARAM B0 (CNF = 0) or Reserved (CNF = 1)
0300	On-chip DARAM B1
0400	Reserved
0800	SARAM (2K) (DON = 1) or Reserved (DON = 0)
1000	Reserved
7000	Non-EV Peripherals
7400	EV Peripherals
7540	Reserved
8000	External
FFFF	


Peripherals
Event Manager
GP Timers
Compare Unit
PWM Outputs
Dead-Band Logic
Capture Unit
Quadrature Encoder Pulse (QEP)
Non-EV Manager
Watchdog Timer
SPI
SCI
A/D Converter
I /O Pins
CAN
Data Bus

Addressing Modes

Data Addressing Modes

Mode	Purpose
Immediate (Constant)	Initialize registers, operate with constants
Direct (Paged)	Access data on a given page in any order
Indirect (Pointer)	Access data from arrays anywhere in data memory in an orderly fashion

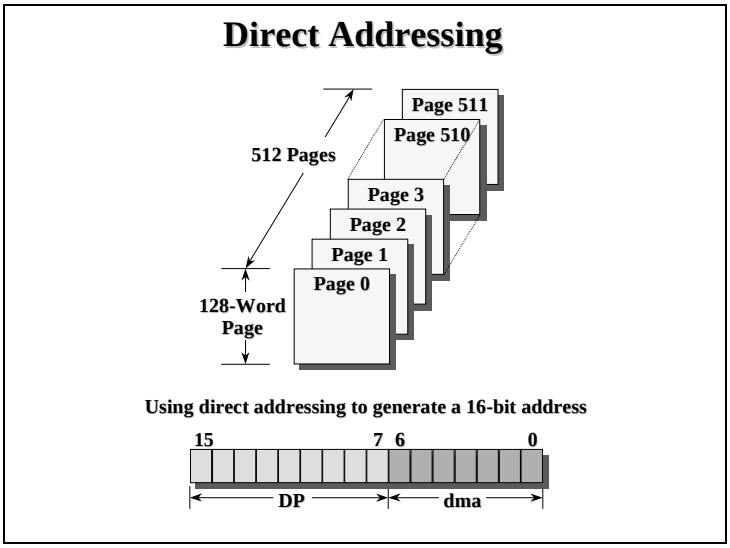
Immediate Addressing

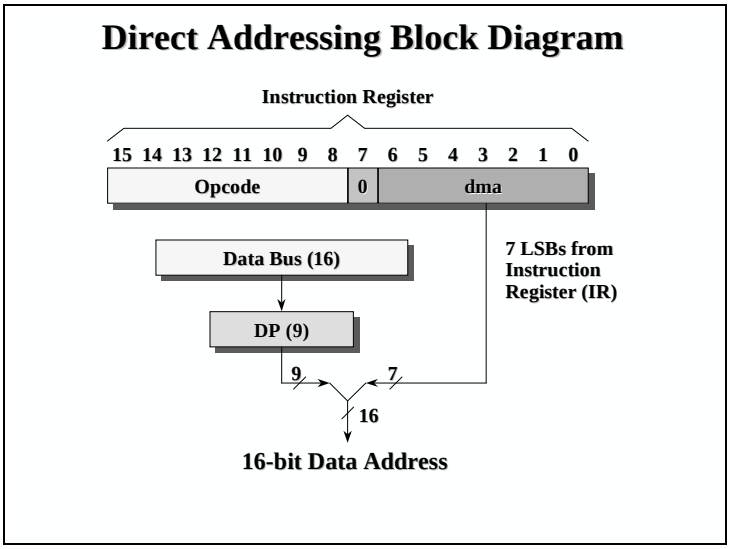
Immediate Addressing Syntax

```
ADD  #8000h,1
```

1) 1 — Shift operand left 1-bit
   Optional, 0 default
2) ADD — Operate on shifted operand
3) Result is stored in Accumulator

Direct Addressing



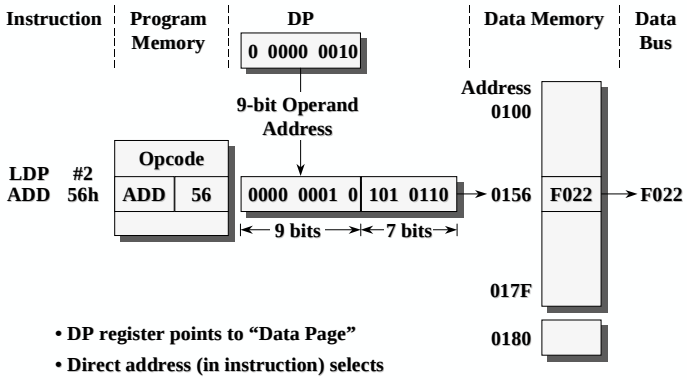


Direct Addressing Syntax

```
ADD  X, 1
```

1)	X	**DP register concatenated with X address operand**
2)	1	**Shift operand left 1-bit optional, 0 default**
3)	ADD	**Operate on shifted operand**
4)		**Result is stored in accumulator**

C240x Direct Addressing Example



- **DP register points to "Data Page"**
- **Direct address (in instruction) selects 1 of 128 locations on the "Data Page"**

Indirect Addressing

Indirect Addressing Syntax

```
ADD   *+,1,AR0
```

1)	*	**ARP selects AR to address operand**
2)	1	**Shift operand left 1-bit optional, 0 default**
3)	ADD	**Operate on shifted operand**
4)	+	**Modify AR value - optional**
5)	AR0	**New ARP value - optional**

Indexing the Auxiliary Registers

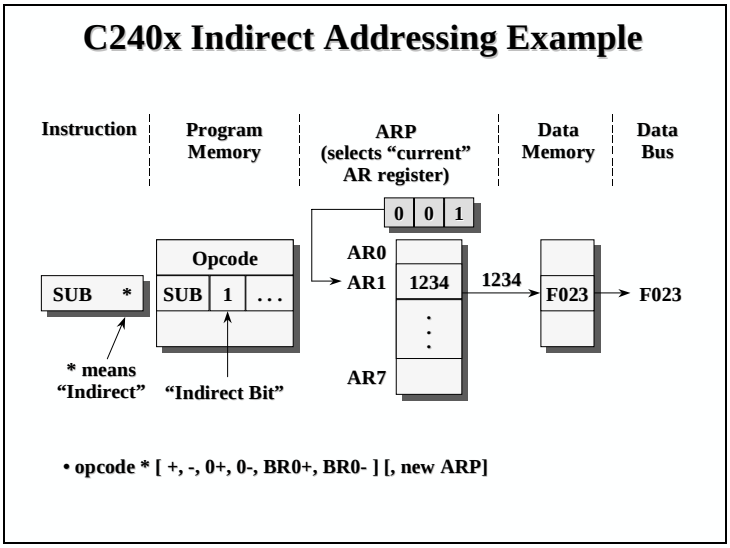
ARP = 101b AR (ARP) = AR5 AR5 = 0200h AR0 = 0004h

Successive indexing operations yield the following in AR5:

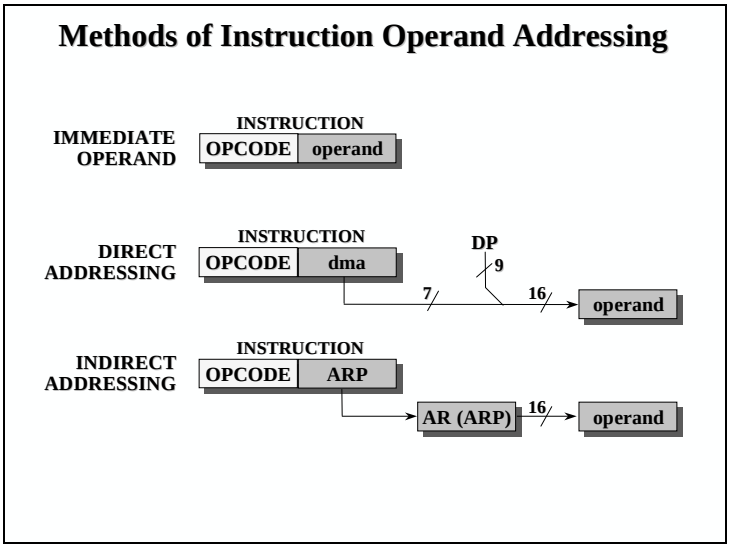
	*	*+/*-	*0+/*0-	*BR0+/*BR0-
AR5	0200	0200	0200	0200
	0200	0201	0204	0204
	0200	0202	0208	0202
	0200	0203	020C	0206
	0200	0204	0220	0201
	0200	0205	0224	0205
	0200	0206	0228	0203
	0200	0207	022C	0207
	0200	0208	0230	0200

FFT Resequencing

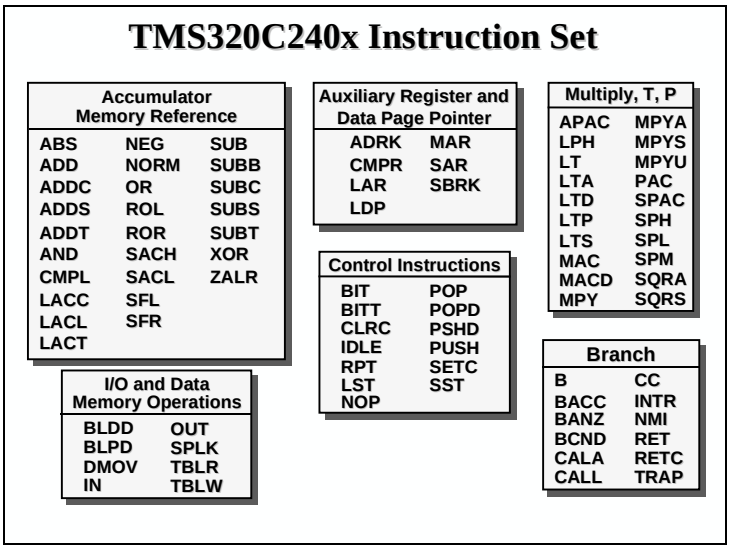
x(0)	200h
x(4)	
x(2)	
x(6)	
x(1)	
x(5)	
x(3)	
x(7)	207h

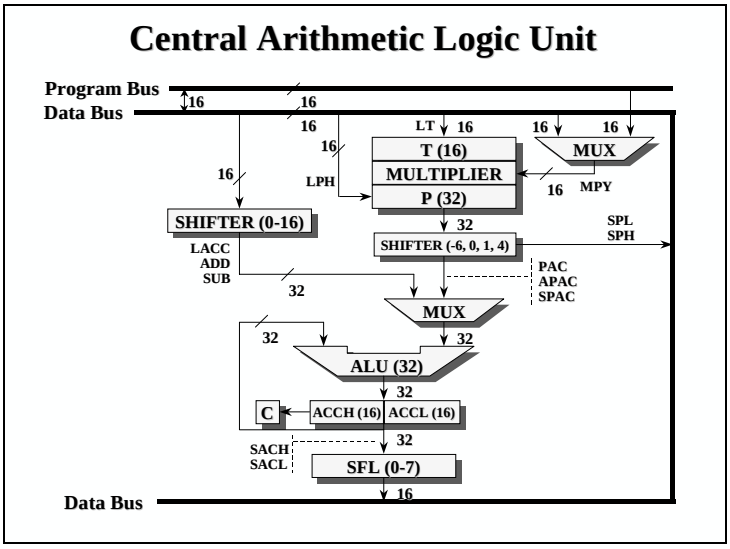


Addressing Summary



Instruction Set





Program Control

BRANCH	CALL	RET
B next	CALL sub	RET
BACC	CALA	—
BCND next,cnd,cnd,…	CC sub,cnd,…	RETC cnd,cnd,…

Conditional Codes			
EQ	ACC = 0	NEQ	ACC ≠ 0
LT	ACC < 0	GT	ACC > 0
LEQ	ACC ≤ 0	GEQ	ACC ≥ 0
C	C = 1	NC	C = 0
OV	OV = 1	NOV	OV = 0
TC	TC = 1	NTC	TC = 0
BIO	$\overline{\text{BIO}}$ = 0	UNC	uncond.

BANZ
Branch on AR ≠ 0

Using SPLK

SPLK means Store Parallel Long Immediate

Problem: Store a 16-bit immediate value to memory

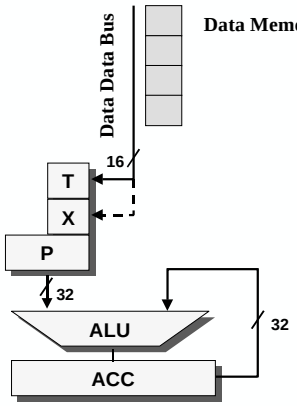
Solution	Code	Words / Cycles
Using LACC	`LACC #1234h` `SACL mem1`	2 / 2 1 / 1 3 / 3
Using SPLK	`SPLK #1234h,mem1`	2 / 2

Benefits:
- **SPLK does not use the ACC**
- **If immediate value is 9-16 bits, you save 1w/1c**
- **The mnemonic is simpler to read**

Note: If immediate value is ≤ 8 bits, you could use LACL instead of LACC and save the extra cycle

Basic Sum-of-Products

LT
MPY
APAC



Sum-of-Products

y = Ax1 + Bx2 + Cx3 + Dx4

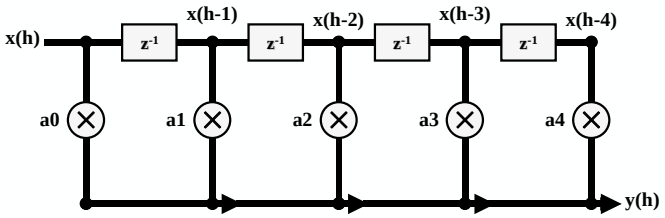
```
LT   X1      ;T = x1
MPY  A       ;P = Ax1
PAC          ;ACC = Ax1
LT   X2      ;T = x2
MPY  B       ;P = Bx2
APAC         ;ACC = Ax1 + Bx2
LT   X3      ;T = x3
MPY  C       ;P = Cx3
APAC         ;ACC = Ax1 + Bx2 + Cx3
LT   X4      ;T = x4
MPY  D       ;P = Dx4
APAC         ;ACC = y
SACL y       ;Store result
```

Enhanced Sum-of-Products

$$y = Ax1 + Bx2 + Cx3 + Dx4$$

```
                LT   X1
PAC             MPY  A
LT X2           LTP  X2
                MPY  B
APAC            LTA  X3
LT X3           MPY  C
                LTA  X4
APAC            MPY  D
LT X4           APAC
                SACL y
```

FIR Filter Implementation

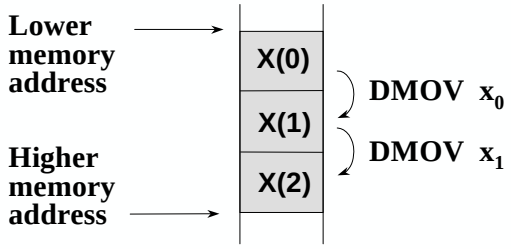


$$y(h) = [a0 \times x(h)] + [a1 \times x(h-1)] + [a2 \times x(h-2)] + [a3 \times x(h-3)] + [a4 \times x(h-4)]$$

- **A delay chain is required!**

Delay Chains: Ripple Buffers

$y(k) = b_0x(k) + b_1x(k-1) + b_2x(k-2)$



Filters = Sum of Products + Delay Chain

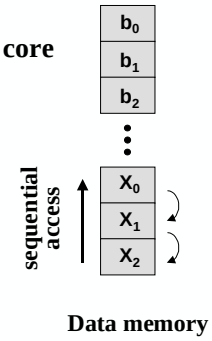
LT	APAC		
MPY	LT	LTA	
APAC	DMOV	DMOV	LTD
DMOV	MPY	MPY	MPY
4 cycles	4 cycles	3 cycles	2 cycles

2nd-order FIR on C240x

$$y(k) = b_0x(k) + b_1x(k-1) + b_2x(k-2)$$

```
FIR:    IN      X0, ADC

        LACL    #0
        LT      X2
        MPY     B2
        LTD     X1
        MPY     B1
        LTD     X0
        MPY     B0
        APAC
        SACH    Y0

        OUT     Y0, DAC
        B       FIR
```

9 cycle core



Data memory

PID algorithm on C240x

$$u(k) = u(k-1) + c_0e(k) + c_1e(k-1) + c_2e(k-2)$$

```
PID:    IN      E0, ADC

        LACC    U1,16
        LT      E2
        MPY     C2
        LTD     E1
        MPY     C1
        LTD     E0
        MPY     C0
        APAC
        SACH    U1

        OUT     U1, DAC
        B       PID
```

9 cycle core

MAC Instruction

MAC <pma>, <dma>

Equivalent to:

APAC, LT <dma>, MPY <pma>

Filters = Sum of Products + Delay Chain

LT	APAC			
MPY	LT	LTA		
APAC	DMOV	DMOV	LTD	
DMOV	MPY	MPY	MPY	MACD ?
4 cycles	4 cycles	3 cycles	2 cycles	1 cycle

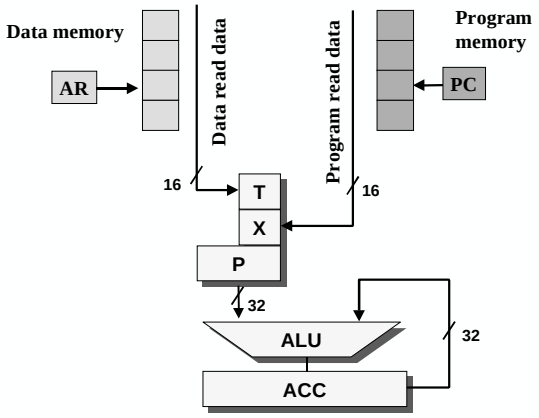
RPT - Repeat Next Instruction

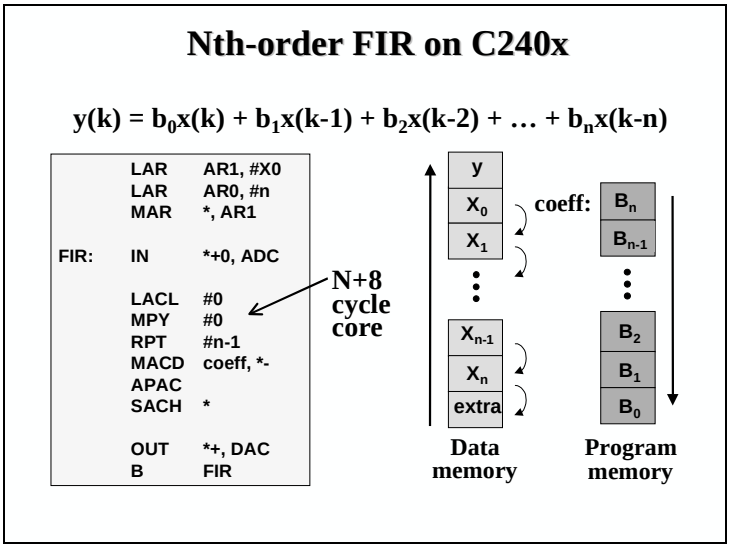
RPT #<n> ;repeat next instruction (n+1) times

Example:

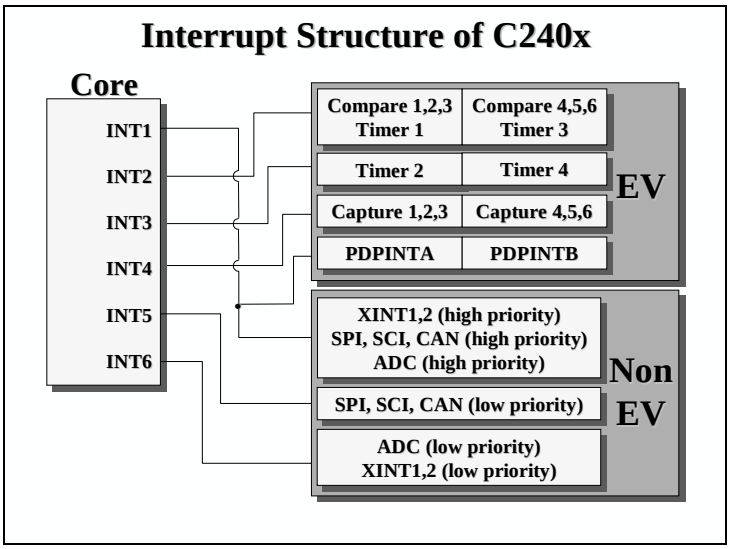
```
RPT     #199        ;repeat next instruction 200 times
MACD    coeff, *+   ;add from * and increment current AR
```

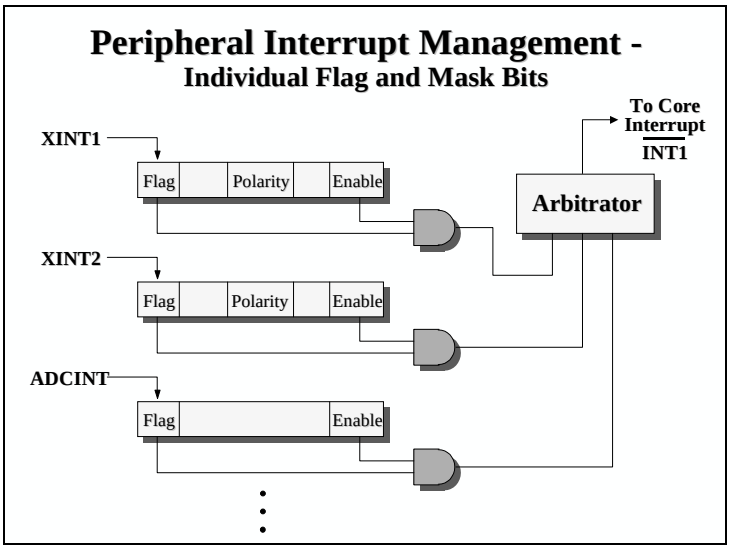
MAC and MACD Instructions



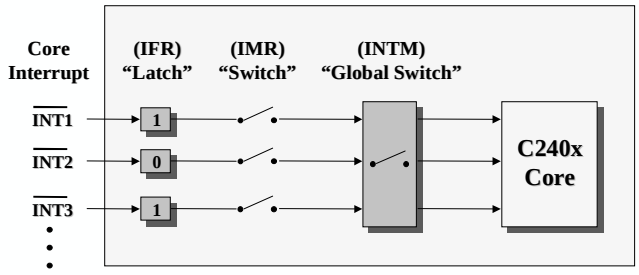

Nth-order FIR on C240x
y(k) = b0x(k) + b1x(k-1) + b2x(k-2) + … + bnx(k-n)
LAR AR1, #X0
LAR AR0, #n
MAR *, AR1
FIR: IN *+0, ADC
LACL #0
MPY #0
RPT #n-1
MACD coeff, *-
APAC
SACH *
OUT *+, DAC
B FIR
N+8 cycle core
y
X0
X1
Xn-1
Xn
extra
coeff:
Bn
Bn-1
B2
B1
B0
Data memory
Program memory

Interrupts


Interrupt Structure of C240x
Core
INT1
INT2
INT3
INT4
INT5
INT6
Compare 1,2,3
Timer 1
Compare 4,5,6
Timer 3
Timer 2
Timer 4
Capture 1,2,3
Capture 4,5,6
PDPINTA
PDPINTB
EV
XINT1,2 (high priority)
SPI, SCI, CAN (high priority)
ADC (high priority)
SPI, SCI, CAN (low priority)
ADC (low priority)
XINT1,2 (low priority)
Non
EV


Peripheral Interrupt Management -
Individual Flag and Mask Bits
To Core
Interrupt
INT1
XINT1
Flag
Polarity
Enable
Arbitrator
XINT2
Flag
Polarity
Enable
ADCINT
Flag
Enable

Core Interrupt Management



A valid signal on a specific interrupt line causes the latch to display a "1" in the appropriate bit. If the individual and global switches are turned on, the interrupt reaches the core.

Determining Interrupt Source

- **Each peripheral interrupt loads a unique offset value into the *Peripheral Interrupt Vector Registers***
  - **location of PIVR @ 701Eh (in data space)**

INT1 (core)	
PDPINTA	0020h
PDPINTB	0019h
ADCINT	0004h
XINT1	0001h
XINT2	0011h
SPIINT	0005h
RXINT	0006h
TXINT	0007h
CANMBINT	0040h
CANERINT	0041h

Phantom Interrupt Vector	0000h

INT2 (core)	
CMP1INT	0021h
CMP2INT	0022h
CMP3INT	0023h
T1PINT	0027h
T1CINT	0028h
T1UFINT	0029h
T1OFINT	002Ah
CMP4INT	0024h
CMP5INT	0025h
CMP6INT	0026h
T3PINT	002Fh
T3CINT	0030h
T3UFINT	0031h
T3OFINT	0032h

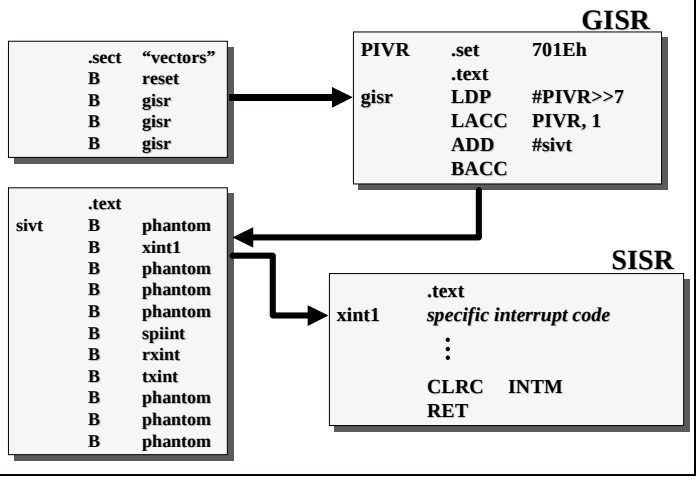
INT3 (core)	
T2PINT	002Bh
T2CINT	002Ch
T2UFINT	002Dh
T2OFINT	002Eh
T4PINT	0039h
T4CINT	003Ah
T4UFINT	003Bh
T4OFINT	003Ch

INT6 (core)	
ADCINT	0004h
XINT1	0001h
XINT2	0011h

INT4 (core)	
CAP1INT	0033h
CAP2INT	0034h
CAP3INT	0035h
CAP4INT	0036h
CAP5INT	0037h
CAP6INT	0038h

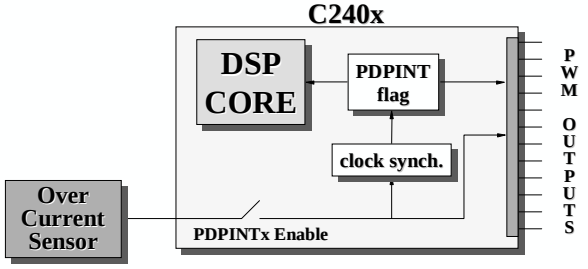
INT5 (core)	
SPIINT	0005h
RXINT	0006h
TXINT	0007h
CANMBINT	0040h
CANERINT	0041h

Basic Approach - (XINT1 example)

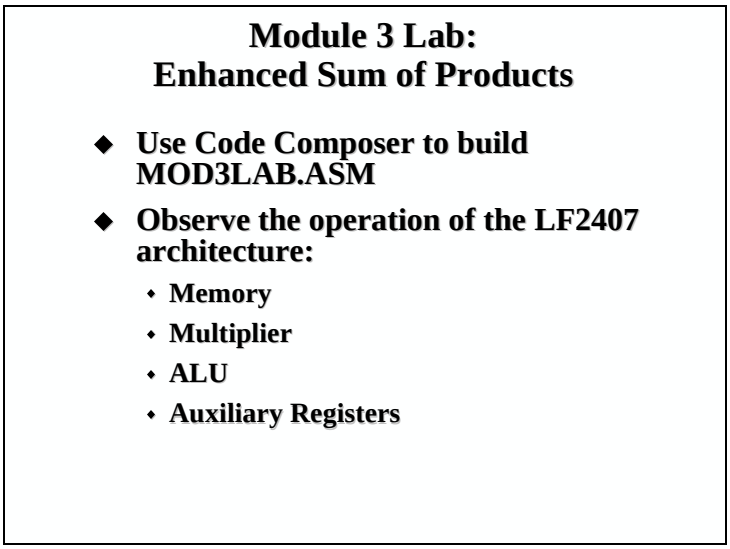


Power Drive Protection - PDPINTA & PDPINTB

- Interrupt latency may not protect hardware when responding to over current through ISR software
- PDPINTx has a fast, clock independent logic path to high-impedance the PWM output pins (~ 45-55 ns)



Module 3 Lab: Enhanced Sum of Products



Introduction

The objective of this lab is to familiarize the student with the programming and operation of the TMS320LF2407 core: the ALU, multiplier, addressing features, accumulator, and the memory. The processor will be initialized with two arrays of four elements each from a table. Pointers will be setup, and the routine will perform the sum-of-products. The result will be stored in data memory.

Recall that there are several ways to execute a sum-of-products

1) A succession of LT, MPY, and APAC instructions

2) A succession of LTA, MPY instructions

3) A RPT instruction followed by a MAC instruction

Note that the order of operations could be extremely important since some instructions make use of previous values. Pointers must be initialized to the correct location. When using the RPT/MAC combination, the product register and accumulator must be initially cleared, since the first operation will be an accumulation and a multiply. The product register can be easily cleared using a MPY #0.

This lab will use method #2. A flow chart of the code is shown in figure 3.1.

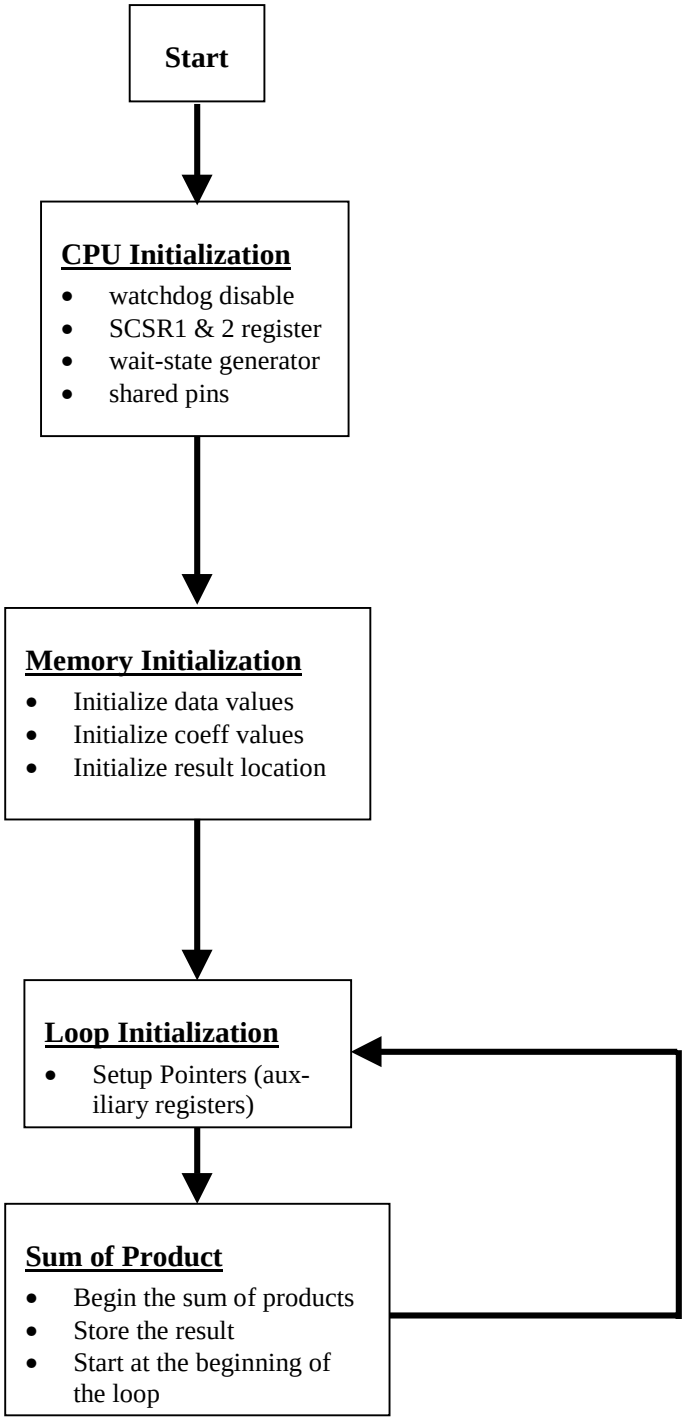


Figure 3.1: Code Flow

➢ **Procedure**

Create a New Project

1. Double-click on the Code Composer icon on the desktop. Maximize Code Composer to fill your screen. The menu bar (at the top) lists File ... Help. Note the horizontal tool bar below the menu bar and the vertical tool bar on the left-hand side. The window on the left is the project window and the large right hand window is your workspace.

2. A *project* is all the files you'll need to develop an executable output file (`.OUT`) which can be run on the target hardware. Let's create a new project for this lab. On the menu bar click:

   `Project → New`

   and make sure the "SAVE IN" location is: `C:\DSP24_1DAY\LABS` and type `MOD3LAB` in the file name window. This will create a *make* file which will invoke all the necessary tools (assembler, linker, compiler) to build your project.

3. Add the assembly file to the new project. Click:

   `Project → Add Files to Project`

   and make sure you're looking in `C:\DSP24_1DAY\LABS`. Change the "files of type" to view assembly files (`.ASM`) and select `MOD3LAB.ASM` and click `OPEN`. This will add the file `MOD3LAB.ASM` to your newly created project.

4. Add `MOD3LAB.CMD` to the project using the same procedure.

5. In the project window on the left click the plus sign (+) to the left of `Project`. Now, click on the plus sign next to `MOD3LAB.MAK`. Notice that the `MOD3LAB.CMD` file is listed. Click on the plus sign next to `Source` to see the current source file list (i.e. `MOD3LAB.ASM`).

Setup the Linker

6. Setup the linker options by clicking:

   `Project → Options`

   on the menu bar. Select the Linker tab. In the middle of the screen select "`No Autoinitialization`". (If needed, a map file can be created by using the `Map Filename [-m]` field). Next, select the Assembler tab. In the middle of the screen check "`Enable Source Level Debugging`". (This feature allows you to debug using both source and assembly, known as "mixed mode"). Then select `OK` to save the Build Options.

Build and Load

7. The top four buttons on the vertical toolbar control code generation. Hover your mouse over each button as you read the following descriptions:

Button	Name	Description
1	Compile File	Compile, assemble the current open file
2	Incremental Build	Compile, assemble only changed files, then link
3	Rebuild All	Compile, assemble all files, then link
4	Stop Build	Stop code generation

8. Click the "Rebuild All" button and watch the tools run in the build window. To open more space, right-click on the build window and Hide the build window.

9. Load the output file onto the target. Click:

   File → Load Program…

   Make sure you are looking in C:\DSP24_1DAY\Labs. Select MOD3LAB.OUT and click OPEN.

Note: Code Composer can automatically load the output file after a successful build. On the menu bar click: Option → Program Load… and select: "Load program after build", then click OK.

---

10. If code generation is successful, the Dis-Assembly window will display the source file (MOD3LAB.ASM) and a yellow highlight on "B start" should appear. This indicates that you are now ready to run.

Running MOD3LAB

11. Inspect MOD3LAB.ASM in the source window. Note how the topics covered in the module are used in the program code.

12. Open a memory window. On the menu bar click:

    View → Memory

    or click the View Memory button on the vertical toolbar. Type "table" into the address to display the contents of the memory starting at label *table*. (Note: Code Composer is case sensitive). You can display as many independent windows as you require. Do you see the initial values in the memory window displaying "table"? Open another memory window starting at label *data*. During initialization, the values from memory starting at label "table" will be copied to the memory locations starting at label "data".

13. To see the CPU registers. On the menu bar click:

    View → CPU Registers → CPU Registers

    Right click in the CPU Registers window and deselect "Allow Docking". You can now move and resize the window as you like.

14. Next, single-step the routine by using the <F8> key or click the single-step button on the vertical toolbar repeatedly past the initialization and to the beginning of the "Sum-of-Products Routine". While in the sum-of-products routine, single-step through the program and watch the memory windows and CPU registers. Did you get the expected answer (28h)?

Optional

15. You might want to experiment with this code by changing some of the values. By double-clicking on a memory location in the memory window you can edit the contents of that memory location. Also, contents of any CPU register can be edited by double-clicking on it.

