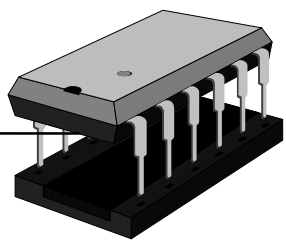
16-Bit-DSP-Microcontroller Texas Instruments TMS320LF2407

Module : Examples EVMLF2407-Kit

Frank Bormann
University of Applied Sciences Zwickau (FH)
Dr.-Friedrichs-Ring 2a
08056 Zwickau , Germany

Tel: +49 375 536 1427
email: Frank.Bormann@fh-zwickau.de

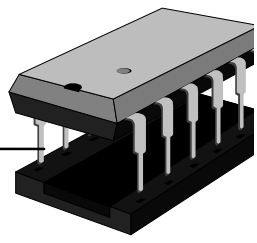


TMS320LF2407 the DSP-MicroController

Important Notice:

The examples used during this presentation are for educational purposes only. Both the author and the university do not take over any responsibility for the use of these examples in commercial products or applications. No part of this document may be reproduced in any form for any purpose without a written permission of

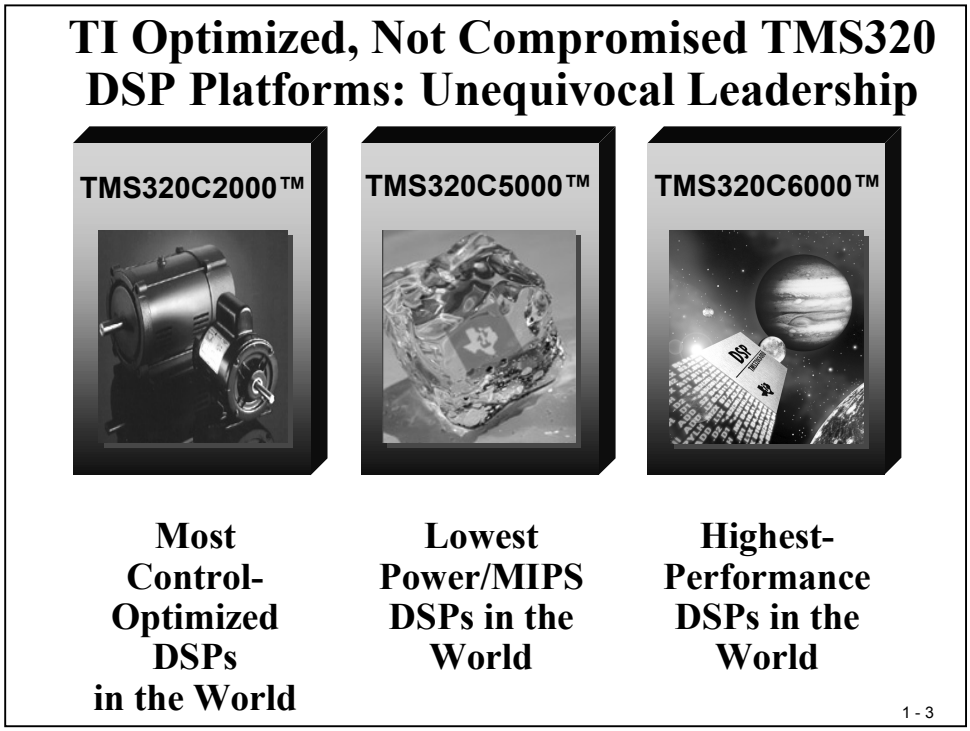
University of Applied Sciences Zwickau

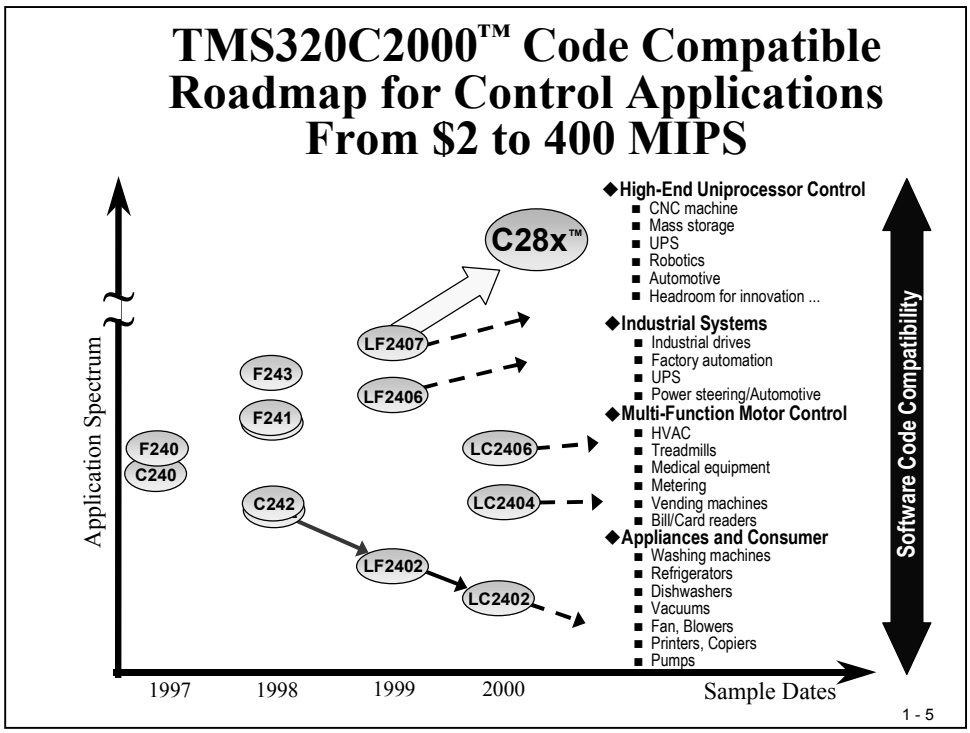


LF2407- Introduction

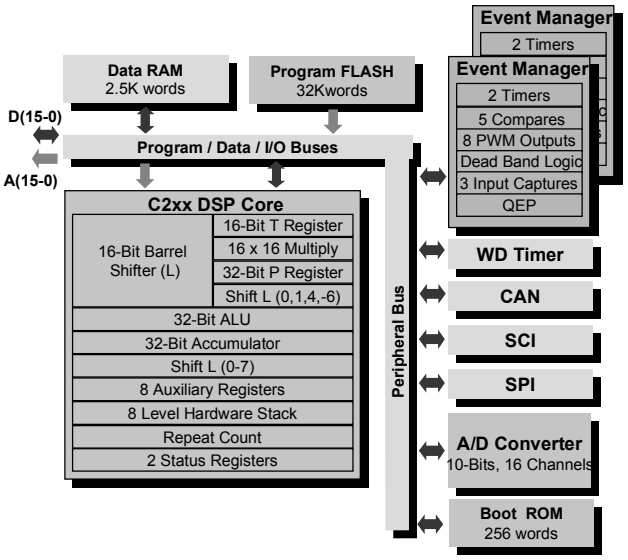
Introduction

This module describes the basic structure of the LF2407-Family.





TMS320LF2407 Architecture



- 0.35um, 3.3V
- 33ns instruction cycle time 30 MIPS (40MIPS option)
- 32K words of on-chip FLASH memory (30 MIPS)
- Flash sectoring and security
- Boot ROM 256 words
- Motor control optimized Event Manager (2x C242 like)
- A/D converter (16 inputs) with 500ns conversion time
- UART (SCI)
- SPI
- On-chip CAN module 2.0B
- 16-bit external memory interface
- up to 37 GPIO Pins
- 144 TQFP

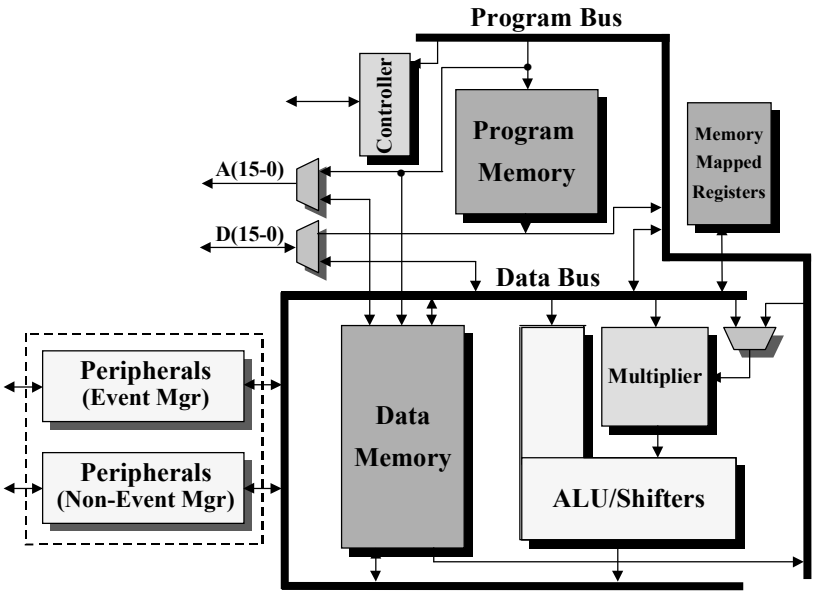
1 - 8

TMS320C24x: The Industry's Broadest DSP Motor Controller Portfolio

	'LF2407	**'LF2406**	**'LF2402**	**'LC2406**	**'LC2404**	**'LC2402**	**'F240**	**'C240**	**'F241**	**'C242**	**'F243**
MIPS	**30**	**30**	**30**	**30**	**30**	**30**	20	20	20	20	20
RAM (16-bit word)	**2.5K**	**2.5K**	**544**	**2.5K**	**1.5K**	**544**	544	544	544	544	544
Flash (16-bit word)	**32K**	**32K**	**8K**	—	—	—	16K	—	8K	—	8K
ROM (16-bit word)	—	—	—	**32K**	**16K**	**4K**	—	16K	—	4K	—
Boot ROM (16-bit word)	**256**	**256**	**256**	—	—	—	—	—	—	—	—
External memory I/F	**Yes**	—	—	—	—	—	Yes	Yes	—	—	Yes
Event manager	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	Yes	Yes	Yes	Yes	Yes
• GP timers	**4**	**4**	**2**	**4**	**4**	**2**	3	3	2	2	2
• CMP/PWM	**10/16**	**10/16**	**5/8**	**10/16**	**10/16**	**5/8**	9/12	9/12	5/8	5/8	5/8
• CAP/QEP	**6/4**	**6/4**	**3/2**	**6/4**	**6/4**	**3/2**	4/2	4/2	3/2	3/2	3/2
Watchdog timer	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	Yes	Yes	Yes	Yes	Yes
10-bit ADC	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	Yes	Yes	Yes	Yes	Yes
• Channels	**16**	**16**	**8**	**16**	**16**	**8**	16	16	8	8	8
• Conv. time (min)	**500 ns**	**500 ns**	**500 ns**	**500 ns**	**500 ns**	**500 ns**	6.6 µs	6.6 µs	850 ns	850 ns	850 ns
SPI	**Yes**	**Yes**	—	**Yes**	**Yes**	—	Yes	Yes	Yes	—	Yes
SCI	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	Yes	Yes	Yes	Yes	Yes
CAN	**Yes**	**Yes**	—	**Yes**	—	—	—	—	Yes	—	Yes
Digital I/O pins	**37**	**37**	**21**	**37**	**37**	**21**	28	28	26	26	32
Voltage range	**3.3 V**	**3.3 V**	**3.3 V**	**3.3 V**	**3.3 V**	**3.3 V**	5 V	5 V	5 V	5 V	5 V
Packaging	**144 TQFP**	**100 TQFP**	**64 PQFP**	**100 TQFP**	**100 TQFP**	**64 PQFP**	132 PQFP	132 PQFP	68 PLCC 64 PQFP	68 PLCC 64 PQFP	144 TQFP

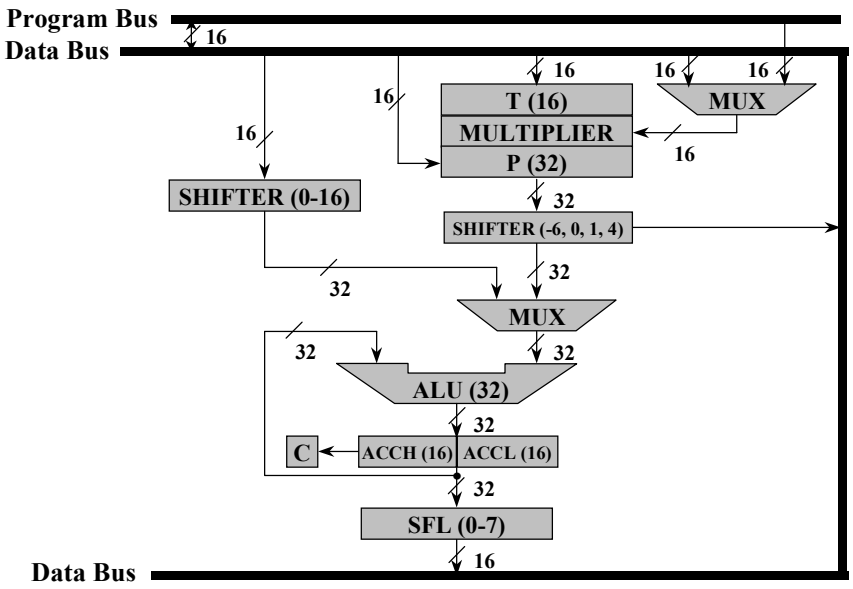
1 - 9

TMS320C240x Block Diagram



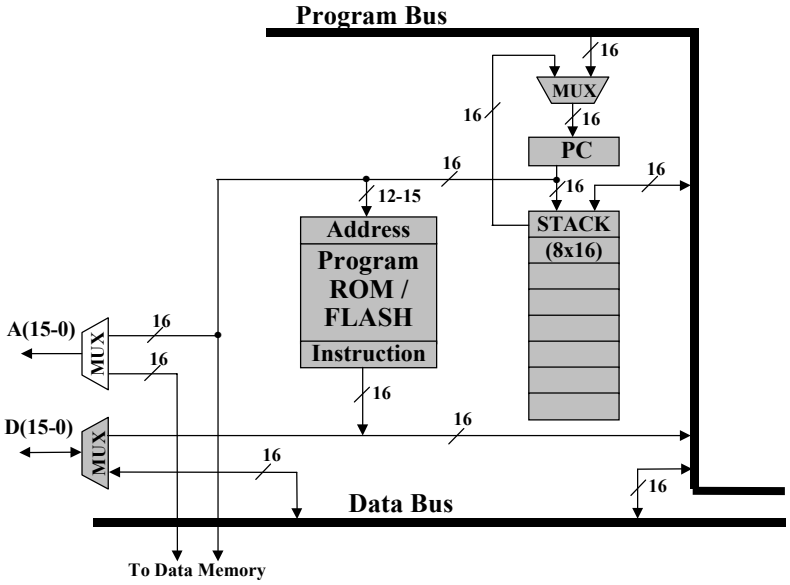
DSP24 1 - 3

Multiplier and ALU / Shifters



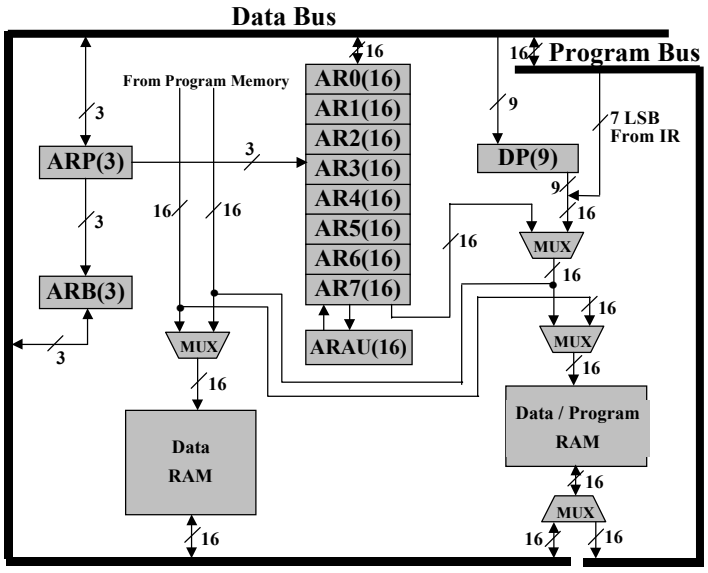
1 - 16

Program Memory



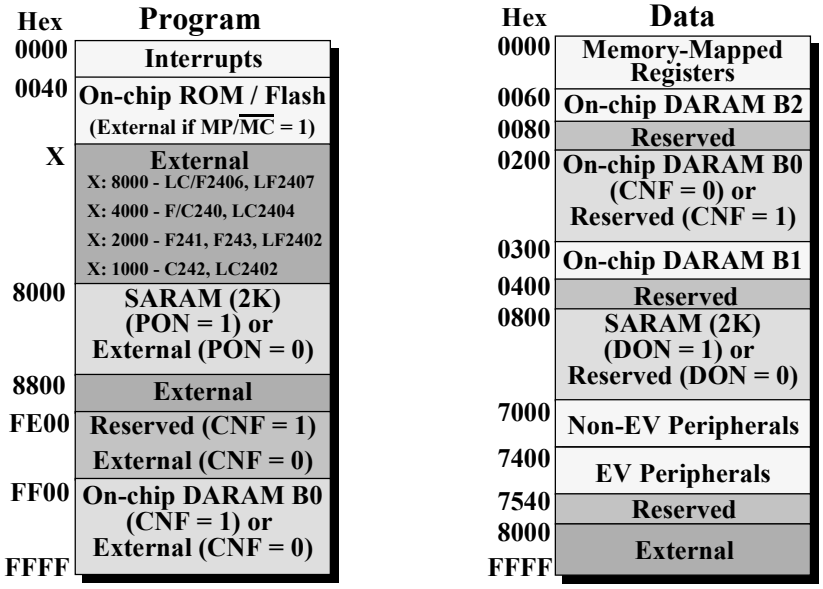
1 - 17

Data Memory



1 - 18

TMS320C240x Memory Map



DSP24 1 - 7

Pipeline Process

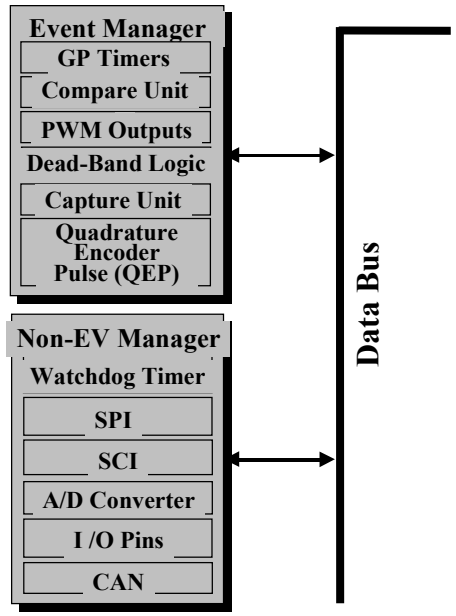
	Cycle						
	100	101	102	103	104	105	106
Add	F_1	D_1	R_1	E_1			
Sub		F_2	D_2	R_2	E_2		
Mpy			F_3	D_3	R_3	E_3	
Store				F_4	D_4	R_4	E_4

↑

Fully loaded pipeline (normal operation)

1 - 20

Peripherals



1 - 21

TMS320C240x Instruction Set

Accumulator Memory Reference

ABS	NEG	SUB
ADD	NORM	SUBB
ADDC	OR	SUBC
ADDS	ROL	SUBS
ADDT	ROR	SUBT
AND	SACH	XOR
CMPL	SACL	ZALR
LACC	SFL	
LACL	SFR	
LACT		

Auxiliary Register and Data Page Pointer

ADRK	MAR
CMPR	SAR
LAR	SBRK
LDP	

Multiply, T, P

APAC	MPYA
LPH	MPYS
LT	MPYU
LTA	PAC
LTD	SPAC
LTP	SPH
LTS	SPL
MAC	SPM
MACD	SQRA
MPY	SQRS

Control Instructions

BIT	POP
BITT	POPD
CLRC	PSHD
IDLE	PUSH
RPT	SETC
LST	SST
NOP	

I/O and Data Memory Operations

BLDD	OUT
BLPD	SPLK
DMOV	TBLR
IN	TBLW

Branch

B	CC
BACC	INTR
BANZ	NMI
BCND	RET
CALA	RETC
CALL	TRAP

DSP24 1 - 11

2 Program Development Tools

Introduction

The goal of this module is to understand the basic functions of the Code Composer V4.12 Integrated Design Environment for the LF2407-Family of Texas Instruments Digital Signal Processors. This involves understanding the basic structure of a project for C and Assembler - coded source files, along with the basic operation of the C-Compiler, Assembler and Linker

Code Composer IDE

The Code Composer is the Environment for Project Development and for all Tools needed to Build an Application for the C2000-Family

Code Composer IDE

- **Code Composer IDE key features for Motion Control applications** :

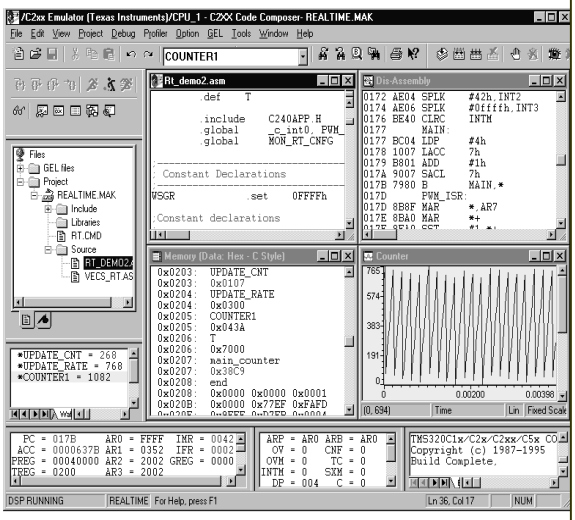
✓ ***Industries first IDE with DSP specific functionality.***

✓ Familiar MS-Visual C++ like environment.

✓ Advanced graphical signal analysis and a leading edge GUI with multiple watch windows.

✓ Edit, build, debug, profile and manage projects from a single unified environment.

✓ RTM(v 4.x) allows code editing and debug without stopping target application.

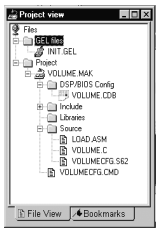


2 - 2

Code Composer IDE

Manage Projects Visually

- Intuitive organization
- Automatically tracks file dependencies

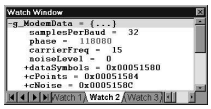


Edit within the IDE

- Edit C and assembly source code together
- Find and replace, and quick search
- Context-sensitive help
- Custom key commands
- Bookmarks
- Column editing

Compile Efficiently

- Integrated code generation tools
- Graphically configure build options
  - Saved with each project
- Background build
- Save time by programming in C



Debug within the IDE

- Debugger is optimized for DSP & MCU
- C and assembly debugging
- Multi-processor debugging
- Multi-board debugging for same processor family

2 - 3

Code Composer v4.1x for ‘C24x

- Includes
  - ✓ Code Composer IDE v 4.1x
  - ✓ Code Generation Tools v 7.00
  - ✓ C24x XDS-510 real-time driver
  - ✓ C2xx XDS-510 driver
  - ✓ C2xx Simulator
  - ✓ Command window plug-in
- Price
  - ✓ Now $1495
  - ✓ Reduced from v3.0x

Note: CC v 4.1x XDS00510PP driver must be acquired separately from Spectrum Digital, Inc.

2 - 4

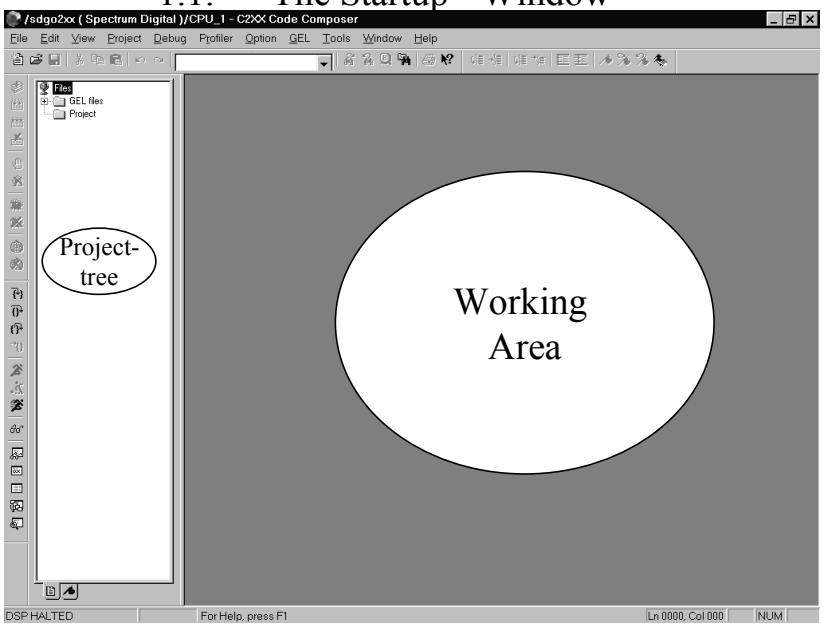
Code Composer - Step by Step

1. Code Composer - The Basics

- 1.1. The Startup - Window
- 1.2. Create a C-code - project
- 1.3. Debug your program
- 1.4. Watch your variables
- 1.5. Perform a Single Step Debug
- 1.6. Adding a Breakpoint
- 1.7. Set a Probe Point
- 1.8. Other View Commands
- 1.9. GEL - General Extension Language

1. Code Composer - The Basics

1.1. The Startup - Window



1.2. Create a C-code - project

Project ==> New

give your project a name , e.g. 'first.mak'

the ASCII-File *.mak stores all setups and options of the project

- **Write a C-Source Code**
  - **File ==> New ==> Source File**

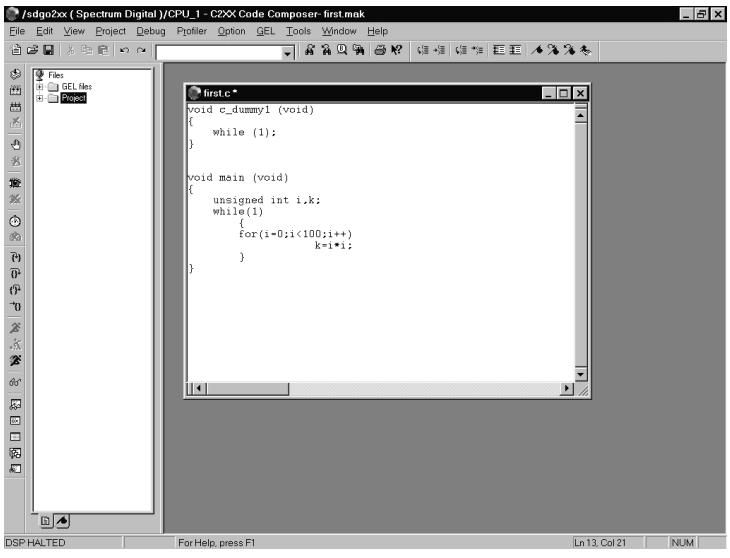
```
void main (void)
{
   unsigned int i,k;
   while(1)
   {
   for (i=0;i<100;i++)
            k=i*i;
   }
}
```

```
void c_dummy1 (void)
{
    while(1);
}
```

- **File ==> Save as : "first.c"**

2 - 7

1.2. Create a C-code - project (cont.)



2 - 8

1.2. Create a C-code-project (cont.)

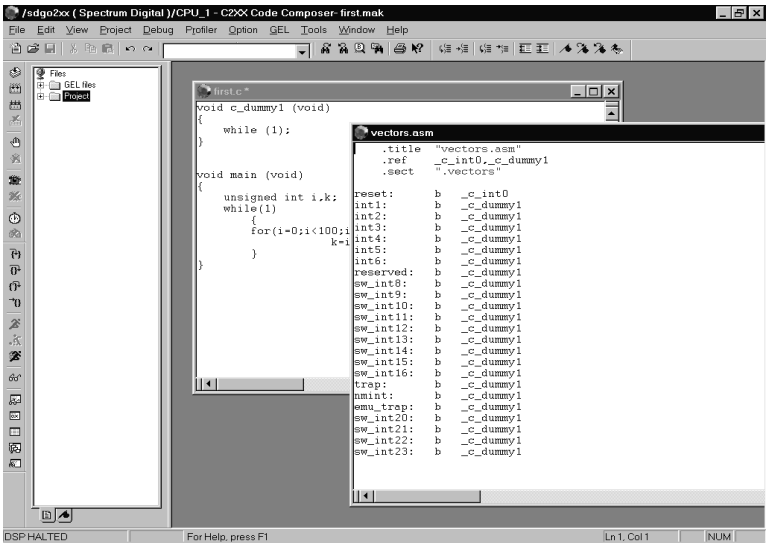
File ==> New ==> Source File

```
    .title    "vectors.asm"
    .ref      _c_int0,_c_dummy1
    .sect     ".vectors"

reset:    b     _c_int0
int1:     b     _c_dummy1
int2:     b     _c_dummy1
int3:     b     _c_dummy1
int4:     b     _c_dummy1
int5:     b     _c_dummy1
int6:     b     _c_dummy1
reserved: b     _c_dummy1
sw_int8:  b     _c_dummy1
sw_int9:  b     _c_dummy1
sw_int10: b     _c_dummy1
sw_int11: b     _c_dummy1
sw_int12: b     _c_dummy1
sw_int13: b     _c_dummy1
sw_int14: b     _c_dummy1
sw_int15: b     _c_dummy1
sw_int16: b     _c_dummy1
trap:     b     _c_dummy1
nmint:    b     _c_dummy1
emu_trap: b     _c_dummy1
sw_int20: b     _c_dummy1
sw_int21: b     _c_dummy1
sw_int22: b     _c_dummy1
sw_int23: b     _c_dummy1
```