MOD3LAB.ASM

```
; MOD3LAB.ASM

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Address Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .include f2407.h        ;address definitions

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Variable and Section Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .bss     temp,1         ;general purpose variable

                .bss    data,4          ;allocate space for data
                .bss    coeff,4         ;allocate space for coeff
                .bss    result,1        ;allocate space for result

                .data
table:          .int    1,2,3,4         ;table values for data
                .int    8,6,4,2         ;table values for coeff
                .int    0               ;table value for result
length          .set    $-table         ;calculate size of table

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Interrupt Vector Table for Core
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .sect   "vectors"
                B       start           ;00h reset
int1:           B       int1            ;02h INT1
int2:           B       int2            ;04h INT2
int3:           B       int3            ;06h INT3
int4:           B       int4            ;08h INT4
int5:           B       int5            ;0Ah INT5
int6:           B       int6            ;0Ch INT6
int7:           B       int7            ;0Eh reserved
int8:           B       int8            ;10h INT8  user-defined
int9:           B       int9            ;12h INT9  user-defined
int10:          B       int10           ;14h INT10 user defined
int11:          B       int11           ;16h INT11 user defined
int12:          B       int12           ;18h INT12 user defined
int13:          B       int13           ;1Ah INT13 user defined
int14:          B       int14           ;1Ch INT14 user defined
int15:          B       int15           ;1Eh INT15 user defined
int16:          B       int16           ;20h INT16 user defined
int17:          B       int17           ;22h TRAP
int18:          B       int18           ;24h NMI
int19:          B       int19           ;26h reserved
int20:          B       int20           ;28h INT20 user defined
int21:          B       int21           ;2Ah INT21 user defined
int22:          B       int22           ;2Ch INT22 user defined
int23:          B       int23           ;2Eh INT23 user defined
int24:          B       int24           ;30h INT24 user defined
int25:          B       int25           ;32h INT25 user defined
int26:          B       int26           ;34h INT26 user defined
int27:          B       int27           ;36h INT27 user defined
int28:          B       int28           ;38h INT28 user defined
```

```
int29:          B       int29           ;3Ah INT29 user defined
int30:          B       int30           ;3Ch INT30 user defined
int31:          B       int31           ;3Eh INT31 user defined

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;                     M A I N   R O U T I N E
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .text
start:

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Disable the watchdog
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #11101000b, WDCR
* bit 7          1:      clear WD flag
* bit 6          1:      disable the dog
* bit 5-3        101:    must be written as 101
* bit 2-0        000:    WDCLK divider = 1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the system control registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #0000000011111101b, SCSR1
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15         0:      reserved
* bit 14         0:      CLKOUT = CPUCLK
* bit 13-12      00:     IDLE1 selected for low-power mode
* bit 11-9       000:    PLL x4 mode
* bit 8          0:      reserved
* bit 7          1:      1 = enable ADC module clock
* bit 6          1:      1 = enable SCI module clock
* bit 5          1:      1 = enable SPI module clock
* bit 4          1:      1 = enable CAN module clock
* bit 3          1:      1 = enable EVB module clock
* bit 2          1:      1 = enable EVA module clock
* bit 1          0:      reserved
* bit 0          1:      clear the ILLADR bit

        SPLK    #0000000000001111b, SCSR2
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15-6       0's:    reserved
* bit 5          0:      DO NOT clear the WD OVERRIDE bit
* bit 4          0:      XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
* bit 3          1:      1 = disable the BOOT ROM
* bit 2          1:      MP/MC*, 1 = Flash addresses mapped external
* bit 1-0        11:     11 = SARAM mapped to prog and data

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Set wait states for external memory interface on eZdsp(TM) LF2407
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #temp           ;set data page
```

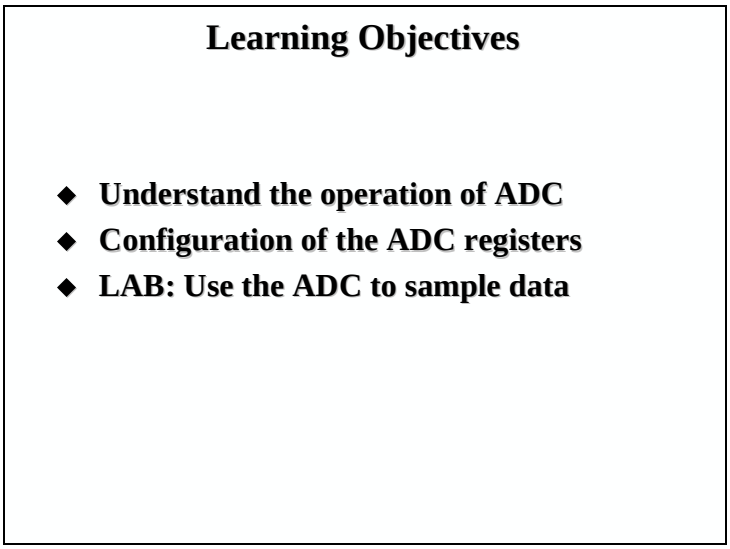
```
        SPLK    #0000000001000000b, temp
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15-11     0's:    reserved
* bit 10-9      00:     bus visibility off
* bit 8-6       001:    1 wait-state for I/O space
* bit 5-3       000:    0 wait-state for data space
* bit 2-0       000:    0 wait-state for program space

        OUT     temp, WSGR

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup shared I/O pins
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page

        SPLK    #0000000000000000b,MCRA
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      0=IOPB7,     1=TCLKINA
* bit 14        0:      0=IOPB6,     1=TDIRA
* bit 13        0:      0=IOPB5,     1=T2PWM/T2CMP
* bit 12        0:      0=IOPB4,     1=T1PWM/T1CMP
* bit 11        0:      0=IOPB3,     1=PWM6
* bit 10        0:      0=IOPB2,     1=PWM5
* bit 9         0:      0=IOPB1,     1=PWM4
* bit 8         0:      0=IOPB0,     1=PWM3
* bit 7         0:      0=IOPA7,     1=PWM2
* bit 6         0:      0=IOPA6,     1=PWM1
* bit 5         0:      0=IOPA5,     1=CAP3
* bit 4         0:      0=IOPA4,     1=CAP2/QEP2
* bit 3         0:      0=IOPA3,     1=CAP1/QEP1
* bit 2         0:      0=IOPA2,     1=XINT1
* bit 1         0:      0=IOPA1,     1=SCIRXD
* bit 0         0:      0=IOPA0,     1=SCITXD

        SPLK    #1111111000000000b,MCRB
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        1:      0=reserved,  1=TMS2 (always write as 1)
* bit 14        1:      0=reserved,  1=TMS  (always write as 1)
* bit 13        1:      0=reserved,  1=TD0  (always write as 1)
* bit 12        1:      0=reserved,  1=TDI  (always write as 1)
* bit 11        1:      0=reserved,  1=TCK  (always write as 1)
* bit 10        1:      0=reserved,  1=EMU1 (always write as 1)
* bit 9         1:      0=reserved,  1=EMU0 (always write as 1)
* bit 8         0:      0=IOPD0,     1=XINT2/ADCSOC
* bit 7         0:      0=IOPC7,     1=CANRX
* bit 6         0:      0=IOPC6,     1=CANTX
* bit 5         0:      0=IOPC5,     1=SPISTE
* bit 4         0:      0=IOPC4,     1=SPICLK
* bit 3         0:      0=IOPC3,     1=SPISOMI
* bit 2         0:      0=IOPC2,     1=SPISIMO
* bit 1         0:      0=IOPC1,     1=BIO*
* bit 0         0:      0=IOPC0,     1=W/R*
```

```
        SPLK    #0000000000000000b,MCRC
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      0=IOPF6,     1=IOPF6
* bit 13        0:      0=IOPF5,     1=TCLKINB
* bit 12        0:      0=IOPF4,     1=TDIRB
* bit 11        0:      0=IOPF3,     1=T4PWM/T4CMP
* bit 10        0:      0=IOPF2,     1=T3PWM/T3CMP
* bit 9         0:      0=IOPF1,     1=CAP6
* bit 8         0:      0=IOPF0,     1=CAP5/QEP4
* bit 7         0:      0=IOPE7,     1=CAP4/QEP3
* bit 6         0:      0=IOPE6,     1=PWM12
* bit 5         0:      0=IOPE5,     1=PWM11
* bit 4         0:      0=IOPE4,     1=PWM10
* bit 3         0:      0=IOPE3,     1=PWM9
* bit 2         0:      0=IOPE2,     1=PWM8
* bit 1         0:      0=IOPE1,     1=PWM7
* bit 0         0:      0=IOPE0,     1=CLKOUT


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Sum of Products Routine
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
top             LAR     AR1,#table      ;AR1 is the source pointer
                LAR     AR2,#data       ;AR2 is the destination pointer
                LAR     AR3,#length-1   ;count value for BANZ loop
                MAR     *,AR1           ;ARP set to AR1

loop:           LACC    *+,0,AR2        ;load/store BANZ loop
                SACL    *+,0,AR3        ;from .data to .bss
                BANZ    loop,*-,AR1     ;branch on AR not equal 0

sop:            LAR     AR1,#data       ;AR1 is data pointer
                LAR     AR2,#coeff      ;AR2 is coeff pointer
                MAR     *,AR1           ;ARP set to AR1
                LT      *+,AR2          ;begin sum-of-products routine
                MPY     *+,AR1
                LTP     *+,AR2
                MPY     *+,AR1
                LTA     *+,AR2
                MPY     *+,AR1
                LTA     *+,AR2
                MPY     *+,AR1
                APAC
                LDP     #result         ;data page ptr set to result
                SACL    result          ;store result
                B       top             ;branch to top
```

This page is intentionally blank.

Analog-to-Digital Converter (ADC)

Learning Objectives



Module Topics

Analog-to-Digital Converter (ADC) 4-1

Module Topics 4-2

Analog-to-Digital Converter (ADC) Overview 4-3

Control Registers 4-5

Sequencer Example 4-8

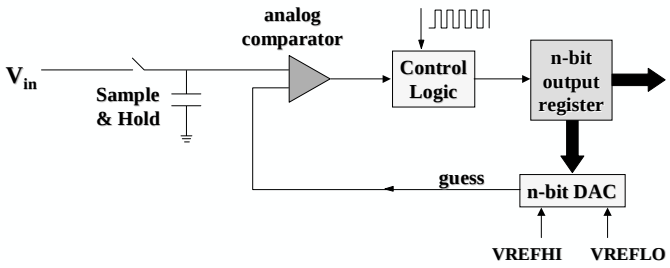
ADC Conversion Result 4-9

Module 4 Lab: ADC Sampling 4-11

MOD4LAB.ASM 4-17

Analog-to-Digital Converter (ADC) Overview

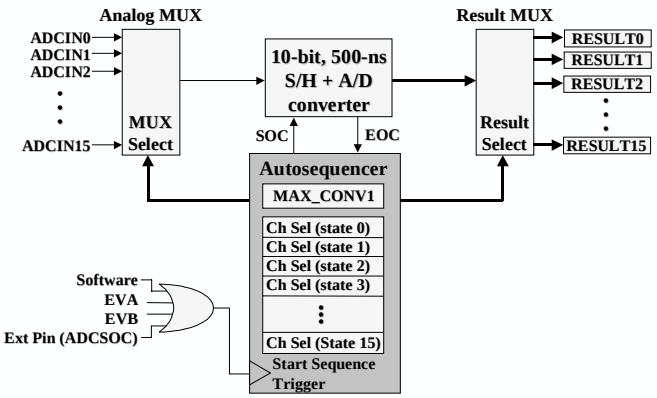
Successive Approximation Method

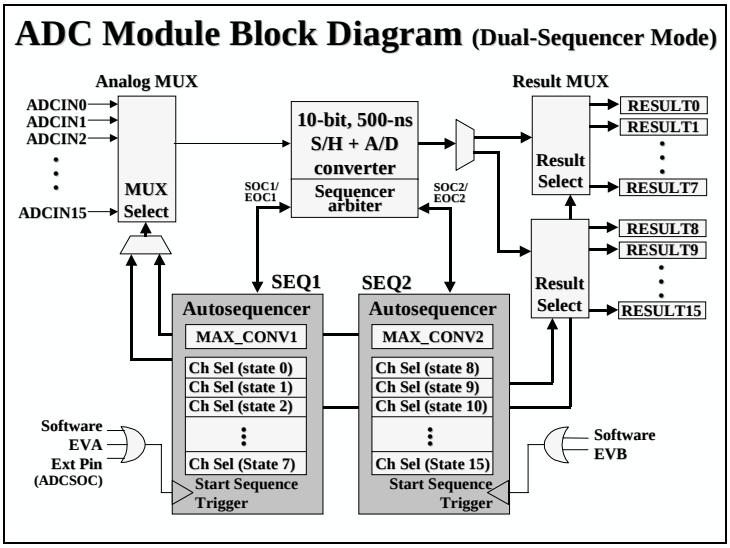


Conversion always takes n steps for an n-bit ADC:

1. Start with all output bits 0
2. Initial guess is made by setting MSB = 1
3. If guess > V_{in} then reset bit, else keep bit set
4. Remaining n-1 bits are sequentially tested, MSB → LSB

ADC Module Block Diagram (Cascaded Mode)

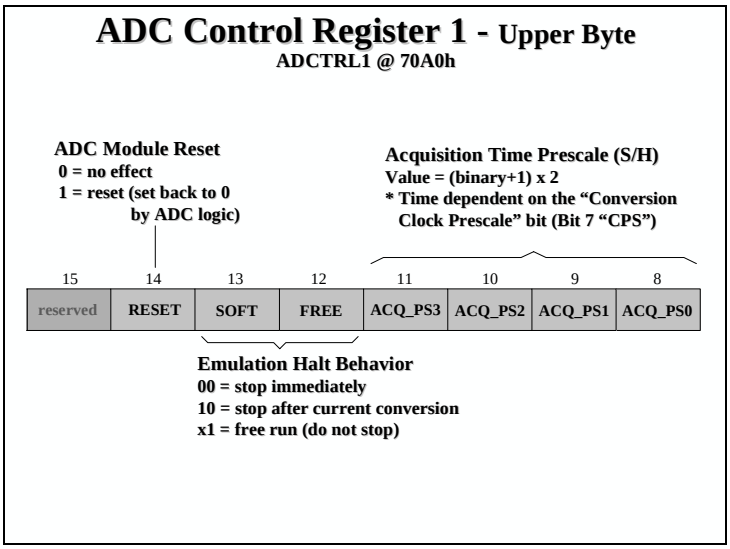


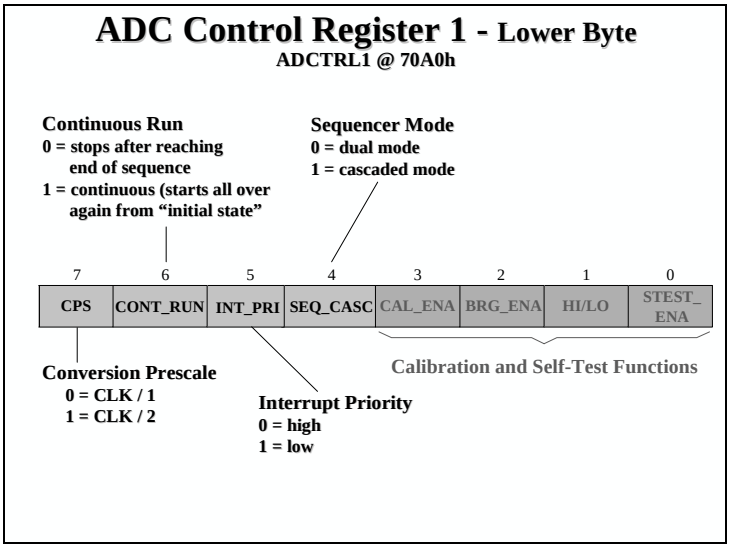


ADC Module

- 10-bit ADC core with built-in Sample & Hold (S/H)
- Sixteen multiplexed analog inputs (8 on C2402)
- Fast conversion time (S/H + conversion) of ≤ 500ns
- Autosequencing capability - up to 16 autoconversions
  - Two independent 8-state sequencers
    - "Dual-sequencer mode"
    - "Cascaded mode"
- Sixteen individually addressable result registers
- Multiple trigger sources for start-of-conversion
- Flexible interrupt control

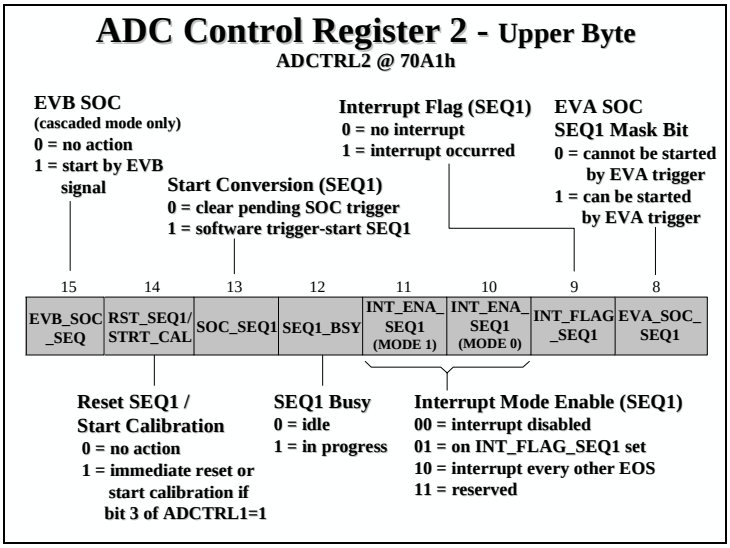
Control Registers





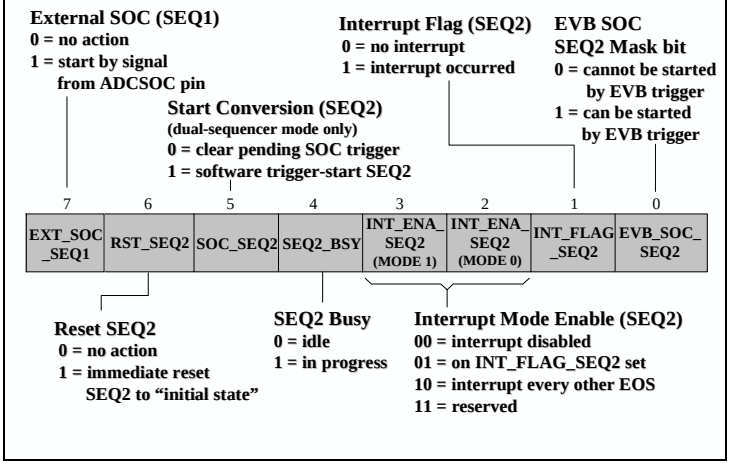
ADC Control Register 2 - Upper Byte

ADCTRL2 @ 70A1h



ADC Control Register 2 - Lower Byte

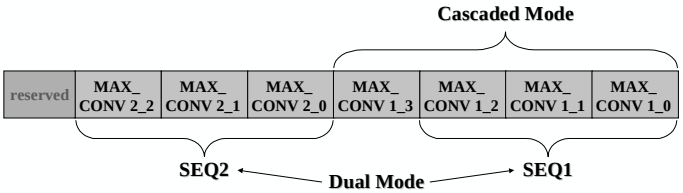
ADCTRL2 @ 70A1h



Maximum Conversion Channels Register

MAX_CONV @ 70A2h

- **Bit fields define the maximum number of autoconversions (binary+1)**



- **Autoconversion session always starts with the "initial state" and continues sequentially until the "end state", if allowed**

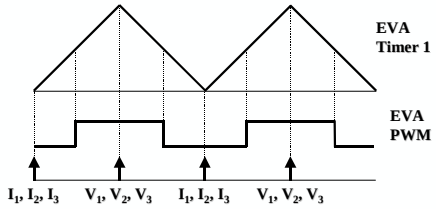
	SEQ1	SEQ2	Cascaded
Initial state	CONV00	CONV08	CONV00
End state	CONV07	CONV15	CONV15

ADC Input Channel Select Sequencing Control Register

	Bits 15-12	Bits 11-8	Bits 7-4	Bits 3-0	
70A3h	CONV03	CONV02	CONV01	CONV00	CHSELSEQ1
70A4h	CONV07	CONV06	CONV05	CONV04	CHSELSEQ2
70A5h	CONV11	CONV10	CONV09	CONV08	CHSELSEQ3
70A6h	CONV15	CONV14	CONV13	CONV12	CHSELSEQ4

Sequencer Example

Example - Sequencer "Start/Stop" Operation



System Requirements:

- Three autoconversions (I_1, I_2, I_3) off trigger 1 (Timer underflow)
- Three autoconversions (V_1, V_2, V_3) off trigger 2 (Timer period)

Event Manager A (EVA) and SEQ1 are used for this example

Example - Sequencer "Start/Stop" Operation

Continued

- MAX_CONV1 is set to 2 and Channel Select Sequencing Control Registers are set to:

Bits →	15-12	11-8	7-4	3-0	
70A3h	V_1	I_3	I_2	I_1	CHSELSEQ1
70A4h	x	x	V_3	V_2	CHSELSEQ2

- Once reset and initialized, SEQ1 waits for a trigger
- First trigger three conversions performed: CONV00 (I_1), CONV01 (I_2), CONV02 (I_3)
- MAX_CONV1 value is reset to 2 (unless changed by software)
- SEQ1 waits for second trigger
- Second trigger three conversions performed: CONV03 (V_1), CONV04 (V_2), CONV05 (V_3)
- End of second autoconversion session, ADC Results registers have the following values:

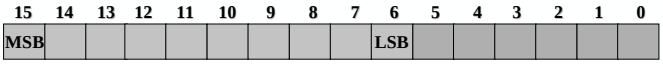
RESULT0	I_1	RESULT3	V_1
RESULT1	I_2	RESULT4	V_2
RESULT2	I_3	RESULT5	V_3

- SEQ1 keeps "waiting" at current state for another trigger
  → User can reset SEQ1 by software to state CONV00 and repeat same trigger 1, 2 session

ADC Conversion Result

ADC Conversion Result Buffer Register

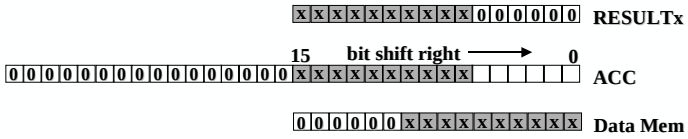
RESULT0 @ 70A8h through RESULT15 @ 70B7h
(Total of 16 Registers)



With V_{REFHI} = 3.3 V, and V_{REFLO} = 0 V, we have:

analog volts	converted value	RESULTx
3.3	3FFh	1111|1111|1100|0000
1.65	1FFh	0111|1111|1100|0000
0.00322	1h	0000|0000|0100|0000
0	0h	0000|0000|0000|0000

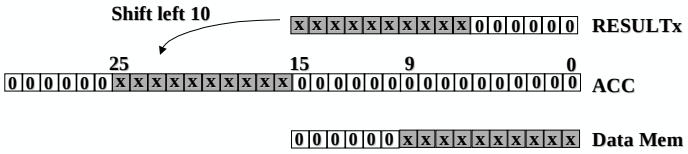
Integer Format: How Do We Handle the Shift?




```
RESULT0   .set    70A8h
          .bss    value, 1

   .text
   LDP    #RESULT0>>7
   LACL   RESULT0
   RPT    #5
   SFR
   LDP    #value
   SACL   value
```

6 words, 13 cycles

Integer Format: A Better Solution for Handling the Shift




```
RESULT0   .set    70A8h
          .bss    value, 1

   .text
   CLRC   SXM
   LDP    #RESULT0>>7
   LACC   RESULT0, 10
   LDP    #value
   SACH   value
```

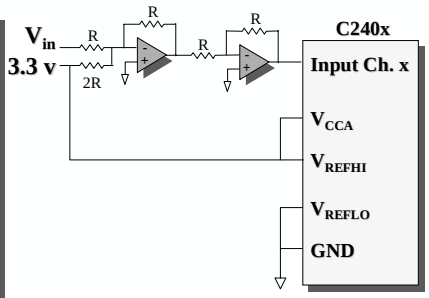
5 words, 7 cycles

What About Signed Input Voltages?

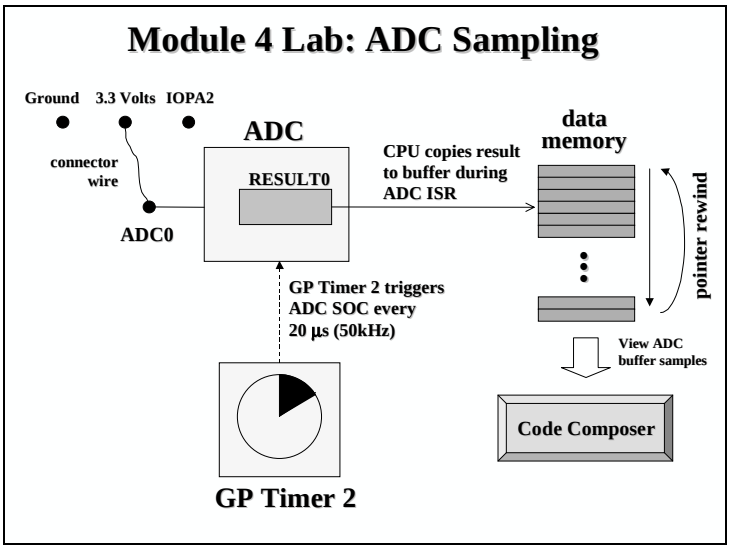
Example: -1.65 V ≤ V_{in} ≤ +1.65 V

Add 1.65 volts to analog input, then subtract "1.65" from digital result

```
RESULT0   .set    70A8h
          .data
offset    .int    1FFh
          .bss    value, 1

   .text
   CLRC   SXM
   LDP    #RESULT0>>7
   LACC   RESULT0, 10
   LDP    #offset
   SUB    offset,16
   LDP    #value
   SACH   value
```



Module 4 Lab: ADC Sampling



Introduction

The objective of this lab is to familiarize the student with the programming and operation of the on-chip ADC on the TMS320LF2407 DSP. The DSP will be setup to sample a single ADC input channel at a prescribed sampling rate and store the conversion result in a buffer in the DSP memory. This buffer will operate in a circular fashion, such that new conversion data continuously overwrites older results in the buffer. Recall that there are three basic ways to initiate an ADC start of conversion (SOC):

1. Using software
    a. ADCIMSTART bit in ADCTRL1 causes an immediate SOC
    b. ADCSOC bit in ADCTRL1 causes an SOC upon completion of the current conversion (if the ADC is currently idle, an SOC occurs immediately)
2. Automatically triggered on user selectable event manager conditions
    a. GP Timer 1 or 2 underflow (e.g. timer count = 0)
    b. GP Timer 1 or 2 period match
    c. GP Timer 1 or 2 compare match
3. Externally triggered using a pin
    a. ADCSOC pin
    b. CAP3 pin (capture unit #3 edge detection)

One or more of these methods may be applicable to your particular application. In this lab, we will be using the ADC for data acquisition. Therefore, one of the GP timers (GP Timer 2) will be configured to automatically trigger an SOC at the desired sampling rate (SOC method 2b above). The ADC end-of-conversion interrupt will be used to prompt the CPU to copy the results of the ADC conversion into a results buffer in memory. This buffer pointer will be managed in a circular fashion, such that new conversion results will continuously overwrite older conversion results in the buffer. In order to generate an interesting input signal, the code also alternately toggles a GPIO pin (IOPA2) high and low in the ADC interrupt service routine. This pin will be connected to the ADC input pin, and sampled. After taking some data, Code Composer will be used to plot the results. A flow chart of the code is shown in Figure 4.1.

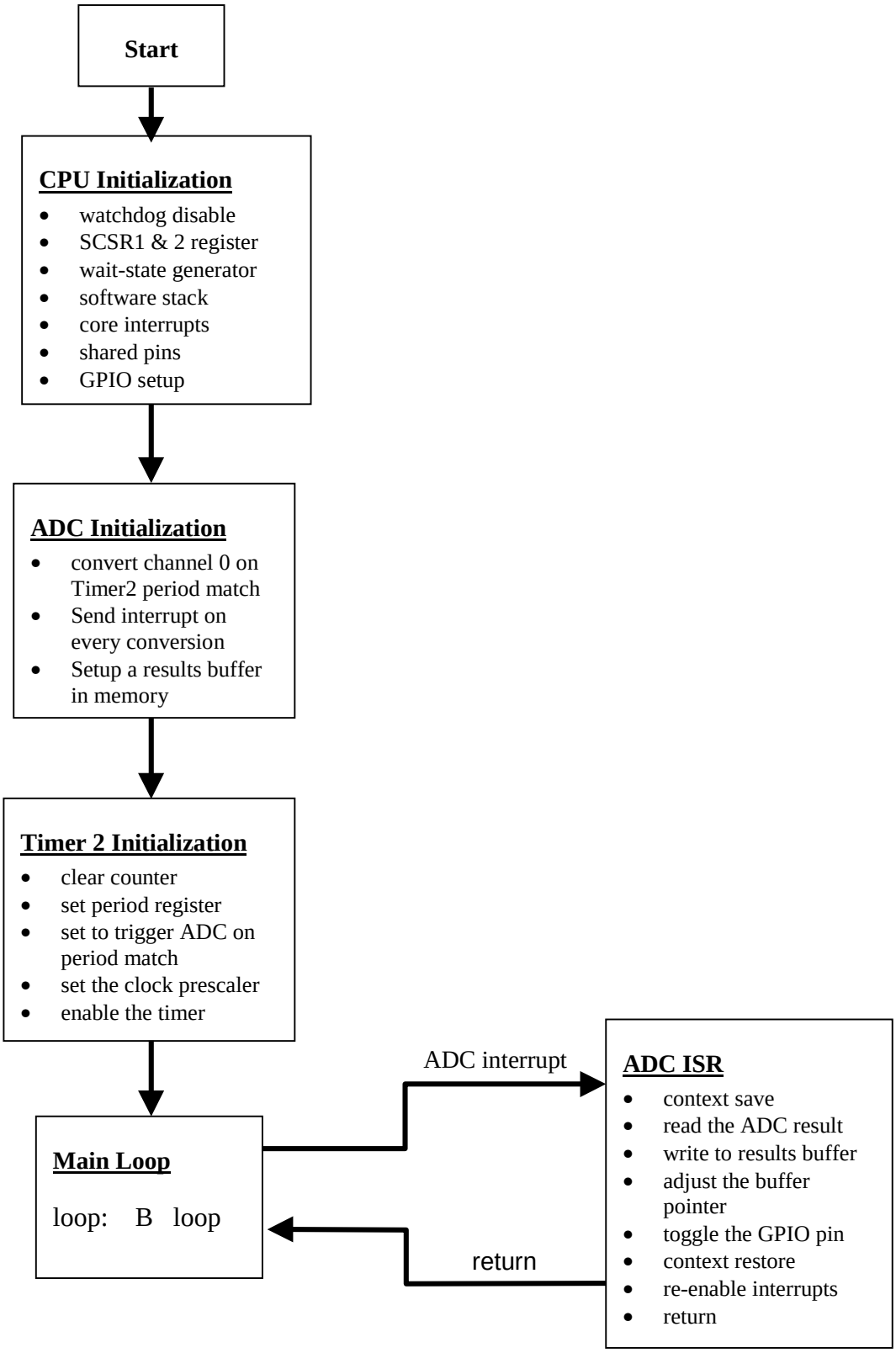


Figure 4.1: Code Flow

➢ **Procedure**

Create a New Project

1. Double-click on the Code Composer icon on the desktop. Maximize Code Composer to fill your screen.

2. Create a new project for this lab. On the menu bar click:

   `Project → New`

   and make sure the "SAVE IN" location is: `C:\DSP24_1DAY\LABS` and type `MOD4LAB` in the file name window. This will create a *make* file which will invoke all the necessary tools to build your project.

3. Add the assembly file to the new project. Click:

   `Project → Add Files to Project`

   and make sure you're looking in `C:\DSP24_1DAY\LABS`. Change the "files of type" to view assembly files (`.ASM`) and select `MOD4LAB.ASM` and click `OPEN`. This will add the file `MOD4LAB.ASM` to your newly created project.

4. Add `MOD4LAB.CMD` to the project using the same procedure.

5. In the project window on the left click the plus sign (+) to the left of `Project`. Now, click on the plus sign next to `MOD4LAB.MAK`. Notice that the `MOD4LAB.CMD` file is listed. Click on the plus sign next to `Source` to see the current source file list (i.e. `MOD4LAB.ASM`).

Setup the Linker

6. Setup the linker options by clicking:

   `Project → Options`

   on the menu bar. Select the Linker tab. In the middle of the screen select "`No Autoinitialization`". (If needed, a map file can be created by using the `Map Filename [-m]` field). Next, select the Assembler tab. In the middle of the screen check "`Enable Source Level Debugging`". (Allows debugging using "mixed mode"). Then select `OK` to save the Build Options.

Build and Load

7. Click the "`Rebuild All`" button and watch the tools run in the build window. To open up more space, close any open files or windows that you do not need.

8. If the "`Load program after build`" option was not selected in Code Composer ("`Option`" menu, click on "`Program Load...`") load the output file into the target. Click: `File → Load Program...`

   Make sure you are looking in `C:\DSP24_1DAY\Labs`. Select `MOD4LAB.OUT` and click `OPEN`.

   Then reset the DSP by clicking on: `Debug → Reset DSP`

9. If code generation is successful, the Dis-Assembly window will display the source file (`MOD4LAB.ASM`) and a yellow highlight on "B start" should appear. This indicates that you are now ready to run.

Running MOD4LAB

10. Inspect `MOD4LAB.ASM` in the source window. Note how the topics covered in the module are used in the program code.

11. Single-step your code a few times into the `MOD4LAB.ASM` file. Then scroll down through the code and place the cursor at the `"loop: B loop"` line. Set a breakpoint by right clicking the mouse key and select `Toggle breakpoint`. Notice that line is highlighted indicating that the breakpoint has been set. Next, locate the code segment that initializes the ADC results buffer with the value 0x2407. Notice the effective use of the RPT instruction coupled with the SACL for this purpose.

12. Open a memory window to view some of the contents of the ADC results buffer. The address label for the ADC results buffer is *adc_buf* .

13. Run the code by using the <F5> key, or using the Run button on the vertical toolbar, or using `Debug → Run` on the menu bar. It should halt when it hits the breakpoint set at the main loop. Verify that the ADC results buffer has been initialized with the expected value (e.g. 0x2407).

14. Remove the breakpoint at `"loop: B loop"` by right clicking the mouse key and select `Toggle breakpoint`.

15. Using the connector wire provided, connect the ADCIN0 (pin # P1-2) to "GND" (pin # P6-49) on the eZdsp™. ***Exercise care when doing this as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™!*** Then run the code again, and halt it after a few seconds. Verify that the ADC results buffer contains the expected value of 0x0000.

16. Adjust the connector wire to connect the ADCIN0 (pin # P1-2) to "+3.3V" (pin # P6-45) on the eZdsp™. ***Exercise care when doing this as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™!*** Then run the code again, and halt it after a few seconds. Verify that the ADC results buffer contains the expected value of 0x03FF.

17. Adjust the connector wire to connect the ADCIN0 (pin # P1-2) to IOPA2 (pin # P2-5) on the eZdsp™. ***Exercise care when doing this as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™!*** Then run the code again, and halt it after a few seconds. Examine the contents of the ADC results buffer (the contents should be alternating 0x0000 and 0x03FF values). Are the contents what you expected?

18. Open and setup a graph to plot a 50-point window of the ADC results buffer.
    Click: `View → Graph → Time/Frequency… and set the following values:`

Start Address	adc_buf
Acquisition Buffer Size	50
Display Data Size	50
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Time Display Unit	µs

Select `OK` to save the graph options.

19. Recall that the code toggled the IOPA2 pin alternately high and low. If you had an oscilloscope available to display IOPA2, you would expect to see a square-wave. Why does Code Composer plot resemble a saw-tooth wave? What is the signal processing term for what is happening here?

20. Recall that the program toggled the IOPA2 pin at a 50kHz rate. Therefore, a complete cycle (toggle high, then toggle low) occurs at half this rate, or 25kHz. We therefore expect the period of the waveform to be 40µs. Confirm this by measuring the period of the saw-tooth wave using the graph (you may want to enlarge the graph window using the mouse). The measurement is best done with the mouse. The lower left-hand corner of the graph window will display the X and Y axis values. Subtract the X-axis values taken over a complete waveform period.

Things to Think About (optional)

1. How could start-of-conversion method 1b (see introduction section) be utilized in this lab instead of SOC method 2b? What advantage does method 2b offer?

2. The ADC uses a successive approximation method that is clocked from a pre-scaled version of the CPU clock. The ADC is capable of running from the CPUCLK with a pre-scale factor of 1 (i.e. maximum speed). Can you think of a reason why you would want to use a larger pre-scale value than 1?

= = = =

MOD4LAB.ASM

```
; MOD4LAB.ASM

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Address Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
               .include f2407.h            ;address definitions

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Constant Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
adc_rate       .set    599                 ;50kHz sampling rate
adc_buf_len    .set    300                 ;ADC results buffer length
stk_len        .set    100                 ;stack length

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Variable and Section Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
               .bss    temp,1              ;general purpose variable

adc_buf_ptr    .usect  "buffer",1          ;ptr to next free buff addr
adc_buf        .usect  "buffer",adc_buf_len ;reserve space for buffer
stk            .usect  "stack",stk_len     ;reserve space for stack

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Interrupt Vector Table for Core
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .sect   "vectors"
                B       start           ;00h reset
int1:           B       adc_isr         ;02h INT1
int2:           B       int2            ;04h INT2
int3:           B       int3            ;06h INT3
int4:           B       int4            ;08h INT4
int5:           B       int5            ;0Ah INT5
int6:           B       int6            ;0Ch INT6
int7:           B       int7            ;0Eh reserved
int8:           B       int8            ;10h INT8  user-defined
int9:           B       int9            ;12h INT9  user-defined
int10:          B       int10           ;14h INT10 user defined
int11:          B       int11           ;16h INT11 user defined
int12:          B       int12           ;18h INT12 user defined
int13:          B       int13           ;1Ah INT13 user defined
int14:          B       int14           ;1Ch INT14 user defined
int15:          B       int15           ;1Eh INT15 user defined
int16:          B       int16           ;20h INT16 user defined
int17:          B       int17           ;22h TRAP
int18:          B       int18           ;24h NMI
int19:          B       int19           ;26h reserved
int20:          B       int20           ;28h INT20 user defined
int21:          B       int21           ;2Ah INT21 user defined
int22:          B       int22           ;2Ch INT22 user defined
int23:          B       int23           ;2Eh INT23 user defined
int24:          B       int24           ;30h INT24 user defined
int25:          B       int25           ;32h INT25 user defined
int26:          B       int26           ;34h INT26 user defined
int27:          B       int27           ;36h INT27 user defined
```

```
int28:          B       int28           ;38h INT28 user defined
int29:          B       int29           ;3Ah INT29 user defined
int30:          B       int30           ;3Ch INT30 user defined
int31:          B       int31           ;3Eh INT31 user defined

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;                   M A I N   R O U T I N E
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              .text
start:

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Disable the watchdog
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #11101000b, WDCR
* bit 7          1:      clear WD flag
* bit 6          1:      disable the dog
* bit 5-3        101:    must be written as 101
* bit 2-0        000:    WDCLK divider = 1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the system control registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #0000000011111101b, SCSR1
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15         0:      reserved
* bit 14         0:      CLKOUT = CPUCLK
* bit 13-12      00:     IDLE1 selected for low-power mode
* bit 11-9       000:    PLL x4 mode
* bit 8          0:      reserved
* bit 7          1:      1 = enable ADC module clock
* bit 6          1:      1 = enable SCI module clock
* bit 5          1:      1 = enable SPI module clock
* bit 4          1:      1 = enable CAN module clock
* bit 3          1:      1 = enable EVB module clock
* bit 2          1:      1 = enable EVA module clock
* bit 1          0:      reserved
* bit 0          1:      clear the ILLADR bit

        SPLK    #0000000000001111b, SCSR2
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15-6       0's:    reserved
* bit 5          0:      DO NOT clear the WD OVERRIDE bit
* bit 4          0:      XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
* bit 3          1:      1 = disable the BOOT ROM
* bit 2          1:      MP/MC*, 1 = Flash addresses mapped external
* bit 1-0        11:     11 = SARAM mapped to prog and data

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Set wait states for external memory interface on eZdsp(TM) LF2407
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
        LDP     #temp     ;set data page

        SPLK    #0000000001000000b, temp
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15-11     0's:    reserved
* bit 10-9      00:     bus visibility off
* bit 8-6       001:    1 wait-state for I/O space
* bit 5-3       000:    0 wait-state for data space
* bit 2-0       000:    0 wait-state for program space

        OUT     temp, WSGR

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the software stack
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LAR     AR1, #stk       ;AR1 is stack pointer
        MAR     *, AR1          ;ARP = AR1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the core interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #0h             ;set data page
        SPLK    #111111b,IFR    ;clear any pending interrupts
        SPLK    #000001b,IMR    ;enable desired interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup shared I/O pins
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page

        SPLK    #0000000000000000b,MCRA
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      0=IOPB7,     1=TCLKINA
* bit 14        0:      0=IOPB6,     1=TDIRA
* bit 13        0:      0=IOPB5,     1=T2PWM/T2CMP
* bit 12        0:      0=IOPB4,     1=T1PWM/T1CMP
* bit 11        0:      0=IOPB3,     1=PWM6
* bit 10        0:      0=IOPB2,     1=PWM5
* bit 9         0:      0=IOPB1,     1=PWM4
* bit 8         0:      0=IOPB0,     1=PWM3
* bit 7         0:      0=IOPA7,     1=PWM2
* bit 6         0:      0=IOPA6,     1=PWM1
* bit 5         0:      0=IOPA5,     1=CAP3
* bit 4         0:      0=IOPA4,     1=CAP2/QEP2
* bit 3         0:      0=IOPA3,     1=CAP1/QEP1
* bit 2         0:      0=IOPA2,     1=XINT1
* bit 1         0:      0=IOPA1,     1=SCIRXD
* bit 0         0:      0=IOPA0,     1=SCITXD

        SPLK    #1111111000000000b,MCRB
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        1:      0=reserved,  1=TMS2 (always write as 1)
* bit 14        1:      0=reserved,  1=TMS  (always write as 1)
* bit 13        1:      0=reserved,  1=TD0  (always write as 1)
```

```
* bit 12        1:      0=reserved,  1=TDI  (always write as 1)
* bit 11        1:      0=reserved,  1=TCK  (always write as 1)
* bit 10        1:      0=reserved,  1=EMU1 (always write as 1)
* bit 9         1:      0=reserved,  1=EMU0 (always write as 1)
* bit 8         0:      0=IOPD0,     1=XINT2/ADCSOC
* bit 7         0:      0=IOPC7,     1=CANRX
* bit 6         0:      0=IOPC6,     1=CANTX
* bit 5         0:      0=IOPC5,     1=SPISTE
* bit 4         0:      0=IOPC4,     1=SPICLK
* bit 3         0:      0=IOPC3,     1=SPISOMI
* bit 2         0:      0=IOPC2,     1=SPISIMO
* bit 1         0:      0=IOPC1,     1=BIO*
* bit 0         0:      0=IOPC0,     1=W/R*

        SPLK    #0000000000000000b,MCRC
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      0=IOPF6,     1=IOPF6
* bit 13        0:      0=IOPF5,     1=TCLKINB
* bit 12        0:      0=IOPF4,     1=TDIRB
* bit 11        0:      0=IOPF3,     1=T4PWM/T4CMP
* bit 10        0:      0=IOPF2,     1=T3PWM/T3CMP
* bit 9         0:      0=IOPF1,     1=CAP6
* bit 8         0:      0=IOPF0,     1=CAP5/QEP4
* bit 7         0:      0=IOPE7,     1=CAP4/QEP3
* bit 6         0:      0=IOPE6,     1=PWM12
* bit 5         0:      0=IOPE5,     1=PWM11
* bit 4         0:      0=IOPE4,     1=PWM10
* bit 3         0:      0=IOPE3,     1=PWM9
* bit 2         0:      0=IOPE2,     1=PWM8
* bit 1         0:      0=IOPE1,     1=PWM7
* bit 0         0:      0=IOPE0,     1=CLKOUT

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup IOPA2 pin for use as output
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page
        LACC    PADATDIR        ;ACC = PADATDIR
        OR      #0400h          ;IOPA2 is output
        SACL    PADATDIR        ;write back to GPIO port register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the ADC
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page

        SPLK    #0100000000000000b, ADCTRL1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 14        1:      1 = reset ADC module

        SPLK    #0000000000000000b, MAX_CONV
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-7      0's:    reserved
* bit 6-4       000:    MAX_CONV2 value
```

```
* bit 3-0       0000:   MAX_CONV1 value (0 means 1 conversion)

        SPLK    #0000000000000000b, CHSELSEQ1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-12     0000:   CONV03 channel
* bit 11-8      0000:   CONV02 channel
* bit 7-4       0000:   CONV01 channel
* bit 3-0       0000:   CONV00 channel (only active conversion)

        SPLK    #0010000000010000b, ADCTRL1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      RESET, 0=no action, 1=reset ADC
* bit 13-12     10:     SOFT and FREE, 10=stop after current conversion
* bit 11-8      0000:   ACQ_Prescaler, 0000 = 1 x Tclk
* bit 7         0:      CPS, 0: Fclk=CPUCLK/1, 1: Fclk=CPUCLK/2
* bit 6         0:      CONT_RUN, 0=start/stop mode, 1=continuous run
* bit 5         0:      0=hi priority int, 1=low priority int
* bit 4         1:      0=dual sequencer, 1=cascaded sequencer
* bit 3         0:      0=calibration mode disabled
* bit 2         0:      BRG_ENA, used in calibration mode only
* bit 1         0:      HI/LO, no effect in normal operation mode
* bit 0         0:      0=self-test mode disabled

        SPLK    #0100011100000010b, ADCTRL2
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      EVB_SOC_SEQ, 0=no action
* bit 14        1:      RST_SEQ1/STRT_CAL, 0=no action
* bit 13        0:      SOC_SEQ1, 0=clear any pending SOCs
* bit 12        0:      SEQ1_BSY, read-only
* bit 11-10     01:     INT_ENA_SEQ1, 01=int on every SEQ1 conv
* bit 9         1:      INT_FLAG_SEQ1, write 1 to clear
* bit 8         1:      EVA_SOC_SEQ1, 1=SEQ1 start from EVA
* bit 7         0:      EXT_SOC_SEQ1, 1=SEQ1 start from ADCSOC pin
* bit 6         0:      RST_SEQ2, 0=no action
* bit 5         0:      SOC_SEQ2, no effect in cascaded mode
* bit 4         0:      SEQ2_BSY, read-only
* bit 3-2       00:     INT_ENA_SEQ2, 00=int disabled
* bit 1         1:      INT_FLAG_SEQ2, write 1 to clear
* bit 0         0:      EVB_SOC_SEQ2, 1=SEQ2 started by EVB

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the buffer for the ADC results
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #adc_buf_ptr      ;set data page
        LAR     AR0, #adc_buf     ;pointer to results buffer
        SAR     AR0, adc_buf_ptr  ;initialize adc_buf_ptr
        MAR     *, AR0            ;ARP = AR0
        LACC    #2407h            ;ACC=0x2407
        LDP     #temp
        SPLK    #adc_buf_len-1, temp
        RPT     temp              ;repeat #adc_buf_len-1 times
        SACL    *+                ;initialize the buffer
```

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup GP Timer2 to trigger an ADC conversion
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA           ;set data page

        SPLK    #0000h, T2CNT     ;clear timer2 counter
        SPLK    #adc_rate, T2PR   ;set timer2 period

        SPLK    #0000010000000000b, GPTCONA  ;init GPTCON register
*                |||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      T2STAT - read only
* bit 13        0:      T1STAT - read only
* bit 12-11     00:     reserved
* bit 10-9      10:     T2TOADC, 10 = ADCSOC on period match
* bit 8-7       00:     T1TOADC, 00 = no ADCSOC
* bit 6         0:      1 = enable all timer compare outputs
* bit 5-4       00:     reserved
* bit 3-2       00:     00 = T2PIN forced low
* bit 1-0       00:     00 = T1PIN forced low

        SPLK    #0001000001000000b, T2CON   ;init T2CON register
*                |||||||||||||||
*                FEDCBA9876543210
* bit 15-14     00:     stop immediately on emulator suspend
* bit 13        0:      reserved
* bit 12-11     10:     10 = continous-up count mode
* bit 10-8      000:    000 = x/1 prescaler
* bit 7         0:      0 = use own TENABLE bit
* bit 6         1:      1 = enable timer
* bit 5-4       00:     00 = CPUCLK is clock source
* bit 3-2       00:     00 = reload compare reg on underflow
* bit 1         0:      0 = disable timer compare
* bit 0         0:      0 = use own period register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the event manager interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA           ;set data page
        SPLK    #0FFFFh, EVAIFRA ;clear all EVA group A interrupts
        SPLK    #0FFFFh, EVAIFRB ;clear all EVA group B interrupts
        SPLK    #0FFFFh, EVAIFRC ;clear all EVA group C interrupts
        SPLK    #00000h, EVAIMRA ;enabled desired EVA group A intrpts
        SPLK    #00000h, EVAIMRB ;enabled desired EVA group B intrpts
        SPLK    #00000h, EVAIMRC ;enabled desired EVA group C intrpts

        LDP     #DP_EVB           ;set data page
        SPLK    #0FFFFh, EVBIFRA ;clear all EVB group A interrupts
        SPLK    #0FFFFh, EVBIFRB ;clear all EVB group B interrupts
        SPLK    #0FFFFh, EVBIFRC ;clear all EVB group C interrupts
        SPLK    #00000h, EVBIMRA ;enabled desired EVB group A intrpts
        SPLK    #00000h, EVBIMRB ;enabled desired EVB group B intrpts
        SPLK    #00000h, EVBIMRC ;enabled desired EVB group C intrpts
```

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Enable global interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        CLRC    INTM            ;enable global interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Main loop
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
loop:   NOP
        B       loop            ;branch to loop



*************************************************************************
*  G E N E R A L  I N T E R R U P T  S E R V I C E  R O U T I N E S  *
*************************************************************************

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ADC Interrupt Service Routine
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
adc_isr:

;context save
        MAR     *,AR1               ;ARP=stack pointer
        MAR     *+                  ;skip one stack location
        SST     #1, *+              ;save ST1
        SST     #0, *+              ;save ST0
        SACH    *+                  ;save ACCH
        SACL    *+                  ;save ACCL
        SAR     AR2, *+             ;save AR2

;clear the INT_FLAG_SEQ1 and read the ADC result
        CLRC    SXM
        LDP     #DP_PF2              ;set data page
        LACC    ADCTRL2              ;read and write ADCTRL2
        SACL    ADCTRL2              ;to clear the INT_FLAG_SEQ1
        LACC    RESULT0,10           ;read ADC RESULT0

;store the data value to the buffer
        LDP     #adc_buf_ptr        ;set data page
        LAR     AR2, adc_buf_ptr    ;AR2 points to the buffer
        MAR     *, AR2              ;set ARP
        SACH    *+                  ;store result
        SAR     AR2, adc_buf_ptr    ;store updated pointer

;brute-force the circular buffer
        LAR     AR0, #(adc_buf+adc_buf_len-1)  ;AR0 points to last
                                               ;buffer entry
        CMPR    2                              ;TC set if AR(ARP) > AR0
        BCND    adc_isr1, NTC                  ;branch if TC not set
        SPLK    #adc_buf, adc_buf_ptr          ;re-init the pointer

adc_isr1:

;reset ADC SEQ1 to CONV00 state
        LDP     #DP_PF2             ;set data page
        LACC    ADCTRL2             ;read ADCTRL2
```

```
        OR      #4000h              ;set bit 14 (RST_SEQ1/STRT_CAL bit)
        SACL    ADCTRL2             ;write back to reset SEQ1

;toggle the IOPA2 pin
        LDP     #DP_PF2             ;set data page
        LACC    PADATDIR            ;ACC = PADATDIR
        XOR     #0004h              ;toggle IOPA2 bit
        SACL    PADATDIR            ;write back to GPIO port register

;context restore
        MAR     *, AR1              ;ARP = AR1
        MAR     *-                  ;SP points to last entry
        LAR     AR2, *-             ;restore AR2
        LACL    *-                  ;restore ACCL
        ADD     *-,16               ;restore ACCH
        LST     #0, *-              ;restore ST0
        LST     #1, *-              ;restore ST1, de-allocate skipped
                                    ;stack location
        CLRC    INTM                ;re-enable global interrupts
        RET
```

PWM Generation & Event Manager

Learning Objectives

Learning Objectives

- **Pulse Width Modulation (PWM) Review**
- **Understand the GP Timers**
- **Understand PWM with GP Compares**
- **Understand PWM with Full Compare Units**
- **Understand Capture and QEP Functions**
- **Lab: PWM Generation**

Note: **Two identical Event Manager (EVA and EVB) modules are available on some devices. For simplicity, only EVA will be explained.**

Module Topics

PWM Generation & Event Manager 5-1

Module Topics 5-2

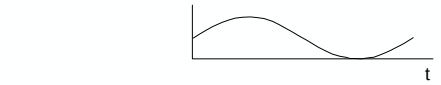
Pulse Width Modulation (PWM) 5-3

Event Manager 5-4
General-Purpose Timer 5-4
Compare Unit 5-7
Dead-Band 5-9
Capture Unit 5-10
Quadrature Encoder Pulse (QEP) 5-13

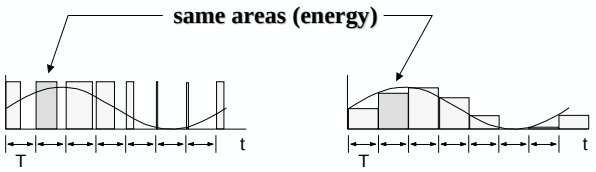
Module 5 Lab: Event Manager and PWM 5-14
MOD5LAB.ASM 5-19

Pulse Width Modulation (PWM)

PWM Signal Representation



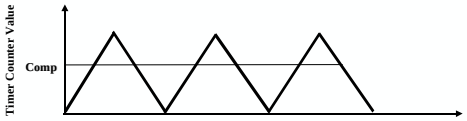
Original Signal



PWM representation PAM representation

What Does a DSP Need to Generate a PWM?

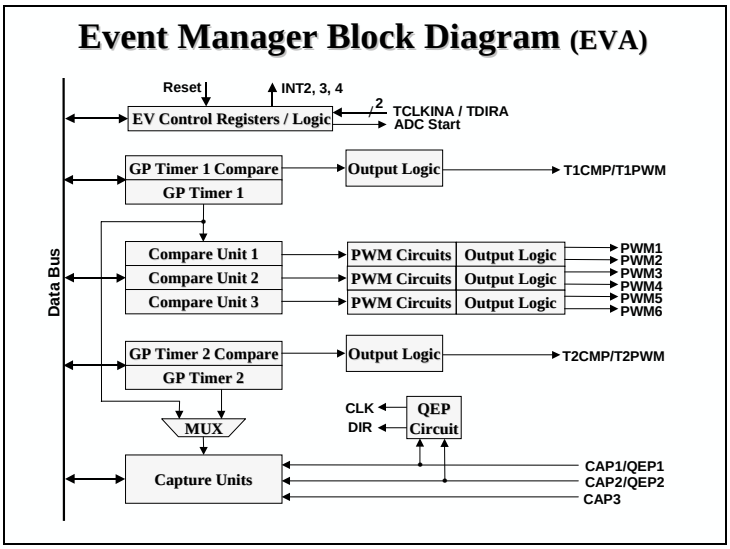
- Clocked counter



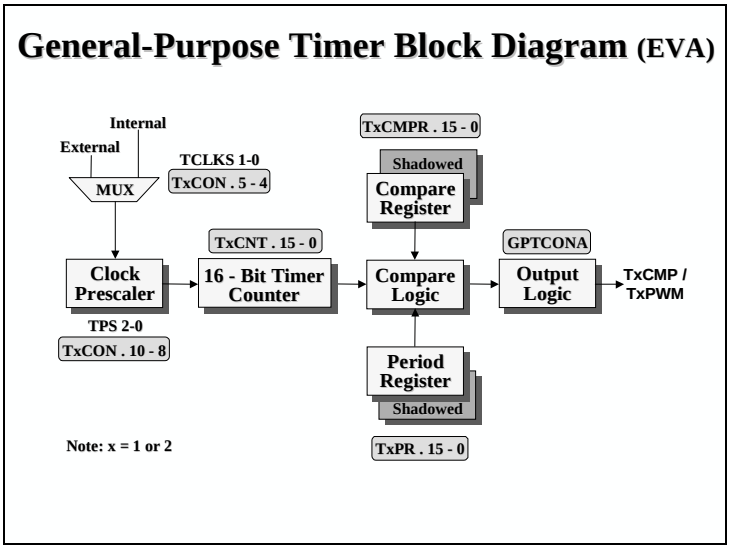
- Compare Function

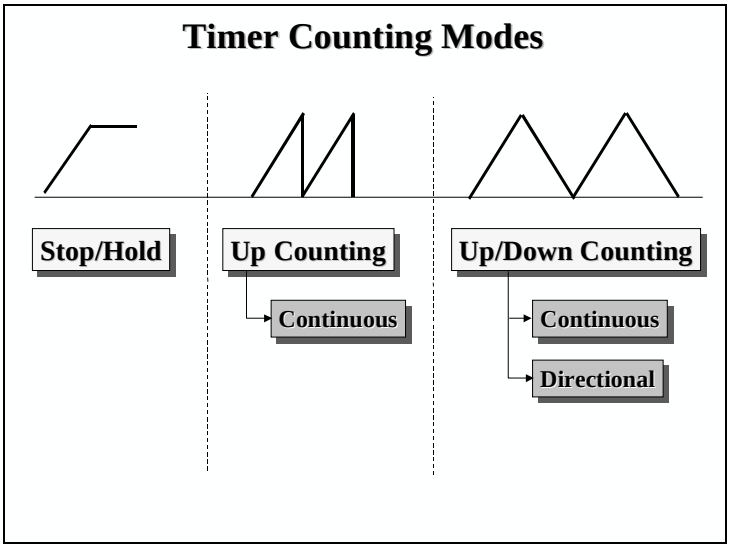
GP Timer	COUNTER
	PERIOD
	COMPARE

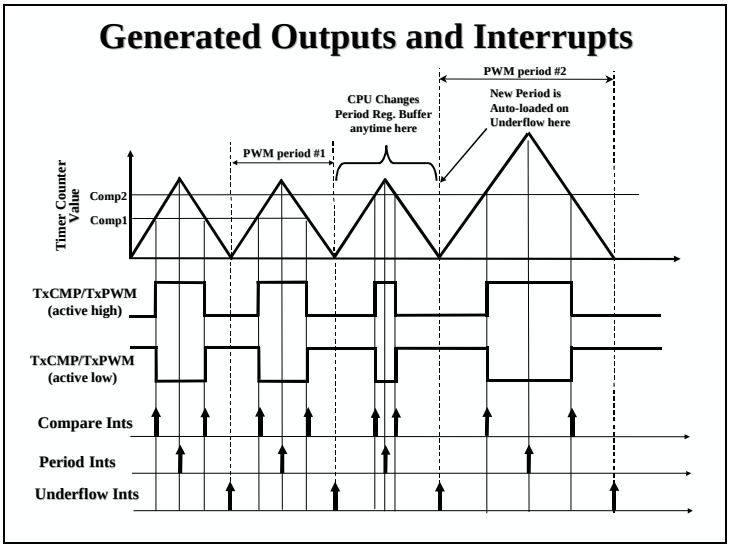
Event Manager



General-Purpose Timer

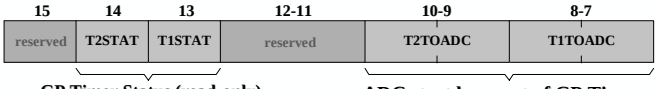



Timer Counting Modes
Stop/Hold
Up Counting
Continuous
Up/Down Counting
Continuous
Directional


Generated Outputs and Interrupts
PWM period #2
CPU Changes Period Reg. Buffer anytime here
New Period is Auto-loaded on Underflow here
PWM period #1
Timer Counter Value
Comp2
Comp1
TxCMP/TxPWM (active high)
TxCMP/TxPWM (active low)
Compare Ints
Period Ints
Underflow Ints

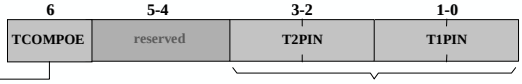
GP Timer Control Register A (EVA)

GPTCONA @ 7400h



GP Timer Status (read-only)
0 = counting down
1 = counting up

ADC start by event of GP Timer x
00: no event starts ADC
01: setting of underflow interrupt flag
10: setting of period interrupt flag
11: setting of compare interrupt



Compare Output Enable

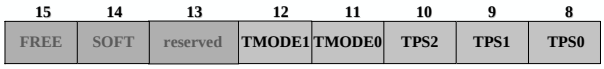
0 = all disabled (hi-impedanced)
1 = all enabled

TxPWM/TxCMP Output Pin Conditioning
00: forced low
01: active low
10: active high
11: forced high

Timer 1/2 Control Register (EVA)

T1CON @ 7404h / T2CON @ 7408h

Upper Byte:

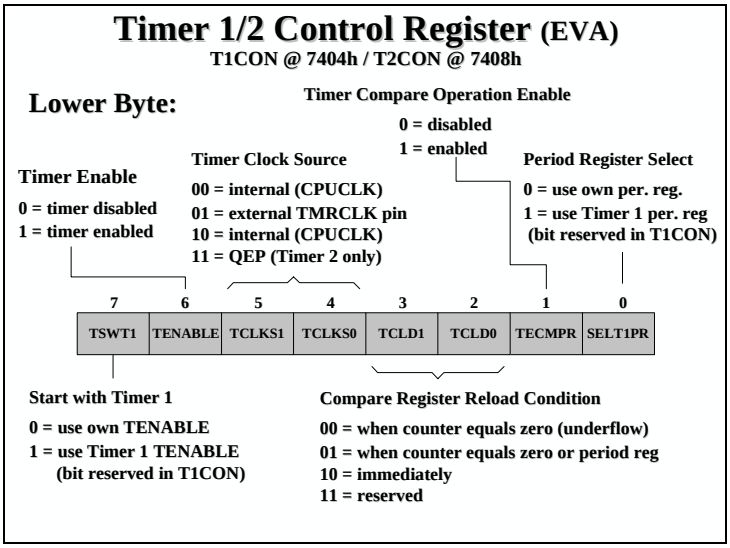


Emulation Halt Behavior

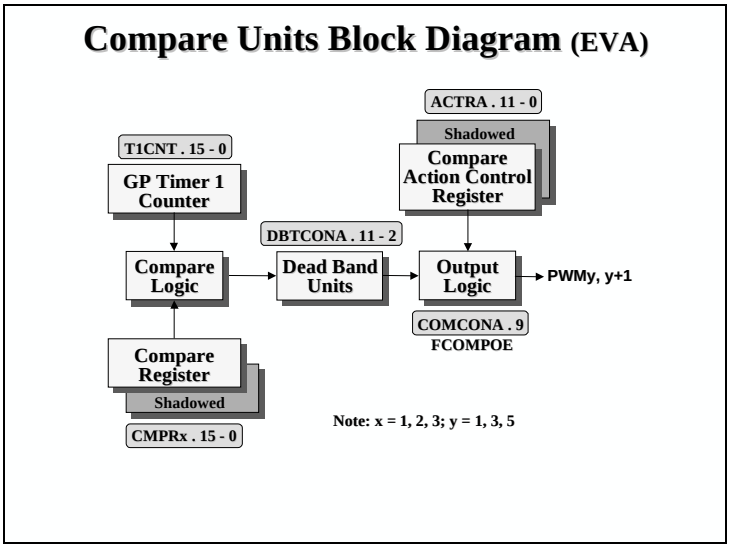
Timer Clock Prescale

000: ÷ 1 **100: ÷ 16**
001: ÷ 2 **101: ÷ 32**
010: ÷ 4 **110: ÷ 64**
011: ÷ 8 **111: ÷ 128**

Count Mode Select
00 = stop/hold
01 = continuous-up/down
10 = continuous-up
11 = directional-up/down



Compare Unit



Compare Control Register (EVA)

COMCONA @ 7411h

Compare Enable
0 = disable
1 = enable

Space Vector PWM
0 = SV disabled
1 = SV enabled

Full Compare Output
0 = disable (hi-Z'd)
1 = enable

15	14	13	12	11	10	9	8 - 0
CENABLE	CLD1	CLD0	SVENABLE	ACTRLD1	ACTRLD0	FCOMPOE	reserved

CMPRx reload condition

00 = when T1CNT = 0
01 = when T1CNT = 0 or T1PR
10 = immediately
11 = reserved

ACTRA reload condition

00 = when T1CNT = 0
01 = when T1CNT = 0 or T1PR
10 = immediately
11 = reserved

Compare Action Control Register (EVA)

ACTRA @ 7413h

Basic Space Vector Bits
can write as 0 when SC not in use

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP6ACT1	CMP6ACT0	CMP5ACT1	CMP5ACT0

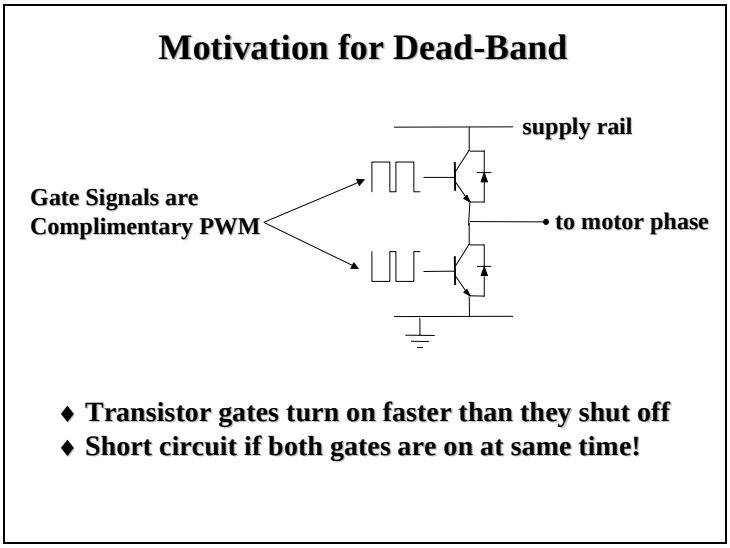
7	6	5	4	3	2	1	0
CMP4ACT1	CMP4ACT0	CMP3ACT1	CMP3ACT0	CMP2ACT1	CMP2ACT0	CMP1ACT1	CMP1ACT0

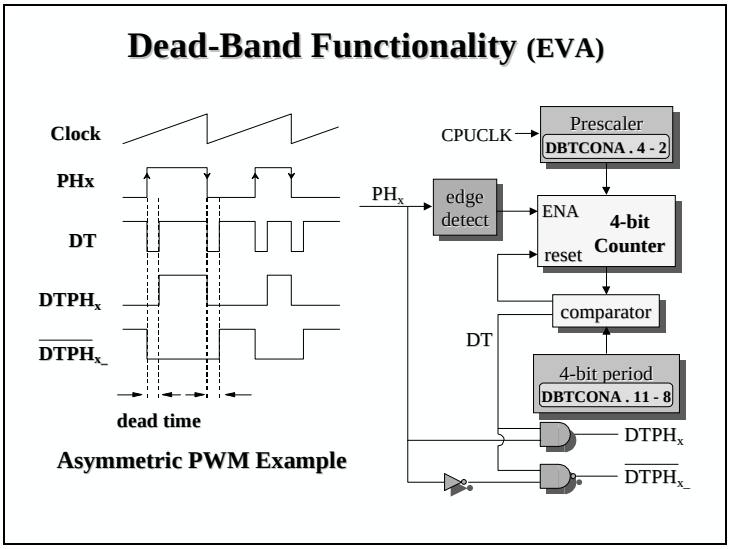
Pin Action on Compare: CMPyACT1-0

00	**force low**
01	**active low**
10	**active high**
11	**forced high**

SV Rotation Direction
can write as 0 when SV not in use

Dead-Band

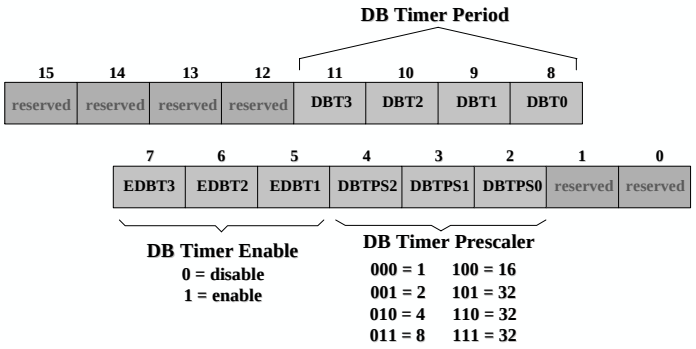

Motivation for Dead-Band
supply rail
Gate Signals are
Complimentary PWM
to motor phase
♦ Transistor gates turn on faster than they shut off
♦ Short circuit if both gates are on at same time!


Dead-Band Functionality (EVA)
Clock
PHx
DT
DTPHx
DTPHx_
dead time
Asymmetric PWM Example
CPUCLK
Prescaler
DBTCONA . 4 - 2
PHx
edge
detect
ENA
reset
4-bit
Counter
comparator
DT
4-bit period
DBTCONA . 11 - 8
DTPHx
DTPHx_

Dead-Band Timer Control Register (EVA)

DBTCONA @ 7415h

dead time = DB period * DB prescaler * CPUCLK period



Capture Unit

Some Uses for the Capture Units

- **Synchronized ADC start with capture event**
- **Measure the time width of a pulse**
- **Low speed velocity estimation from incr. encoder:**

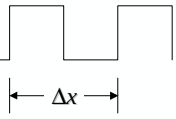
Problem: At low speeds, calculation of speed based on a measured position change at fixed time intervals produces large estimate errors

$$v_k \approx \frac{x_k - x_{k-1}}{\Delta t}$$

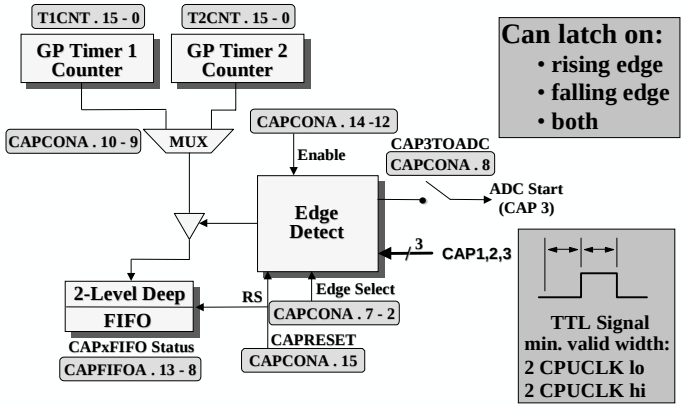
Alternative: Estimate the speed using a measured time interval at fixed position intervals

$$v_k \approx \frac{\Delta x}{t_k - t_{k-1}}$$

Signal from one quadrature encoder channel

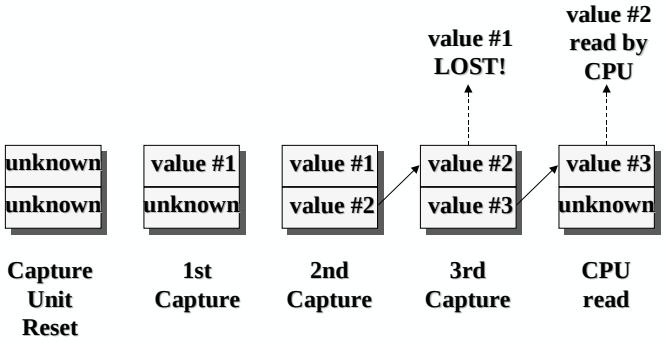


Capture Units Block Diagram (EVA)



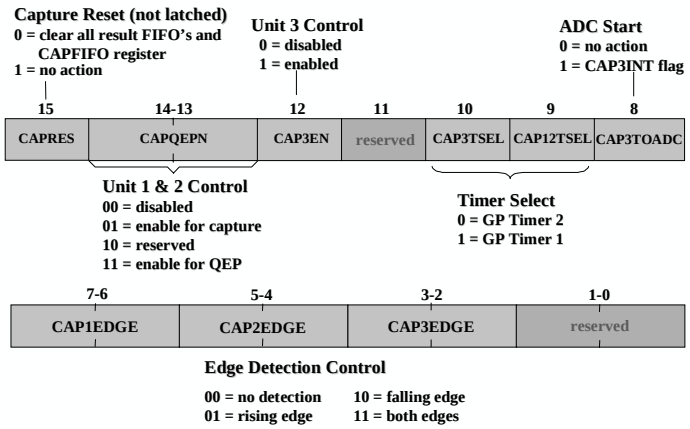
Capture Unit FIFO Operation

- CPU can read both upper and lower FIFO registers separately; Standard approach is to read upper, but lower can be read



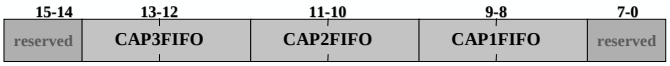
Capture Control Register (EVA)

CAPCONA @ 7420h



Capture FIFO Status Register (EVA)

CAPFIFOA @ 7422h



CAPxFIFO bits are automatically adjusted on a capture or FIFO read

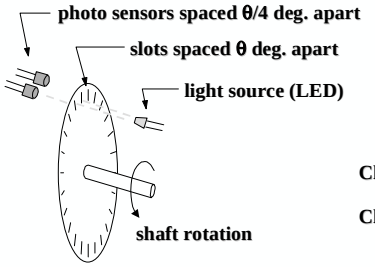
FIFOx Status:

00 = empty
01 = one entry
10 = two entries
11 = three entries attempted, 1st entry lost

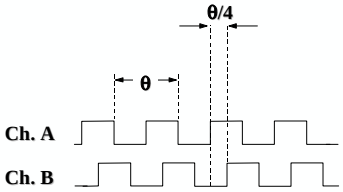
Quadrature Encoder Pulse (QEP)

What is an Incremental Quadrature Encoder?