• File ==> Save as : „vectors.asm“

2 - 9

1.2. Create a C-code-project (cont.)



2 - 10

1.2. Create a C-code-project (cont.)

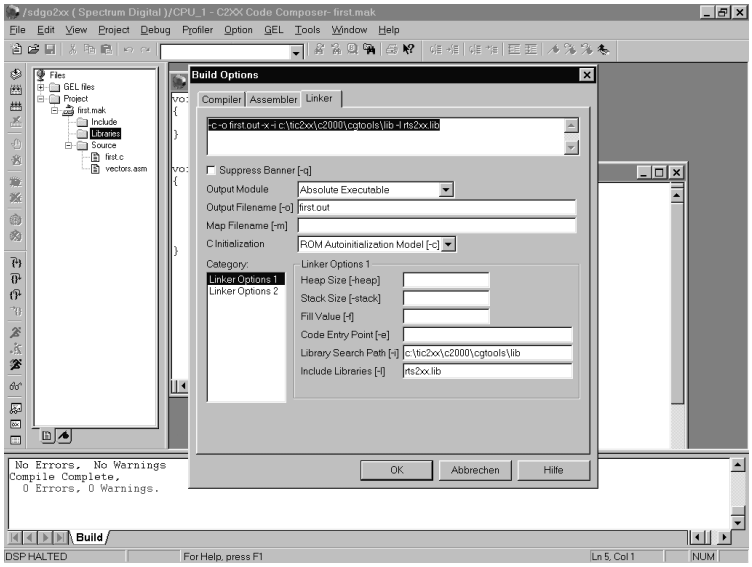
- **Add your files to your project :**
  - **Project ==> Add files to project**
  - **Add: first.c**
  - **Add: vectors.asm**

- **Compile / Assemble your two files :**
  - **Project ==> Compile File ==> active window will be compiled / assembled**
  - **The Code Composer distinguishes between C and ASM - sources**
  - **in case of syntax errors : modify your source code**

- **Add the C-runtime-library to your project :**
  - **Project ==> Options ==> Linker ==> Library Search Path :**
    c:\tic2xx\c2000\cgtools\lib
  - **Project ==> Options ==> Linker ==> Include Libraries :**
    rts2xx.lib

2 - 11

1.2. Create a C-code-project (cont.)



2 - 12

1.2. Create a C-code-project (cont.)

- **Prepare the Linker Command File :**

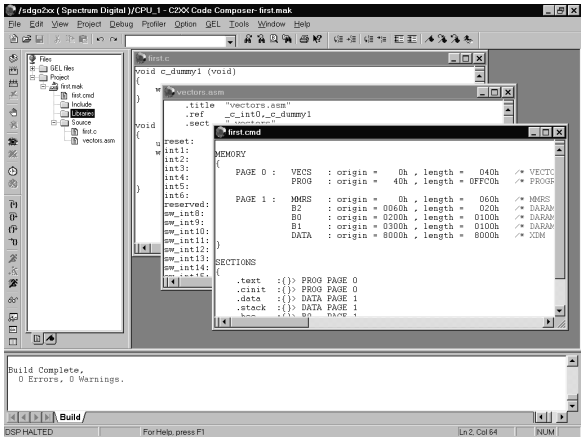
```
MEMORY
{
        PAGE 0 :    VECS   : origin =   0h , length =  040h     /* VECTORS */
                    PROG   : origin =  40h , length = 0FFC0h /* PROGRAM */

        PAGE 1 :    MMRS   : origin =   0h , length =  060h     /* MMRS    */
                    B2     :      origin = 0060h , length =  020h   /* DARAM   */
                    B0     :      origin = 0200h , length = 0100h   /* DARAM   */
                    B1     :      origin = 0300h , length = 0100h   /* DARAM   */
                    DATA   :   origin = 8000h , length = 8000h   /* XDM     */
}

SECTIONS
{
    .text           :{}> PROG    PAGE 0
    .cinit          :{}> PROG    PAGE 0
    .data           :{}> DATA    PAGE 1
    .stack          :{}> DATA    PAGE 1
    .bss            :{}> B0      PAGE 1
    .vectors        :{}> VECS    PAGE 0
}
```

• create a new file, save it as first.cmd and add it to your project

2 - 13

1.2. Create a C-code-project (cont.)

- **Build the whole project (compile / assemble / link) :**
  - **Project ==> build**



2 - 14

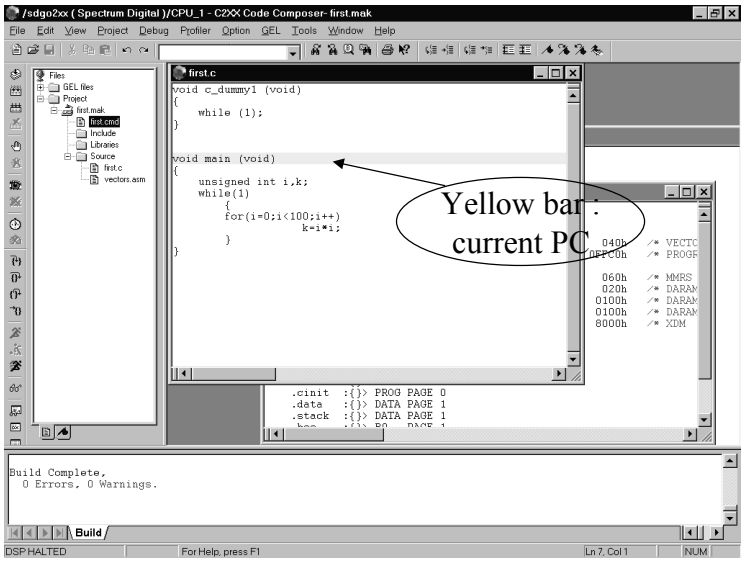
1.2. Create a C-code-project (cont.)

- **Load the binary code into the DSP :**
  - **File ==> Load Program : first.out**

- **Run the program until label : Main**
  - **Debug ==> Go main**

2 - 15

1.2. Create a C-code-project (cont.)



2 - 16

1.3. Debug your program

- **Perform a real time run :**
  ==> Debug ==> Run

  note : bottom left corner marked : "DSP Running".
  You'll see no action on the peripherals because the program does not use it so far

  note : the yellow bar is no longer visible !

- **Stop the real time run :**
  ==> Debug ==> Halt

- **Reset the DSP :**
  ==>Debug ==> Reset DSP

- **Run again until main :**
  ==> Debug ==> Go Main

2 - 17

1.4. Watch your variables

- **Open the Watch Window :**
  ==> View ==> Watch Window

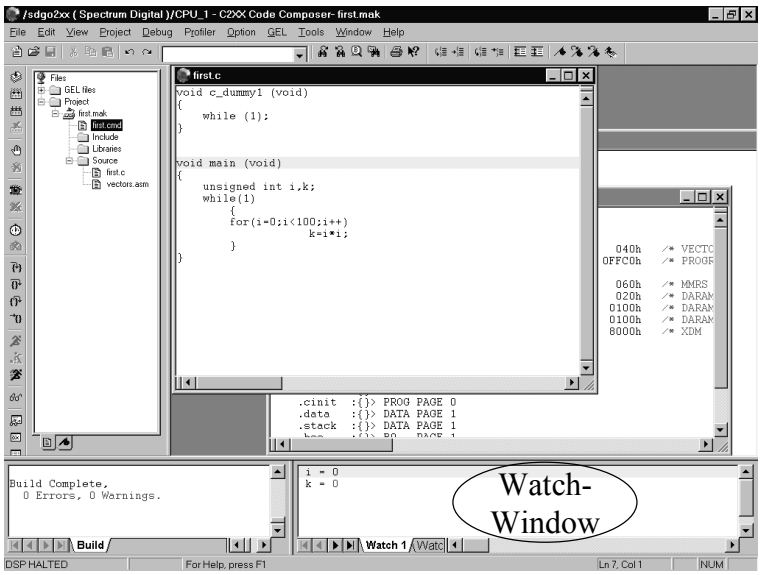
- **Add your variables to the Watch Window :**
  - **inside the Watch Window click right mouse button and select**
    - » **"Insert New Expression"**
    - » **type in the Expression window : i**
  - **do it again for variable k**

  - **note : the input 'i,x' will display the content of i in a hexadecimal manner**

  - **note : the input 'i,x ; revolution' will display the text following the semicolon as comment in the watch window**

2 - 18

1.4. Watch your variables



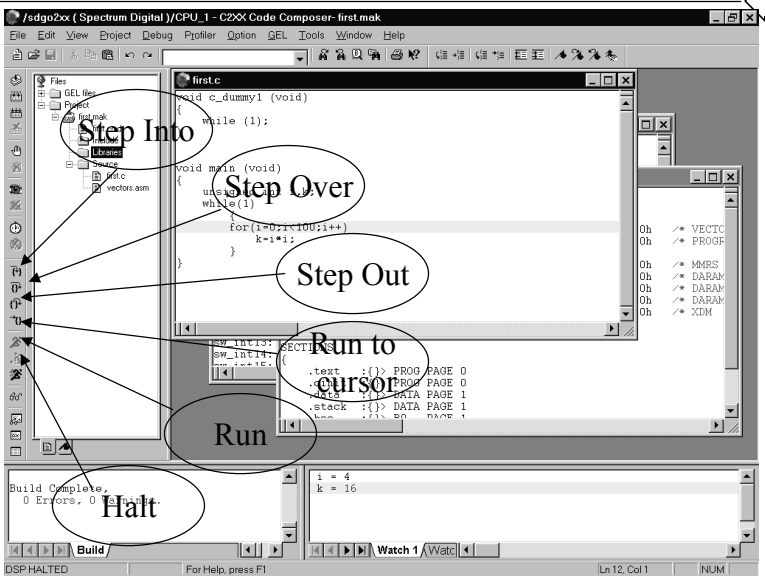
2 - 19

1.5. Perform a Single Step Debug

- **Go until main :**
  ==> Debug ==> Go Main
- **Perform a single step trough the program :**
  ==> Debug ==> Step Into (or Key F8)
- **Watch the numerical values of variable i and k in the Watch Window**

2 - 20

1.5. Perform a Single Step Debug



2 - 21

1.6. Adding a Breakpoint

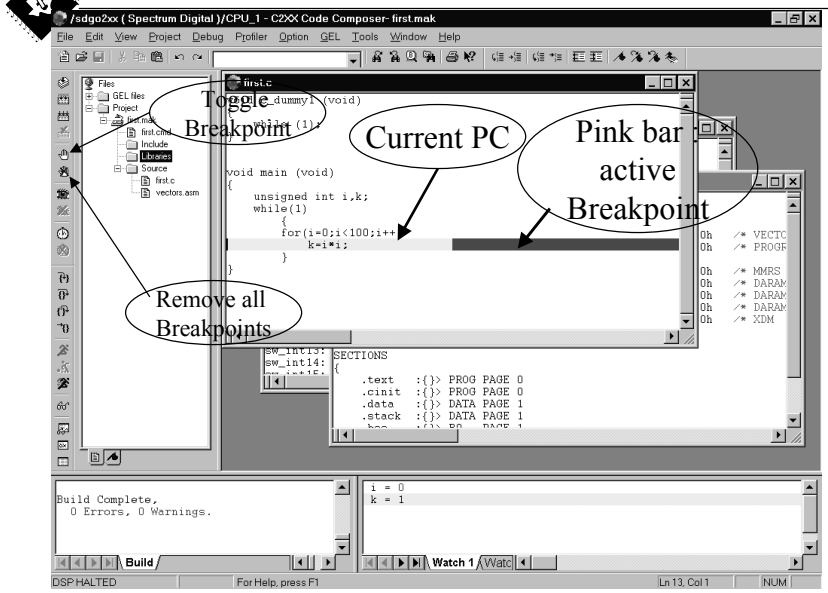
- **Set a Breakpoint :**
  - **Place the Cursor in first.c on line : k=i*i;**
  - **Click right mouse and select 'Toggle Breakpoint'**
  - **the line is highlighted in pink to mark an active breakpoint**

  Note : most Code Composer Commands are also available through buttons or trough Command -Keys (see manual, or help)

- **Reset the Program**
  ==> Debug ==> Reset DSP

- **Perform a real time run**
  ==> Debug ==> Run (or Key F5)
- **a yellow - pink bar marks that the DSP has reached an active Breakpoint**
- **repeat Run and watch your variables**

2 - 22



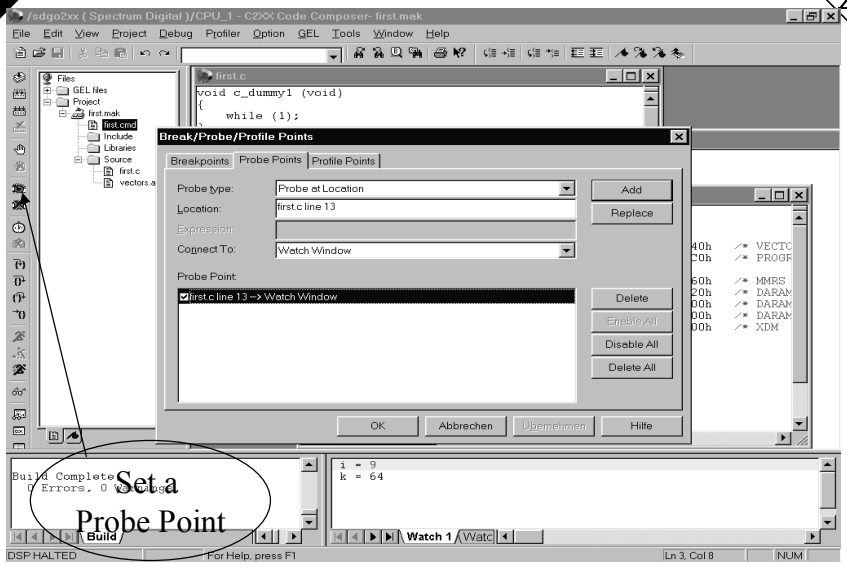
2 - 23

1.7. Set a Probe Point

- **causes an update of a particular window at a specific point in your program.**
- **When a window is created it is updated at every breakpoint. However, you can change this so the window is updated only when the program reaches the connected Probe Point. When the window is updated, execution of the program is continued.**
- **To set a Probe - Point :**
  - **Click right mouse on the line 'k = i*i;' in the program first.c**
  - **select : 'Toggle Probe Point ' (indicated by a blue bar)**
  - **select ==> Debug ==> Probe Points...**
  - **In the Probe Point Window click on the line 'first.c line 13 -> no Connection'**
  - **in the 'Connect to' - selector select 'Watch Window'**
  - **exit this dialog with the 'Replace' and 'OK' - Button**
- **Run the program , you'll see the watch window updating**

2 - 24

1.7. Set a Probe Point (cont.)



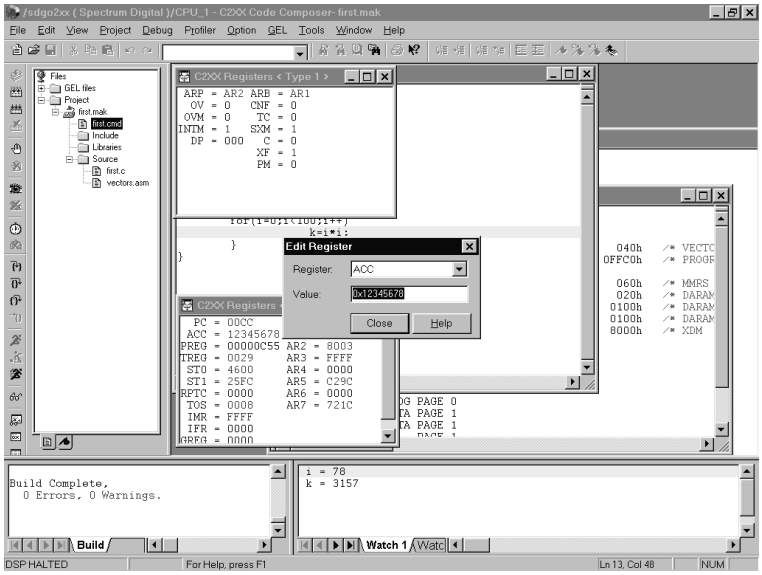
2 - 25

1.8. Other View Commands

- **The View menu includes more useful windows to monitor and control the DSP**
- **==> View ==> CPU Registers ==> CPU Register**
- **==> View ==> CPU Registers ==> Status Register**
  - **click right mouse inside these windows and select 'Float in Main Window'**
  - **click right mouse on line 'ACC' and select 'Edit Register' ;**
  - **modify the value inside the Accumulator ACC**