A digital (angular) position sensor

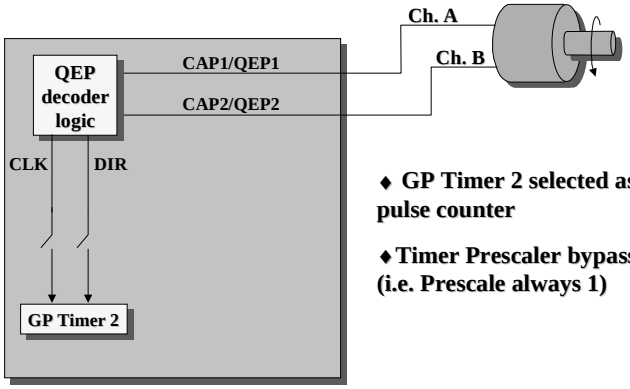


Incremental Optical Encoder



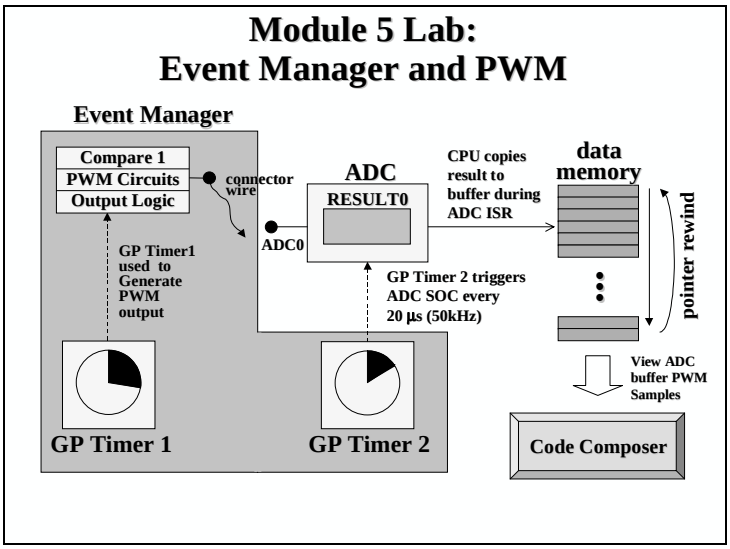
Quadrature Output from Photo Sensors

Incremental Encoder Connections (EVA)



- GP Timer 2 selected as pulse counter
- Timer Prescaler bypassed (i.e. Prescale always 1)

Module 5 Lab: Event Manager and PWM



Introduction

The objective of this lab is to familiarize the student with the programming and operation of the Event Manager on the TMS320LF2407 DSP. General-Purpose Timer 1 and Compare 1 will be setup to generate a 2kHz, 25% duty cycle symmetric PWM waveform. The waveform will then be sampled with the on-chip analog-to-digital converter and displayed using the graphing feature of Code Composer.

Recall that there are two timebases associated with Event Manager "EVA". The timer and its associated control registers will be configured to produce a PWM waveform on the eZdsp™ LF2407:

General Purpose Timer #1 – PWM Generation

- Used as timebase for Compare1 generation of PWM waveform

General Purpose Timer #2 – ADC Conversion Trigger & PWM Generation

- Used as timebase for triggering ADC samples (same as previous lab)

The software for this lab configures the Event Manager and the ADC. It is entirely interrupt driven. A flow chart of the code is shown in Figure 5.1.

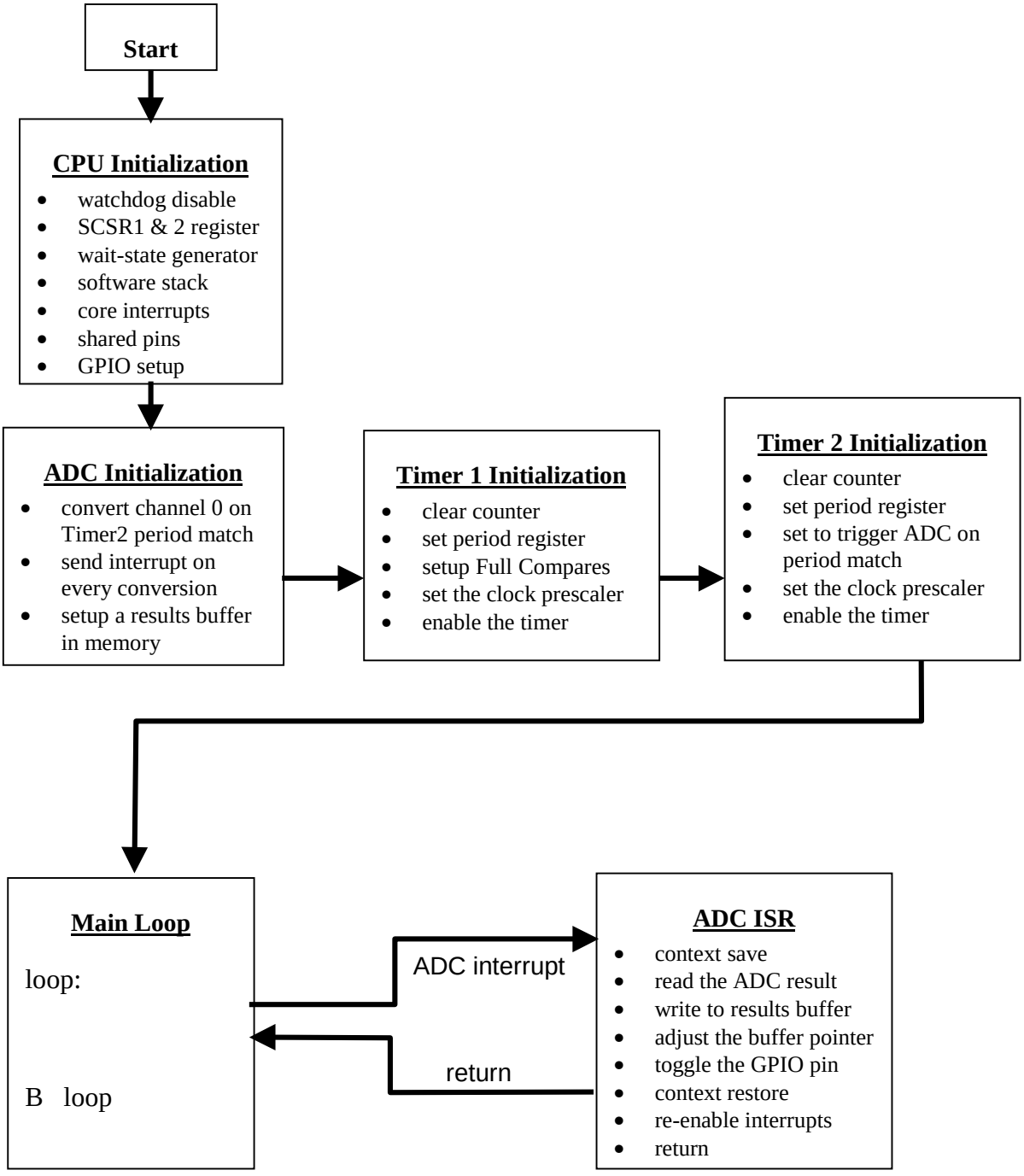


Figure 5.1: Code Flow

➢ **Procedure**

Create a New Project

1. Double-click on the Code Composer icon on the desktop. Maximize Code Composer to fill your screen.

2. Create a new project for this lab. On the menu bar click:

   `Project → New`

   and make sure the "SAVE IN" location is: `C:\DSP24_1DAY\LABS` and type `MOD5LAB` in the file name window. This will create a *make* file which will invoke all the necessary tools to build your project.

3. Add the assembly file to the new project. Click:

   `Project → Add Files to Project`

   and make sure you're looking in `C:\DSP24_1DAY\LABS`. Change the "files of type" to view assembly files (`.ASM`) and select `MOD5LAB.ASM` and click `OPEN`. This will add the file `MOD5LAB.ASM` to your newly created project.

4. Add `MOD5LAB.CMD` to the project using the same procedure.

5. In the project window on the left click the plus sign (+) to the left of `Project`. Now, click on the plus sign next to `MOD5LAB.MAK`. Notice that the `MOD5LAB.CMD` file is listed. Click on the plus sign next to `Source` to see the current source file list (i.e. `MOD5LAB.ASM`).

Setup the Linker

6. Setup the linker options by clicking:

   `Project → Options`

   on the menu bar. Select the Linker tab. In the middle of the screen select "`No Autoinitialization`". (If needed, a map file can be created by using the `Map Filename [-m]` field). Next, select the Assembler tab. In the middle of the screen check "`Enable Source Level Debugging`". (Allows debugging using "mixed mode"). Then select `OK` to save the Build Options.

Build and Load

7. Click the "`Rebuild All`" button and watch the tools run in the build window. To open up more space, close any open files or windows that you do not need.

8. If the "`Load program after build`" option was not selected in Code Composer ("`Option`" menu, click on "`Program Load...`") load the output file into the target. Click: `File → Load Program...`

   Make sure you are looking in `C:\DSP24_1DAY\Labs`. Select `MOD5LAB.OUT` and click `OPEN`.

   Then reset the DSP by clicking on: `Debug → Reset DSP`

9. If code generation is successful, the Dis-Assembly window will display the source file (`MOD5LAB.ASM`) and a yellow highlight on "B start" should appear. This indicates that you are now ready to run.

Running MOD5LAB

10. Inspect `MOD5LAB.ASM` in the source window. Note how the topics covered in the module are used in the program code.

11. Single-step your code a few times into the `MOD5LAB.ASM` file.

12. Open a memory window to view some of the contents of the ADC results buffer. The address label for the ADC results buffer is *adc_buf*.

13. Using the connector wire provided, connect the PWM1 (pin # P2-9) to ADCIN0 (pin # P1-2) on the eZdsp™. ***Exercise care when doing this as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™!*** Then run the code, and halt it after a few seconds. Verify that the ADC results buffer contains updated values.

14. Open and setup a graph to plot a 50-point window of the ADC results buffer.
    Click: `View` → `Graph` → `Time/Frequency...` and set the following values:

Start Address	adc_buf
Acquisition Buffer Size	50
Display Data Size	50
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Time Display Unit	µs

    Select `OK` to save the graph options.

15. The graphical display should show the generated 2kHz, 25% duty cycle symmetric PWM waveform. The period of a 2kHz signal is 500µs. You can confirm this by measuring the period of the waveform using the graph (you may want to enlarge the graph window using the mouse). The measurement is best done with the mouse. The lower left-hand corner of the graph window will display the X and Y-axis values. Subtract the X-axis values taken over a complete waveform period (you can use the PC calculator program found in Microsoft Windows to do this).

Frequency Domain Graphing Feature of Code Composer

16. Code Composer also has the ability to make frequency domain plots. It does this by using the PC to perform a Fast Fourier Transform (FFT) of the DSP data. Let's make a frequency domain plot of the contents in the ADC results buffer (i.e. the PWM waveform).

Click: `View` → `Graph` → `Time/Frequency…` and set the following values:

Display Type	FFT Magnitude
Start Address	adc_buf
Acquisition Buffer Size	256
FFT Framesize	256
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000

Select `OK` to save the graph options.

17. On the plot window, left-click the mouse to move the vertical marker line and observe the frequencies of the different magnitude peaks. Do the peaks occur at the expected frequencies?

Optional

18. You might want to experiment with this code by changing some of the values or modifying the code. Try generating another waveform of a different frequency and duty cycle. Also, try to generate complementary pair PWM outputs. Next, try to generate additional simultaneous waveforms by using Compare 1, Compare 2 and Compare 3. (Hint: don't forget to setup the proper shared I/O pins, etc.).

MOD5LAB.ASM

```
; MOD5LAB.ASM

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Address Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
               .include f2407.h            ;address definitions

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Constant Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
adc_rate       .set    599                 ;50kHz sampling rate

adc_buf_len    .set    300                 ;ADC results buffer length
stk_len        .set    100                 ;stack length

pwm_half_per   .set    7500                ;period/2 for 2kHz symmet-
ric PWM
pwm_duty       .set    5625                ;25% duty cycle

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Variable and Sections Definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
               .bss    temp,1              ;general purpose variable

adc_buf_ptr    .usect  "buffer",1          ;ptr to next free buff addr
adc_buf        .usect  "buffer",adc_buf_len ;reserve space for buffer
stk            .usect  "stack",stk_len     ;reserve space for stack

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Interrupt Vector Table for Core
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                .sect   "vectors"
                B       start           ;00h reset
int1:           B       adc_isr         ;02h INT1
int2:           B       int2            ;04h INT2
int3:           B       int3            ;06h INT3
int4:           B       int4            ;08h INT4
int5:           B       int5            ;0Ah INT5
int6:           B       int6            ;0Ch INT6
int7:           B       int7            ;0Eh reserved
int8:           B       int8            ;10h INT8  user-defined
int9:           B       int9            ;12h INT9  user-defined
int10:          B       int10           ;14h INT10 user defined
int11:          B       int11           ;16h INT11 user defined
int12:          B       int12           ;18h INT12 user defined
int13:          B       int13           ;1Ah INT13 user defined
int14:          B       int14           ;1Ch INT14 user defined
int15:          B       int15           ;1Eh INT15 user defined
int16:          B       int16           ;20h INT16 user defined
int17:          B       int17           ;22h TRAP
int18:          B       int18           ;24h NMI
int19:          B       int19           ;26h reserved
int20:          B       int20           ;28h INT20 user defined
int21:          B       int21           ;2Ah INT21 user defined
int22:          B       int22           ;2Ch INT22 user defined
```

```
int23:          B       int23           ;2Eh INT23 user defined
int24:          B       int24           ;30h INT24 user defined
int25:          B       int25           ;32h INT25 user defined
int26:          B       int26           ;34h INT26 user defined
int27:          B       int27           ;36h INT27 user defined
int28:          B       int28           ;38h INT28 user defined
int29:          B       int29           ;3Ah INT29 user defined
int30:          B       int30           ;3Ch INT30 user defined
int31:          B       int31           ;3Eh INT31 user defined

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;                   M A I N   R O U T I N E
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              .text
start:

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Disable the watchdog
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #11101000b, WDCR
* bit 7          1:      clear WD flag
* bit 6          1:      disable the dog
* bit 5-3        101:    must be written as 101
* bit 2-0        000:    WDCLK divider = 1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the system control registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #0000000011111101b, SCSR1
;                |||||||||||||||
;               FEDCBA9876543210
* bit 15         0:      reserved
* bit 14         0:      CLKOUT = CPUCLK
* bit 13-12      00:     IDLE1 selected for low-power mode
* bit 11-9       000:    PLL x4 mode
* bit 8          0:      reserved
* bit 7          1:      1 = enable ADC module clock
* bit 6          1:      1 = enable SCI module clock
* bit 5          1:      1 = enable SPI module clock
* bit 4          1:      1 = enable CAN module clock
* bit 3          1:      1 = enable EVB module clock
* bit 2          1:      1 = enable EVA module clock
* bit 1          0:      reserved
* bit 0          1:      clear the ILLADR bit

        SPLK    #0000000000001111b, SCSR2
;                |||||||||||||||
;               FEDCBA9876543210
* bit 15-6       0's:    reserved
* bit 5          0:      DO NOT clear the WD OVERRIDE bit
* bit 4          0:      XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
* bit 3          1:      1 = disable the BOOT ROM
* bit 2          1:      MP/MC*, 1 = Flash addresses mapped external
```

```
* bit 1-0       11:     11 = SARAM mapped to prog and data

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Set wait states for external memory interface on eZdsp(TM) LF2407
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #temp     ;set data page

        SPLK    #0000000001000000b, temp
;                ||||||||||||||||
;                FEDCBA9876543210
* bit 15-11     0's:    reserved
* bit 10-9      00:     bus visibility off
* bit 8-6       001:    1 wait-state for I/O space
* bit 5-3       000:    0 wait-state for data space
* bit 2-0       000:    0 wait-state for program space

        OUT     temp, WSGR

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the software stack
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LAR     AR1, #stk       ;AR1 is stack pointer
        MAR     *, AR1          ;ARP = AR1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the core interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #0h             ;set data page
        SPLK    #111111b,IFR    ;clear any pending interrupts
        SPLK    #000001b,IMR    ;enable desired interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup shared I/O pins
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page

        SPLK    #0000000001000000b,MCRA
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      0=IOPB7,     1=TCLKINA
* bit 14        0:      0=IOPB6,     1=TDIRA
* bit 13        0:      0=IOPB5,     1=T2PWM/T2CMP
* bit 12        0:      0=IOPB4,     1=T1PWM/T1CMP
* bit 11        0:      0=IOPB3,     1=PWM6
* bit 10        0:      0=IOPB2,     1=PWM5
* bit 9         0:      0=IOPB1,     1=PWM4
* bit 8         0:      0=IOPB0,     1=PWM3
* bit 7         0:      0=IOPA7,     1=PWM2
* bit 6         1:      0=IOPA6,     1=PWM1
* bit 5         0:      0=IOPA5,     1=CAP3
* bit 4         0:      0=IOPA4,     1=CAP2/QEP2
* bit 3         0:      0=IOPA3,     1=CAP1/QEP1
* bit 2         0:      0=IOPA2,     1=XINT1
* bit 1         0:      0=IOPA1,     1=SCIRXD
* bit 0         0:      0=IOPA0,     1=SCITXD
```

```
        SPLK    #1111111000000000b,MCRB
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        1:      0=reserved,  1=TMS2 (always write as 1)
* bit 14        1:      0=reserved,  1=TMS  (always write as 1)
* bit 13        1:      0=reserved,  1=TD0  (always write as 1)
* bit 12        1:      0=reserved,  1=TDI  (always write as 1)
* bit 11        1:      0=reserved,  1=TCK  (always write as 1)
* bit 10        1:      0=reserved,  1=EMU1 (always write as 1)
* bit 9         1:      0=reserved,  1=EMU0 (always write as 1)
* bit 8         0:      0=IOPD0,     1=XINT2/ADCSOC
* bit 7         0:      0=IOPC7,     1=CANRX
* bit 6         0:      0=IOPC6,     1=CANTX
* bit 5         0:      0=IOPC5,     1=SPISTE
* bit 4         0:      0=IOPC4,     1=SPICLK
* bit 3         0:      0=IOPC3,     1=SPISOMI
* bit 2         0:      0=IOPC2,     1=SPISIMO
* bit 1         0:      0=IOPC1,     1=BIO*
* bit 0         0:      0=IOPC0,     1=W/R*

        SPLK    #0000000000000000b,MCRC
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      0=IOPF6,     1=IOPF6
* bit 13        0:      0=IOPF5,     1=TCLKINB
* bit 12        0:      0=IOPF4,     1=TDIRB
* bit 11        0:      0=IOPF3,     1=T4PWM/T4CMP
* bit 10        0:      0=IOPF2,     1=T3PWM/T3CMP
* bit 9         0:      0=IOPF1,     1=CAP6
* bit 8         0:      0=IOPF0,     1=CAP5/QEP4
* bit 7         0:      0=IOPE7,     1=CAP4/QEP3
* bit 6         0:      0=IOPE6,     1=PWM12
* bit 5         0:      0=IOPE5,     1=PWM11
* bit 4         0:      0=IOPE4,     1=PWM10
* bit 3         0:      0=IOPE3,     1=PWM9
* bit 2         0:      0=IOPE2,     1=PWM8
* bit 1         0:      0=IOPE1,     1=PWM7
* bit 0         0:      0=IOPE0,     1=CLKOUT

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup IOPA2 pin for use as output
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page
        LACC    PADATDIR        ;ACC = PADATDIR
        OR      #0400h          ;IOPA2 is output
        SACL    PADATDIR        ;write back to GPIO port register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the ADC
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF2         ;set data page

        SPLK    #0100000000000000b, ADCTRL1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 14        1:      1 = reset ADC module
```

```
        SPLK    #0000000000000000b, MAX_CONV
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-7      0's:    reserved
* bit 6-4       000:    MAX_CONV2 value
* bit 3-0       0000:   MAX_CONV1 value (0 means 1 conversion)

        SPLK    #0000000000000000b, CHSELSEQ1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-12     0000:   CONV03 channel
* bit 11-8      0000:   CONV02 channel
* bit 7-4       0000:   CONV01 channel
* bit 3-0       0000:   CONV00 channel (only active conversion)

        SPLK    #0010000000010000b, ADCTRL1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      RESET, 0=no action, 1=reset ADC
* bit 13-12     10:     SOFT and FREE, 10=stop after current conversion
* bit 11-8      0000:   ACQ_Prescaler, 0000 = 1 x Tclk
* bit 7         0:      CPS, 0: Fclk=CPUCLK/1, 1: Fclk=CPUCLK/2
* bit 6         0:      CONT_RUN, 0=start/stop mode, 1=continuous run
* bit 5         0:      0=hi priority int, 1=low priority int
* bit 4         1:      0=dual sequencer, 1=cascaded sequencer
* bit 3         0:      0=calibration mode disabled
* bit 2         0:      BRG_ENA, used in calibration mode only
* bit 1         0:      HI/LO, no effect in normal operation mode
* bit 0         0:      0=self-test mode disabled

        SPLK    #0100011100000010b, ADCTRL2
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      EVB_SOC_SEQ, 0=no action
* bit 14        1:      RST_SEQ1/STRT_CAL, 0=no action
* bit 13        0:      SOC_SEQ1, 0=clear any pending SOCs
* bit 12        0:      SEQ1_BSY, read-only
* bit 11-10     01:     INT_ENA_SEQ1, 01=int on every SEQ1 conv
* bit 9         1:      INT_FLAG_SEQ1, write 1 to clear
* bit 8         1:      EVA_SOC_SEQ1, 1=SEQ1 start from EVA
* bit 7         0:      EXT_SOC_SEQ1, 1=SEQ1 start from ADCSOC pin
* bit 6         0:      RST_SEQ2, 0=no action
* bit 5         0:      SOC_SEQ2, no effect in cascaded mode
* bit 4         0:      SEQ2_BSY, read-only
* bit 3-2       00:     INT_ENA_SEQ2, 00=int disabled
* bit 1         1:      INT_FLAG_SEQ2, write 1 to clear
* bit 0         0:      EVB_SOC_SEQ2, 1=SEQ2 started by EVB

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the buffer for the ADC results
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #adc_buf_ptr      ;set data page
        LAR     AR0, #adc_buf     ;pointer to results buffer
        SAR     AR0, adc_buf_ptr  ;initialize adc_buf_ptr
        MAR     *, AR0            ;ARP = AR0
```

```
        LACC    #2407h            ;ACC=0x2407
        LDP     #temp
        SPLK    #adc_buf_len-1, temp
        RPT     temp              ;repeat #adc_buf_len-1 times
        SACL    *+                ;initialize the buffer

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup GP Timer2 to trigger an ADC conversion
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA           ;set data page

        SPLK    #0000h, T2CNT     ;clear timer2 counter
        SPLK    #adc_rate, T2PR   ;set timer2 period

        SPLK    #0000010000000000b, GPTCONA  ;init GPTCON register
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      T2STAT - read only
* bit 13        0:      T1STAT - read only
* bit 12-11     00:     reserved
* bit 10-9      10:     T2TOADC, 10 = ADCSOC on period match
* bit 8-7       00:     T1TOADC, 00 = no ADCSOC
* bit 6         0:      1 = enable all timer compare outputs
* bit 5-4       00:     reserved
* bit 3-2       00:     00 = T2PIN forced low
* bit 1-0       00:     00 = T1PIN forced low

        SPLK    #0001000001000000b, T2CON   ;init T2CON register
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-14     00:     stop immediately on emulator suspend
* bit 13        0:      reserved
* bit 12-11     10:     10 = continous-up count mode
* bit 10-8      000:    000 = x/1 prescaler
* bit 7         0:      0 = use own TENABLE bit
* bit 6         1:      1 = enable timer
* bit 5-4       00:     00 = CPUCLK is clock source
* bit 3-2       00:     00 = reload compare reg on underflow
* bit 1         0:      0 = disable timer compare
* bit 0         0:      0 = use own period register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup Full Compare 1 and GP Timer 1 to generate PWM
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA             ;set data page
        SPLK    #0000h, T1CON       ;disable timer 1
        SPLK    #0000h, T1CNT       ;clear timer 1 counter
        SPLK    #pwm_half_per, T1PR  ;setup timer 1 period

        SPLK    #0000h, DBTCONA     ;deadband units off
        SPLK    #pwm_duty, CMPR1    ;set PWM1 duty cycle

        SPLK    #0000000000000010b, ACTRA  ;PWM1 set for active high
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      space vector dir is CCW (don't care)
```

```
* bit 14-12     000:    basic space vector is 000 (dont' care)
* bit 11-10     00:     PWM6/IOPB3 pin forced low
* bit 9-8       00:     PWM5/IOPB2 pin forced low
* bit 7-6       00:     PWM4/IOPB1 pin forced low
* bit 5-4       00:     PWM3/IOPB0 pin forced low
* bit 3-2       00:     PWM2/IOPA7 pin forced low
* bit 1-0       10:     PWM1/IOPA6 pin active high

        SPLK    #1000001000000000b, COMCONA  ;configure COMCON
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        1:      1 = enable full compare operation
* bit 14-13     00:     00 = reload CMPRx regs on timer 1 underflow
* bit 12        0:      0 = space vector disabled
* bit 11-10     00:     00 = reload ACTR on timer 1 underflow
* bit 9         1:      1 = enable PWM pins
* bit 8-0       0's:    reserved

        SPLK    #0000100001000000b, T1CON   ;init T1CON register
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-14     00:     stop immediately on emulator suspend
* bit 13        0:      reserved
* bit 12-11     01:     01 = continous-up/down count mode
* bit 10-8      000:    000 = x/1 prescaler
* bit 7         0:      0 = use own TENABLE bit
* bit 6         1:      1 = enable timer
* bit 5-4       00:     00 = CPUCLK is clock source
* bit 3-2       00:     00 = reload compare reg on underflow
* bit 1         0:      0 = disable timer compare
* bit 0         0:      0 = use own period register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the event manager interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA          ;set data page
        SPLK    #0FFFFh, EVAIFRA ;clear all EVA group A interrupts
        SPLK    #0FFFFh, EVAIFRB ;clear all EVA group B interrupts
        SPLK    #0FFFFh, EVAIFRC ;clear all EVA group C interrupts
        SPLK    #00000h, EVAIMRA ;enabled desired EVA group A intrpts
        SPLK    #00000h, EVAIMRB ;enabled desired EVA group B intrpts
        SPLK    #00000h, EVAIMRC ;enabled desired EVA group C intrpts

        LDP     #DP_EVB          ;set data page
        SPLK    #0FFFFh, EVBIFRA ;clear all EVB group A interrupts
        SPLK    #0FFFFh, EVBIFRB ;clear all EVB group B interrupts
        SPLK    #0FFFFh, EVBIFRC ;clear all EVB group C interrupts
        SPLK    #00000h, EVBIMRA ;enabled desired EVB group A intrpts
        SPLK    #00000h, EVBIMRB ;enabled desired EVB group B intrpts
        SPLK    #00000h, EVBIMRC ;enabled desired EVB group C intrpts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Enable global interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        CLRC    INTM            ;enable global interrupts
```

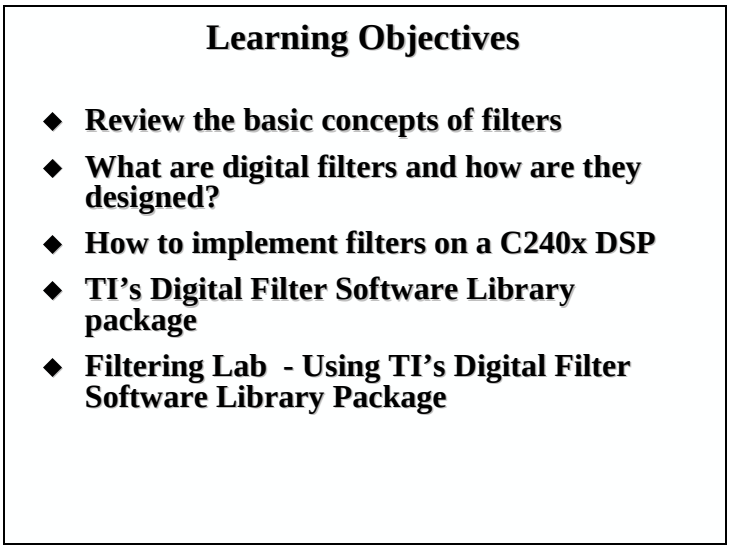
```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Main loop
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
loop:   NOP
        B       loop            ;branch to loop



*************************************************************************
*  G E N E R A L  I N T E R R U P T  S E R V I C E  R O U T I N E S  *
*************************************************************************

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ADC Interrupt Service Routine
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
adc_isr:

;context save
        MAR     *,AR1               ;ARP=stack pointer
        MAR     *+                  ;skip one stack location
        SST     #1, *+              ;save ST1
        SST     #0, *+              ;save ST0
        SACH    *+                  ;save ACCH
        SACL    *+                  ;save ACCL
        SAR     AR2, *+             ;save AR2

;clear the INT_FLAG_SEQ1 and read the ADC result
        CLRC    SXM
        LDP     #DP_PF2              ;set data page
        LACC    ADCTRL2              ;read and write ADCTRL2
        SACL    ADCTRL2              ;to clear the INT_FLAG_SEQ1
        LACC    RESULT0,10           ;read ADC RESULT0

;store the data value to the buffer
        LDP     #adc_buf_ptr         ;set data page
        LAR     AR2, adc_buf_ptr     ;AR2 points to the buffer
        MAR     *, AR2               ;set ARP
        SACH    *+                   ;store result
        SAR     AR2, adc_buf_ptr     ;store updated pointer

;brute-force the circular buffer
        LAR     AR0, #(adc_buf+adc_buf_len-1)  ;AR0 points to last
buffer entry
        CMPR    2                              ;TC set if AR(ARP) > AR0
        BCND    adc_isr1, NTC                  ;branch if TC not set
        SPLK    #adc_buf, adc_buf_ptr          ;re-init the pointer

adc_isr1:

;reset ADC SEQ1 to CONV00 state
        LDP     #DP_PF2              ;set data page
        LACC    ADCTRL2              ;read ADCTRL2
        OR      #4000h               ;set bit 14 (RST_SEQ1/STRT_CAL bit)
        SACL    ADCTRL2              ;write back to reset SEQ1

;toggle the IOPA2 pin
        LDP     #DP_PF2              ;set data page
```

```
        LACC    PADATDIR            ;ACC = PADATDIR
        XOR     #0004h              ;toggle IOPA2 bit
        SACL    PADATDIR            ;write back to GPIO port register

;context restore
        MAR     *, AR1              ;ARP = AR1
        MAR     *-                  ;SP points to last entry
        LAR     AR2, *-             ;restore AR2
        LACL    *-                  ;restore ACCL
        ADD     *-,16               ;restore ACCH
        LST     #0, *-              ;restore ST0
        LST     #1, *-              ;restore ST1, de-allocate skipped
                                    ;stack location
        CLRC    INTM                ;re-enable global interrupts
        RET
```

This page is intentionally blank.

Filtering Techniques

Learning Objectives



Module Topics

Filtering Techniques 6-1

Module Topics 6-2

Filtering 6-3
TI Modular Software Library 6-6
TI Digital Filter Software Package 6-7
Intro to Lab Exercise 6-9

Module 6 Lab: Digital Filter Basics 6-12
Appendix 6A 6-27
Appendix 6B 6-30
Appendix 6C – Code Listings 6-32

Filtering

Filtering & DSP

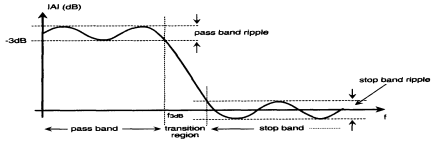
- **What is Filtering**
  - **The selective attenuation of certain frequency components of a signal**
    - **Isolate desired portions of a signal**
    - **Remove interfering signals**
- **Why use digital filtering?**
  - **Programmability - *adaptive filters***
  - **Stability**
  - **Repeatability**
  - **Realization of systems not possible or feasible with analog techniques**

Analog Filter Examples

Highpass Filter

Lowpass Filter

Typical Filter Performance Design Parameters



Digital Filters

- **The mathematical theory of digital filters is based on the Z-transform**
- **In practice, digital filters are realized by multiplying the incoming data by a set of *coefficient* values and summing the results to produce the filtered output signal (Basically, a Sum-of-Products…)**
- **The number of coefficients is directly related to the filter performance (Each coefficient-data multiplier junction is referred to as a "TAP"); more "TAPS" generally means better performance**
- **Filter coefficients are normally generated mathematically with special filter design software (ie: Matlab, or other Filter Generation Package…)**

Coefficients Determine the Filter Response

Digital Filters are designed by using Software Packages (ie: Matlab, or other Filter Generation Packages…) to mathematically generate coefficient tables to match the desired filter type (LowPass, Bandpass…) and performance specifications (Rolloff, Ripple, Passband…) .

Basic Filter Types:

Lowpass - passes everything below a designated frequency

Highpass - passes everything above a designated frequency

Bandpass - passes everything between two designated frequencies

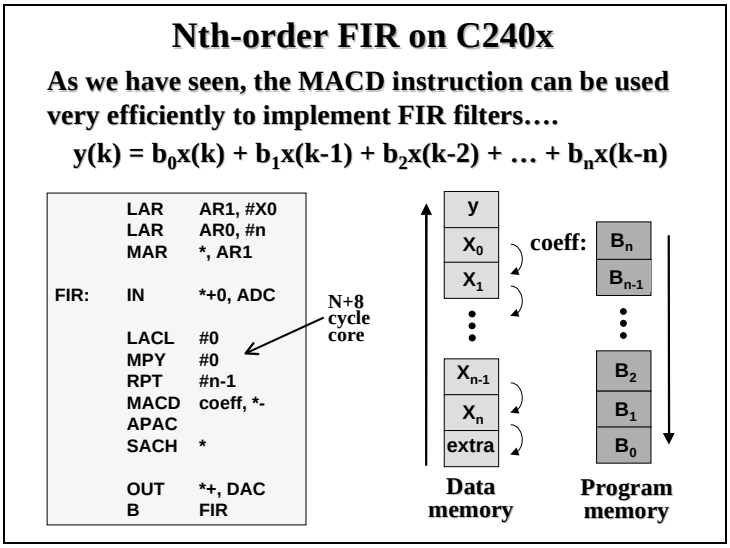
Bandstop - rejects everything between two designated frequencies.

Digital Filter Types

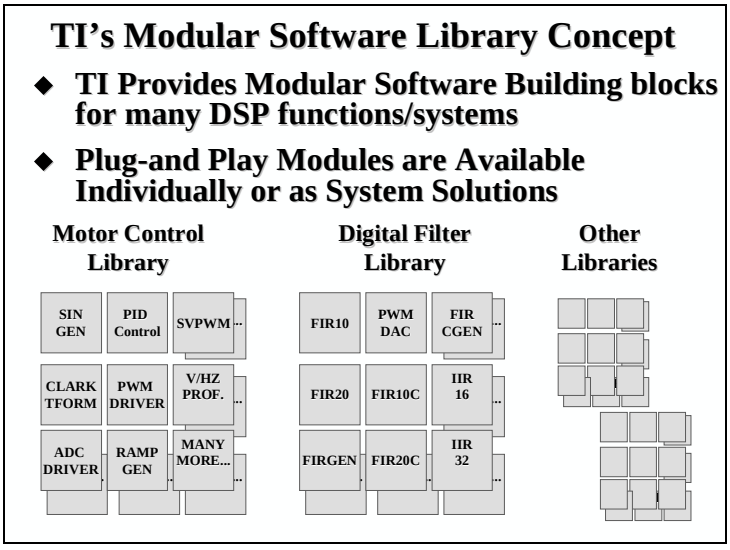
- **Finite Impulse Response (FIR)**
  - **The impulse response of an FIR filter decays to zero in a finite amount of time**
  - **FIR filters exhibit linear phase response**
    - **Highly desirable in signal processing**
    - **Difficult to achieve with analog techniques**
  - **FIR filters are feedforward filters**
- **Infinite Impulse Response (IIR)**
  - **The impulse response of an IIR filter theoretically never decays to zero**
  - **IIR filters do not exhibit linear phase response**
  - **IIR filters are feedback filters**
  - **IIR filters have steeper roll-off/tap than FIR Filters**

FIR Filter Implementation

- **As we have seen, DSPs have specialized instructions for the implementation of FIR filters**
  - **MAC - Multiply and accumulate**
  - **MACD - Multiply and accumulate and move data**
- **DSPs have multiple memory buses to access data and coefficient values in the same clock cycle**
- **DSPs execute FIR filters at a 1-tap per clock cycle rate by using a "Repeat" instruction in combination with a "MACD" instruction**



TI Modular Software Library



TI Digital Filter Software Package

TI's Digital Filter Software Package

TI Provides a Digital Filter Software Package that can be used to implement various forms of Digital Filters

- **The Digital Filter Software is provided in:**
  - **Assembly Language (ASM)**
  - **C-Callable Assembly (CcASM)**
- **Filters types provided are FIR & IIR**
- **Coefficient tables are provided for several Lowpass, Highpass, Bandpass Filters**
- **Digital Filters are implemented on TI's 'C2000 DSP's using different techniques:**
  - **Linear Buffer Delay Line Approach**
  - **Circular Buffer Approach (Uses Bit-Reversed Addressing)**
- **Filter design scripts are provided for MATLAB Signal Processing Toolbox**

Types of Filters in TI's Digital Filter Software Package

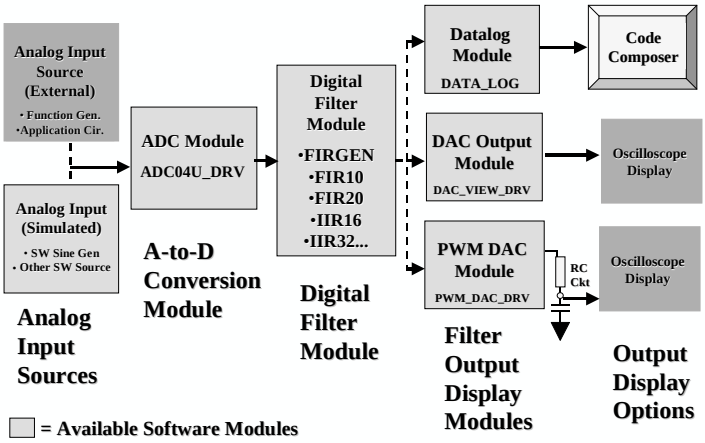
- **All Software provided follows TI's Modular Software Library Methodology**
- **The Following Types of Filters are Provided:**
  - FIRGEN - Generic FIR Filter w/ Linear Buffer (var. Order)
  - FIR10 – Fixed 10th Order FIR Filter using Linear Buffer
  - FIR20 – Fixed 20th Order FIR Filter using Linear Buffer
  - FIRCGEN- Generic FIR w/ Circular Buffer (var. Order)
  - FIR10C – Fixed 10th Order FIR Filter using Circular Buffer
  - FIR20C – Fixed 20th Order FIR Filter using Circular Buffer
  - IIR16 - 16-bit IIR Filter
  - IIR32 - 32-bit IIR Filter

TI's Complete Digital Filter Demo Systems

To Better Understand and Use the Digital Filter Software Package, TI provides complete Demo Systems for each filter type

- **Digital Filter Demo's Include:**
  - All necessary software (.asm, .c, .cmd, .out, .wks…)
  - Complete Documentation and Code Composer Workspace
  - Instructions for running demo on eZdsp™ or EVM hardware platforms
  - Several Data Input Options
    - Analog Signal from External Sources (function gen, other…)
    - Software Simulated Analog Signal
  - Several Filter Output Options
    - Code Composer Graph Windows
    - Oscilloscope - Measurement of PWM-based DAC
    - Oscilloscope - Measurement of DAC (EVM only)

Digital Filter System Block Diagram



Intro to Lab Exercise

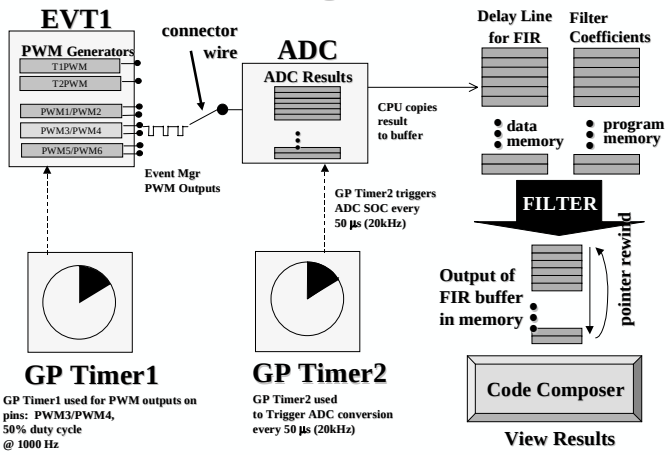
Running Lab6 - Digital Filter Demo

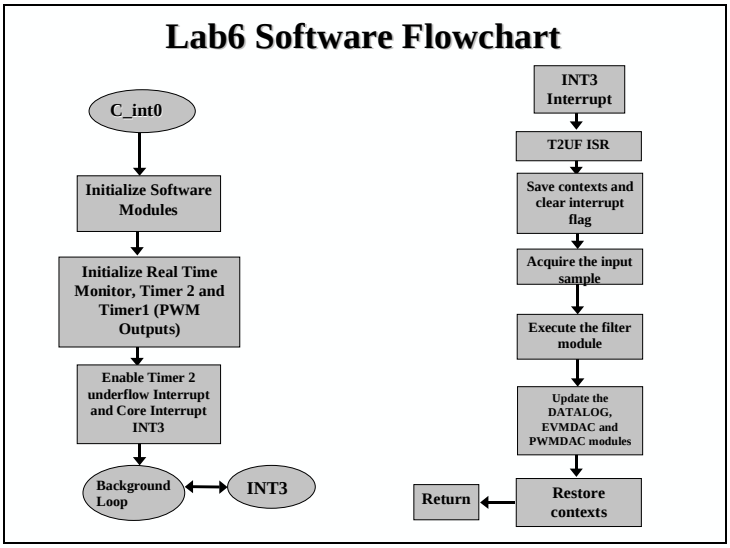
For Purposes of this Lab, we will review and demonstrate a <u>modified*</u> Filter Demo from TI's Digital Filter software package

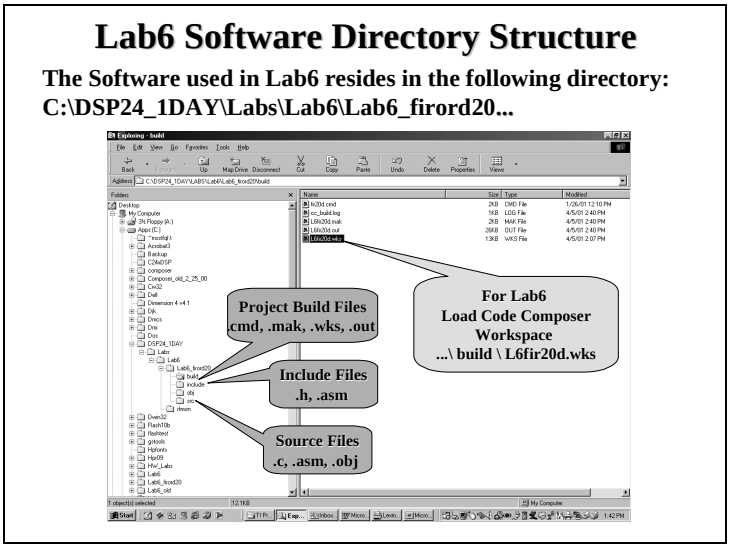
- **<u>Lab6 Basics:</u>**
  - Review components and features of TI's Digital Filter Demo package
  - Review LAB6 Components. Lab6 is based on a modified* copy of Filter Package Demo FIR20 (ASM version) - Fixed 20th Order FIR Filter.
  - Build and Load Lab6_FIRORD20 Filter Demo code using Code Composer and eZdsp™ LF2407
  - Run the Filter Demo using a DSP PWM output as the Analog Input Source (via jumper wire on eZdsp™ LF2407 board)
  - Vary Filter performance using different Coefficient tables
  - Display Filter Output using Code Composer Graphs

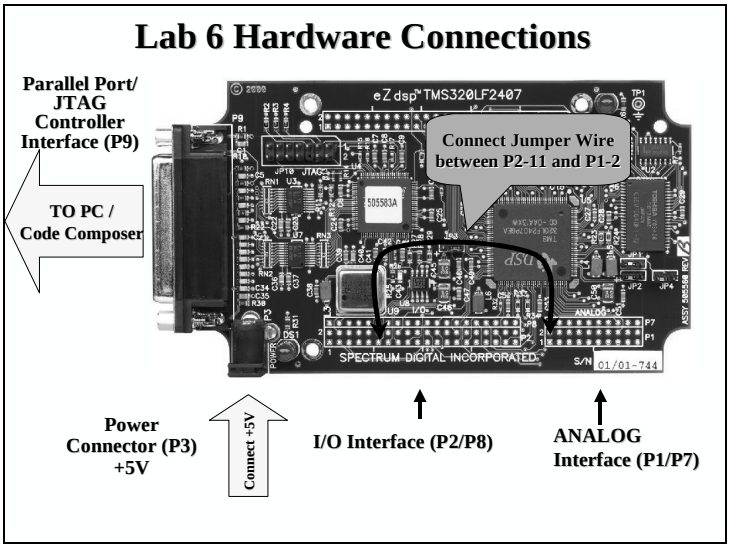
**** Modifications include Lab6 directory structure and addition of PWM output***

Module 6 Lab: Digital Filter Demo




Lab6 Software Flowchart
C_int0
Initialize Software Modules
Initialize Real Time Monitor, Timer 2 and Timer1 (PWM Outputs)
Enable Timer 2 underflow Interrupt and Core Interrupt INT3
Background Loop
INT3
INT3 Interrupt
T2UF ISR
Save contexts and clear interrupt flag
Acquire the input sample
Execute the filter module
Update the DATALOG, EVMDAC and PWMDAC modules
Restore contexts
Return


Lab6 Software Directory Structure
The Software used in Lab6 resides in the following directory:
C:\DSP24_1DAY\Labs\Lab6\Lab6_firord20...
Project Build Files
cmd, .mak, .wks, .out
Include Files
.h, .asm
Source Files
.c, .asm, .obj
For Lab6
Load Code Composer
Workspace
...\ build \ L6fir20d.wks


Lab 6 Hardware Connections
Parallel Port/ JTAG Controller Interface (P9)
TO PC / Code Composer
Connect Jumper Wire between P2-11 and P1-2
eZdsp™TMS320LF2407
SPECTRUM DIGITAL INCORPORATED
Power Connector (P3) +5V
Connect +5V
I/O Interface (P2/P8)
ANALOG Interface (P1/P7)

Module 6 Lab: Digital Filter Basics

Outline

1) Introduction – Overview of Digital Filter SW Library

2) Objective and Components (10 mins)

- Data Input/Output Options
- Software Library Modules Used
- Directory Structure
- Filter demo Block Diagram
- Software Flowchart
- Hardware Setup

3) Lab6 Procedure (20-30 min)

- Review Lab Components
- Connecting the Hardware
- Building and Loading the Filter Project
- Running and Viewing Filter Performance
  - LowPass Filter
  - HighPass Filter
  - BandPass Filter (OPTIONAL)
  - StopBand Filter (OPTIONAL)

APPENDIX 6A (Reference)

- Theory of FIR filter Used in this Lab

APPENDIX 6B (Reference)

- Overview of the Complete Digital Filter SW Library Package

APPENDIX 6C - Code Listings (Reference)

- L6fir20d.asm Main System File
- Fir20a.asm FIR Filter Algorithm
- Fircoeff.asm Filter Coefficients

1. Introduction

TI offers a Digital Filter Library Software package containing Modular Sourcecode, Documentation, and Demonstrations for several different types of Digital Filter Implementations (see Appendix for Details). This package is available Free-of-Charge from Texas Instruments (subject to TI's Free Software license agreement) and contains the flexibility and filter coefficient tables necessary to demonstrate Low-Pass, HighPass, BandPass and StopBand Filters with various cutoff frequencies. ***Lab6 will focus on understanding and using a modified version of the FIR20 package for Fixed 20th Order FIR Filters.***

TI's C2000 DSP Digital Filter Library Options:

1) FIRGEN - Generic FIR Filter using Linear Buffer (variable Order)
2) FIR10 – Fixed 10th Order FIR Filter using Linear Buffer
3) ***FIR20 – Fixed 20th Order FIR Filter using Linear Buffer*** ← LAB6 FOCUS
4) FIRCGEN - Generic FIR Filter using Circular Buffer (variable Order)
5) FIR10C – Fixed 10th Order FIR Filter using Circular Buffer
6) FIR20C – Fixed 20th Order FIR Filter using Circular Buffer
7) IIR16 - 16-bit IIR Filter
8) IIR32 - 32-bit IIR Filter

2. Objective and Components

Lab6 uses software modules from TI's Software Library to create a complete system to demonstrate digital filtering on the eZdsp™ LF2407 hardware platform. This lab uses a modified version of the **FIRORD20 Digital Filter Demo**, which is one of several Filter Demo's available in TI's Digital Filter Library. ***Only a portion of the Digital Filter Package's capabilities are explored in this lab.*** Refer to the Appendix for details of the Digital Filter Library package.

The idea behind Lab6 is simple. A jumper wire will be used to connect the PWM signal to one of the ADC inputs on the eZdsp™ LF2407 board. This PWM signal will be sampled at 20KHz through one of the DSP's ADC inputs, and the sampled digital representation will be passed through the Filter module to perform Digital Filtering. Both the input data sequence and filtered output are logged and displayed in Code Composer Graph Windows. This is done to minimize the required hardware for this lab (no signal generator or Oscilloscope are needed). Note that the Filter Demo has several possible Data Input and Output Display options in addition to what is explored in this lab (see below)

Digital Filter Demo - Data Input Options:

1) Analog signal from Signal generator
2) **Analog Signal from EVM or eZdsp™ board (through jumpers)** ← Method Used in Lab
3) Software Simulated Analog Signal

Digital Filter Demo - Output Visualization options:

1) **Code Composer Graph Windows** ← Method Used in Lab
2) Oscilloscope – Measurement of PWM-based DAC
3) Oscilloscope – Measurement of hardware DAC signals (2407EVM only)

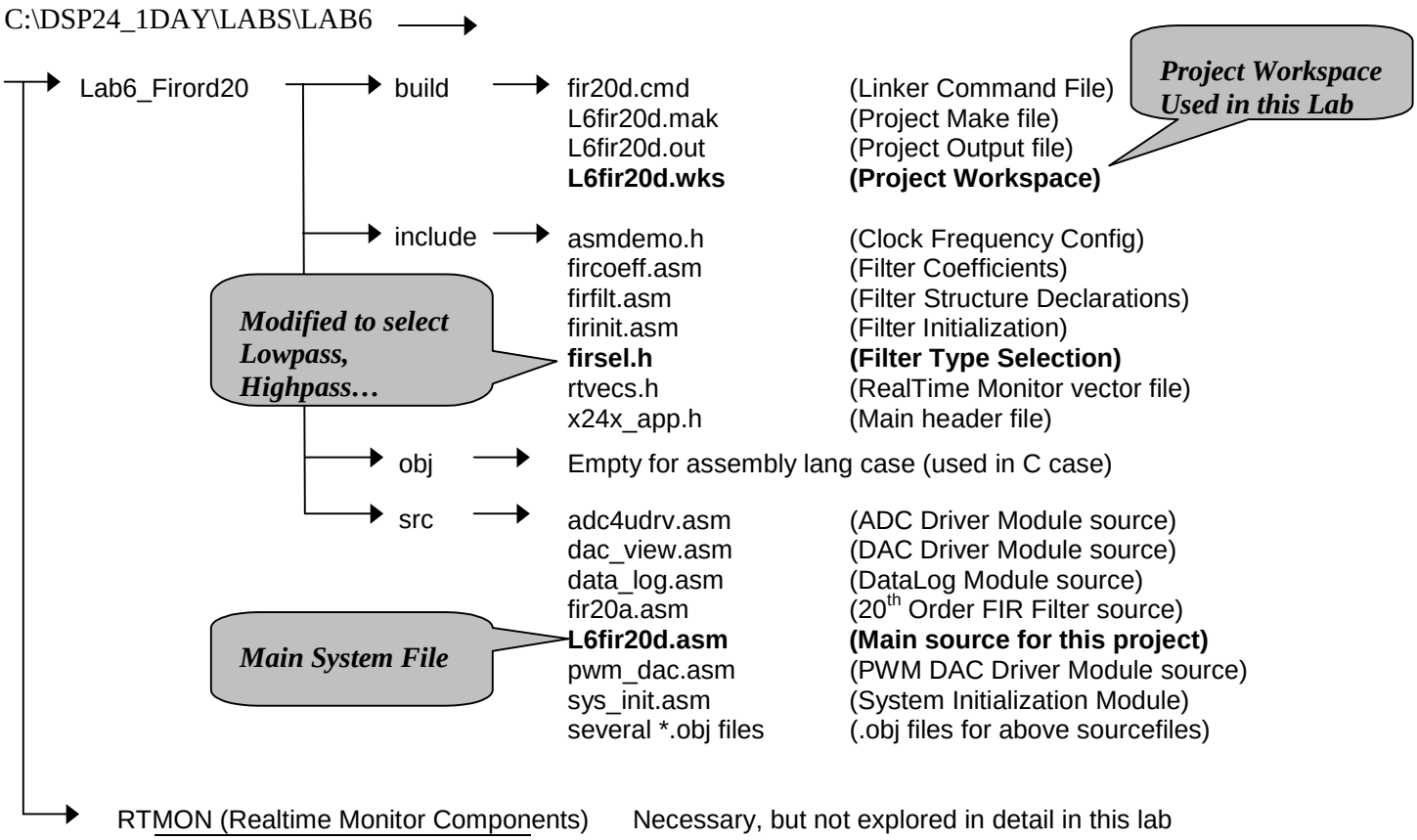
Software Library Modules Used:

The following Assembly Language modules are used in this lab. Most Modules are available in both ASM and CcASM. Refer to Modular Library Documentation for complete details.

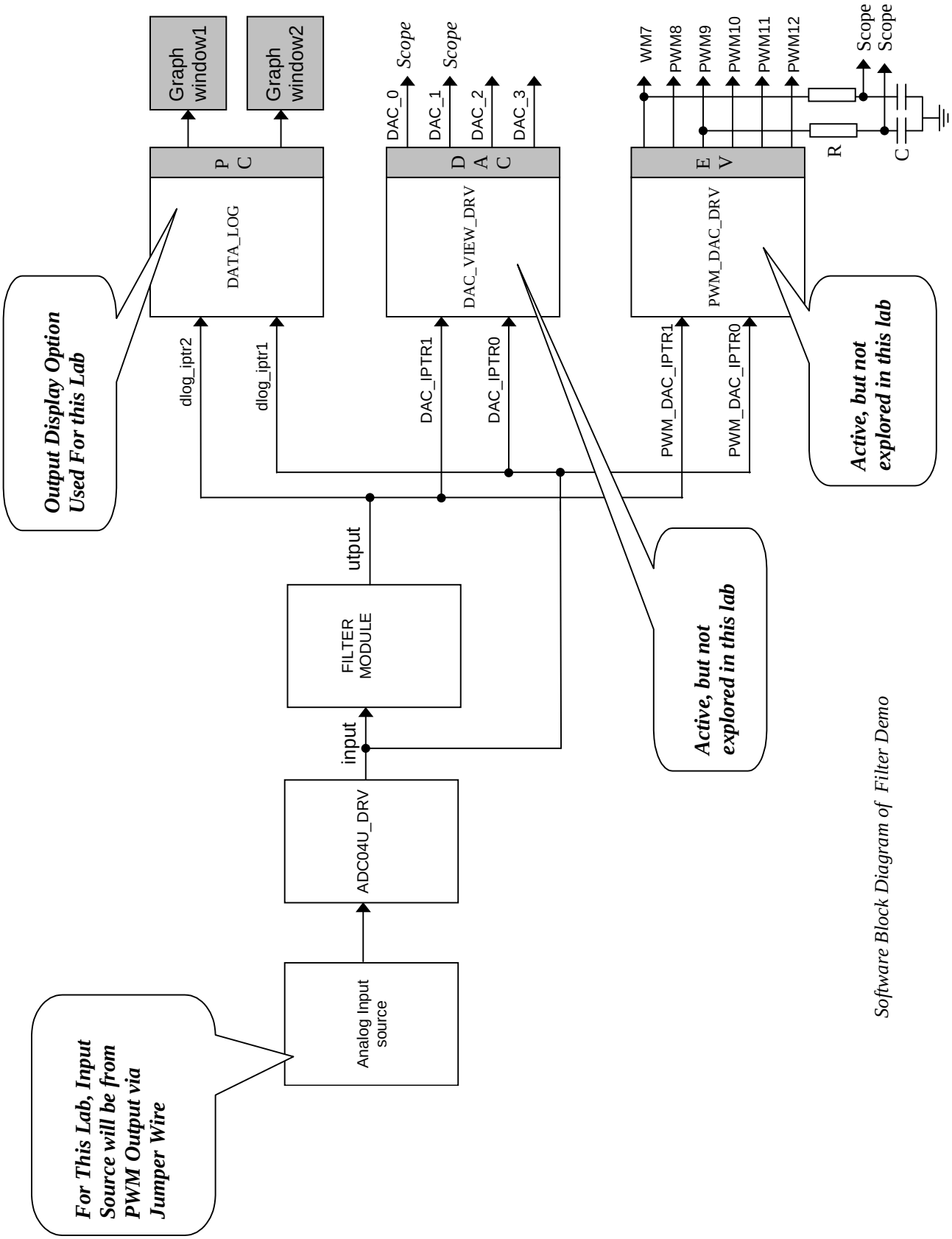
1. **ADC04U_DRV** - Converts Analog Inputs to Digital Format
2. **PWM_DAC_DRV** (PWMDAC) Generates a modulated PWM signal on DSP output
3. **DAC_VIEW_DRV** (EVMDAC) Sends Digital Signals to External Hardware DAC (HW DAC is only present on LF2407EVM, not available on eZdsp™ LF2407)
4. **DATA_LOG** - Displays signals in Code Composer Graph Window
5. **SYS_INIT** • System Initialization
6. **FIRORD20** - Fixed 20th Order FIR Filter using Linear Buffer

Lab6 Directory Structure and Files Used:

The software modules used in lab6 consist of several assembly language sourcefiles, which are described in the following directory snapshot. Keep in mind that **all** of the modules/files shown below are active in this project, **but some are not reviewed in this lab** (DAC Driver, PWM DAC driver…). The main files examined/modified in this lab are highlighted. Only the files **L6fir20d.asm** and **firsel.h** are actually modified in this lab. All other sourcefiles are copies of the original software library sourcecode. Refer to the SW Library Documentation for complete details on using any of the modules.

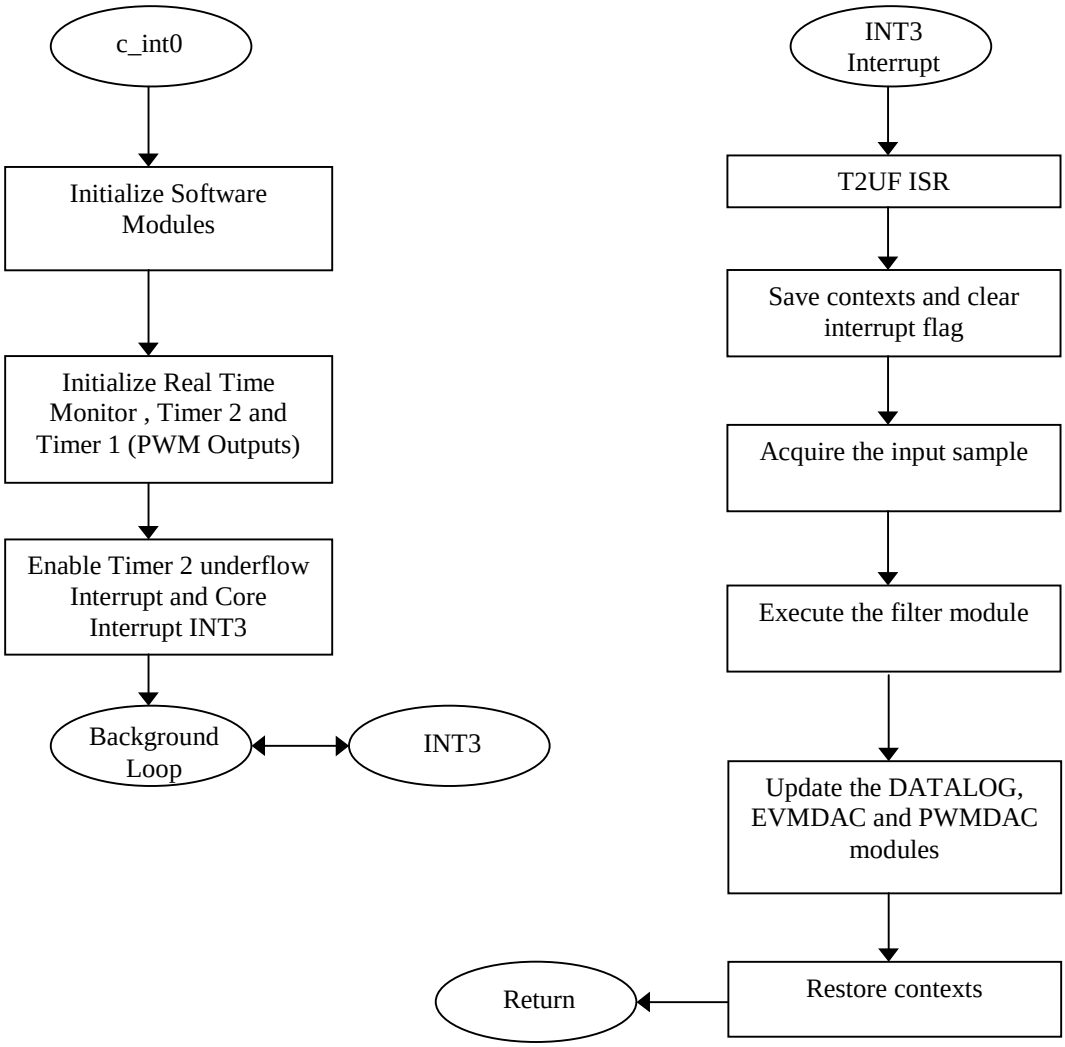


Lab6 Block Diagram



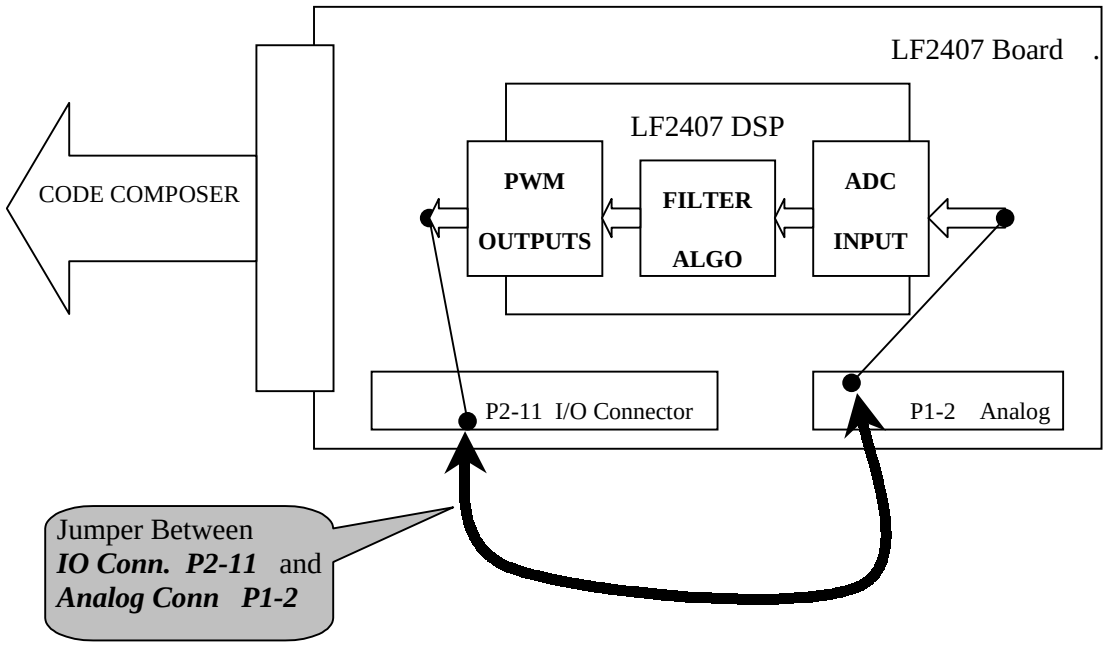
Software Block Diagram of Filter Demo

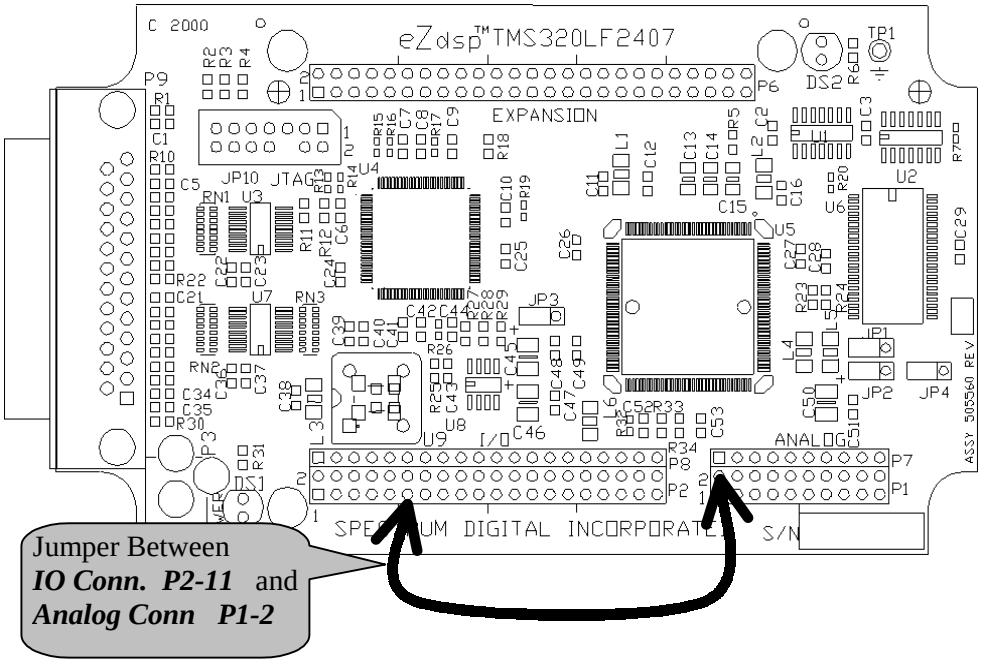
Software Flowchart



Lab6 - Hardware Setup:

The hardware used in this lab consists of the eZdsp™ LF2407 board and Jumpers used to connect the PWM outputs to the ADC Input. The Block Diagram and Board Layout are as follows:





3. Procedure

1) Spend a few minutes reviewing Sections 1 & 2 of this document to understand the Objective and Components of Lab6:
   - Software Modules Used
   - Directory Structure and Filenames
   - Hardware Setup
   - System Block Diagram
   - Software Flowchart

2) Connect and Power-up Hardware. Connect Jumper Wire between **I/O Connector P2-11 and Analog Connector P1-2** *(refer to Hardware Setup Diagram)*

3) Launch Code Composer and Load the Workspace: ***File =>Workspace=>LoadWorkspace***=> **C:\DSP24_1DAY\LABS\LAB6\Lab6_firord20\build\ L6fir20d.wks**

4) **Refer to the Screen Capture Below (on next page) for Diagram of the next several steps.** In the Project Window, expand the Project Directories *(click on the +'s)* and spend a moment reviewing the names and locations of the sourcefiles (*.asm), header files (*.h), and command files (*.cmd) used in this project.

SELECT LOWPASS FILTER OPTION

5) Locate the main system file **L6fir20d.asm.** *(should already be open in the Code Composer Workspace, if not then it can be found in Source directory*). Spend a few minutes examining the sourcecode to understand the general structure/flow of the software. Locate the section called **Filter Initialization** *(Hint…one convenient way to do this is to type Filter Init in the Code Composer Search Window and hit <Return>).* Under the Filter Initialization section, **UNCOMMENT** *(remove the semicolon ;)* the **Select LowPass** option. Comment out the other selections.