2 - 26

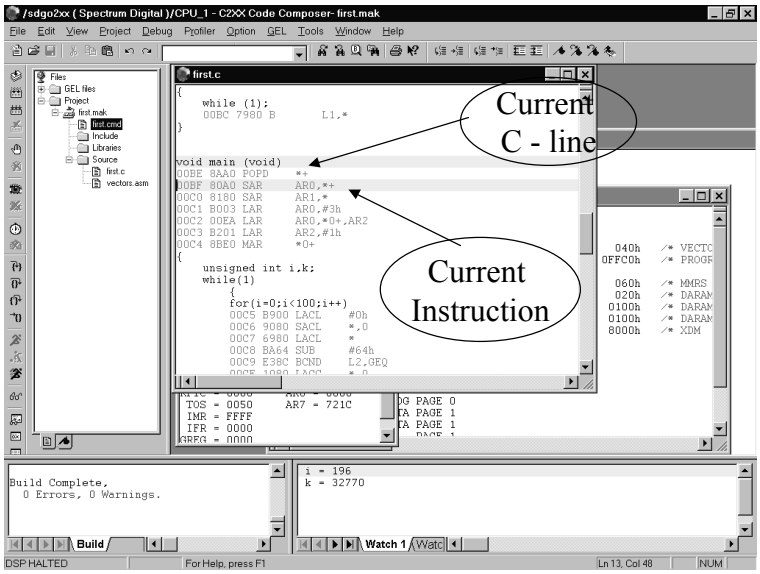
1.8. Modify a Register



1.8. Other View Commands (cont.)

- **To view both the Assembler code and the C Source Code :**
- **==> View ==> Mixed Source /ASM**
- **The Instruction Code generated by the Compiler is added and printed in grey colour**
- **Single Step ('Step Into') is now possible on instruction level**
  - **Perform : ==> Debug ==> Reset DSP**
    ==> Debug ==> Go Main
    ==> Debug ==> Step Into (F8)
  - **You'll see two yellow bars , one on C-lines, the second on instruction-lines**

1.8. View C and Disassembly



2 - 29

1.9. GEL - General Extension Language

- **language similar to C**
- **lets you create functions to extend Code Composer's features**
- **to create GEL functions use the GEL grammar**
- **load GEL-files into Code Composer**

- **With GEL, you can:**
  - **access actual/simulated target memory locations**
  - **add options to Code Composer's GEL menu**

- **GEL is useful for automated testing and user workspace customization.**

- **GEL - files are ASCII with extension *.gel**

2 - 30

Lab 1: First project

- Objective

  The objective of this lab is to practice and verify the basics of the Code Composer Integrated Design Environment.

- Procedure

  Open Files, Create Project File

  1. Using Code Composer, create a new project , called myfirst.MAK in C:\One2407\Labs\first. **NOTE** : There is a project file first.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 7.

  2. Write a new source code file by clicking : File → New → Source File. A new window in the workspace area will open. Type in this window the following tiny program :

```
void c_dummy1 (void)
{
    while(1);
}

void main (void)
{
  unsigned int i,k;
  while(1)
    {
    for (i=0;i<100;i++)
          k=i*i;
     }
}
```

  Save this file by clicking File → Save as and type in : myfirst.c

  3. Add the Source Code Files: myfirst.c and the provided files : vectors.asm and first.cmd from C:\One2407\Labs\first\ to your project by clicking : Project → Add Files to project

  4. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

  5. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

  6. Close the Build Options Menu by clicking OK

Build and Load

7. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

8. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. Note: The binary output file has the same name as the project with the extension .out, so you have to load either first.out or myfirst.out depending on your project name.

   Note: Code Composer can automatically load the output file after a successful build. To do this by default , click on the menu bar : Option → Program Load and select: "Load Program after build", then click OK.

Test

9. Reset the DSP by clicking on Debug → Reset DSP

10. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "first.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

11. Perform a real time run by clicking : Debug → Run

12. Verify the note at the bottom left corner of the screen :"DSP Running" to sign the real time run. Because we do nothing with peripherals so far, there is no other visible action.

13. Halt the program by clicking : Debug → Halt , reset the DSP (Debug → Reset DSP) and go again until main (Debug → Go main)

14. Open the Watch Window to watch your variables. Click: View → Watch Window. Add the variables i and k to this window : Click right mouse inside the Watch Window and select : "Insert New Expression". Type in the name of the variable to be shown : i . Repeat this step for variable k.

15. Perform a single stepping through your program by clicking : Debug → Step Into (or use function Key F8). Repeat F8 and watch your variables.

16. Place a Breakpoint in the first.c – window at line "k=i*i;" . Do this by placing the cursor at this line ,click right mouse and select : "Toggle Breakpoint". The line is highlighted with a pink bar to mark an active breakpoint. Perform a real time run by Debug → Run (or F5). The program will stop execution when it reaches the active breakpoint (indicated by a yellow and pink highlighted line). Remove the breakpoint after this step (click right mouse and "Toggle Breakpoint").

17. Set a Probe Point . Click right mouse on the line k=i*i; . Select “Toggle Probe Point” . The color of the line changes into blue. From the menu-bar then select . Debug → Probe Points... In the dialog window click on the line ‘first.c line 13 → No Connection’ . Change the ‘connect to’-selector to “Watch Window”, click on ‘Replace’ and ‘OK’. Run the program again (F5). You will see that the the probe point updates the watch window each time the program passes the probe point.

18. Have a look into the DSP-Registers. Perform View → CPU Registers → CPU Register and View → CPU Registers → Status Register. Click right mouse inside these windows and select ‘Float in Main Window’. Click right mouse on the line ACC in the CPU-Register and select ‘Edit Register’. Modify the value inside the Accumulator.

19. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as “first.wks”

20. Delete the active probe by clicking on the button “Remove all Probe Points”, close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab1

3 C2407- Peripherals – a first glance

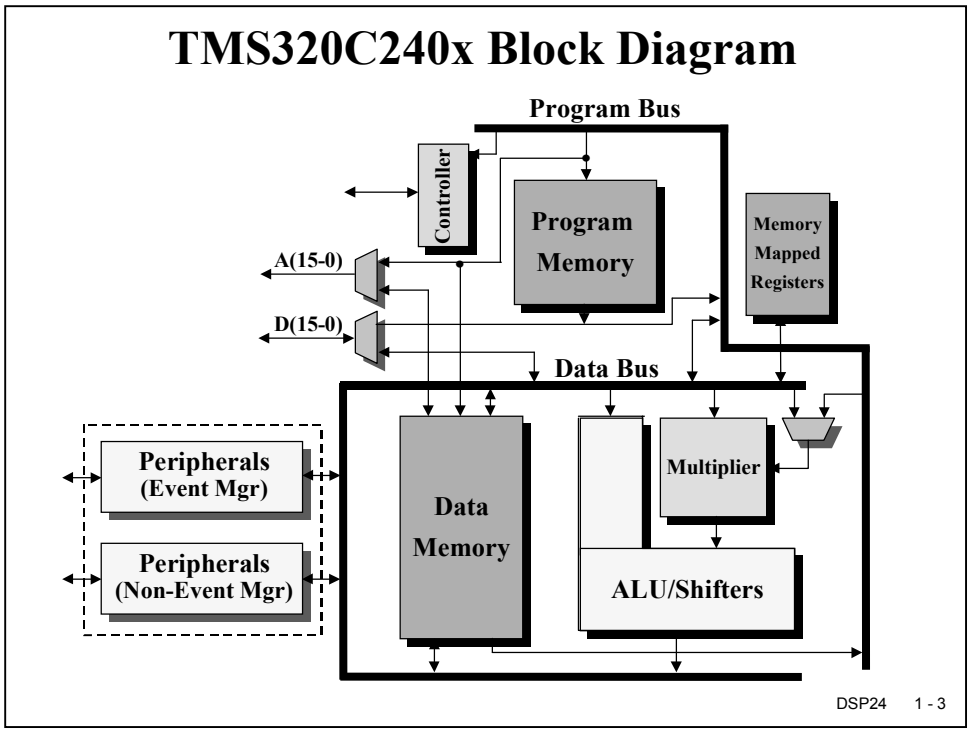
Introduction

This module discusses the operation of the memory mapped peripherals in general. The first part of these peripherals used in exercises are the general purpose I/O's. Also, the watchdog-timer and the wait state generator will be covered.

TMS320C2407 Architecture Basis

The TMS320C207 uses a *modified Harvard* architecture. These modifications consist of enhancements to the strict Harvard architecture in the form of features from the von Neumann architecture. Those features include the ability to initialize data memory from program memory, and the ability to transfer data memory to program memory. This capability allows the 'C2407 to time-division multiplex its memory between tasks as well as to initialize its data memory with constants (for example, a coefficient) stored in the system's program ROM. This minimizes system cost by eliminating the need for a data ROM and maximizes data memory utilization by allowing dynamic redefinition of data memory's function.

The most important reason for basing the 'C2407 on the Harvard architecture is speed. Separate data and program space allow simultaneous fetching of program instructions and data. In a mathematically intensive application, this effectively doubles algorithm throughput compared to (standard) von Neumann-type processors.



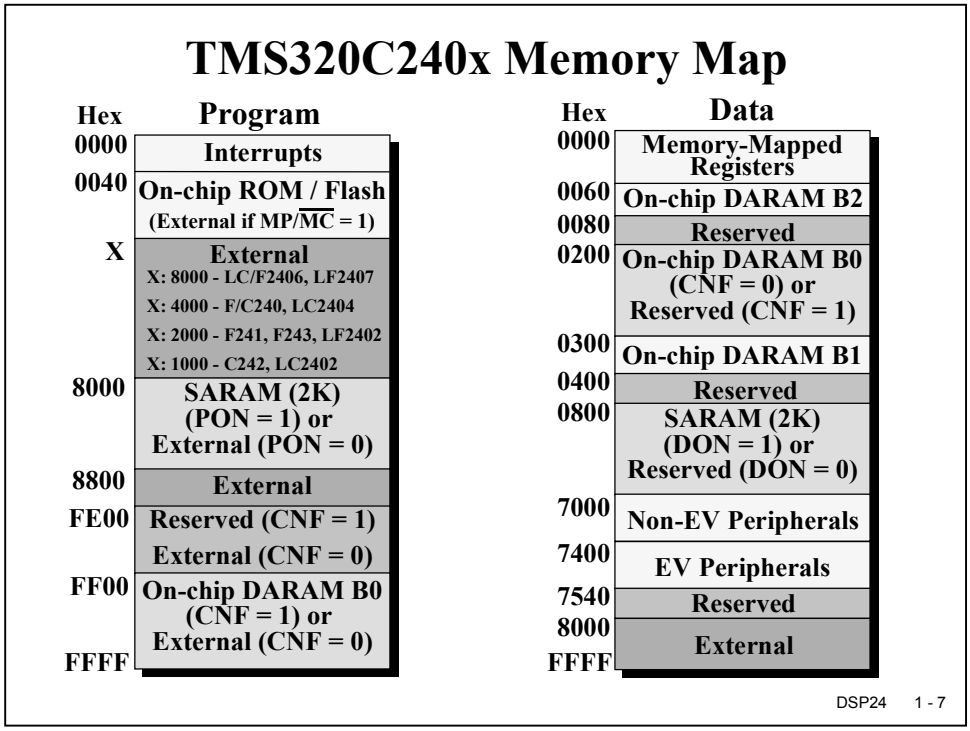
Memory Map

The 'LF2407 memory space is divided into three regions:

- 64K program
- 64K data
- 64K I/O

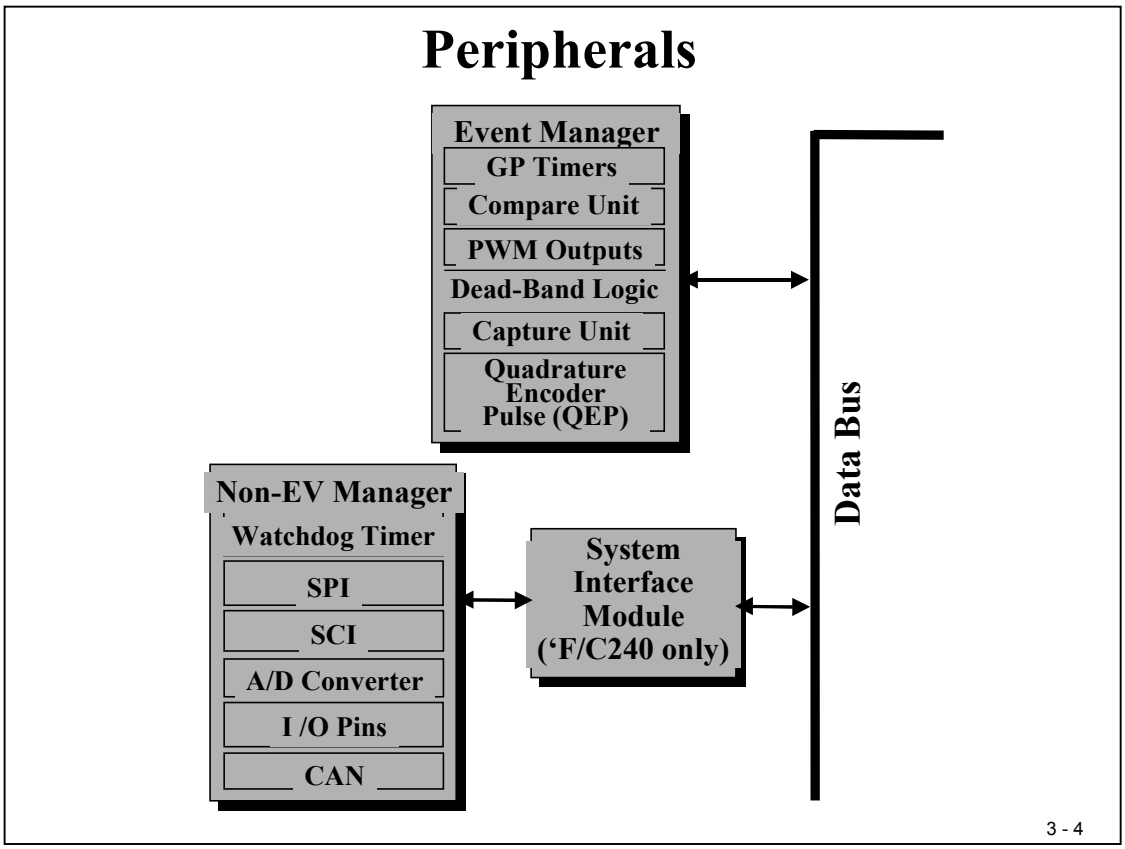
The 'LF2407 has a minimum of 544 words of on-chip RAM, which is sometimes referred to as dual-access RAM. It can implement the action of a delay line without the need for “circular buffering” as any other memory would. On reset, this memory is found in data space, but a portion of it may be relocated under software control to program space.