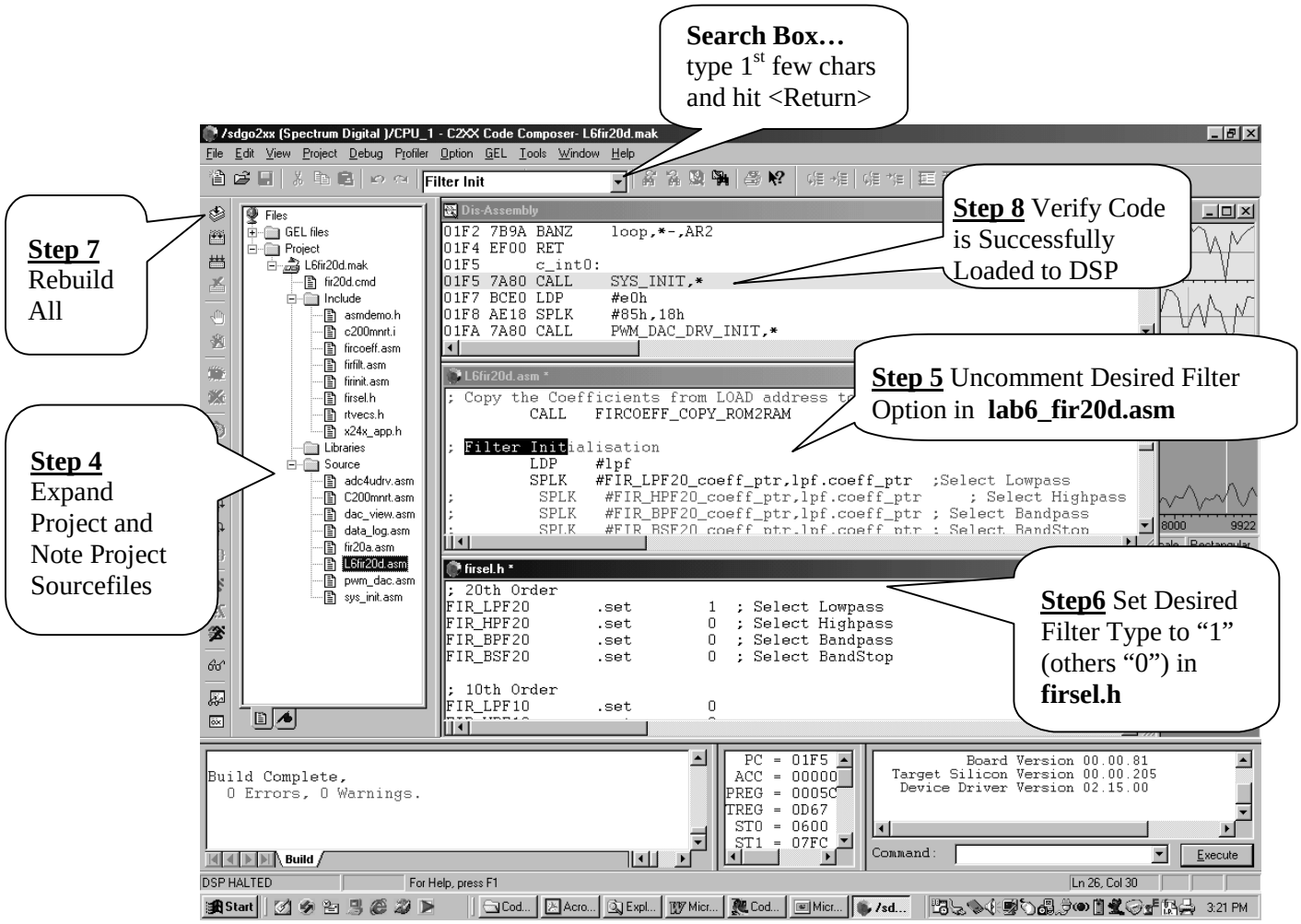
```
; Filter Initialization
        LDP   #lpf
➔ SPLK   #FIR_LPF20_coeff_ptr,lpf.coeff_ptr ; Select Lowpass
        ; SPLK   #FIR_HPF20_coeff_ptr,lpf.coeff_ptr  ; Select Highpass
        ; SPLK   #FIR_BPF20_coeff_ptr,lpf.coeff_ptr ; Select Bandpass
        ; SPLK   #FIR_BSF20_coeff_ptr,lpf.coeff_ptr ; Select BandStop
```

6) Locate the file **firsel.h** *(should already be open in the Code Composer Workspace, if not then it can be found in INCLUDE directory*). Enable the 20th Order **Lowpass** option by putting a "1" in the 20th Order **Lowpass** selection. Make sure all other options are "0":

```
;    20th Order
➔ FIR_LPF20        .set        1  ; Select Lowpass
    FIR_HPF20      .set        0  ; Select Highpass
    FIR_BPF20      .set        0  ; Select Bandpass
    FIR_BSF20      .set        0  ; Select BandStop
```

7) From the Toolbar, select Option => Program Load and make certain the "Load Program After Build" box is checked. Select Rebuild All to assemble and link the Project *(use Toolbar RebuildAll button, or Pulldown menu Project => RebuildAll).*

8) Ensure that **L6fir20d.out** is successfully loaded to DSP target. You should see a Yellow Bar highlighting the line CALL SYS_INIT in the Dis-Assembly window *(if this is not the case, then use Toolbar to select File=> Load Program to load file L6fir20d.out)*



Enable Code Composer Realtime Mode

9) Invoke **REALTIME MODE**: using the following Steps

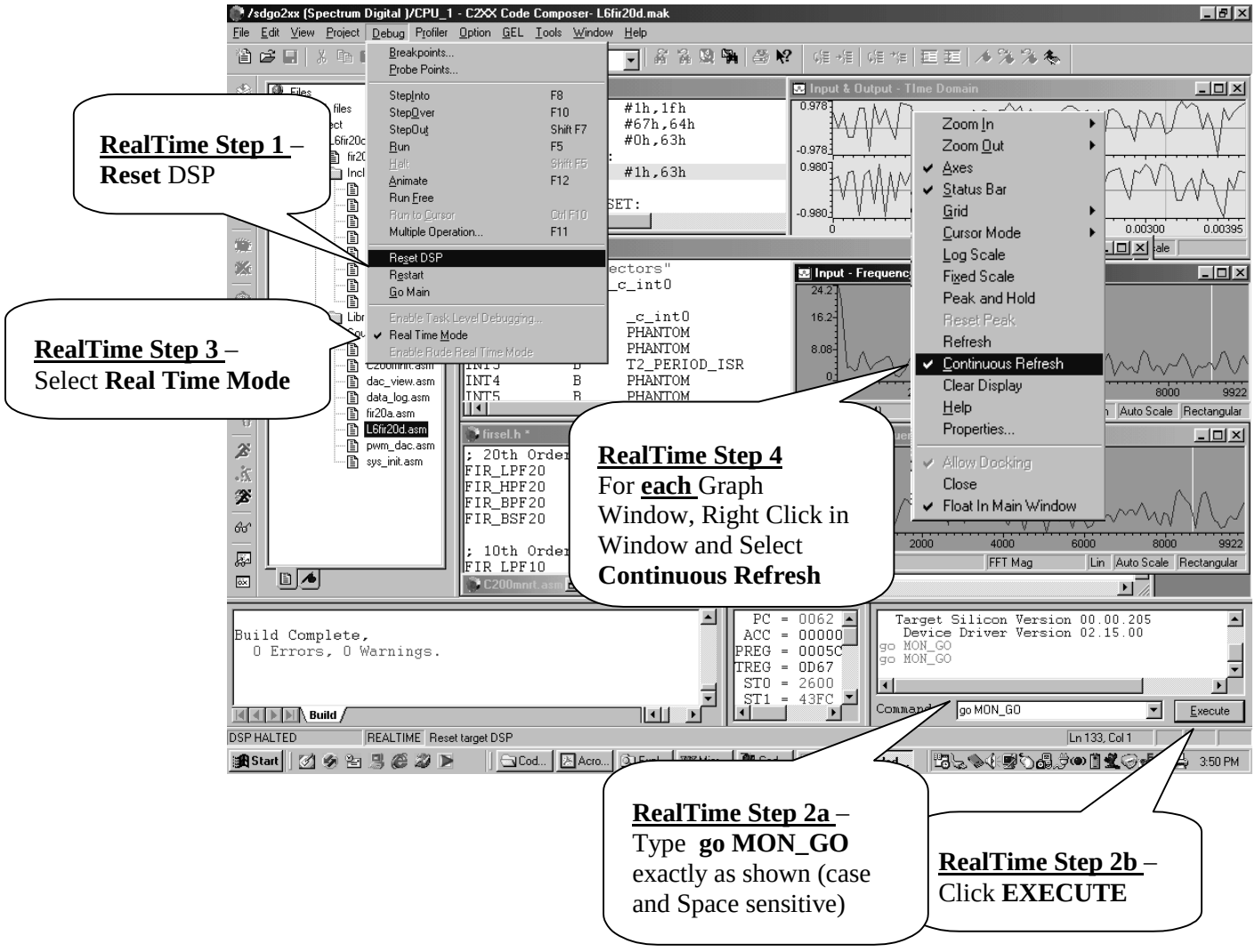
RealTime Step 1) Select Menu Item DEBUG => **RESET DSP**

RealTime Step 2a) In Command Window type **go MON_GO** ***(type this exactly as is)***

Step 2b) Click **EXECUTE** *(you may see the file C200mnrt pop up… just minimize this file as it will not be reviewed in this lab)*

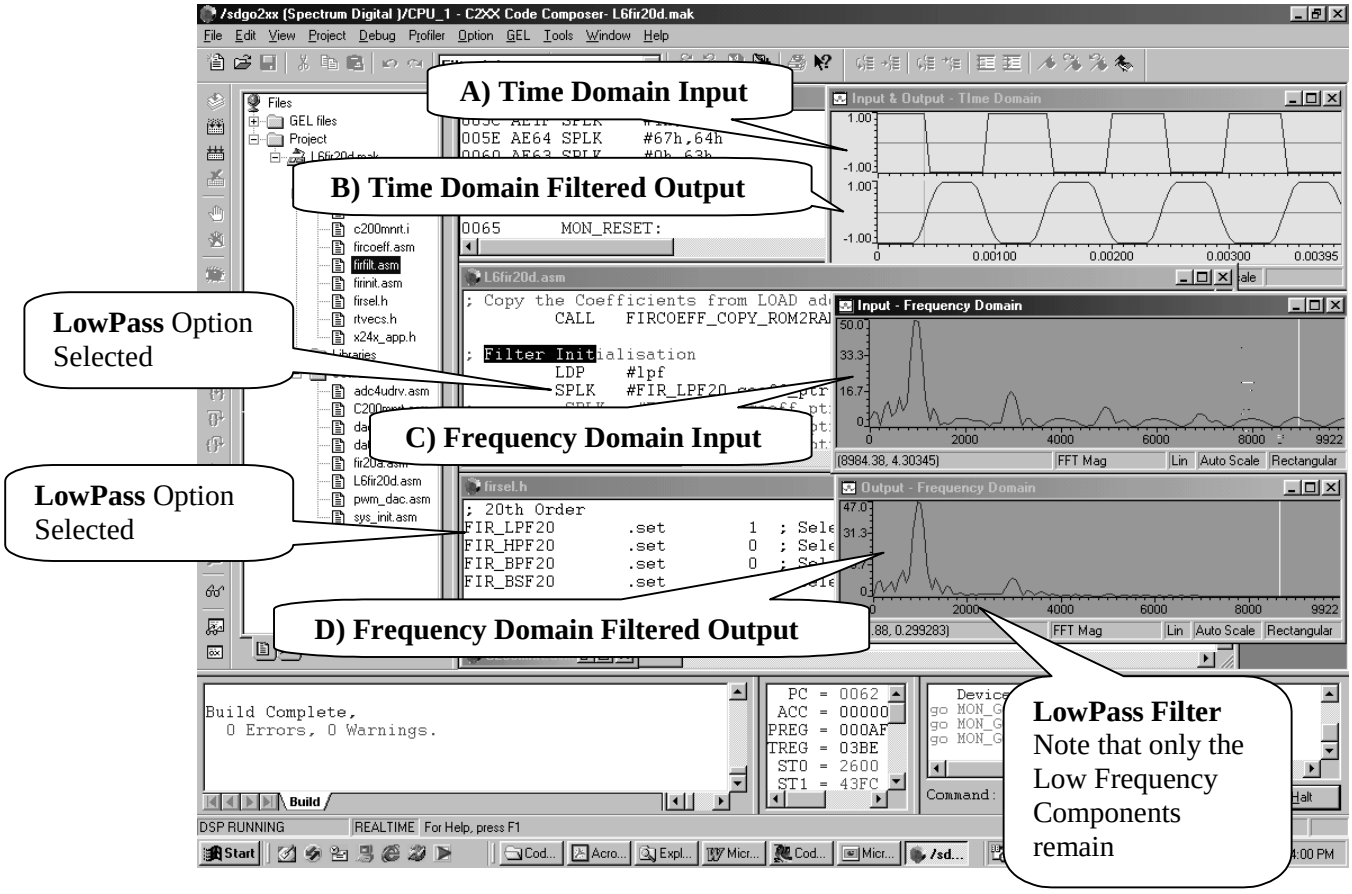
RealTime Step 3) Select Menu Item DEBUG => REALTIME MODE

RealTime Step 4) For each Graph Window, Right Click in the Graph Window and select =>Continuous Refresh *(note: once this is done once for each graph, it does not need to be repeated unless Continuous Refresh is somehow turned off)*



View the LowPass Filter Output

10) Select RUN *(from Icon or Debug=>RUN)* and verify the following Filter Display



11) Verify that display is as expected:

A) Time Domain Input = 1000 Hz Square Wave from DSP PWM Output *(you can use x-axis in TimeDomain Graph to verify the signal period 0.00100msec = 1000Hz)*

B) Time Domain Filtered Output = Filtered Result for **LowPass** Filter with **3000Hz** Corner Frequency. Frequency components >**~3000Hz** have been filtered out.

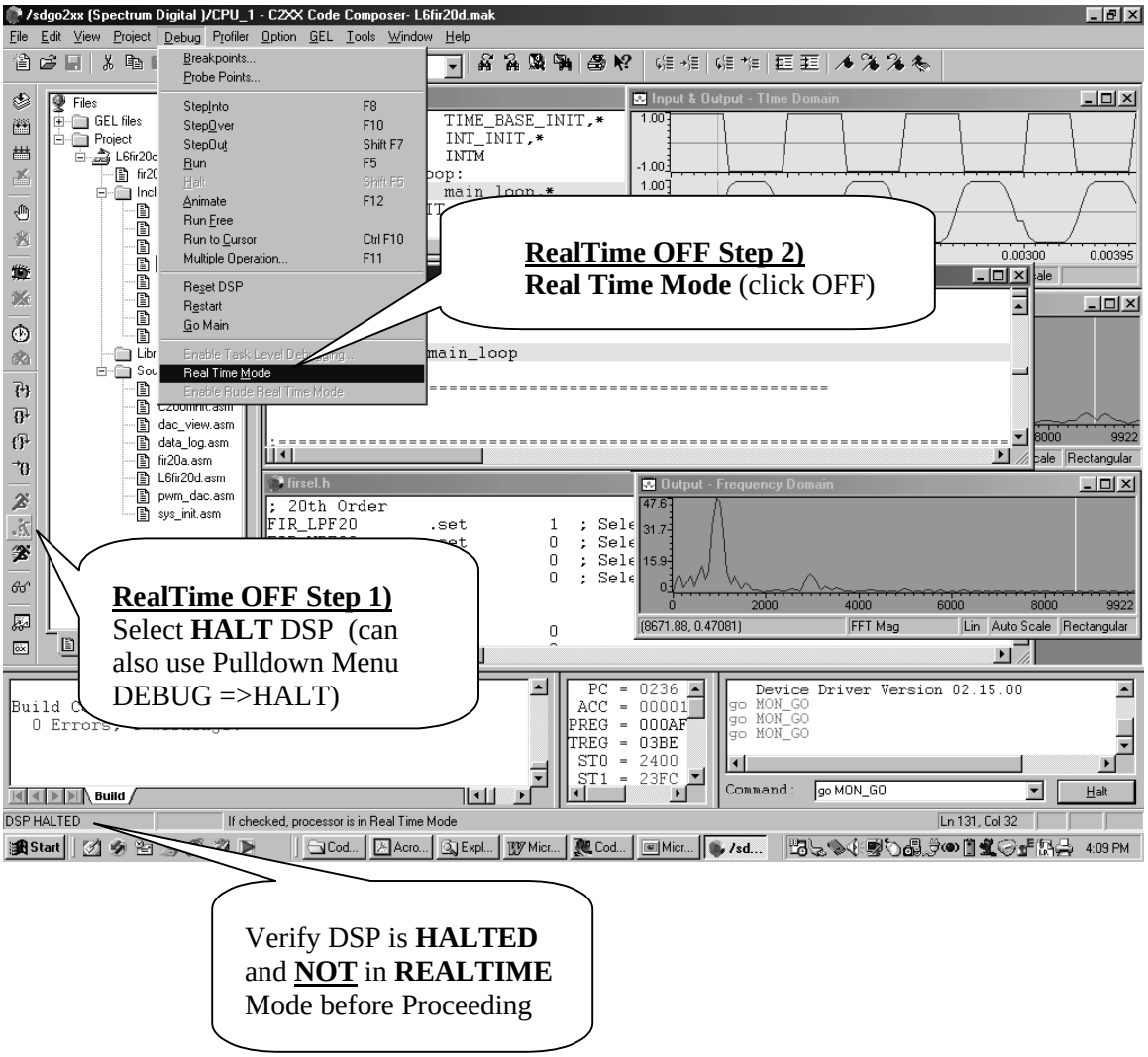
C) Frequency Domain Input = 1000Hz Square Wave (PWM Output)

D) Frequency Domain Filtered Output = Filtered Result for **LowPass** Filter with **3000Hz** Corner Frequency. Frequency components >**~3000Hz** have been filtered out.

!!!!!!! IMPORTANT !!!!!!

12) Turn Realtime Mode OFF before proceeding ***(IMPORTANT – Failure to turn RealTime Mode OFF before making Code changes and loading new Code can result in Unpredictable Operation):***

- RealTime OFF Step 1) Select HALT *(Use Halt Icon, or DEBUG =>HALT)*
- RealTime OFF Step 2) Select DEBUG=>REALTIME MODE (click OFF)
- Verify DSP is **HALTED** and **NOT** in **REALTIME** Mode before Proceeding



SELECT HIGHPASS FILTER OPTION

13) Ensure **DSP is HALTED** and **NOT** in REALTIME Mode *(bottom left corner of screen)*

14) Locate the main system file **L6fir20d.asm.** *(should already be open in the Code Composer Workspace, if not then it can be found in SRC directory*). Locate the section called **Filter Initialization** *(Hint…one convenient way to do this is to type Filter Init in the Code Composer Search Window and hit <Return>)* Under the Filter Initialization section, **UNCOMMENT** *(remove the semicolon ;)* the **Select HighPass** option. Comment out the other selections.

```
; Filter Initialization
      LDP   #lpf
     ;SPLK   #FIR_LPF20_coeff_ptr,lpf.coeff_ptr  ; Select Lowpass
➔   SPLK   #FIR_HPF20_coeff_ptr,lpf.coeff_ptr  ; Select Highpass
      ; SPLK   #FIR_BPF20_coeff_ptr,lpf.coeff_ptr ; Select Bandpass
      ; SPLK   #FIR_BSF20_coeff_ptr,lpf.coeff_ptr ; Select BandStop
```

15) Locate the file **firsel.h** *(should already be open in the Code Composer Workspace, if not then it can be found in INCLUDE directory*). Enable the 20th Order **Highpass** option by putting a "1" in the 20th Order **Highpass** selection. Make sure all other options are "0":

```
;     20th Order
      FIR_LPF20          .set          0  ; Select Lowpass
➔   FIR_HPF20          .set          1  ; Select Highpass
      FIR_BPF20          .set          0  ; Select Bandpass
      FIR_BSF20          .set          0  ; Select BandStop
```

16) Select **ReBuild All** and Ensure that **L6fir20d.out** is successfully loaded to DSP target *(refer to Steps 7 & 8 if you need details on accomplishing this).*

17) Invoke **REALTIME MODE** using the following steps *(refer to earlier diagram as needed)*:

RealTime Step 1) Select Menu Item DEBUG => **RESET DSP**

RealTime Step 2a) In Command Window type **go MON_GO** ***(type this exactly as is)***

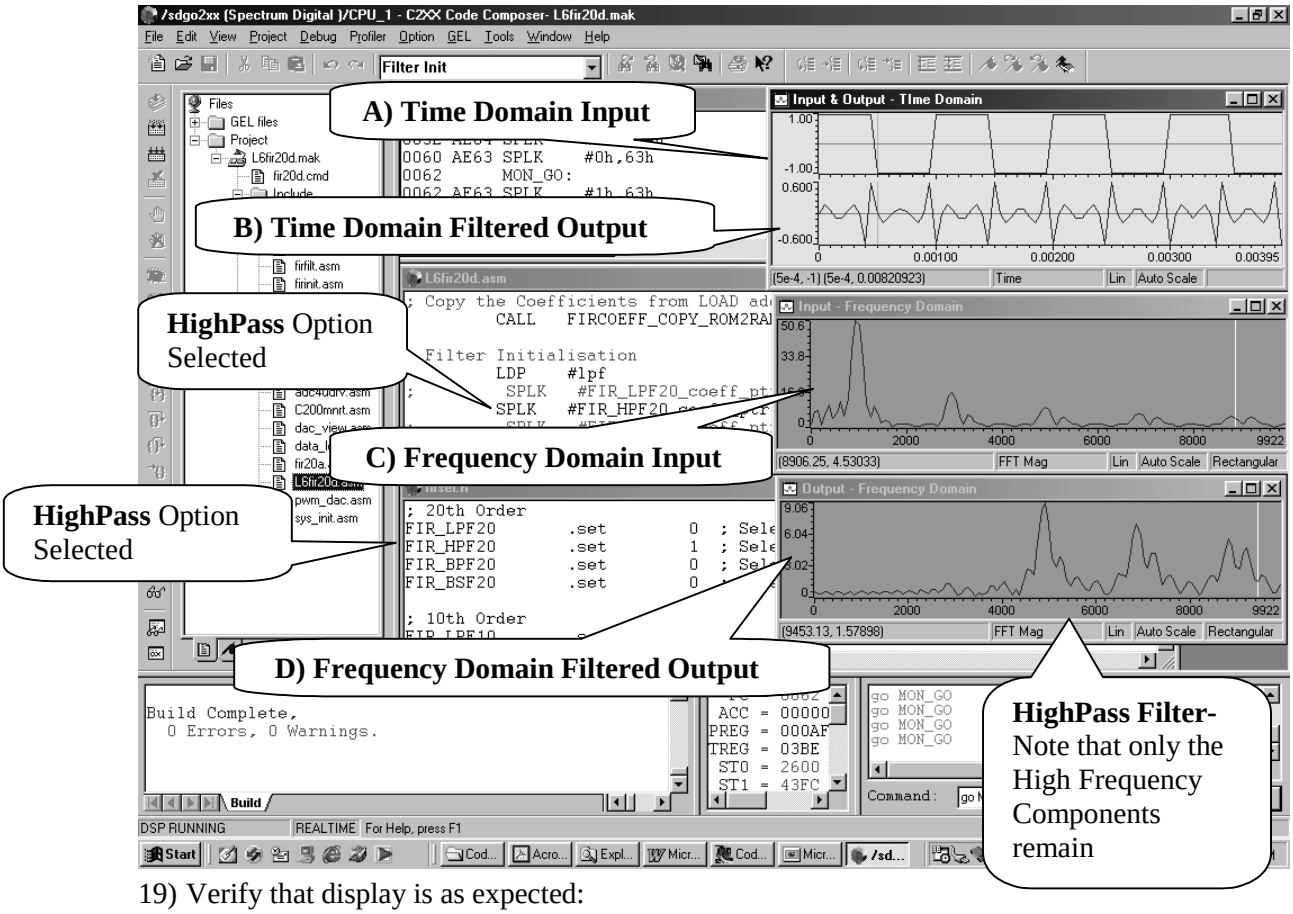
Step 2b) Click **EXECUTE** *(you may see the file C200mnrt pop up… just minimize this file as it will not be reviewed in this lab)*

RealTime Step 3) Select Menu Item DEBUG => REALTIME MODE

RealTime Step 4) For each Graph Window, Right Click in the Graph Window and select =>Continuous Refresh *(note: once this is done once for each graph, it does not need to be repeated unless Continuous Refresh is somehow turned off)*

View the HighPass Filter Output

18) Select RUN *(from Icon or Debug=>RUN)* and verify the following Filter Display



19) Verify that display is as expected:

A) Time Domain Input = from DSP PWM Output *(you can use x-axis in TimeDomain Graph to verify the signal period 0.00100msec = 1000Hz)*

B) Time Domain Filtered Output = Filtered Result for **HighPass** Filter with **4000Hz** Corner Frequency. Frequency components **less than ~4000Hz** have been filtered out.

C) Frequency Domain Input = 1000Hz Square Wave (PWM Output)

D) Frequency Domain Filtered Output = Filtered Result for **HighPass** Filter with **4000Hz** Corner Frequency. Frequency components **less than ~4000Hz** have been filtered out.

!!!!!!! IMPORTANT !!!!!!

20) Turn Realtime Mode OFF before proceeding ***(IMPORTANT – Failure to turn RealTime Mode OFF before making Code changes and loading new Code can result in Unpredictable Operation):***

- RealTime OFF Step 1) Select HALT DSP *(Use Halt Icon, or DEBUG =>HALT)*
- RealTime OFF Step 2) Select DEBUG=>REALTIME MODE (click OFF)
- Verify DSP is **HALTED** and **NOT** in **REALTIME** Mode before Proceeding

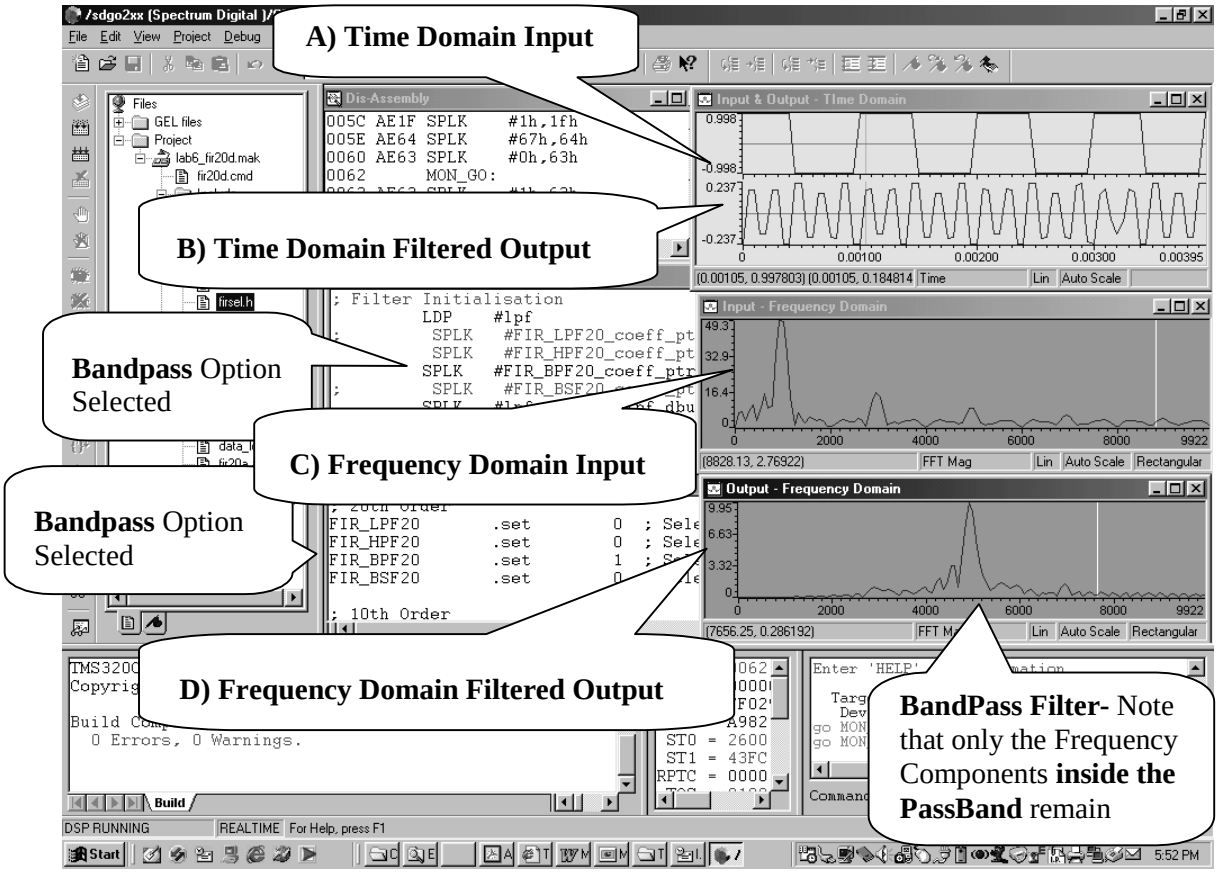
OPTIONAL –

The following Section is Optional and uses similar methods to explore other Filter Options.

SELECT BANDPASS FILTER OPTION

21) Follow RealTime Off steps outlined earlier (step 12) and ensure **DSP is HALTED** and **NOT** in REALTIME Mode *(bottom left corner of screen)*

22) Following the steps outlined in previous sections *(refer to steps 5-12 if needed)*, make the necessary code adjustments to **L6fir20d.asm** and **firsel.h** to SELECT and RUN **BANDPASS** filter option.



23) Verify that display is as expected:

A) Time Domain Input = 1000 Hz Square Wave (PWM Output)\

B) Time Domain Filtered Output = Filtered Result for **BandPass** Filter with **4000Hz and 6000Hz** Corner Frequencies. Frequency components **outside of~4000Hz to ~6000Hz** Band have been filtered out.

C) Frequency Domain Input = 1000Hz Square Wave (PWM Output)

D) Frequency Domain Filtered Output = Filtered Result for **BandPass** Filter with **4000Hz and 6000Hz** Corner Frequencies. Frequency components **outside of~4000Hz to ~6000Hz** Band have been filtered out.

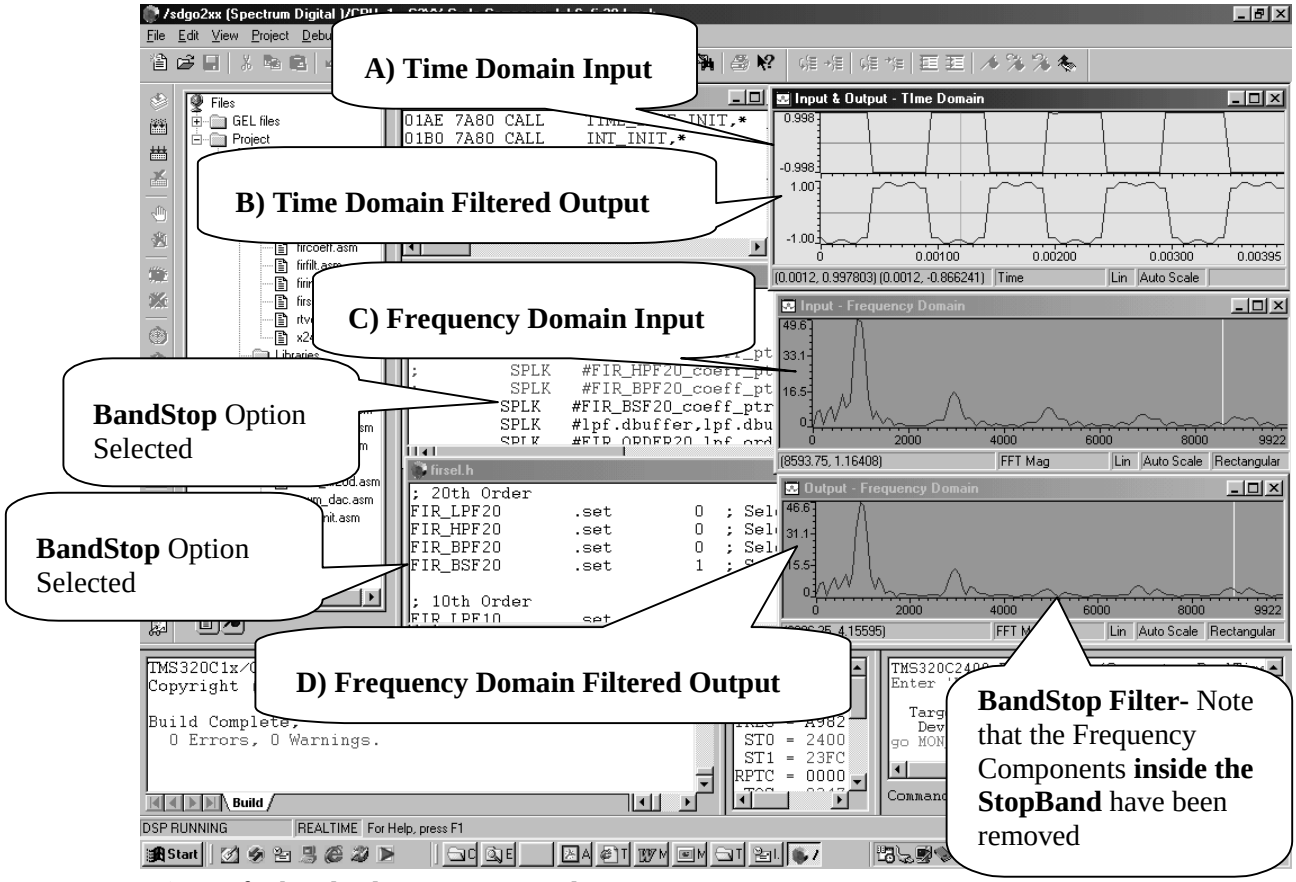
OPTIONAL –

The following Section is Optional and uses similar methods to explore other Filter Options.

SELECT BANDSTOP FILTER OPTION

24) Follow RealTime Off steps outlined earlier (step 12) and ensure **DSP is HALTED** and **NOT** in REALTIME Mode *(bottom left corner of screen)*

25) Following the steps outlined in previous sections *(refer to steps 5-12 if needed)*, make the necessary code adjustments to **L6fir20d.asm** and **firsel.h** to SELECT and RUN **BANDSTOP** filter option.



26) Verify that display is as expected:

A) Time Domain Input = 1000 Hz Square Wave (PWM Output)\

B) Time Domain Filtered Output = Filtered Result for **BandStop** Filter with **4000Hz and 6000Hz** Corner Frequencies. Frequency components **inside of~4000Hz to ~6000Hz** have been filtered out.

C) Frequency Domain Input = 1000Hz Square Wave (PWM Output)

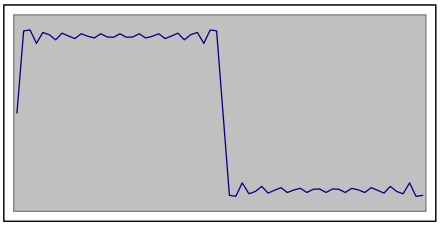
D) Frequency Domain Filtered Output = Filtered Result for **BandStop** Filter with **4000Hz and 6000Hz** Corner Frequencies. Frequency components **inside of~4000Hz to ~6000Hz** have been filtered out.

Appendix 6A

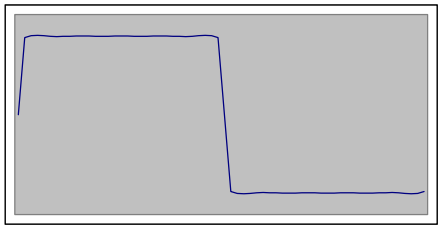
Filter Theory – Used in Lab6 and FIRORD20

Background Theory -- Fourier Analysis:

- Any signal can be constructed by summing sinusoids of appropriate frequency and amplitude.
- Any signal can be decomposed into a sum of sinusoids



First 10 Harmonics



First 100 Harmonics

Waveforms created by summing the first X odd harmonics of a sinusoid

Conversely, it is possible to remove the upper harmonic components of a square-wave signal using a lowpass filter. For example, if a lowpass filter with a cutoff frequency between the 1st and 3rd harmonic of the square-wave signal is used, the remaining signal will contain only the fundamental and the 1st harmonic. If the cutoff frequency of the filter is between the 3rd and 5th harmonic, the output from the filter should contain the fundamental, 1st and 3rd harmonic of the original signal.

Background Information for *FIR20 – Fixed 20th Order FIR Filter using Linear Buffer*

A 20th Order FIR filter is described by the difference equation

$$y(n) = \sum_{k=0}^{20} h_k \times x(n-k)$$

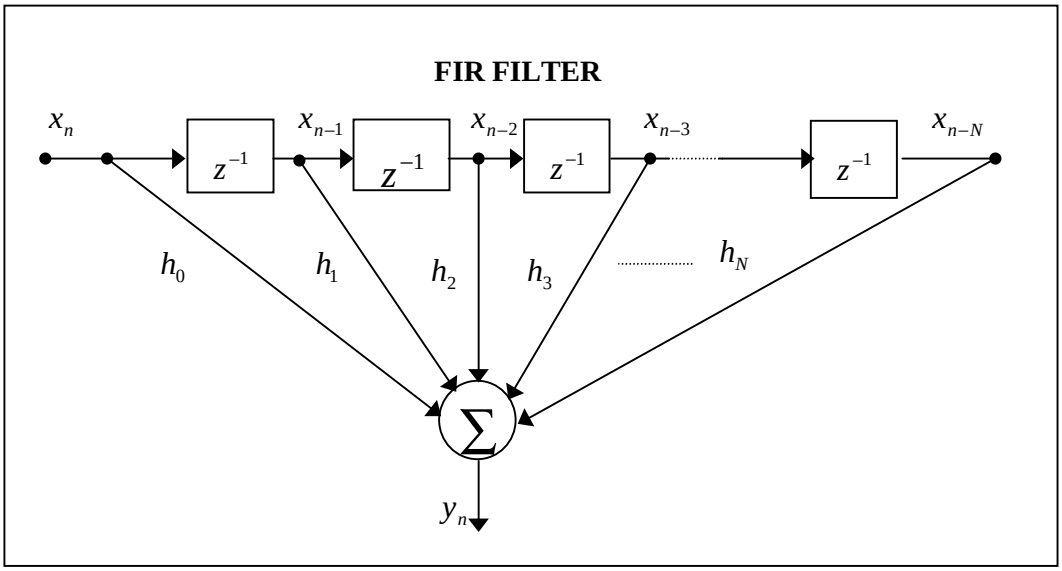
or, equivalently, by the system function

$$H(z) = \sum_{k=0}^{20} h_k \times z^{-k}$$

Furthermore, the unit sample response of the FIR system is identical to the coefficients $h(k)$, that is

$$h(n) = \begin{cases} h_n, & 0 < n < 20 \\ 0, & otherwise \end{cases}$$

The signal flow diagram of the FIR filter, representing the above mentioned difference equation is given below. The following Tapped delay implementation is canonical, which means that the number of storage element in the structure is equal to the order of the Filter.

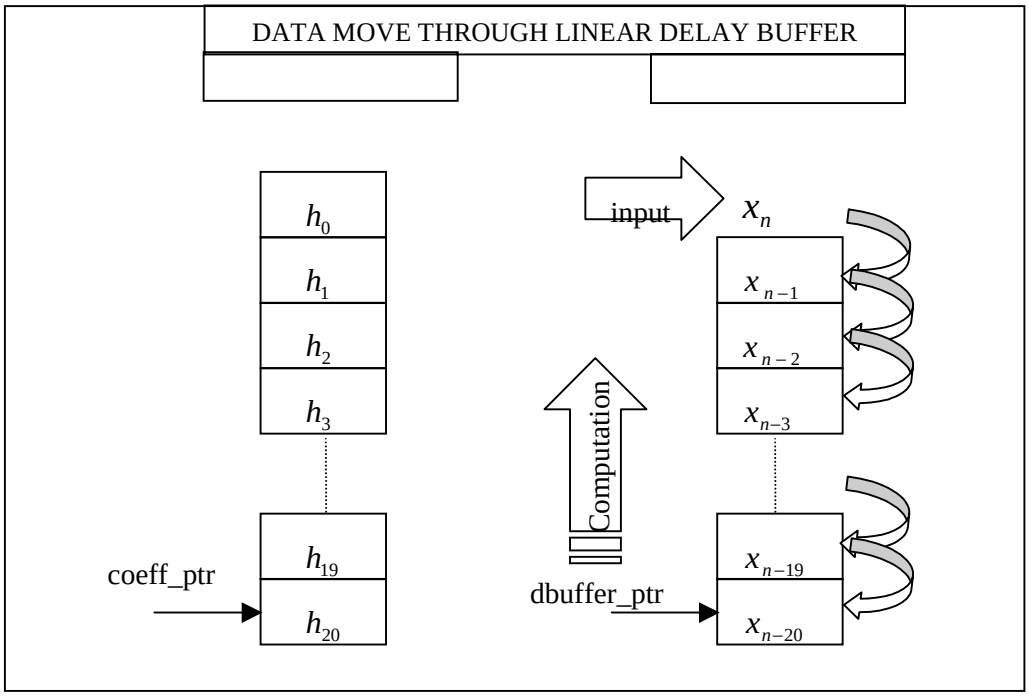


The FIR filter is essentially a sum of product operating on an array of values maintained in a delay line. The delay line for this routine is implemented as linear buffer, which requires manual data move through the delay line. The new data is written to the recently vacated spot at the top of the buffer. Although C24x/C240x DSP can perform this data movement in parallel with arithmetic processing, use of this feature is limited to the on-chip memory. Due to this restriction, it is absolutely necessary to place the delay buffer of the filter for this routine in internal DARAM B0, B1 & B2 memory blocks

The following diagram depicts the filter tap computation, the computation proceeds from the oldest values to the newest. The coeff_ptr and dbuffer_ptr are manipulated to point to the end of the buffer in the initialization routine though the beginning of the respective buffers are assigned to the corresponding pointers before calling the initialization routine.