The second on-chip memory is ROM, which is located at the beginning of program space. It may be used or bypassed under control of the MP/ MC signal line. The amount of on-chip non-volatile memory (ROM or flash) varies based on the ‘LF2407 device.

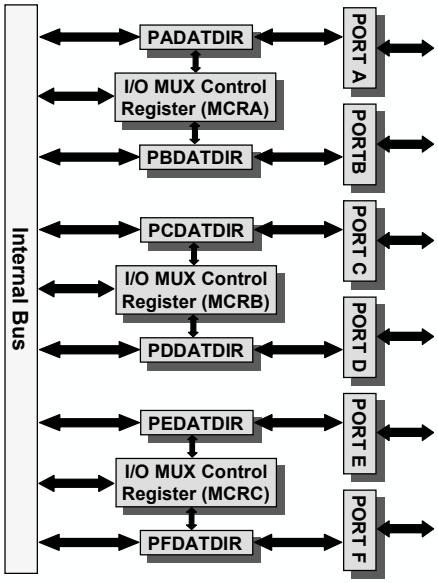


Peripherals

The ‘LF2407 devices contain peripherals optimized for motor/motion control applications. For
many systems, this devices provide a low-cost, high performance solution.



C240x Digital I/O Pin & Register Structure



PORT A
IOPA0 / SCITXD
IOPA1 / SCIRXD
IOPA2 / XINT1
IOPA3 / CAP1 / QEP1
IOPA4 / CAP2 / QEP2
IOPA5 / CAP3
IOPA6 / PWM1
IOPA7 / PWM2

PORT B
IOPB0 / PWM3
IOPB1 / PWM4
IOPB2 / PWM5
IOPB3 / PWM6
IOPB4 / T1PWM / T1CMP
IOPB5 / T2PWM / T2CMP
IOPB6 / TDIRA
IOPB7 / TCLKINA

PORT C
W/R / IOPC0
BIO / IOPC1
IOPC2 / SPISIMO
IOPC3 / SPISOMI
IOPC4 / SPICLK
IOPC5 / SPISTE
IOPC6 / CANTX
IOPC7 / CANRX

PORT D
IOPD0 / XINT2 / ADCSOC
EMU0 / --
EMU1 / --
TCK / --
TDI / --
TDO / --
TMS / --
TMS2 / --

PORT E
CLKOUT / IOPE0
IOPE1 / PWM7
IOPE2 / PWM8
IOPE3 / PWM9
IOPE4 / PWM10
IOPE5 / PWM11
IOPE6 / PWM12
IOPE7 / CAP4 / QEP3

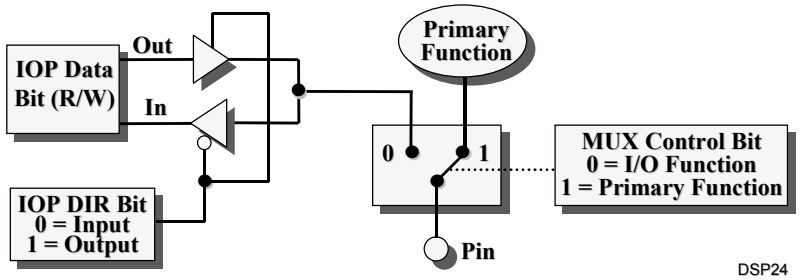
PORT F
IOPF0 / CAP5 / QEP4
IOPF1 / CAP6
IOPF2 / T3PWM / T3CMP
IOPF3 / T4PWM / T4CMP
IOPF4 / TDIRB
IOPF5 / TCLKINB
IOPF6 / IOPF6

Note: label names on left are pin functions at reset

DSP24 9 - 18

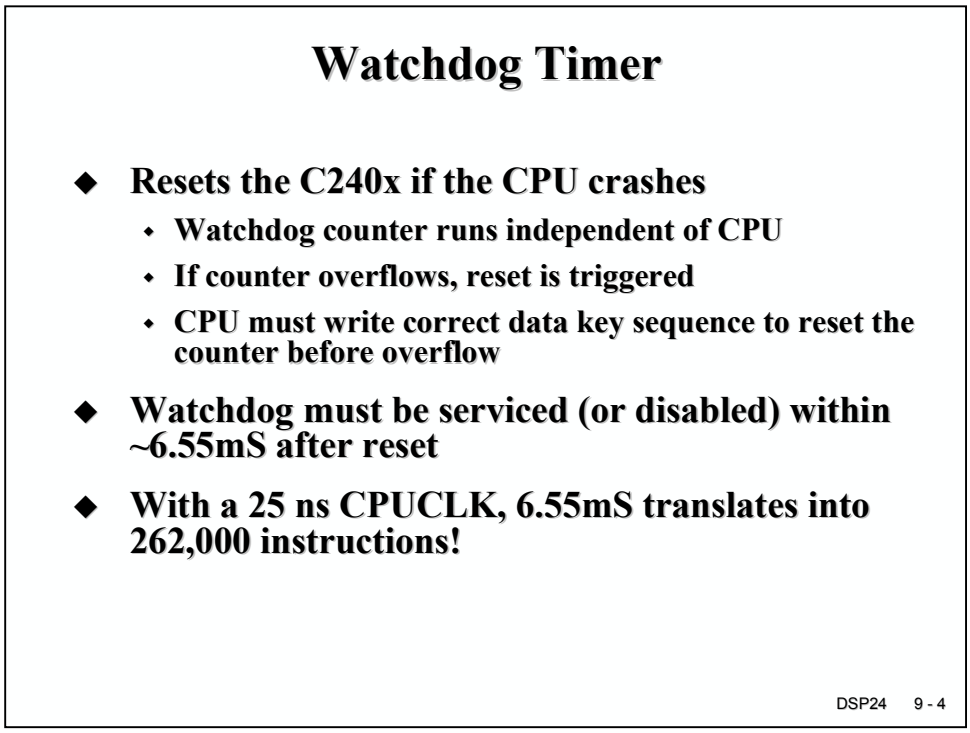
Digital I/O Port Registers

Address	Register	Name
7090h	MCRA	I/O Mux Control Register A
7092h	MCRB	I/O Mux Control Register B
7094	MCRC	I/O Mux Control Register C
7098h	PADATDIR	I/O Port A Data and Direction Register
709Ah	PBDATDIR	I/O Port B Data and Direction Register
709Ch	PCDATDIR	I/O Port C Data and Direction Register
709Eh	PDDATDIR	I/O Port D Data and Direction Register
7095h	PEDATDIR	I/O Port E Data and Direction Register
7096h	PFDATDIR	I/O Port F Data and Direction Register



DSP24 9 - 19

Watchdog Timer



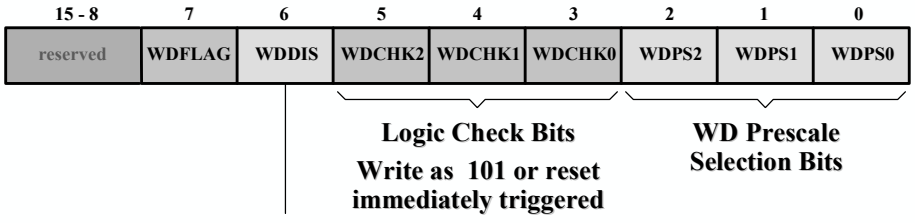
The watchdog timer provides a safeguard against CPU crashes by automatically initiating a reset if it is not serviced by the CPU at regular intervals. In motor control applications, this helps protect the motor and drive electronics when control is lost due to a CPU lockup. Any CPU reset will revert the PWM outputs to a high-impedance state, which should turn off the power converters in a properly designed system.

The watchdog timer is running immediately after system power-up/reset, and must be dealt with by software soon after. Specifically, you have 6.55 ms after any reset before a watchdog initiated reset will occur. For a 25 ns CPU clock, this translates into 262,000 instruction cycles, which is a seemingly tremendous amount! Indeed, this is plenty of time to get the watchdog configured as desired and serviced. A failure of your software to properly handle the watchdog after reset could cause an endless cycle of watchdog initiated resets to occur.

Watchdog Timer Control Register

WDCR @ 7029h

WD Flag Bit
Gets set when the WD causes a reset
- **Writing a 1 clears this bit**
- **Writing a 0 has no effect**



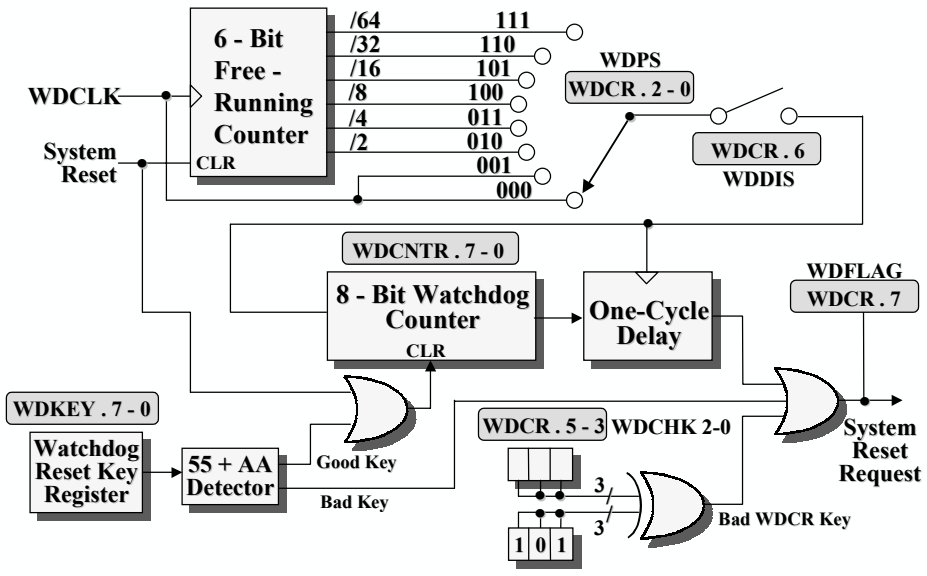
Logic Check Bits
Write as 101 or reset immediately triggered

WD Prescale Selection Bits

Watchdog Disable Bit
(Functions only if WD OVERRIDE bit in SCSR2 is equal to 1)

DSP24 9 - 7

Watchdog Timer Module



DSP24 9 - 5

System Control and Status Register 1 and 2

System Control and Status Register 1

SCSR1 @ 7018h

CLKOUT Pin Source Select
0 = CPU clock
1 = Watchdog Clock

PLL Clock Prescale

15	14	13	12	11	10	9	8
reserved	CLKSCR	LPM1	LPM0	CLK_PS2	CLK_PS1	CLK_PS0	reserved

7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	reserved	ILLADR

Module Clock Enable Bit
0 = Disabled
1 = Enabled

Illegal Address Detect Bit
- Gets set when an illegal address is accessed
- An illegal address event also triggers an NMI
- This bit it is not cleared by reset
- Writing a 1 clears this bit
- Writing a 0 has no effect

DSP24 9 - 10

System Control and Status Register 2

SCSR2 @ 7019h

WD Protect Bit
After reset - bit gives user ability to disable WDDIS in WDCR
0 = protects WD from being disabled by s/w
1 = allows WD to be disabled using WDDIS bit in WDCR

Boot Enable
0 = Enable Boot ROM
1 = Disable Boot ROM

Microprocessor / Microcontroller Select

15 - 6	5	4	3	2	1	0
reserved	WD OVERRIDE	XMIF_HI-Z	$\overline{\text{BOOT\_EN}}$	MP / $\overline{\text{MC}}$	DON	PON

XMIF_Hi-Z Control
0 = normal
1 = Hi-Z state

SARAM Program / Data Space Select

DSP24 9 - 11

Lab 2: Digital Output on Port E (IOPE0..E7)

Aim :

- Use the 8 LED‘s connected to outputs E0-E7 to show a ‚running light‘ going from left to right and reverse
- Use a software delay loop to generate the pause interval

Project - Files :

- LED2407C.c - C - source file
- vectors.asm - jump table to Interrupt Service Routines
- regs2407.h - pointer definition to peripherals
- LED2407C.cmd - linker command file
- rts2xx.lib - Runtime Library

Project -Directory:

- all files are included in : \Labs\lab2
- the source code files are printed in the student guide

1 - 13

Lab 2: Digital Output on Port E (IOPE0..E7)

Registers involved in LAB 2 :

- **Initialize Registers:**
  - Watchdog - Timer - Control : WDCR
  - Waitstate Generator : WSGR
  - System Control & Status 1 : SCSR1
  - Multiplex Control Register A : MCRA
  - Multiplex Control Register B : MCRB
  - Multiplex Control Register C : MCRC

- **Access to GPIO - Port E**
  - Port D Data & Direction : PEDATDIR
    - MS-Byte : direction (1 = out | 0 = in)
    - LS -Byte : data

1 - 14

Lab 2: Digital Output

- **Objective**

  The objective of this lab is to practice and verify the behaviour of digital output. In this process we make use of the general purpose I/O-Port E, which is memory mapped into the data memory. Additionally, we again exercise how to run and observe the operation of code using Code Composer. In this lab, we will initialize only those registers, that are important for this exercise. These registers are :

MCRA	- Multiplex Control Register A
MCRB	- Multiplex Control Register B
MCRC	- Multiplex Control Register C
PEDATDIR	- Port D Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1

  The aim of this exercise is to produce an endless loop of outputs to the PEDATDIR - Register. There are 8 LED's connected to E7 to E0. A LED is switched on by putting the equivalent data-bit to 1, it is switched off by clearing the bit. The upper byte of PEDATDIR controls the direction of Port E, in this exercise all 8 pins of Port E are used as outputs, therefore the upper byte has to be 0xFF. The aim of the endless loop is to produce a 'Knight Rider' light , running from left to right and reverse.

- **Procedure**

  ## Open Files, Create Project File

  1. Using Code Composer, create a new project , called myLab2.MAK in C:\One2407\Labs\Lab2. NOTE : There is a project file Lab2.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

  2. Add the Source Code Files: LED2407C.c and vectors.asm as well as the Linker Command File: LED2407C.cmd from C:\One2407\Labs\Lab2\ to your project by clicking : Project → Add Files to project

  3. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

  4. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

  5. Close the Build Options Menu by clicking OK

Build and Load

6. Click the “Rebuild All” button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

7. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. Note: The binary output file has the same name as the project with the extension .out, so you have to load either lab2.out or mylab2.out depending on your project name.

   Note: Code Composer can automatically load the output file after a successful build. To do this by default , click on the menu bar : Option → Program Load and select: “Load Program after build”, then click OK.

Test

8. Reset the DSP by clicking on Debug → Reset DSP

9. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code “LED2407.c” is highlighted and that the yellow bar for the current Program Counter is placed on the line ‘void main(void)’.

10. Perform a real time run by clicking : Debug → Run

11. Verify the program running on the external LED’s as supposed. Also note that the bottom left corner of the screen is marked “DSP Running” to sign the real time run.

12. Halt the program by clicking : Debug → Halt , reset the DSP (Debug → Reset DSP) and go again until main (Debug → Go main)

13. Place a Breakpoint in the LED2407.c – window at line “wait();” . Do this by placing the cursor at this line ,click right mouse and select : “Toggle Breakpoint”. The line is highlighted with a pink bar to mark an active breakpoint.

14. Perform a real time run by clicking : Debug → Run or Key F5. Verify that the program halts when it reaches the Breakpoint line. Watch the LED’s on Port E. Continue the program execution by clicking : Debug → Run or Key F5 and watch the next cycle on the LED’s.

15. Open the Watch-Window by clicking : View → Watch Window. Add the Register PEDATDIR to the watch window by clicking right mouse inside the watch window and select : “ Insert New Expression” . Type in the Expression Window : *(int *)0x7095@data,x ; PEDATDIR

Note: Adding Variables and Registers to the watch window can be done automatically by using the General Extension Language (GEL) and adding a dedicated GEL-file to the project environment. For further details see the Code-Composer Manual.

16. Perform again a real time run by clicking : Debug → Run or Key F5. Make sure that the breakpoint (step 13) is still active. Each time the program execution reaches the breakpoint both the LED's and the Watch window are updated.

17. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab2.wks"

18. Delete the active breakpoint by clicking on the button "Remove all Breakpoints", close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab2

Source Code Files Lab Exercise 2

LED2407C.c

```
/*********************************************************************/
/* Testprogram for digital I/O on Port E                             */
/* running on TMS320LF2407 EVM                                       */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/* date :  17.08.2000                                                */
/*********************************************************************/
/* digital I/O on Port E0...E7                                       */
/* 8 LED's connected to Port E0...E7 ; LED-on : 1  LED off : 0       */
/* "Knight-rider" :1 of 8 LED switched on, then shift light one LED etc */
/* Time delay 0.1 sec generated by Software-delay-loop               */
/* program-name :  LED2407C.c    / project : LAB2                    */
/* NEW :  SCSR1 - clock enable EVA , PLL, ILLADR                     */
/* NEW :  MCRC  - for multiplex Port E,F                             */
/*********************************************************************/

#include "regs2407.h"

/*************    SETUP for the MCRA - Register   **************/
#define MCRA15          0       /* 0 : IOPB7       1 : TCLKIN      */
#define MCRA14          0       /* 0 : IOPB6       1 : TDIR        */
#define MCRA13          0       /* 0 : IOPB5       1 : T2PWM       */
#define MCRA12          0       /* 0 : IOPB4       1 : T1PWM       */
#define MCRA11          0       /* 0 : IOPB3       1 : PWM6        */
#define MCRA10          0       /* 0 : IOPB2       1 : PWM5        */
#define MCRA9           0       /* 0 : IOPB1       1 : PWM4        */
#define MCRA8           0       /* 0 : IOPB0       1 : PWM3        */
#define MCRA7           0       /* 0 : IOPA7       1 : PWM2        */
#define MCRA6           0       /* 0 : IOPA6       1 : PWM1        */
#define MCRA5           0       /* 0 : IOPA5       1 : CAP3        */
#define MCRA4           0       /* 0 : IOPA4       1 : CAP2/QEP2   */
#define MCRA3           0       /* 0 : IOPA3       1 : CAP1/QEP1   */
#define MCRA2           0       /* 0 : IOPA2       1 : XINT1       */
#define MCRA1           0       /* 0 : IOPA1       1 : SCIRXD      */
#define MCRA0           0       /* 0 : IOPA0       1 : SCITXD      */
/*************************************************************/
/*************    SETUP for the MCRB - Register   **************/
#define MCRB9           0       /* 0 : IOPD1       1 : XINT2/EXTSOC */
#define MCRB8           1       /* 0 : CKLKOUT  1 : IOPD0          */
#define MCRB7           0       /* 0 : IOPC7       1 : CANRX       */
#define MCRB6           0       /* 0 : IOPC6       1 : CANTX       */
#define MCRB5           0       /* 0 : IOPC5       1 : SPISTE      */
#define MCRB4           0       /* 0 : IOPC4       1 : SPICLK      */
#define MCRB3           0       /* 0 : IOPC3       1 : SPISOMI     */
#define MCRB2           0       /* 0 : IOPC2       1 : SPISIMO     */
#define MCRB1           1       /* 0 : BIO         1 : IOPC1       */
#define MCRB0           1       /* 0 : XF          1 : IOPC0       */
/*************************************************************/
/*************    SETUP for the MCRC - Register   **************/
#define MCRC13          0       /* 0 : IOPF5       1 : TCLKIN2     */
#define MCRC12          0       /* 0 : IOPF4       1 : TDIR2       */
#define MCRC11          0       /* 0 : IOPF3       1 : T4PWM/T4CMP */
#define MCRC10          0       /* 0 : IOPF2       1 : T3PWM/T3CMP */
#define MCRC9           0       /* 0 : IOPF1       1 : CAP6        */
#define MCRC8           0       /* 0 : IOPF0       1 : CAP5/QEP3   */
```

```
#define MCRC7           0       /* 0 : IOPE7      1 : CAP4/QEP2                */
#define MCRC6           0       /* 0 : IOPE6      1 : PWM12                    */
#define MCRC5           0       /* 0 : IOPE5      1 : PWM11                    */
#define MCRC4           0       /* 0 : IOPE4      1 : PWM10                    */
#define MCRC3           0       /* 0 : IOPE3      1 : PWM9                     */
#define MCRC2           0       /* 0 : IOPE2      1 : PWM8                     */
#define MCRC1           0       /* 0 : IOPE1      1 : PWM7                     */
#define MCRC0           0       /* 0 : IOPE0      1 : CLKOUT                   */
/*************************************************************/
/*************    SETUP for the WDCR - Register   **************/
#define WDDIS           1       /* 0 : Watchdog enabled   1: disabled          */
#define WDCHK2          1       /* 0 : System reset       1:  Normal OP        */
#define WDCHK1          0       /* 0 : Normal Oper.       1: sys reset         */
#define WDCHK0          1       /* 0 : System reset       1: Normal OP         */
#define WDSP            7       /* Watchdog prescaler 7 : div 64               */
/*************************************************************/
/*************    SETUP for the SCSR1 - Register  **************/
#define CLKSRC          0       /* 0 : intern(20MHz)                           */
#define LPM             0       /* 0 : Low power mode 0 if idle                */
#define CLK_PS          1       /* 001 : PLL multiply by 2                     */
#define ADC_CLKEN       0       /* 0 : No ADC-service in this test             */
#define SCI_CLKEN       0       /* 0 : No SCI-service in this test             */
#define SPI_CLKEN       0       /* 0 : No SPI-servide in this test             */
#define CAN_CLKEN       0       /* 0 : No CAN-service in this test             */
#define EVB_CLKEN       0       /* 0 : No EVB-Service in this test             */
#define EVA_CLKEN       0       /* 1 : No EVA-Service in this test             */
#define ILLADR          1       /* 1 : Clear ILLADR during startup             */
/*************************************************************/
/*************    SETUP for the WSGR - Register   **************/
#define BVIS            0       /* 10-9 : 00 Bus visibility OFF                */
#define ISWS            0       /* 8 -6 : 000 0 Waitstates for IO              */
#define DSWS            0       /* 5 -3 : 000 0 Waitstates  data               */
#define PSWS            0       /* 2 -0 : 000 0 Waitstaes code                 */
/*************************************************************/

unsigned int i=0;
unsigned int LED[8]= {0xFF01,0xFF02,0xFF04,0xFF08,
                      0xFF10,0xFF20,0xFF40,0xFF80};
                        /* lookup table for Port E                             */

void wait(void)
        {
        unsigned int i;
        for(i=0;i<65000;i++);
        }


void c_dummy1(void)
        {
        while(1);               /* Dummy ISR used to trap spurious interrupts  */
        }

void main(void)
{
 asm (" setc INTM");    /*Disable all interrupts                               */
 asm (" clrc SXM");     /*Clear Sign Extension Mode bit                        */
 asm (" clrc OVM");     /*Reset Overflow Mode bit                              */
 asm (" clrc CNF");     /*Configure block B0 to data mem.                      */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                        /* set the external waitstates          WSGR           */
```

```
WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                        /* Initialize Watchdog-timer                    */

SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                        /* Initialize SCSR1                             */

MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                +(MCRC1<<1)+MCRC0);
                        /* Initialize master control register C         */

MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                        /* Initialize master control register B         */

 MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
       (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
       (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
       (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                        /* Initialize master control register A         */

PEDATDIR = 0xFF00;      /* Clear Port E                                 */


while(1)
{
        for(i=0;i<14;i++){
                if(i<7)    PEDATDIR=LED[i];
                else PEDATDIR=LED[14-i];
                wait();
                }
}
}
```

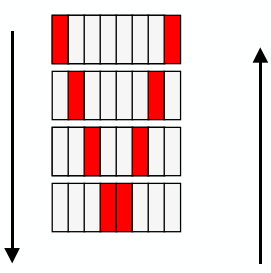
Vectors.asm

```
.title      "vectors.asm"
            .ref      _c_int0,_c_dummy1
            .sect     ".vectors"

reset:      b         _c_int0
int1:       b         _c_dummy1
int2:       b         _c_dummy1
int3:       b         _c_dummy1
int4:       b         _c_dummy1
int5:       b         _c_dummy1
int6:       b         _c_dummy1
reserved:   b         _c_dummy1
sw_int8:    b         _c_dummy1
sw_int9:    b         _c_dummy1
sw_int10:   b         _c_dummy1
sw_int11:   b         _c_dummy1
sw_int12:   b         _c_dummy1
sw_int13:   b         _c_dummy1
sw_int14:   b         _c_dummy1
sw_int15:   b         _c_dummy1
sw_int16:   b         _c_dummy1
trap:       b         _c_dummy1
nmint:      b         _c_dummy1
emu_trap:   b         _c_dummy1
sw_int20:   b         _c_dummy1
sw_int21:   b         _c_dummy1
sw_int22:   b         _c_dummy1
sw_int23:   b         _c_dummy1
```

2.5. Exercise 2-A

Modify the C -source - code :

- **switch 2 LED's on (E7 and E0)**
- **let the 'light' move one step to the center of the LED-bar (E6 and E1 switched on)**
- **continue the move until the 'lights' touch each other**
- **move the 'lights' in the opposite direction**



E7 and E0 = on
E6 and E1 = on
E5 and E2 = on
E4 and E3 = on

1 - 23

Lab 2A: Digital Output

- **Objective**

  The objective of this lab is to modify the source code of the previous exercise Lab2. Again we use only the general purpose I/O-Port E, which is memory mapped into the data memory. In this lab, we will initialize only those registers, that are important for this exercise. These registers are :

MCRA	- Multiplex Control Register A
MCRB	- Multiplex Control Register B
MCRC	- Multiplex Control Register C
PEDATDIR	- Port E Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1

  The aim of this exercise is to produce a modification of the 'Knight Rider' light , so that always 2 of the LED's are switched on while the others are switched off. An endless cycle should produce these steps of LED's switched on: E7+E0 ; E6+E1 ; E5+E2 ; E4+E3 ; E5+E3 ; E6+E2 .

- **Procedure**

  Open Files, Create Project File

1. Using Code Composer, create a new project , called Lab2A.MAK in C:\One2407\Labs\Lab2.

2. Open LED2407C.C and save it as LED2407A.C by clicking : File → Open and File → Save as... .

3. Modify the program LED2407A.C. There are two spots to be changed inside main:

   1. The array LED[8] , from where we take the data's for Port E, has to be adapted to the new task. We can also reduce the number of elements in this array.

   2. Inside main we find a for – loop to count the different steps of the task. The loop – counter, variable i, must be modified too.

   Note : A solution directory c:\one2407\solutions\lab2A contains a solution for this modifications and for all other lab exercises. If you can't solve the task now or your own solution does not work you can copy the source file from there.

4. Add the Source Code Files: LED2407A.C and vectors.asm as well as the Linker Command File: LED2407C.cmd from C:\One2407\Labs\Lab2\ to your project by clicking : Project → Add Files to project

5. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

6. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

7. Close the Build Options Menu by clicking OK

Build and Load

8. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

9. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. **Note:** The binary output file has the same name as the project with the extension ".out" , so you have to load either lab2.out or mylab2.out depending on your project name.

Note: Code Composer can automatically load the output file after a successful build. To do this by default , click on the menu bar : Option → Program Load and select: "Load Program after build", then click OK.

Test

10. Reset the DSP by clicking on Debug → Reset DSP

11. Perform the Test – steps as we have done before in Lab2 to verify the correct function of your project Lab2A.

12. Delete the active breakpoint by clicking on the button "Remove all Breakpoints", close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab2A

Lab 3: Digital Input Port B (IOPB0..B7)

Aim :

- 8 DIP-Switches connected to GPIO-Port B (B0...B7)
- 8 LED‘s connected to E0-E7
- read the switches and show their status on the LED’s

Project - Files :

• F2407DIL.c	- C - source file
• vectors.asm	- jump table to Interrupt Service Routines
• regs2407.h	- pointer definition to peripherals
• F2407DIL.cmd	- linker command file
• rts2xx.lib	- Runtime Library

Project -Directory:

- all files are included in : \Labs\lab3
- the source code files are printed in the student guide

1 - 24

Lab 3: Digital Input Port B (IOPB0..B7)

Registers , used in LAB 3 :

- **Initialize Registers:**
  - Watchdog - Timer - Control : WDCR
  - Waitstate Generator : WSGR
  - System Control & Status 1 : SCSR1
  - Multiplex Control Register A : MCRA
  - Multiplex Control Register B : MCRB
  - Multiplex Control Register C : MCRC
- **Access to GPIO‘s:**
  - **Port E** Data & Direction : PEDATDIR
  - **Port B** Data & Direction : PBDATDIR
    - MS-Byte : direction (1 = out | 0 = in)
    - LS -Byte : data

1 - 25

Lab 3: Digital Input

- **Objective**

  The objective of this lab is to practice and verify the behaviour of digital input and digital output. In this process we make use of the general purpose I/O-Port E and Port B, that are memory mapped into the data memory. In this lab, we will initialize only those registers, that are used in this test. These registers are :

MCRA	- Multiplex Control Register A
MCRB	- Multiplex Control Register B
MCRC	- Multiplex Control Register C
PEDATDIR	- Port E Data & Direction Register
PBDATDIR	- Port B Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1

  The aim of this exercise is to read the inputs, connected to Port B, and to copy this input value to the LED's, connected to Port E in an endless loop.
  There are 8 LED's connected to D7 to D0. A LED is switched on by putting the equivalent data-bit to 1, it is switched off by clearing the bit. The upper byte of PDDATDIR controls the direction of Port E, in this exercise all 8 pins of Port E are used as outputs, therefore the upper byte has to be 0xFF.
  The 8 pins of Port B(B7 to B0) are used as inputs, therefore the upper byte of PBDATDIR ahs to be 0x00. The input level of B7 to B0 can be toggled between ground and Vcc by 8 DIP-switches.

- **Procedure**

Open Files, Create Project File

1. Using Code Composer, create a new project , called myLab3.mak in C:\One2407\Labs\Lab3. **NOTE** : There is a project file Lab3.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

2. Add the Source Code Files: F2407DIL.C and vectors.asm as well as the Linker Command File: F2407DIL.cmd from C:\One2407\Labs\Lab3\ to your project by clicking : Project → Add Files to project

3. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

4. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

5. Close the Build Options Menu by clicking OK

Build and Load

6. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

7. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. Note: The binary output file has the same name as the project with the extension .out, so you have to load either lab3.out or mylab3.out depending on your project name.