Computing from the newest sample to oldest with DMOV would not implement a delay line, but instead would copy the newest value through the entire delay line.

Another important thing that the user should understand is the fact that the length of the filter will be always greater then the order of the filter by 1. That is why the size of the coefficient array is greater then the order of filter by 1.



Notes:

- The FIRFILT_ORD20 routine provides efficient execution time, but the user can no longer be able to modify the order of the filter
- Filter taps computation routine is unrolled, which in turn increases the program memory space
- Delay buffer array should be located in DARAM Memory Blocks
- Intermediate and final result of sum-of-product computation is saturated, and the filter output is represented in Q15 format. Hence, if the filter gain is over '1' in the pass band, then the user needs to restrict the input amplitude so that the output will not be saturated. Otherwise, the filter will saturate the output to maximum +ve/-ve values.

Appendix 6B

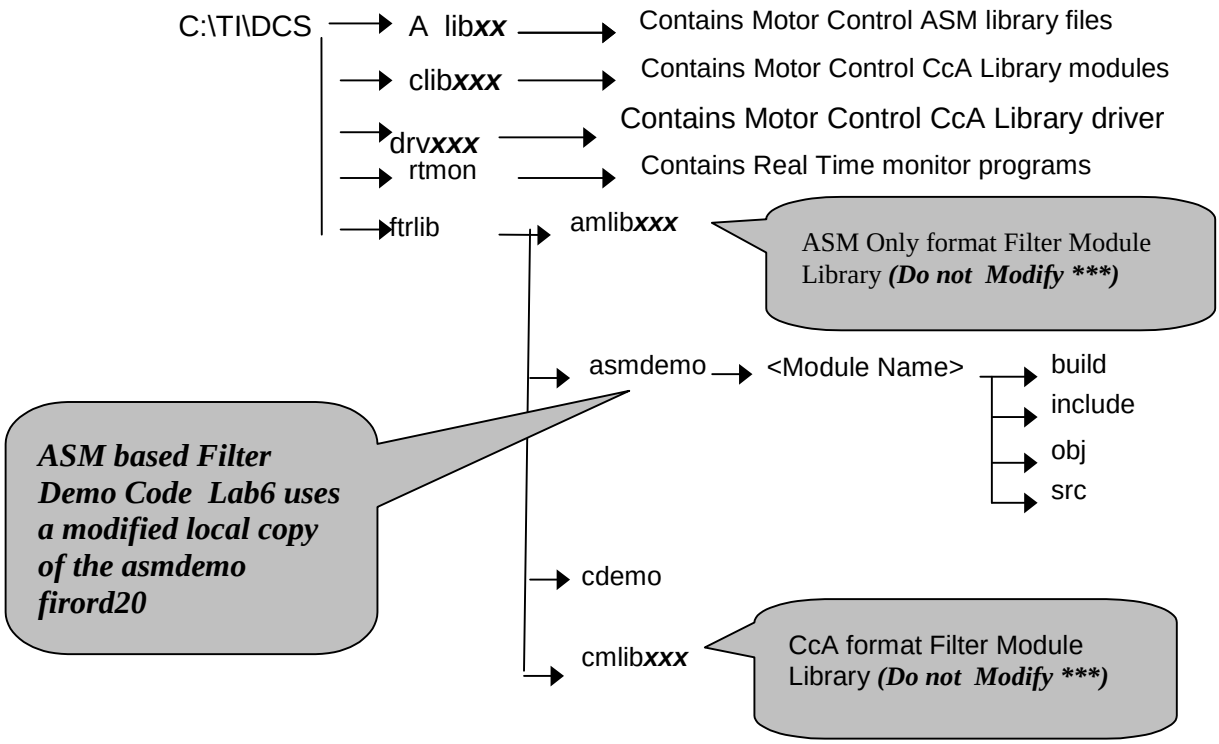
Overview of Complete Digital Filter SW Library

The Digital Filter Library is one of several Modular Software Libraries avaivable from TI for the C2000 Family of DSP's (Motor Control Library, Math Library, Filter Library…). TI provides these Libraries Free-of-Charge (subject to TI's Free Software license agreement). Refer to the files SYSTEM.DOC and USER_DOC.DOC included with the Filter Library for complete documentation.

General Software Library Directory Structure

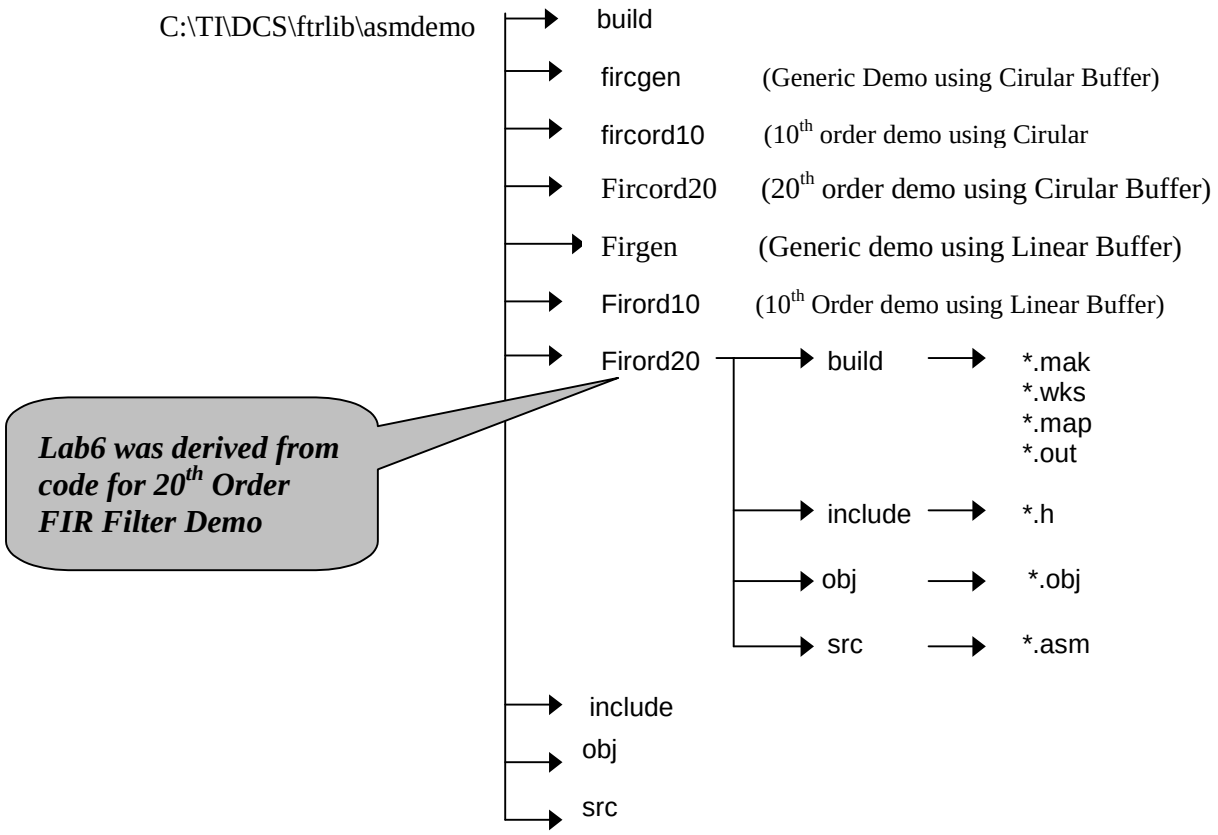
All of these libraries come in .zip Files that should be extracted into the c:\TI\DCS directory structure.

******* It is recommended that users make <u>COPIES</u> of the necessary modules into their own working directory to avoid corruption of the Software Library Material *******



***** <u>Important :</u>** The source code found in the **ftrlib → asmlibxxx** and **ftrlib → cmlibxxx** directories should not be modified as it is the Original Digital Filter Library code. It is recommended that the User make a local copy of the necessary modules from these directories to avoid corrupting the original source code. Lab6 uses a local copy of the material fornd in the Filter Library package

Within the C:\TI\DCS\ftrlib\asmdemo directory, will be several directories as shown below, containing the Digital Filter Demo Software in ASM format. The Code used in Lab6 is a modified version of ***firord20 - Fixed 20th Order FIR Filter using Linear Buffer.***



Each Filter Demo directory (firord10, firord20…) contains all of the necessary files to build and run the appropriate filter demo. The main subdirectories contain the code as follows:

The ***build*** directory has the work space files (*.wks), make files (*.mak) along with the map files (*.map) and output files (*.out) for individual functions demo

The ***include*** directory contains the include file for the ezDSK demo system.

The ***obj*** directory contains the systems object file, generated only in the C framework

The ***src*** directory contains the main system file (*.asm for ASM demo, *.C for C demo).

Appendix 6C – Code Listings

- L6fir20d.asm Main System File
- Fir20a.asm FIR Filter Algorithm
- Fircoeff.asm Filter Coefficients

```
;===============================================================================================
; L6fir20d.asm
;
; Main System file for LF2407 1-Day Workshop Lab #6
; This code is a modified version of one of the Digital Filter Demo's included
; with TI's Digital Filter Library Package.  See Digital Filter Library documentation
; for complete details on using all elements of this package.  This code originated
; from fir20d.asm included in the asmdemo section of TI's Digital Filter Library
; Package.  The changes made to the original code for purposes of Lab#6 are denoted
; with a "Lab6" notation in the comments fields. All other code/comments are as
; included in the original Digital Filter package.
;
;
; Revision 1.0         3/27/01
;
;
;===============================================================================================

;System Name:  ASM Framework for FIRFILT_ORD20 module
;
;File Name:    fir20d.ASM
;
;Description:  Primary System file for demonstrating the FIR Filter
;
;Originator:           Digital control systems Group - Texas Instruments
;
;Target dependency:    x2407/x241/2/3
;
;Description:
;============
;
; This software demonstrates the FIR Filter Usage
;
;   The frame work, for demonstrating the FILTER is given below
;
;
;  CH0  |--------------|           |------------|               |-----------|
;  ---->|              |           |            |               |           |
;  CH0  |              |           |            |               |           |
;  ---->|              |  xn=CH0   |            |  yn           |  EVMDAC   |
;  CH0  |   ADC04U_DRV |-----|---->|    FIR     |-------------->|  PWMDAC   |
;  ---->|              |     |     |   FILTER   |               |  DATALOG  |
;  CH0  |              |     |     |            |          |--->|           |
;  --|->|              |     |     |            |          |    |           |
;       |--------------|     |     |------------|          |    |-----------|
;                            |                             |
;                            |-----------------------------|
;
;===============================================================================================


;*******************************************************************************
;                                    SYSTEM OPTIONS
;*******************************************************************************
REAL_TIME      .set    1    ; 1 for real time mode, otherwise set 0
;*******************************************************************************

;-------------------------------------------------------------------------------
; External references
;-------------------------------------------------------------------------------
                      .include       "asmdemo.h"
```

```
; EVMDAC Module Externel referance symbol declaration
                        .ref    DAC_VIEW_DRV,DAC_VIEW_DRV_INIT              ;function call
                        .ref    DAC_IPTR0,DAC_IPTR1,DAC_IPTR2,DAC_IPTR3     ;Inputs

; DATALOG Module Externel Referance symbol declaration
                        .ref    DATA_LOG, DATA_LOG_INIT                     ;function call
                        .ref    dlog_iptr1, dlog_iptr2                      ;Inputs
                        .ref    trig_value                                  ;Inputs

; PWM DAC Module - Only for x2407 DSP
                        .if x2407
                        .ref    PWM_DAC_DRV, PWM_DAC_DRV_INIT              ;Function Calls
                        .ref    PWM_DAC_IPTR0, PWM_DAC_IPTR1, PWM_DAC_IPTR2 ;Inputs
                        .endif

; ADC Module External Referance Symbol Declaration
                        .ref    ADC04U_DRV, ADC04U_DRV_INIT                 ;function call
                        .ref    A4_ch_sel,C1_gain,C2_gain,C3_gain,C4_gain   ;inputs
                        .ref    C1_out, C2_out, C3_out, C4_out              ;Outputs

; REAL TIME MONITOR External Referance
                        .global MON_RT_CNFG                                 ;function call

; SYS_INIT Routine external referance
                        .ref   SYS_INIT

; Global Varible declaration in main function
                        .def GPR0,GPR1,GPR2,GPR3,GPR4

; Variable declaration for this main function, map the .bss section in DATA Mem area B0

                        .bss    GPR0,1          ;General purpose registers.
                        .bss    GPR1,1
                        .bss    GPR2,1
                        .bss    GPR3,1
                        .bss    GPR4,1
                        .bss    xn,1
                        .bss    yn,1

; Lab6 - Variable declaration for T1 based PWM outputs
;T1CMP = 1000Hz PWM
;CMP1 & CMP2 = 1000Hz @ 10% Duty Cycle Complementary Asymmetric PWM's
;CMP3 & CMP4 = 1000Hz @ 50% Duty Cycle Complementary Asymmetric PWM's
;CMP5 & CMP6 = 1000Hz @ 75% Duty Cycle Complementary Asymmetric PWM's
T1_period       .set    30303           ;1000 Hz PWM @ 33ns Clock
T1_compare      .set    3030            ;10% Duty Cycle @ 1000 Hz PWM
Full_CMPR1      .set    3030            ; Full Compare #1 value for 10% @ 1000Hz
Full_CMPR2      .set    15151           ; Full Compare #2 value for 50% @ 1000Hz
Full_CMPR3      .set    22727           ; Full Compare #3 value for 75% @ 1000Hz
; Lab6 - End Lab6 additions

; Filter function external reference and Symbolic constants
                        .ref    FIRFILT_ORD20_calc
FIR_ORDER20     .set    20

; Create an Instance of FIRFILT_ORD20 module and place the object in "firfilt" section
lpf             .usect "firfilt",FIRFILT_ORD20_LEN,1
lpf             .tag FIRFILT_ORD20              ;Assign the FIRFILT_ORD20 structure
                                                ;properties to the
                                                ;alloted memory locations

;=============================================================================
; V E C T O R    T A B L E    ( including RT monitor traps )
;=============================================================================
           .include "c200mnrt.i"     ; Include conditional assembly options.

          .sect "vectors"
          .def        _c_int0

RESET     B     _c_int0                   ;00
INT1      B     PHANTOM                   ;02
```

```
INT2       B      PHANTOM                    ;04
INT3       B      T2_PERIOD_ISR              ;06
INT4       B      PHANTOM                    ;08
INT5       B      PHANTOM                    ;0A
INT6       B      PHANTOM                    ;0C

               .include       "rtvecs.h"

; Note : The above include line must be AFTER the user configurable
;        vectors. Do not change the place where this line is included.

;=============================================================================
; M A I N   C O D E  - starts here
;=============================================================================
         .text
_c_int0:
         CALL  SYS_INIT
         .if x2407
         POINT_PF1                      ;Load DP with the Periperal Frame 1 page no
         SPLK  #SCSR1_REG_VALUE,SCSR1  ;PLL MF=4, EVA&ADC Enable, clear illegal addr bit
         CALL  PWM_DAC_DRV_INIT
         .endif
         CALL   DAC_VIEW_DRV_INIT
         CALL  DATA_LOG_INIT
         CALL  ADC04U_DRV_INIT

; Copy the Coefficients from LOAD address to Run Address
         CALL  FIRCOEFF_COPY_ROM2RAM

; Filter Initialisation
          LDP  #lpf
          SPLK   #FIR_LPF20_coeff_ptr,lpf.coeff_ptr  ;Select Lowpass
;         SPLK   #FIR_HPF20_coeff_ptr,lpf.coeff_ptr  ;Select Highpass
;         SPLK   #FIR_BPF20_coeff_ptr,lpf.coeff_ptr ;Select Bandpass
;         SPLK   #FIR_BSF20_coeff_ptr,lpf.coeff_ptr ;Select BandStop
          SPLK #lpf.dbuffer,lpf.dbuffer_ptr
          SPLK #FIR_ORDER20,lpf.order
          MAR  *,AR1
          SPLK #lpf,*+        ; Store the address of the instanced filter structure
          CALL FIRFILT_LIN_INIT   ;Call the Linear buffer FIR filter Init routine clear
                                  ;the delay buffer
          MAR  *-
;-----------------------------------------------------------
; Initialise the Real time monitor
;-----------------------------------------------------------
;---Real Time option---------------
               .if (REAL_TIME)
               CALL    MON_RT_CNFG           ;For Real-Time
               .endif

;----------------------------------------
; EVMDAC Module properties initialisation
;----------------------------------------
               LDP     #DAC_IPTR0

; Assign the Address of 'xn' variable to DAC_iptr0
               SPLK    #xn,DAC_IPTR0

; Assign the Address of 'yn' variable to DAC_iptr0
               SPLK    #yn,DAC_IPTR1

;----------------------------------------
; DATALOG Module properties initialisation
;----------------------------------------
               LDP     #dlog_iptr1

; Assign the Address of 'xn' variable to dlog_iptr1
               SPLK    #xn,dlog_iptr1

; Assign the Address of 'yn' variable to dlog_iptr2
               SPLK    #yn,dlog_iptr2
```

```
; Trigger Value =0
               SPLK    #0800h,trig_value

;-----------------------------------------------------
; PWMDAC Module properties initialisation - Only for x2407 DSPs
;-----------------------------------------------------
               .if x2407
               LDP             #PWM_DAC_IPTR0

; Assign the Address of 'xn' variable to PWM_DAC_IPTR0
               SPLK    #xn,PWM_DAC_IPTR0

; Assign the Address of 'yn' variable to PWM_DAC_IPTR1
               SPLK    #yn,PWM_DAC_IPTR1
               .endif

;---------------------------------------------------------------
; ADC Module Properties Initialisation
;---------------------------------------------------------------
               ldp     #A4_ch_sel
        SPLK   #00000h, A4_ch_sel     ; Select CH0 all the 4 ADC module input
               SPLK    #1FFFh, C1_gain               ;gain = 1.0 (Q13)
               SPLK    #1FFFh, C2_gain               ;gain = 1.0 (Q13)
               SPLK    #1FFFh, C3_gain               ;gain = 1.0 (Q13)
               SPLK    #1FFFh, C4_gain               ;gain = 1.0 (Q13)

;---------------------------------------------------------------
; PWM Output Driver Properties Initialisation
;---------------------------------------------------------------
;Lab6   Setup PWM Outputs
        CALL   PWM_INIT

;---------------------------------------------------------------
; Initialise the Timer 2 to generate 20Khz interrupt burst
; and configure the interrupt levels to accept the timer interrupt
;---------------------------------------------------------------
               CALL    TIME_BASE_INIT
               CALL    INT_INIT
               EINT                    ;Enable global Ints

;=======================================================
; Infinite loop
;=======================================================

main_loop:
               B               main_loop

;=============================================================================================
=
; Initialise the Interrupt      INT_INIT
;=============================================================================================

INT_INIT:

               POINT_EV
               .if x2407
               SPLK    #00004h,EVAIMRB        ;Enable T2 UF interrupt
               .endif

               .if x243
               SPLK    #0004h,EVIMRB          ;Enable T2 UF interrupt
               .endif

;C2xx Core
               POINT_PG0

;---Real Time option ------------------------------------------------
               .if (REAL_TIME)
               SPLK    #0000000001000100b,IMR ;En Int lvl 3,7 (T2 ISR)
                        ;|||||||||!||||!|
```

```
                ;5432109876543210
        .endif


        .if (REAL_TIME != 1)
        SPLK    #0000000000000100b,IMR   ;En Int lvl 3 (T2 ISR)
                ;||||!!!!!!!!!!!!
                ;5432109876543210
        .endif
        SPLK    #0FFFFh, IFR             ;Clear any pending Ints
        RET

;=================================================================================================
; Initialise Timer 2 to generate 20Khz Interrupt
;=================================================================================================

TIME_BASE_INIT:
;         Timer 2 Configuration
;     Timer Prescalar= x1
;     Counting Mode: Continuous up counting mode
;     Clock Source: Internal Clock source
;     Reload Condition: When Counter 0
;     Enable the Timer
;     Operation not affected by Emulation suspend

     POINT_EV
     SPLK      #09040h,T2CON
     SPLK      #SYSTEM_INT_PERIOD,T2PER
     RET

;=================================================================================================
; Lab6 - Initialise Timer 1 to generate PWM Outputs
;=================================================================================================
;Setup GP Timer1 for Full Compare PWM outputs on CMP1-CMP6
;Setup T1CMP and T2CMP PWM outputs
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PWM_INIT:
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup shared I/O pins
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #OCRA>>7                          ;set data page
        SPLK    #0011111111000011b,OCRA
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      0=IOPB7, 1=TCLKIN
* bit 14        0:      0=IOPB6, 1=TDIR
* bit 13        1:      0=IOPB5, 1=T2PWM/T2CMP   ;Turn On all PWM's
* bit 12        1:      0=IOPB4, 1=T1PWM/T1CMP          ;
* bit 11        1:      0=IOPB3, 1=PWM6                 ;
* bit 10        1:      0=IOPB2, 1=PWM5                 ;
* bit 9         1:      0=IOPB1, 1=PWM4                 ;
* bit 8         1:      0=IOPB0, 1=PWM3                 ;
* bit 7         1:      0=IOPA7, 1=PWM2                 ;
* bit 6         1:      0=IOPA6, 1=PWM1                 ;
* bit 5         0:      0=IOPA5, 1=CAP3
* bit 4         0:      0=IOPA4, 1=CAP2/QEP1
* bit 3         0:      0=IOPA3, 1=CAP1/QEP0
* bit 2         0:      0=IOPA2, 1=XINT1
* bit 1         1:      0=IOPA1, 1=SCIRXD
* bit 0         1:      0=IOPA0, 1=SCITXD

       POINT_EV              ;set data page
        SPLK    #0000000001000101b, GPTCON     ;init GPTCON register
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      T2STAT - read only
* bit 13        0:      T1STAT - read only
* bit 12-11     00:     reserved
* bit 10-9      10:     T2TOADC, 00 = no ADCSOC
* bit 8-7       00:     T1TOADC, 00 = no ADCSOC
```

```
* bit 6         1:      1 = enable all timer compare outputs
* bit 5-4       00:     reserved
* bit 3-2       01:     01 = T2PIN Active HIGH
* bit 1-0       01:     01 = T1PIN Active low

        SPLK    #0001000000000010b, T1CON      ;init T1CON register
*                ||||||||||||||||
*               FEDCBA9876543210
* bit 15-14     00:     stop immediately on emulator suspend
* bit 13        0:      reserved
* bit 12-11     10:     10 = continous-up count mode ;ASymmetric PWM
* bit 10-8      000:    000 = x/1 prescaler
* bit 7         0:      0 = Reserved in T1CON
* bit 6         0:      0 = disable timer      ;Keep Timer Off for now
* bit 5-4       00:     00 = CPUCLK is clock source
* bit 3-2       00:     00 = reload compare reg on underflow
* bit 1         1:      1 = enable timer compare
* bit 0         0:      0 = Reserved in T1CON

        SPLK    #1000001000000000b, COMCON      ;init COMCON register
*                ||||||||||||||||
*               FEDCBA9876543210
* bit 15        1:      1 = Enable Compare Operation
* bit 14-13     00:     00 = reload compare reg on underflow
* bit 12        0:      0= Disable Space Vector
* bit 11-10     00:     00= Reload Action Control Register on Underflow
* bit 9         1:     1 = Full Compare Outputs Enabled
* bit 8-0      00000000: Reserved


        SPLK    #0000100110011001b, ACTR     ;init ACTR register
*                ||||||||||||||||
*               FEDCBA9876543210
* bit 15        0:      Ignore if not using Space Vector
* bit 14-12     000:    Ignore if not using Space Vector
* bit 11-10     10:     10= Output pin CMP6 Active High
* bit 9-8       01:     01= Output pin CMP5 Active Low
* bit 7-6       10:     10= Output pin CMP4 Active High
* bit 5-4       01:     01= Output pin CMP3 Active Low
* bit 3-2       10:     10= Output pin CMP2 Active High
* bit 1-0       01:     01= Output pin CMP1 Active Low


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Set Periods and Duty's for Compare and FullCompare PWM outputs
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        POINT_EV                        ;set data page
        SPLK    #T1_period, T1PER       ;set timer1 period
        SPLK    #T1_compare, T1CMP      ;Set Timer1 for chosen Duty Cycle
        SPLK    #Full_CMPR1,CMPR1       ;Set Full Compare #1 compare value
        SPLK    #Full_CMPR2,CMPR2       ;Set Full Compare #2 compare value
        SPLK    #Full_CMPR3,CMPR3       ;Set Full Compare #3 compare value
        SPLK    #0000h, T1CNT           ;clear timer1 counter

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Turn on Timer T1 to Start Enabled PWM outputs
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                LACL    T1CON
                OR      #0040h
                SACL    T1CON   ;Set Bit #6 of T1CON to Start Counter
; Lab6 - End Lab6 additions

;=============================================================================
; Routine Name: T1_PERIOD_ISR                           Routine Type: ISR
;
; Description:
;
;
; Originator: Digital control systems Group - Texas Instruments
;
```

```
; History:
;
;
; Last Update: 20-Dec  00
;=============================================================================

T2_PERIOD_ISR:
;Context save regs
               MAR     *,AR1                 ;AR1 is stack pointer
               MAR     *+                    ;skip one position
               SST     #1,*+                 ;save ST1
               SST     #0,*+                 ;save ST0
               SACH    *+                    ;save acc high
               SACL    *+                    ;save acc low

;NOTE: should use "read-modify-write" to clear Int flags & not SPLK!
               POINT_EV
               .if x2407
               SPLK    #0FFFFh,EVAIFRB       ;Clear all Group B interrupt flags (T2 ISR)
               .endif

               .if x243
               SPLK    #0FFFFh,EVIFRB        ;Clear all Group B interrupt flags (T2 ISR)
               .endif

               CALL    ADC04U_DRV
               LDP     #xn
               BLDD    #C1_out,xn            ;Get the latest sample
               LACC    xn                    ;ACC=xn (Q15 unipolar)
               XOR     #4000h
               SACL    xn,1                  ;Bipolar Q15
               LDP     #lpf                  ;Store the input
               BLDD    #xn,lpf.input
;-----------------------------------------------------------------------------
; Filter
;-----------------------------------------------------------------------------
               SPM     #0
               MAR     *,AR1
               SPLK    #lpf,*+
               CALL    FIRFILT_ORD20_calc
               MAR     *-
;-----------------------------------------------------------------------------
               LDP            #yn
               BLDD    #lpf.output,yn

               CALL    DAC_VIEW_DRV          ;Update sx&sy in EVMDAC CH0 & 1 respectively
               CALL    DATA_LOG              ;Update the data log module

               .if x2407
               CALL    PWM_DAC_DRV
               .endif



               MAR     *, AR1                ;make stack pointer active
               SBRK    #1
               LACL    *-                    ;Restore Acc low
               ADDH    *-                    ;Restore Acc high
               LST     #0, *-                ;load ST0
               LST     #1, *-                ;load ST1
               EINT
               RET

;=============================================================================
; I S R  -  PHANTOM
;
; Description: Dummy ISR, used to trap spurious interrupts.
;
; Modifies:
;
; Last Update: 12-29-00
```

```
;===========================================================================
PHANTOM         B       PHANTOM
PHANTOM1        B       PHANTOM1
PHANTOM2        B       PHANTOM2
PHANTOM3        B       PHANTOM3
PHANTOM4        B       PHANTOM4
PHANTOM5        B       PHANTOM5
PHANTOM6        B       PHANTOM6
```

FIR20A.ASM

```
;=====================================================================
;
; File Name:   fir20a.asm
;
; Originator:  Digital Control Systems Group
;              Texas Instruments
;
; Description: This file contain source code for 20th order (FIXED ORDER) Filter
;
; Date:        13/1/2001
;=====================================================================
;
;
; Routine Name : Generic Function
; Routine Type : C Callable
;
; Description  :
; void FIRFILT_ORD20_calc(FIRFILT_ORD20_handle)
; void FIRFILT_ORD20_calc(FIRFILT_ORD20S_handle)
;
; This routine implements the non-recursive difference equation of an
; all-zero filter(FIR), of order 20. All the coefficients of all-zero
; filter are assumed to be less than 1 in magnitude.

;=====================================================================
;
; Difference Equation :
;
;       y(n)=H(0)*x(n)+H(1)*x(n-1)+H(2)*x(n-2)+....+H(20)*x(n-20)
;
;      where
;              y(n)=output sample of the filter at index n
;              x(n)=input sample of the filter at index n
;
; Transfer Function :
;
;              Y(z)                -1        -2              -19        -20
;             ----- = h(0) + h(1) z  + h(2) z  + ... +h(19) z   + h(20) z
;              X(z)
;
;     Network Diagram  :

;     dbuffer[0]          dbuffer[1]    dbuffer[2]    dbuffer[20}
;     Input           -1  x(n-1)  -1    x(n-2)        x(n-20)
;   x(n) >------o----z---->-o----z---->-o---  - ->- - o
;               |           |           |             |
;               |           |           |             |
;               |           |           |             |
;               v H(0)      v H(1)      v H(2)        v H(20)
;               |           |           |             |
;               |           |           |             |        output
;               |---->-----(+)---->----(+)-- - -> -  (+)-----> y(n)
;
;       Symbols Used :
;             H(0),H(1),H(2),...,H(20) : filter coefficients
;            x(n-1),x(n-2),...,x(n-20) : filter states
;                                 x(n) : filter input
;                                 y(n) : filter output
;=====================================================================================
;  Function Input: This function accepts the handle of the below structure
;
;  typedef struct
;     {int *coeff_ptr;        /* Pointer to Filter co-efficient array */
;         int *dbuffer_ptr;   /* Delay buffer pointer */
;         int order;          /* Order of the filter */
;         int input;          /* Input data  */
;         int output;         /* Output data */
;        int dbuffer[11];     /* Delay buffer */
;        int coeff[11];       /* Coefficient Buffer */
```

```
;          void (*init)(void *)   /* Pointer to init fun */
;         void (*calc)(void *);  /* Pointer to the calculation function */
;     }FIRFILT_ORD20;
;
;  Filter input (x) :
;    x(n) = | s.fff ffff | ffff ffff |
;
;  Filter coefficients(h[]):
;       h(k) for k=0,1,2,...,20
;       = | s.fff ffff | ffff ffff |  are stored as shown below.
;
       LOWER MEMORY ADDRESS
;
;                      |----------|
;                      |  H(0)    |        |          |
;                      |----------|        |----------|
;                      |  H(1)    |        | x(n-1)   | dbuffer[1]       |
;                      |----------|        |----------|                  |
;                      |  H(2)    |        | x(n-2)   | dbuffer[0]       |
;                      |----------|        |----------|                  |
;                      |    .     |        |    .     |                  |
;                      |    .     |        |    .     |                  |
;                      |----------|        |----------|                  V
;                      |  H(18)   |        |  x(n-18) |                  |
;                      |----------|        |----------|                  |
;                      |  H(19)   |        |  x(n-19) | dbuffer[18]      V
;                      |----------|        |----------|
;             AR3----->|  H(20)   |        |  x(n-20) | dbuffer[19] <--- AR4
;                      |----------|        |----------|
;                filter coefficients       filter states
;                                                        HIGHER MEMORY ADDRESS
; Implicit inputs(dbuffer[]):
;    To compute the present output y(n), inputs x(n-i) for i=1,2,...20
;    are needed. These are stored in the filter states buffer, as shown
;    above.
;
; Filter order(n):
;       This member of the structure holds the order of the filter
;
; Filter output(y):
;       y(n)= | s.fff ffff | ffff ffff |  Q15 Format
;=====================================================================
; Function Local Frame
;=====================================================================
;   |_______|
;   |_______|<- Stack Pointer                              (FP+1) <---AR1
;   |_______|<- Register to Register Tfr & Computation     (FP)   <---AR0
;   |_______|<- Old FP                                     (FP-1)
;   |_______|<- Return Address of the Caller               (FP-2)
;   |_______|<- Formal parameter FIRFILT_GEN_handle        (FP-3)
;=====================================================================

; Symbolic Constant

ORDER                .set    20

; Module definition for external referance
                     .def    FIRFILT_ORD20_calc


__fir_calc_frs .set    00001h               ; Local frame size for this routine


FIRFILT_ORD20_calc:
                     POPD    *+                      ; Store the Return Address in stack
                     SAR     AR0,*+          ; Store the Caller's Frame Pointer
                      SAR    AR1,*
                     LAR     AR2,*+,AR2      ; ARP=AR2, AR2=FP, AR1=FP+1

                     SETC    SXM
                     SETC     OVM
```

```
SBRK    #3      ;ARP=AR2, AR2=FP-3 points to the input argument
LAR     AR2,*   ;ARP=AR2, AR2=FIRFILT_GEN_handle->coeff_ptr
LAR     AR3,*+  ;ARP=AR2, AR2->dbuffer_ptr, AR3=coeff_ptr->coeff[N]
LAR     AR4,*+  ;ARP=AR2, AR2->order, AR4=dbuffer_ptr->dbuffer[N-1]
MAR     *+,AR4  ;ARP=AR4, AR4=dbuffer_ptr->dbuffer[N-1], AR2->input
LACL    #0      ;Clear ACC
LT      *-,AR3  ;ARP=AR3, TREG=dbuffer[N-1], AR4->dbuffer[N-2]

.loop   (ORDER-1)
MPY     *-,AR4  ;Filter Tap execution from oldest sample
LTD     *-,AR3
.endloop

MPY     *-,AR2          ; ARP=AR2, AR2->input
LTA     *+,AR3          ; ARP=AR3, AR2->output
MPY     *,AR2
APAC
SACH    *     ;32 bit of ACC is added with itself to get Q31 result
ADDH    *
SACL    *
ADDS    *       ;ARP=AR2, AR2->output
ADD     #1,15   ;Round the result
SACH    *-      ;ARP=AR2, AR2->input, Store ACCH in Q15 format
LACL    *,AR4
MAR     *+
SACL    *,0,AR1    ;Store the latest sample in the delay buffer

CLRC    OVM
SBRK    #(__fir_calc_frs+1)    ;Clear the local frame
LAR     AR0,*-          ;Retrive Caller's frame pointer
PSHD    *               ;Push the return address to TOS
RET                    ;Return to the caller
```