Note: Code Composer can automatically load the output file after a successful build. To do this by default , click on the menu bar : Option → Program Load and select: "Load Program after build", then click OK.

Test

8. Reset the DSP by clicking on Debug → Reset DSP

9. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407DIL.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

10. Perform a real time run by clicking : Debug → Run

11. Verify the program running on the external boards behave as supposed. Change the switches and watch the LED's. Debug as necessary.

12. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab3.wks"

13. Delete the active breakpoint by clicking on the button "Remove all Breakpoints", close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab3

Source Code Files Lab Exercise 3

F2407DIL.c
Source Code Files Lab Exercise 3

```
/*********************************************************************/
/* Testprogram for digital I/O on Port E and B                       */
/* running on TMS320LF2407 EVAL-Board, PLL is set to multiply by 2   */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/* date :  17.08.2000                                                */
/*********************************************************************/
/* digital OUT on Port E0...E7 , digital IN on Port B0..B7           */
/* 8 LED's connected to Port E0...E7 ; LED-on : 1  LED off : 0       */
/* 8 Switches to GND connected to Port B0..B7                        */
/* LED's show the status of the Switches                             */
/* program-name :  F2407DIL.c     / project : Lab3                   */
/*********************************************************************/

#include "regs2407.h"

/*************    SETUP for the MCRA - Register   **************/
#define MCRA15                  0       /* 0 : IOPB7      1 : TCLKIN          */
#define MCRA14                  0       /* 0 : IOPB6      1 : TDIR            */
#define MCRA13                  0       /* 0 : IOPB5      1 : T2PWM           */
#define MCRA12                  0       /* 0 : IOPB4      1 : T1PWM           */
#define MCRA11                  0       /* 0 : IOPB3      1 : PWM6            */
#define MCRA10                  0       /* 0 : IOPB2      1 : PWM5            */
#define MCRA9                   0       /* 0 : IOPB1      1 : PWM4            */
#define MCRA8                   0       /* 0 : IOPB0      1 : PWM3            */
#define MCRA7                   0       /* 0 : IOPA7      1 : PWM2            */
#define MCRA6                   0       /* 0 : IOPA6      1 : PWM1            */
#define MCRA5                   0       /* 0 : IOPA5      1 : CAP3            */
#define MCRA4                   0       /* 0 : IOPA4      1 : CAP2/QEP2       */
#define MCRA3                   0       /* 0 : IOPA3      1 : CAP1/QEP1       */
#define MCRA2                   0       /* 0 : IOPA2      1 : XINT1           */
#define MCRA1                   0       /* 0 : IOPA1      1 : SCIRXD          */
#define MCRA0                   0       /* 0 : IOPA0      1 : SCITXD          */
/*************************************************************/
/*************    SETUP for the MCRB - Register   **************/
#define MCRB9                   0       /* 0 : IOPD1      1 : XINT2/EXTSOC    */
#define MCRB8                   1       /* 0 : CKLKOUT  1 : IOPD0             */
#define MCRB7                   0       /* 0 : IOPC7      1 : CANRX           */
#define MCRB6                   0       /* 0 : IOPC6      1 : CANTX           */
#define MCRB5                   0       /* 0 : IOPC5      1 : SPISTE          */
#define MCRB4                   0       /* 0 : IOPC4      1 : SPICLK          */
#define MCRB3                   0       /* 0 : IOPC3      1 : SPISOMI         */
#define MCRB2                   0       /* 0 : IOPC2      1 : SPISIMO         */
#define MCRB1                   1       /* 0 : BIO        1 : IOPC1           */
#define MCRB0                   1       /* 0 : XF         1 : IOPC0           */
/*************************************************************/
/*************    SETUP for the MCRC - Register   **************/
#define MCRC13                  0       /* 0 : IOPF5      1 : TCLKIN2         */
#define MCRC12                  0       /* 0 : IOPF4      1 : TDIR2           */
#define MCRC11                  0       /* 0 : IOPF3      1 : T4PWM/T4CMP     */
#define MCRC10                  0       /* 0 : IOPF2      1 : T3PWM/T3CMP     */
#define MCRC9                   0       /* 0 : IOPF1      1 : CAP6            */
#define MCRC8                   0       /* 0 : IOPF0      1 : CAP5/QEP3       */
#define MCRC7                   0       /* 0 : IOPE7      1 : CAP4/QEP2       */
```

```
#define MCRC6                   0       /* 0 : IOPE6      1 : PWM12               */
#define MCRC5                   0       /* 0 : IOPE5      1 : PWM11               */
#define MCRC4                   0       /* 0 : IOPE4      1 : PWM10               */
#define MCRC3                   0       /* 0 : IOPE3      1 : PWM9                */
#define MCRC2                   0       /* 0 : IOPE2      1 : PWM8                */
#define MCRC1                   0       /* 0 : IOPE1      1 : PWM7                */
#define MCRC0                   0       /* 0 : IOPE0      1 : CLKOUT              */
/*****************************************************************/
/*************    SETUP for the WDCR - Register   **************/
#define WDDIS                   1       /* 0 : Watchdog enabled   1: disabled     */
#define WDCHK2                  1       /* 0 : System reset       1: Normal OP    */
#define WDCHK1                  0       /* 0 : Normal Oper.       1: sys reset    */
#define WDCHK0                  1       /* 0 : System reset       1: Normal OP    */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64          */
/*****************************************************************/
/*************    SETUP for the SCSR1 - Register  **************/
#define CLKSRC                  0       /* 0 : intern(20MHz)                      */
#define LPM                     0       /* 0 : Low power mode 0 if idle           */
#define CLK_PS                  1       /* 001 : PLL multiply by 2                */
#define ADC_CLKEN               0       /* 0 : No ADC-service in this test        */
#define SCI_CLKEN               0       /* 0 : No SCI-service in this test        */
#define SPI_CLKEN               0       /* 0 : No SPI-servide in this test        */
#define CAN_CLKEN               0       /* 0 : No CAN-service in this test        */
#define EVB_CLKEN               0       /* 0 : No EVB-Service in this test        */
#define EVA_CLKEN               0       /* 0 : No EVA-Service in this test        */
#define ILLADR                  1       /* 1 : Clear ILLADR during startup        */
/*****************************************************************/
/*************    SETUP for the WSGR - Register   **************/
#define BVIS                    0       /* 10-9 : 00 Bus visibility OFF           */
#define ISWS                    0       /* 8 -6 : 000 0 Waitstates for IO         */
#define DSWS                    0       /* 5 -3 : 000 0 Waitstates  data          */
#define PSWS                    0       /* 2 -0 : 000 0 Waitstaes code            */
/*****************************************************************/

unsigned int LED[8]={0xFF01,0xFF02,0xFF04,0xFF08,
                     0xFF10,0xFF20,0xFF40,0xFF80};
                        /* lookup table for Port E                                */

void c_dummy1(void)
        {
        while(1);               /*Dummy ISR used to trap spurious interrupts*/
        }

void main(void)
{
 asm (" setc INTM");    /*Disable all interrupts                                  */
 asm (" clrc SXM");     /*Clear Sign Extension Mode bit                           */
 asm (" clrc OVM");     /*Reset Overflow Mode bit                                 */
 asm (" clrc CNF");     /*Configure block B0 to data mem.                         */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                        /* setup external waitstates                              */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                        /* Initialize Watchdog-timer                              */

 SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                        /* Initialize SCSR1                                       */
```

```
MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                +(MCRC1<<1)+MCRC0);
                        /* Initialize master control register C                 */

MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                 (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                 (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                        /* Initialize master control register B                 */

MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
      (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
      (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
      (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                        /* Initialize master control register A                 */

PBDATDIR = 0x0000;    /* use IOPB0..B7 as inputs                                */

PEDATDIR = 0xFF00;    /* Clear Port E                                           */


while(1) PEDATDIR = 0xFF00 + (PBDATDIR & 0x00FF);
                        /* endless loop :  put input B0-B7 to output E0-E7      */

}
```

vectors.asm

```
.title      "vectors.asm"
            .ref        _c_int0,_c_dummy1
            .sect       ".vectors"

reset:      b       _c_int0
int1:       b       _c_dummy1
int2:       b       _c_dummy1
int3:       b       _c_dummy1
int4:       b       _c_dummy1
int5:       b       _c_dummy1
int6:       b       _c_dummy1
reserved:   b       _c_dummy1
sw_int8:    b       _c_dummy1
sw_int9:    b       _c_dummy1
sw_int10:   b       _c_dummy1
sw_int11:   b       _c_dummy1
sw_int12:   b       _c_dummy1
sw_int13:   b       _c_dummy1
sw_int14:   b       _c_dummy1
sw_int15:   b       _c_dummy1
sw_int16:   b       _c_dummy1
trap:       b       _c_dummy1
nmint:      b       _c_dummy1
emu_trap:   b       _c_dummy1
sw_int20:   b       _c_dummy1
sw_int21:   b       _c_dummy1
sw_int22:   b       _c_dummy1
sw_int23:   b       _c_dummy1
```

3.3. : Exercise 3-A

Modify the C -source - code :

- **modify Lab 2 :**
  - **read the DIP switches (Port B)**
  - **modify the frequency of the 'running light' subject to the status of the DIP switches, let's say between 10sec and 0.01 sec per step**

3 - 24

Lab 3A: Digital Input to modify the frequency of LED's

- Objective

In this exercise we will combine Lab2 and Lab3. The aim is to read the DIP-switches, connected to I/O-Port B and to modify the frequency of the 'Knight-Rider'- LED-program. By toggling the DIP-Switches we can gain a value between 0..255 from input port B. This result multiplied by 10000 should then be used as the repeat counter for the delay-function wait().

In this exercise we make use of the general purpose I/O-Port E and Port B, that are memory mapped into the data memory. In this lab, we will initialize only those registers, that are used in this test. These registers are :

MCRA	- Multiplex Control Register A
MCRB	- Multiplex Control Register B
MCRC	- Multiplex Control Register C
PEDATDIR	- Port E Data & Direction Register
PBDATDIR	- Port B Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1

- Procedure

Open Files, Create Project File

1. Using Code Composer, create a new project , called Lab3A.mak in C:\One2407\Labs\Lab3.

2. Open the source-file LED2407C.c in c:\One2407\labs\lab2 and save it as LED2407B.c in c:\One2407\labs\lab3 by clicking File → Open and File → Save as..

3. Modify the program LED2407B.C. Take into account these modifications :
    1. The delay function wait() must include now a variable repeat counter in the for-loop. One possibility to do this, is to declare a global variable 'period' and to repeat the for-loop as long as i<period. After reading the DIP-switches we will calculate a new value for period. The data-type for period should be long integer to include high numbers for low frequencies.

    2. Inside main we have to add the calculation of the new period depending on the value received from the DIP-switches. A good position would be to include this calculation just before calling wait(). You should read Port B, take the lower byte (it includes the data from Port B) multiply it by 10.000 and put the result into the long integer period. Because the value derived from Port B is between 0 and 255 , period is then set between 0 and 255*10.000 = 2.550.000 .

    Note : A solutions directory c:\one2407\solutions\lab3A contains a solution for this modifications and for all other lab exercises. If you can't solve the task now or your own solution does not work you can copy the source file from there.

4. Add the Source Code Files: LED2407B.C and vectors.asm as well as the Linker Command File: F2407DIL.cmd from C:\One2407\Labs\Lab3\ to your project by clicking : Project → Add Files to project

5. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

6. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

7. Close the Build Options Menu by clicking OK

Build and Load

8. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

9. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. Note: The binary output file has the same name as the project with the

extension .out, so you have to load either lab3.out or mylab3.out depending on your project name.

Note: Code Composer can automatically load the output file after a successful build. To do this by default , click on the menu bar : Option → Program Load and select: "Load Program after build", then click OK.

Test

10. Reset the DSP by clicking on Debug → Reset DSP

11. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407LEDB.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

12. Perform a real time run by clicking : Debug → Run

13. Verify the program running on the external boards behave as supposed. Change the switches and watch the LED's. Debug as necessary.

Optional Exercise : Probe Point

14. Open the Watch-Window by clicking : View → Watch Window. Add the Registers PEDATDIR, PBDATDIR and the variable 'period' to the watch window by clicking right mouse inside the watch window , repeat 3 times the selection : " Insert New Expression" and type in the Expression Window :
    *(int *)0x7095@data,x ; PEDATDIR
    *(int *)0x709A@data,x ; PBDATDIR and
    period

15. In the program LED2407B.c place the cursor at the line : wait(). Click right mouse and select : "Toggle Probe Point" . The line is then highlighted in blue.

16. From the menu bar select : Debug → Probe Points. Inside this window click on first line in the probe points window. With the "Connect to"-selector select "Watch Window". Finish this dialog by clicking on "Replace" and "OK"

17. Run the program. You will see that the watch-window is updated each time the program passes the probe-point.

End of Exercise Lab3A

4 C2407- Interrupt System

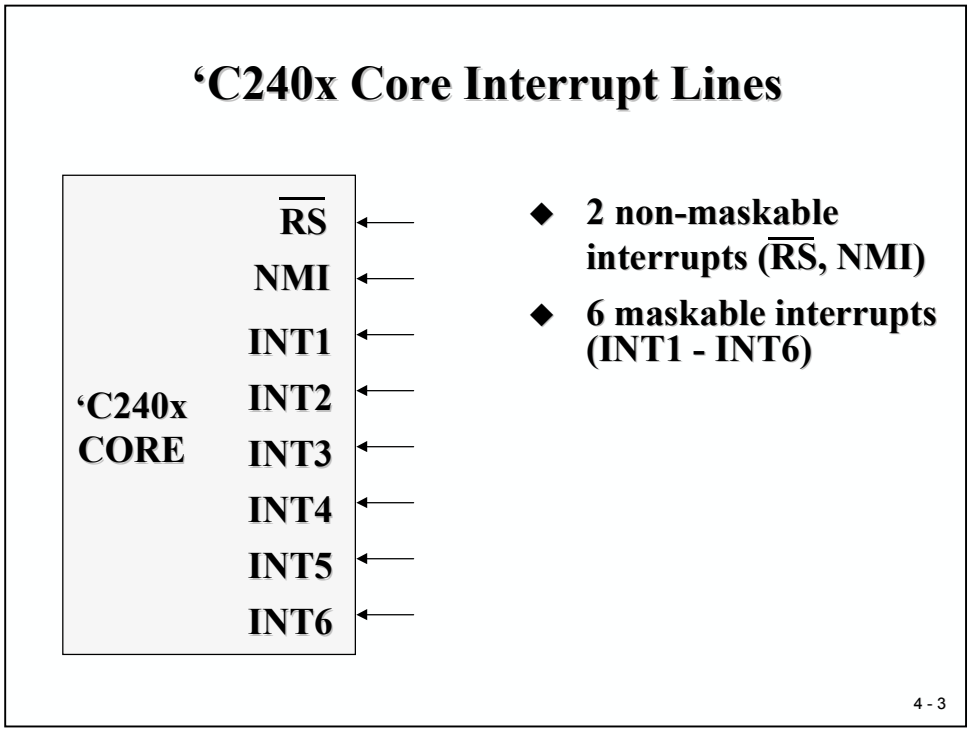
Introduction

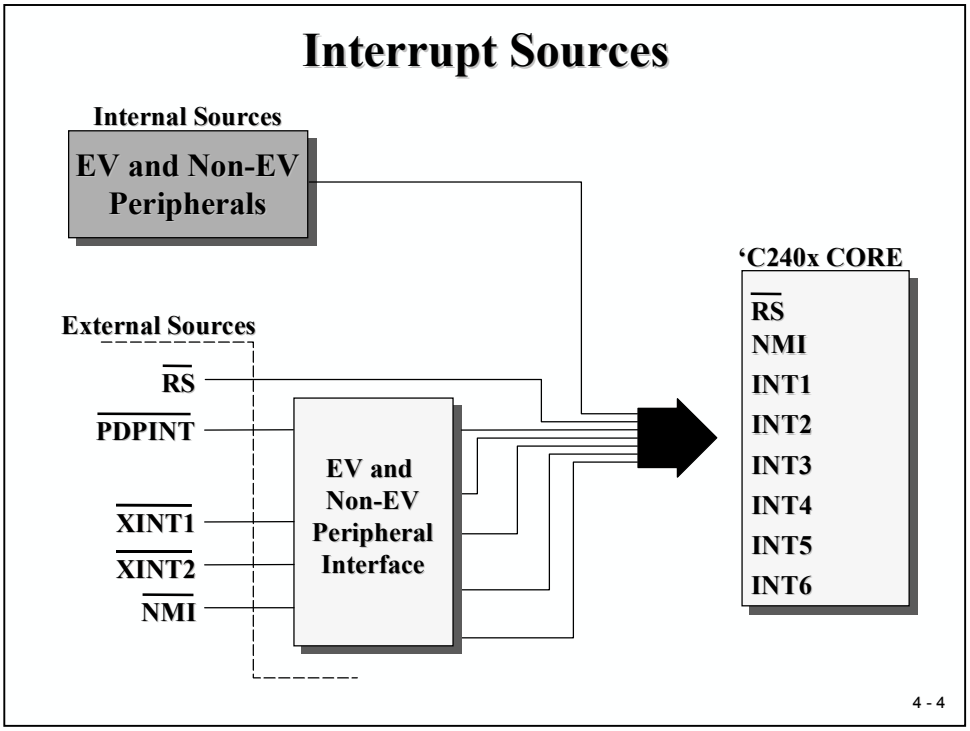
This module describes the interrupt structure on the C240x.