FIRCOEFF.ASM

```
;===============================================================================
; File name:    FIRCOEFF.ASM
;
; Originator:  Digital Control Systems Group
;              Texas Instruments
; Description:
; This file contains Filter coefficients and the initialisation routine
;===============================================================================
; History:
;-------------------------------------------------------------------
; 24-1-2000    Release Rev 1.0
;-------------------------------------------------------------------
;Header file containing FIR Filter coefficients to initialise the coefficient array.
;4 coefficient sets for Order 50, 20, & 10 are available for demo, user can expand it
;using ezFIR script
;===============================================================================

; Include firsel.h header file
                      .include  firsel.h

; External declaration to copy the FIR coeff from ROM to RAM
                       .def   FIRCOEFF_COPY_ROM2RAM

; FIRCOEFF_COPY_ROM2RAM routine that copies the filter coefficients from ROM to RAM

FIRCOEFF_COPY_ROM2RAM:
                     .text
                      LAR     AR2,#coeff_RUN_start_adr
                      LACC    #coeff_LOAD_start_adr
                      MAR     *,AR2
                      LAR     AR3, #(coeff_LOAD_end_adr-coeff_LOAD_start_adr-1)
loop:                 TBLR    *+,AR3
                      ADD             #1
                      BANZ    loop,*-,AR2
                      RET

                      .sect   "fircoeff"
                      .label  coeff_LOAD_start_adr   ;Load Address of the section
                                                     ;coeff_RUN_start_adr
                                                     ;Run Address of the section

;*****************************************************
; 50th Order Filter
;*****************************************************

                      .if FIR_LPF50
FIR_LPF50_coeff_ptr   .word -34,-22,13,50,55,-1,-91,-137,-56,131
                      .word 276,198,-127,-471,-481,0,690,976,384,-896
                      .word -1905,-1447,1040,4892,8415,9839,8415,4892,1040,-1447
                      .word -1905,-896,384,976,690,0,-481,-471,-127,198
                      .word 276,131,-56,-137,-91,-1,55,50,13,-22
                      .word -34
                      .endif

                      .if FIR_HPF50
FIR_HPF50_coeff_ptr   .word 0,34,25,-32,-65,-1,106,83,-106,-214
                      .word 0,321,240,-291,-565,-1,810,602,-730,-1447
                      .word 0,2337,1976,-3019,-9877,19646,-9877,-3019,1976,2337
                      .word 0,-1447,-730,602,810,-1,-565,-291,240,321
                      .word 0,-214,-106,83,106,-1,-65,-32,25,34
                      .word 0
                      .endif

                      .if FIR_BPF50
FIR_BPF50_coeff_ptr   .word 0,69,-1,-63,0,-1,-1,167,-1,-427
                      .word 0,642,-1,-581,-1,-1,0,1203,-1,-2890
                      .word 0,4670,-1,-6033,-1,6543,-1,-6033,-1,4670
                      .word 0,-2890,-1,1203,0,-1,-1,-581,-1,642
                      .word 0,-427,-1,167,-1,-1,0,-63,-1,69
```

```
                        .word 0
                        .endif

                        .if FIR_BSF50
FIR_BSF50_coeff_ptr     .word -1,-70,0,62,-1,-1,0,-169,0,427
                        .word 0,-645,-1,582,0,-1,0,-1208,0,2900
                        .word 0,-4688,0,6054,0,26267,0,6054,0,-4688
                        .word 0,2900,0,-1208,0,-1,0,582,-1,-645
                        .word 0,427,0,-169,0,-1,-1,62,0,-70
                        .word -1
                        .endif

;******************************************************
; 20th Order Filter
;******************************************************
; Coeff's for 3000Hz Corner Freq 20th Order LowPass Filter
                        .if FIR_LPF20
FIR_LPF20_coeff_ptr     .word 0,95,207,123,-406,-1125,-1044,869,4517,8236
                        .word 9815,8236,4517,869,-1044,-1125,-406,123,207,95
                        .word 0
                        .endif

; Coeff's for 4000Hz Corner Freq 20th Order HighPass Filter
                        .if FIR_HPF20
FIR_HPF20_coeff_ptr     .word -1,113,128,-237,-659,0,1694,1658,-2801,-9711
                        .word 19689,-9711,-2801,1658,1694,0,-659,-237,128,113
                        .word -1
                        .endif

; Coeff's for 4000Hz-6000Hz Corner Freq 20th BandPass Filter
                       .if FIR_BPF20
FIR_BPF20_coeff_ptr     .word -1,0,304,-1,-1560,0,4009,-1,-6628,-1
                        .word 7766,-1,-6628,-1,4009,0,-1560,-1,304,0
                        .word -1
                        .endif

; Coeff's for 4000Hz-6000Hz Corner Freq 20th BandStop Filter
                        .if FIR_BSF20
FIR_BSF20_coeff_ptr     .word -1,0,-258,0,1316,0,-3386,0,5595,0
                        .word 26229,0,5595,0,-3386,0,1316,0,-258,0
                        .word -1
                        .endif

;******************************************************
; 10th Order Filter
;******************************************************

                        .if FIR_LPF10
FIR_LPF10_coeff_ptr     .word -171,-264,437,3464,7882,10067,7882,3464,437,-264
                        .word -171
                        .endif

                        .if FIR_HPF10
FIR_HPF10_coeff_ptr     .word 0,416,812,-2091,-9045,19652,-9045,-2091,812,416
                        .word 0
                        .endif

                        .if FIR_BPF10
FIR_BPF10_coeff_ptr     .word 0,1645,-1,-8264,-1,12949,-1,-8264,-1,1645
                        .word 0
                        .endif

                        .if FIR_BSF10
FIR_BSF10_coeff_ptr     .word 0,-829,0,4163,0,26098,0,4163,0,-829
                        .word 0
                        .endif

                        .label coeff_LOAD_end_adr
```

Learning Objectives

Recap / What's Next

- **Great!! You have the basics of a DSP Control System**
  - **Signal capture-digital control-output generation**
- **Beyond the eZdsp™: What are you going to do NOW!!**
  - **Software development tools**
  - **Hardware development tools**
  - **Additional training**
  - **Web Resources**
  - **Third Party Resources**

Module Topics

What's Next **7-1**

Module Topics *7-2*

Development Support *7-3*

Development Support

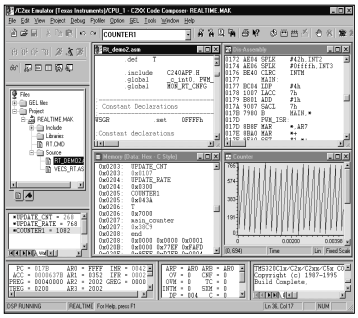
Code Generation & Programming

- **Code Generation Tools, v7.0**
  - **Highly efficient C-Compiler/Assembler/Linker**
  - **ANSI C source code compliant.**
  - **Variety of optimizations for improved code efficiency.**
  - **Includes archiver, hex conversion and cross reference utilities, and absolute lister.**
- **Flash Programming Utilities**
  - **Support for JTAG and Serial Port flash programming.**
  - **LF240x & F240/241/243 utilities available today with free WEB download:**

    http://www.ti.com/sc/docs/tools/dsp/c2000/software.htm#C2000

Code Composer 4.12

Offers Real-time Monitor and Plug-in Capabilities



- New interface style
- C2000 Code Gen Tools included
- Command window plug-in
- C24x real-time debugging:
  - Halt the application and still handle interrupts; ability to update graphs, watch and memory windows, without halting the processor
  - Single-stepping of background code while ISR's continue to run in real-time
  - Tuning of system parameters, e.g. PID loops, Motors char, etc.
  - Software switches to control/invoke motor wind-down sequence
- Drag and drop setup program
- Emulation and simulation drivers include
- BIOS supported on C28x

Recommended Next Step: DMC Software Modules

Module	Description	Type
FC_PWM_DRV	Full Compare PWM driver with dead-band control	Driver
CAP_EVT_DRV	Capture input event driver with 16 bit time stamps	Driver
QEP_THETA_DRV	Quadrature Encoder Pulse interface driver for position	Driver
ILEG2_DRV	Inverter leg based 3 phase current measurement	Driver
ADC04_DRV	General purpose 4 conversion ADC driver with gain control	Driver
BLDC_PWM_DRV	BLDC PWM driver – uses High side chopping	Driver
VHz_PROF	Volts / Hertz profile for ACI (Voltage vs Frequency)	TI / AC
CLARK	Clark transform – 3 phase to 2 phase (quadrature) conversion	TI / AI
CLARK_I	Inverse Clark transform – 2 phase to 3 phase conversion	TI / AI
PARK	Park Transform – Stationary to Rotating reference frame conv.	TI / AI
PARK_I	Inverse Park Transform – Rotating to Stationary reference frame	TI / AI
PMSM_SMO	PMSM Sliding mode observer for position estimation	TI / AC
ACI_MRAS	Model Reference Adaptive System speed estimator	TI / AC

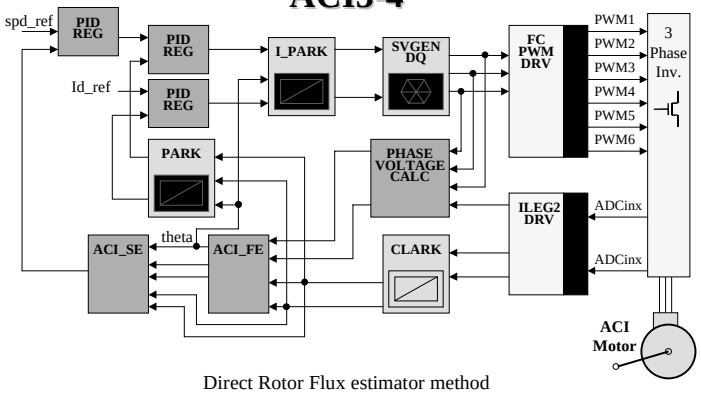
Many more software modules are available at
http://www.ti.com/sc/dmcsoftware

Note: Driver = Peripheral dependent driver
TI/AC = Target independent / Application configurable
TI/AI = Target independent / Application independent

Recommended Next Step: DMC Software Systems

System	Motor Type	Description
ACI1-1	1 ph AC induction	**Sensored** – Tacho i/p VHz / SinePWM / Closed loop (CL) speed PID
ACI3-1	3 ph AC induction	**Sensored** – Tacho i/p VHz / SVPWM / Closed loop speed PID
ACI3-2	3 ph AC induction	**Sensorless** – MRAS (speed estimator) VHz / SVPWM / Closed loop speed PID
ACI3-3	3 ph AC induction	**Sensored** – Tacho i/p FOC / SVPWM / Closed loop current PID for D,Q / CL speed PID
ACI3-4	3 ph AC induction	**Sensorless** – Direct Flux estimator + speed estimator FOC / SVPWM / Closed loop current PID for D,Q / CL speed PID
ACI3-5	3 ph AC induction	**Sensorless** – Direct Torque Control (DTC) Modified SVPWM / Closed loop current / CL speed
PMSM3-1	3 Ph Permanent Magnet Synch	**Sensored** - QEP FOC / SVPWM / Closed loop current PID for D,Q / CL speed PID
PMSM3-2	3 Ph Permanent Magnet Synch	**Sensorless** – SMO (Sliding mode observer) position estimator FOC / SVPWM / Closed loop current PID for D,Q / CL speed PID
BLDC3-1	3 ph Trapezoidal Brushles DC	**Sensored** – 3 Hall effect i/p Trapezoidal / Closed loop current PID / CL Speed PID
BLDC3-2	3 ph Trapezoidal Brushles DC	**Sensorless** – BEMF / Zero crossing detection Trapezoidal / Closed loop current PID / CL Speed PID

Recommended Next Step: System example 4: ACI FOC Sensorless "ACI3-4"



Direct Rotor Flux estimator method

Recommended Next Step: C2000 Evaluation Modules

C2000 Evaluation Modules include:

- C2000 Evaluation Board
- Code Generation Tools
- Code Composer Debugger
- XDS510PP+ Emulator
- Universal Power Supply
- Documentation & cables

Everything you need to develop your own LF240x DSP motor control based algorithms!

Recommended Next Step: 4-day Training Course

Learn the LF2407 Evaluation Module

In-depth C240x Core and Peripheral Training

http://www.ti.com/sc/docs/training/training.htm

Recommended Next Step: Explore the Web

Material Available For Web Download

More than 80,000 downloads

Type	Number Available	Description
Application Notes	**60**	Variety of topics including ACI, BLDC, PMSM, SR Motors, and CPU/Peripheral reference guides.
Application Design Kits (ADK)	**5**	Full system designs, including software, technical documentation, and schematics (BLDC, PMSM, ACI, UPS, SRM)
Software Library	**15**	Segmented 'packages' of source code for functions such as ramp generators, sine and cosine generators, and current sampling.
Flash Programming Utilities	**5**	Flash programming software for F240, F241/F243, LF24xx (parallel and serial).

Large Third Party Network

TECHNOSOFT

FSI systems inc.

NFO®

BP MICROSYSTEMS The Standard in Device Programming

Spectrum Digital

Drivetec, Inc.

PENSAR

SOFTRONICS

International Rectifier

Oztek Corp. Motion Control Specialists

Hyperception www.hyperception.com The Leader in DSP

SCHMIDHAUSER AG

Visual Solutions

Thank You!!

Appendix

Module Topics

Appendix **A-1**

Module Topics *A-2*

eZdsp™ LF2407 ..... *A-3*
eZdsp™ LF2407 Connector / Header and Pin Diagram A-3
P1 / P7 – Analog Interface A-4
P2 / P8 – I/O Interface A-5
P6 – Expansion Interface A-6

F2407.h *A-7*

eZdsp™ LF2407

eZdsp™ LF2407 Connector / Header and Pin Diagram

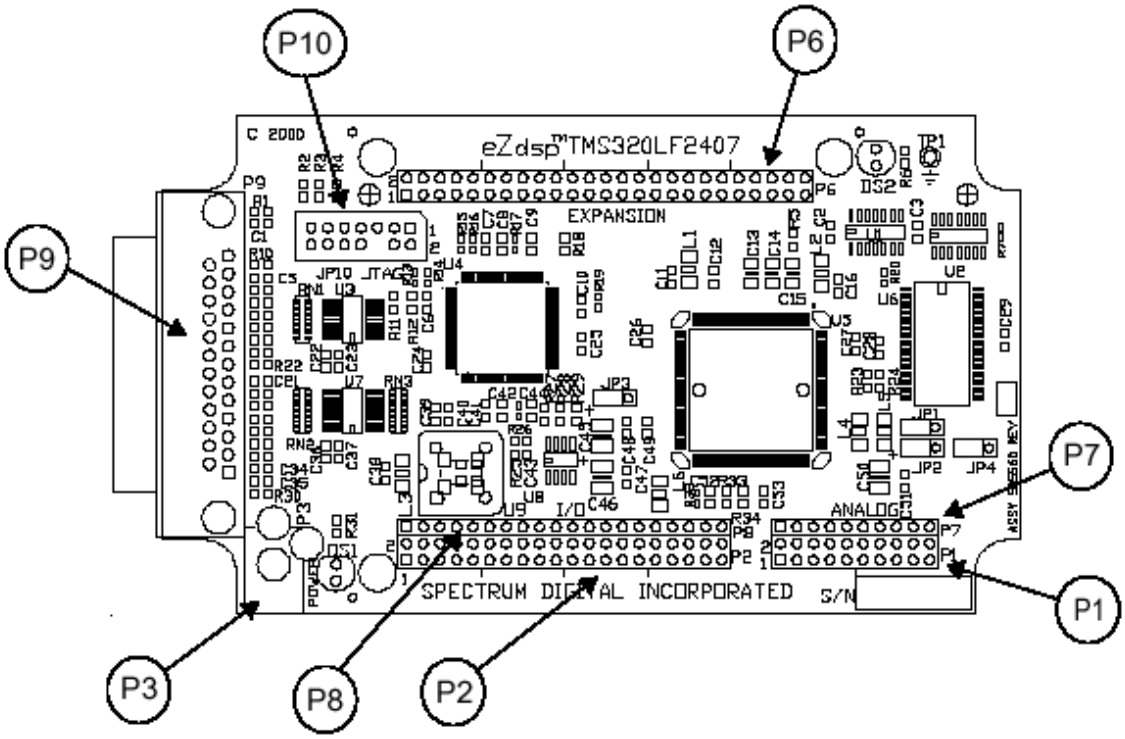


Table 1: eZdsp™ LF2407 Connectors

Connector	Function
P1/P7	Analog Interface
P2/P8	I/O Interface
P3	Power Connector
P6	I/O Expansion Connector
P9	Parallel Port/JTAG Controller Interface
P10	JTAG Interface

P1 / P7 – Analog Interface

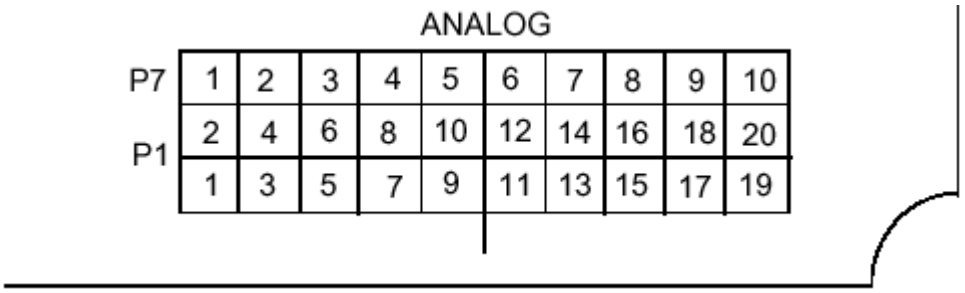


Table 2: P1, Analog Interface Connector

Pin #	Signal	Pin #	Signal
1	GND	2	ADCIN0
3	GND	4	ADCIN1
5	GND	6	ADCIN2
7	GND	8	ADCIN3
9	GND	10	ADCIN4
11	GND	12	ADCIN5
13	GND	14	ADCIN6
15	GND	16	ADCIN7
17	GND	18	VREFLO
19	GND	20	VREFHI

Table 3: P7, Analog Interface Connector

Pin #	Signal
1	ADCIN8
2	ADCIN9
3	ADCIN10
4	ADCIN11
5	ADCIN12
6	ADCIN13
7	ADCIN14
8	ADCIN15
9	RESERVED
10	RESERVED

P2 / P8 – I/O Interface

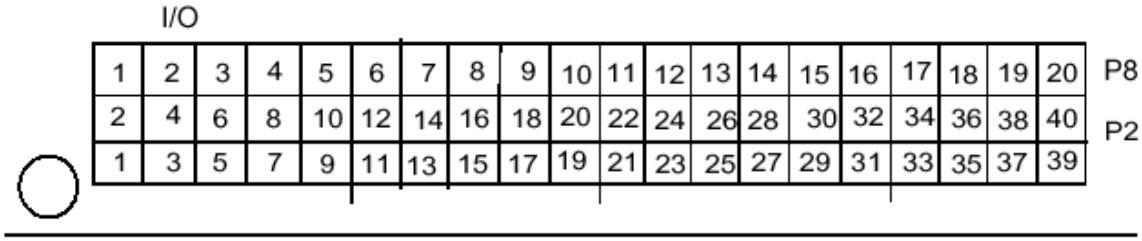


Table 4: P2, I/O Interface Connector

Pin #	Signal	Pin #	Signal
1	+5V	2	+5V
3	SCITXD/IOPA0	4	SCIRXD/IOPA1
5	XINT1-/IOPA2	6	CAP1/QEP1/IOPA3
7	CAP2/QEP2/IOPA4	8	CAP3/IOPA5
9	PWM1/IOPA6	10	PWM2/IOPA7
11	PWM3/IOPB0	12	PWM4/IOPB1
13	PWM5/IOPB2	14	PWM6/IOPB3
15	T1PWM/T1CMP/IOPB4	16	T2PWM/T2CMP/IOPB5
17	TDIRA/IOPB6	18	TCLKINA/IOPB7
19	GND	20	GND
21	WNR/IOPC0	22	BIO-/IOPC1
23	SPISIMO/IOPC2	24	SPISOMI/IOPC3
25	SPICLK/IOPC4	26	SPISTE/IOPC5
27	CANTX/IOPC6	28	CANRX/IOPC7
29	CLKOUT/IOPE0	30	PWM7/IOPE1
31	PWM8/IOPE2	32	PWM9/IOPE3
33	PWM10/IOPE4	34	PWM11/IOPE5
35	PWM12/IOPE6	36	CAP4/QEP3/IOPE7
37	PDPINTA-	38	PDPINTB-
39	GND	40	GND

Table 5: P8, I/O Interface Connector

Pin #	Signal
1	+5V
2	XINT2-/ADCSOC/ IOPD0
3	EMU0/IOPD1
4	EMU1/IOPD2
5	TCK/IOPD3
6	TDI/IOPD4
7	TDO/IOPD5
8	TMS/IOPD6
9	TMS2/IOPD7
10	GND
11	CAP5/QEP4/IOPF0
12	CAP6/IOPF1
13	T3PWM/T3CMP/IOPF2
14	T4PWM/T4CMP/IOPF3
15	TDIRB/IOPF4
16	TCLKINB/IOPF5
17	IOPF6
18	RESERVED
19	RESERVED
20	GND

P6 – Expansion Interface

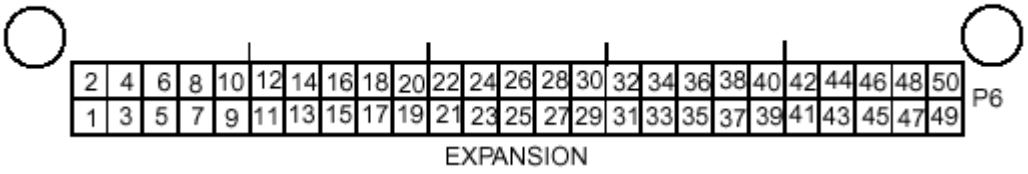


Table 6: P6, Expansion Interface Connector

Pin #	Signal	Pin #	Signal
1	+5V	2	+5V
3	D0	4	D1
5	D2	6	D3
7	D4	8	D5
9	D6	10	D7
11	D8	12	D9
13	D10	14	D11
15	D12	16	D13
17	D14	18	D15
19	A0	20	A1
21	A2	22	A3
23	A4	24	A5
25	A6	26	A7
27	A8	28	A9
29	A10	30	A11
31	A12	32	A13
33	A14	34	A15
35	GND	36	GND
37	PS-	38	DS-
39	READY	40	IS-
41	R/W-	42	STRB-
43	WE-	44	RD-
45	3.3V**	46	RESERVED
47	RS-*	48	RESERVED
49	GND	50	GND

F2407.h

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
;Filename: f2407.h                                                          |
;                                                                           |
;Last Modified: 10/24/00                                                    |
;                                                                           |
;Description: LF2407 DSP register definitions                               |
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;Core registers
IMR                .set 0004h   ;Interrupt mask reg
GREG               .set 0005h   ;Global memory allocation reg
IFR                .set 0006h   ;Interrupt flag reg

;System configuration and interrupt registers
PIRQR0             .set 7010h   ;Peripheral interrupt request reg 0
PIRQR1             .set 7011h   ;Peripheral interrupt request reg 1
PIRQR2             .set 7012h   ;Peripheral interrupt request reg 2
PIACKR0            .set 7014h   ;Peripheral interrupt acknowledge reg 0
PIACKR1            .set 7015h   ;Peripheral interrupt acknowledge reg 1
PIACKR2            .set 7016h   ;Peripheral interrupt acknowledge reg 2
SCSR1              .set 7018h   ;System control & status reg 1
SCSR2              .set 7019h   ;System control & status reg 2
DINR               .set 701Ch   ;Device identification reg
PIVR               .set 701Eh   ;Peripheral interrupt vector reg

;Watchdog timer (WD) registers
WDCNTR             .set 7023h   ;WD counter reg
WDKEY              .set 7025h   ;WD reset key reg
WDCR               .set 7029h   ;WD timer control reg

;Serial Peripheral Interface (SPI) registers
SPICCR             .set 7040h   ;SPI configuration control reg
SPICTL             .set 7041h   ;SPI operation control reg
SPISTS             .set 7042h   ;SPI status reg
SPIBRR             .set 7044h   ;SPI baud rate reg
SPIRXEMU           .set 7046h   ;SPI emulation buffer reg
SPIRXBUF           .set 7047h   ;SPI serial receive buffer reg
SPITXBUF           .set 7048h   ;SPI serial transmit buffer reg
SPIDAT             .set 7049h   ;SPI serial data reg
SPIPRI             .set 704Fh   ;SPI priority control reg

;SCI registers
SCICCR             .set 7050h   ;SCI communication control reg
SCICTL1            .set 7051h   ;SCI control reg 1
SCIHBAUD           .set 7052h   ;SCI baud-select reg, high bits
SCILBAUD           .set 7053h   ;SCI baud-select reg, low bits
SCICTL2            .set 7054h   ;SCI control reg 2
SCIRXST            .set 7055h   ;SCI receiver status reg
SCIRXEMU           .set 7056h   ;SCI emulation data buffer reg
SCIRXBUF           .set 7057h   ;SCI receiver data buffer reg
SCITXBUF           .set 7059h   ;SCI transmit data buffer reg
SCIPRI             .set 705Fh   ;SCI priority control reg

;External interrupt configuration registers
XINT1CR            .set 7070h   ;Ext interrupt 1 config reg
```

```
XINT2CR             .set 7071h   ;Ext interrupt 2 config reg

;Digital I/O registers
MCRA                .set 7090h   ;I/O mux control reg A
MCRB                .set 7092h   ;I/O mux control reg B
MCRC                .set 7094h   ;I/O mux control reg C
PADATDIR            .set 7098h   ;I/O port A data & dir reg
PBDATDIR            .set 709Ah   ;I/O port B data & dir reg
PCDATDIR            .set 709Ch   ;I/O port C data & dir reg
PDDATDIR            .set 709Eh   ;I/O port D data & dir reg
PEDATDIR            .set 7095h   ;I/O port E data & dir reg
PFDATDIR            .set 7096h   ;I/O port F data & dir reg

;Analog-to-Digital Converter (ADC) registers
ADCTRL1             .set 70A0h   ;ADC control reg 1
ADCTRL2             .set 70A1h   ;ADC control reg 2
MAX_CONV            .set 70A2h   ;Maximum conversion channels reg
CHSELSEQ1           .set 70A3h   ;Channel select sequencing cntrl reg 1
CHSELSEQ2           .set 70A4h   ;Channel select sequencing cntrl reg 2
CHSELSEQ3           .set 70A5h   ;Channel select sequencing cntrl reg 3
CHSELSEQ4           .set 70A6h   ;Channel select sequencing cntrl reg 4
AUTO_SEQ_SR         .set 70A7h   ;Autosequence status reg
RESULT0             .set 70A8h   ;Conversion result buffer reg 0
RESULT1             .set 70A9h   ;Conversion result buffer reg 1
RESULT2             .set 70AAh   ;Conversion result buffer reg 2
RESULT3             .set 70ABh   ;Conversion result buffer reg 3
RESULT4             .set 70ACh   ;Conversion result buffer reg 4
RESULT5             .set 70ADh   ;Conversion result buffer reg 5
RESULT6             .set 70AEh   ;Conversion result buffer reg 6
RESULT7             .set 70AFh   ;Conversion result buffer reg 7
RESULT8             .set 70B0h   ;Conversion result buffer reg 8
RESULT9             .set 70B1h   ;Conversion result buffer reg 9
RESULT10            .set 70B2h   ;Conversion result buffer reg 10
RESULT11            .set 70B3h   ;Conversion result buffer reg 11
RESULT12            .set 70B4h   ;Conversion result buffer reg 12
RESULT13            .set 70B5h   ;Conversion result buffer reg 13
RESULT14            .set 70B6h   ;Conversion result buffer reg 14
RESULT15            .set 70B7h   ;Conversion result buffer reg 15
CALIBRATION         .set 70B8h   ;Calibration result reg

;Controller Area Network (CAN) registers
MDER                .set 7100h   ;CAN mailbox direction/enable reg
TCR                 .set 7101h   ;CAN transmission control reg
RCR                 .set 7102h   ;CAN receive control reg
MCR                 .set 7103h   ;CAN master control reg
BCR2                .set 7104h   ;CAN bit config reg 2
BCR1                .set 7105h   ;CAN bit config reg 1
ESR                 .set 7106h   ;CAN error status reg
GSR                 .set 7107h   ;CAN global status reg
CEC                 .set 7108h   ;CAN trans and rcv err counters
CAN_IFR             .set 7109h   ;CAN interrupt flag reg
CAN_IMR             .set 710ah   ;CAN interrupt mask reg
LAM0_H              .set 710bh   ;CAN local acceptance mask MBX0/1
LAM0_L              .set 710ch   ;CAN local acceptance mask MBX0/1
LAM1_H              .set 710dh   ;CAN local acceptance mask MBX2/3
LAM1_L              .set 710eh   ;CAN local acceptance mask MBX2/3
```

```
MSGID0L             .set 7200h   ;CAN msg ID for mailbox 0 (lower 16b)
MSGID0H             .set 7201h   ;CAN msg ID for mailbox 0 (upper 16b)
MSGCTRL0            .set 7202h   ;CAN RTR and DLC for mailbox 0
MBX0A               .set 7204h   ;CAN 2 of 8 bytes of mailbox 0
MBX0B               .set 7205h   ;CAN 2 of 8 bytes of mailbox 0
MBX0C               .set 7206h   ;CAN 2 of 8 bytes of mailbox 0
MBX0D               .set 7207h   ;CAN 2 of 8 bytes of mailbox 0

MSGID1L             .set 7208h   ;CAN msg ID for mailbox 1 (lower 16b)
MSGID1H             .set 7209h   ;CAN msg ID for mailbox 1 (upper 16b)
MSGCTRL1            .set 720Ah   ;CAN RTR and DLC for mailbox 1
MBX1A               .set 720Ch   ;CAN 2 of 8 bytes of mailbox 1
MBX1B               .set 720Dh   ;CAN 2 of 8 bytes of mailbox 1
MBX1C               .set 720Eh   ;CAN 2 of 8 bytes of mailbox 1
MBX1D               .set 720Fh   ;CAN 2 of 8 bytes of mailbox 1

MSGID2L             .set 7210h   ;CAN msg ID for mailbox 2 (lower 16b)
MSGID2H             .set 7211h   ;CAN msg ID for mailbox 2 (upper 16b)
MSGCTRL2            .set 7212h   ;CAN RTR and DLC for mailbox 2
MBX2A               .set 7214h   ;CAN 2 of 8 bytes of mailbox 2
MBX2B               .set 7215h   ;CAN 2 of 8 bytes of mailbox 2
MBX2C               .set 7216h   ;CAN 2 of 8 bytes of mailbox 2
MBX2D               .set 7217h   ;CAN 2 of 8 bytes of mailbox 2

MSGID3L             .set 7218h   ;CAN msg ID for mailbox 3 (lower 16b)
MSGID3H             .set 7219h   ;CAN msg ID for mailbox 3 (upper 16b)
MSGCTRL3            .set 721Ah   ;CAN RTR and DLC for mailbox 3
MBX3A               .set 721Ch   ;CAN 2 of 8 bytes of mailbox 3
MBX3B               .set 721Dh   ;CAN 2 of 8 bytes of mailbox 3
MBX3C               .set 721Eh   ;CAN 2 of 8 bytes of mailbox 3
MBX3D               .set 721Fh   ;CAN 2 of 8 bytes of mailbox 3

MSGID4L             .set 7220h   ;CAN msg ID for mailbox 4 (lower 16b)
MSGID4H             .set 7221h   ;CAN msg ID for mailbox 4 (upper 16b)
MSGCTRL4            .set 7222h   ;CAN RTR and DLC for mailbox 4
MBX4A               .set 7224h   ;CAN 2 of 8 bytes of mailbox 4
MBX4B               .set 7225h   ;CAN 2 of 8 bytes of mailbox 4
MBX4C               .set 7226h   ;CAN 2 of 8 bytes of mailbox 4
MBX4D               .set 7227h   ;CAN 2 of 8 bytes of mailbox 4

MSGID5L             .set 7228h   ;CAN msg ID for mailbox 5 (lower 16b)
MSGID5H             .set 7229h   ;CAN msg ID for mailbox 5 (upper 16b)
MSGCTRL5            .set 722Ah   ;CAN RTR and DLC for mailbox 5
MBX5A               .set 722Ch   ;CAN 2 of 8 bytes of mailbox 5
MBX5B               .set 722Dh   ;CAN 2 of 8 bytes of mailbox 5
MBX5C               .set 722Eh   ;CAN 2 of 8 bytes of mailbox 5
MBX5D               .set 722Fh   ;CAN 2 of 8 bytes of mailbox 5

;Event Manager A (EVA) registers
GPTCONA             .set 7400h   ;GP timer control reg A
T1CNT               .set 7401h   ;GP timer 1 counter reg
T1CMPR              .set 7402h   ;GP timer 1 compare reg
T1PR                .set 7403h   ;GP timer 1 period reg
T1CON               .set 7404h   ;GP timer 1 control reg
T2CNT               .set 7405h   ;GP timer 2 counter reg
T2CMPR              .set 7406h   ;GP timer 2 compare reg
T2PR                .set 7407h   ;GP timer 2 period reg
```

```
T2CON               .set 7408h   ;GP timer 2 control reg
COMCONA             .set 7411h   ;Compare control reg A
ACTRA               .set 7413h   ;Compare action control reg A
DBTCONA             .set 7415h   ;Dead-band timer control reg A
CMPR1               .set 7417h   ;compare reg 1
CMPR2               .set 7418h   ;compare reg 2
CMPR3               .set 7419h   ;compare reg 3
CAPCONA             .set 7420h   ;Capture control reg A
CAPFIFOA            .set 7422h   ;Capture FIFO status reg A
CAP1FIFO            .set 7423h   ;Capture Channel 1 FIFO top
CAP2FIFO            .set 7424h   ;Capture Channel 2 FIFO top
CAP3FIFO            .set 7425h   ;Capture Channel 3 FIFO top
CAP1FBOT            .set 7427h   ;Bottom reg of capture FIFO stack 1
CAP2FBOT            .set 7427h   ;Bottom reg of capture FIFO stack 2
CAP3FBOT            .set 7427h   ;Bottom reg of capture FIFO stack 3
EVAIMRA             .set 742Ch   ;EVA interrupt mask reg A
EVAIMRB             .set 742Dh   ;EVA interrupt mask reg B
EVAIMRC             .set 742Eh   ;EVA interrupt mask reg C
EVAIFRA             .set 742Fh   ;EVA interrupt flag reg A
EVAIFRB             .set 7430h   ;EVA interrupt flag reg B
EVAIFRC             .set 7431h   ;EVA interrupt flag reg C

;Event Manager B (EVB) registers
GPTCONB             .set 7500h   ;GP timer control reg B
T3CNT               .set 7501h   ;GP timer 3 counter reg
T3CMPR              .set 7502h   ;GP timer 3 compare reg
T3PR                .set 7503h   ;GP timer 3 period reg
T3CON               .set 7504h   ;GP timer 3 control reg
T4CNT               .set 7505h   ;GP timer 4 counter reg
T4CMPR              .set 7506h   ;GP timer 4 compare reg
T4PR                .set 7507h   ;GP timer 4 period reg
T4CON               .set 7508h   ;GP timer 4 control reg
COMCONB             .set 7511h   ;Compare control register B
ACTRB               .set 7513h   ;Compare action control register B
DBTCONB             .set 7515h   ;Dead-band timer control reg B
CMPR4               .set 7517h   ;Compare reg 4
CMPR5               .set 7518h   ;Compare reg 5
CMPR6               .set 7519h   ;Compare reg 6
CAPCONB             .set 7520h   ;Capture control reg B
CAPFIFOB            .set 7522h   ;Capture FIFO status reg B
CAP4FIFO            .set 7523h   ;Capture channel 4 FIFO top
CAP5FIFO            .set 7524h   ;Capture channel 5 FIFO top
CAP6FIFO            .set 7525h   ;Capture channel 6 FIFO top
CAP4FBOT            .set 7527h   ;Bottom reg of capture FIFO stack 4
CAP5FBOT            .set 7527h   ;Bottom reg of capture FIFO stack 5
CAP6FBOT            .set 7527h   ;Bottom reg of capture FIFO stack 6
EVBIMRA             .set 752Ch   ;EVB interrupt mask reg A
EVBIMRB             .set 752Dh   ;EVB interrupt mask reg B
EVBIMRC             .set 752Eh   ;EVB interrupt mask reg C
EVBIFRA             .set 752Fh   ;EVB interrupt flag reg A
EVBIFRB             .set 7530h   ;EVB interrupt flag reg B
EVBIFRC             .set 7531h   ;EVB interrupt flag reg C

;I/O space mapped registers
FCMR                .set 0FF0Fh  ;Flash control mode reg
WSGR                .set 0FFFFh  ;Wait-state generator reg
```

```
;***********************************************
;Other useful definitions below (not addresses) *
;***********************************************

;Data page definitions for LDP instruction
DP_PF1              .set 224    ;sys regs,WD,SPI,SCI,(0x7000 - 0x707F)
DP_PF2              .set 225    ;ADC, GPIO (0x7080 - 0x70FF)
DP_CAN1             .set 226    ;CAN control regs (0x7100 - 0x717F)
DP_CAN2             .set 228    ;CAN mailboxes 1-5 (0x7200 - 0x727F)
DP_EVA              .set 232    ;Event manager A (0x7400 - 0x747F)
DP_EVB              .set 234    ;Event manager A (0x7500 - 0x757F)

;Bit codes for test bit instruction (BIT)
BIT15               .set 0000h  ;Bit code for bit 0
BIT14               .set 0001h  ;Bit code for bit 1
BIT13               .set 0002h  ;Bit code for bit 2
BIT12               .set 0003h  ;Bit code for bit 3
BIT11               .set 0004h  ;Bit code for bit 4
BIT10               .set 0005h  ;Bit code for bit 5
BIT9                .set 0006h  ;Bit code for bit 6
BIT8                .set 0007h  ;Bit code for bit 7
BIT7                .set 0008h  ;Bit code for bit 8
BIT6                .set 0009h  ;Bit code for bit 9
BIT5                .set 000Ah  ;Bit code for bit 10
BIT4                .set 000Bh  ;Bit code for bit 11
BIT3                .set 000Ch  ;Bit code for bit 12
BIT2                .set 000Dh  ;Bit code for bit 13
BIT1                .set 000Eh  ;Bit code for bit 14
BIT0                .set 000Fh  ;Bit code for bit 15
```

This page is intentionally blank.