Overview

Interrupts provide a means of directing the 'C240x to suspend its main program in order to respond to a hardware-driven event. This eliminates the need to poll external events via software (unless desired), and improves response time and decrease processor overhead dramatically. Typically, interrupts are generated by devices which need to give or take data from the 'C240x. Examples of such devices are A/D and D/A converters and other processors. Interrupt may also be used as a signal to inform the 'C240x when any event of interest has occurred within the system. When the 'C240x recognizes the interrupt signal, it suspends execution of the main program and begins execution of the code specific to the particular interrupt event. On the ‘C240x, the programmer can dynamically select when interrupts may be taken, and which interrupts will be recognized.

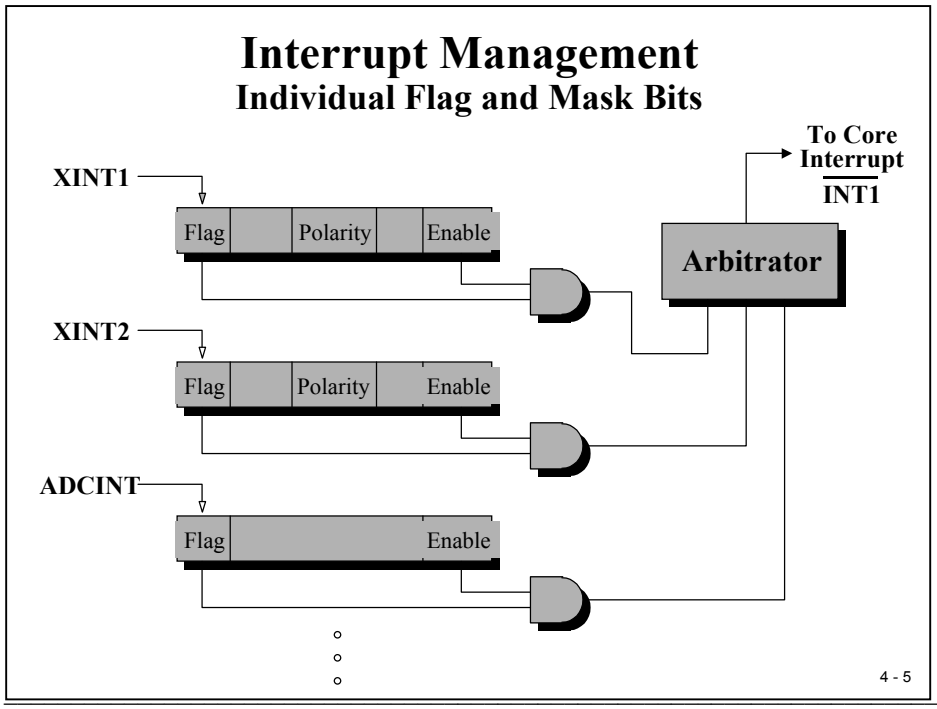
The ‘C240x core of the ‘C240x processor supports six user-maskable interrupts. These interrupts are then fanned out and shared among numerous on-chip peripherals and external pins. Interrupts can be generated by internal or external sources or by software interrupt instructions. A reset function, a non-maskable interrupt, and a power-drive protection interrupt are also supported on all ‘C240x devices.



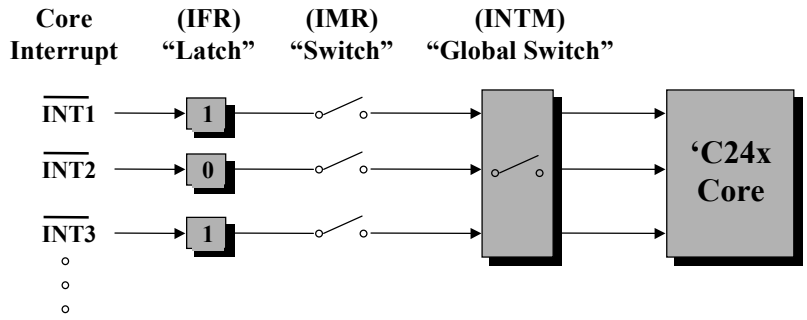


Interrupt Management - Individual Flags and Masks

Peripheral and externally generated interrupts are arranged in groups and associated with one of the core interrupts, INT1 - INT6. When a valid signal is generated by a peripheral or by an external source, the corresponding individual flag bit is set to 1 in the control register associated with that peripheral or external interrupt . If the individual enable bit is set for that interrupt, an interrupt request will be passed to the arbitration logic, which prioritizes all pending interrupts in its group and then sends the highest priority interrupt request to its associated core interrupt. The individual flag bits are set regardless of the condition of the individual enable bits. The schematic below depicts the signal flow to core INT1.



Interrupt Management
Conceptual Core Overview



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.

4 - 6

Interrupt Management - Core Flags and Masks

When an interrupt request is sent to the CPU core by the arbitration logic, the corresponding bit in the Interrupt Flag Register (IFR) is set. Flags are set regardless of the condition of the masks (IMR) or global interrupt switch (INTM). The core flag bits are automatically cleared by the processor after the interrupt is recognized by the CPU.

4. Interrupt Management - Core Flags
IFR @ 0006h

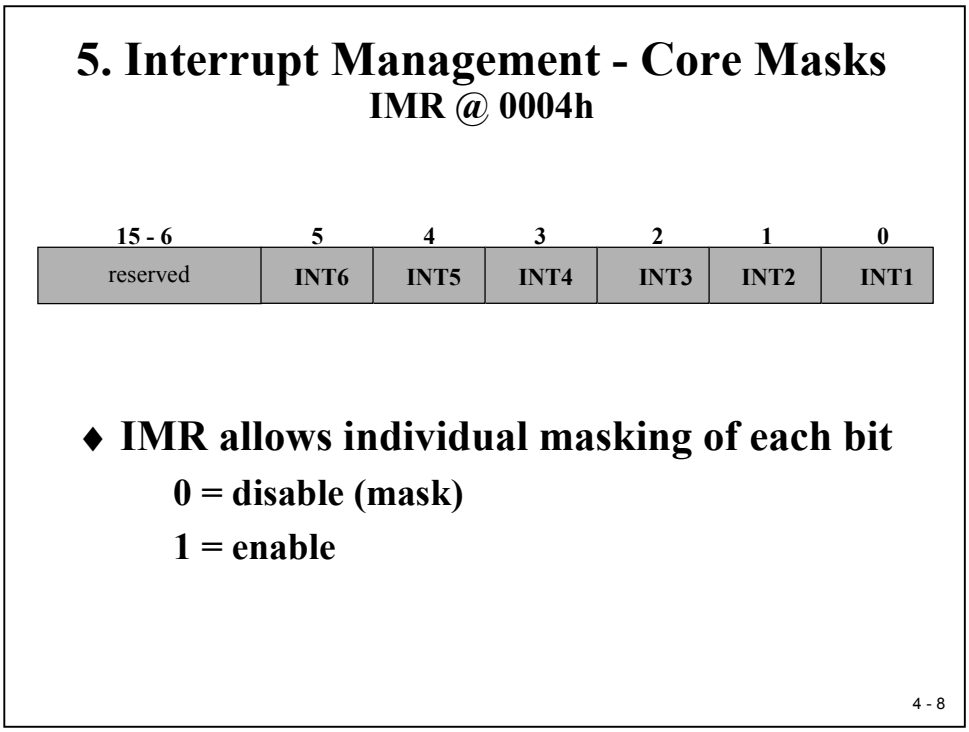
15 - 6	5	4	3	2	1	0
reserved	INT6	INT5	INT4	INT3	INT2	INT1

- **IFR indicates when a valid interrupt has occurred**
- **"1" is displayed in the corresponding bit**

4 - 7

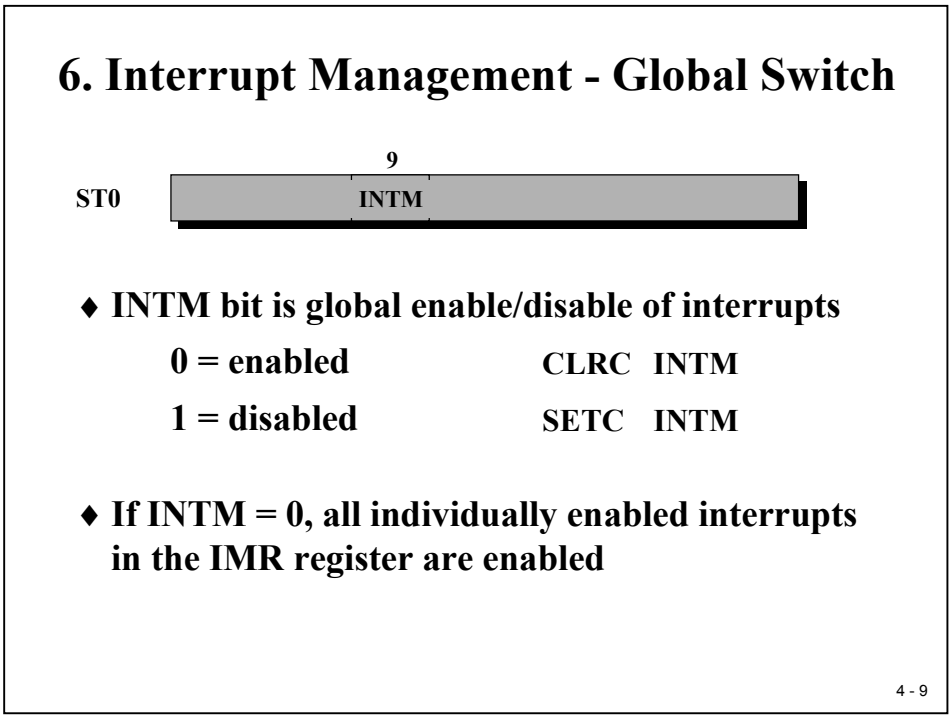
Interrrupt Management – Masks

The Interrupt Mask Register (IMR) allows the user to select which core interrupts the ‘C240x should respond to at a given time. A logic 1 written to any mask bit enables the corresponding interrupt. Notice that RS and NMI cannot be masked.



Interrupt Management - Global Switch

The INTM bit (bit 9 in ST0) can be used to globally enable or disable all maskable interrupts. When INTM = 0, all enabled interrupts (as indicated in the IMR register) are allowed. INTM = 1 inhibits these interrupts. Note that the IFR and IMR are not affected by the state of INTM.



Interrupt Management — Summary

The complete interrupt story can now be revealed by observing all four registers related to interrupt operation. For example, if a valid interrupt signal occurred on the XINT1 pin, the following bits would be set:

- Individual flag bit = 1 (for XINT1, this is bit 15 at data address 7070h)

If the following condition were met, an interrupt request would be sent to the arbitration logic:

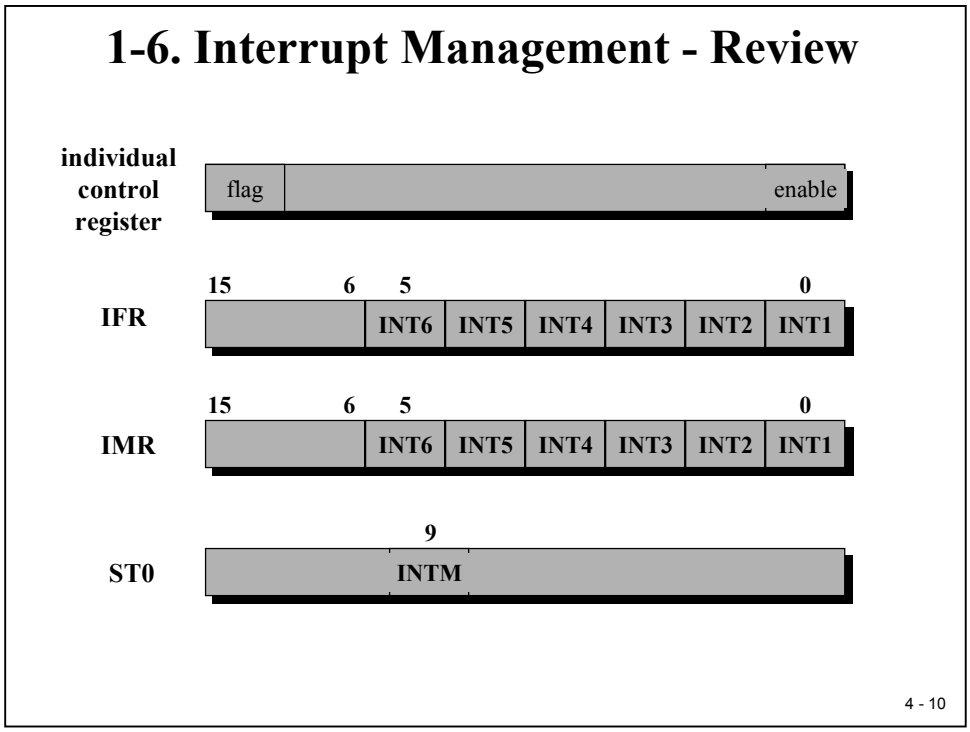
- Individual enable bit = 1 (for XINT1, this is bit 0 at data address 7070h)

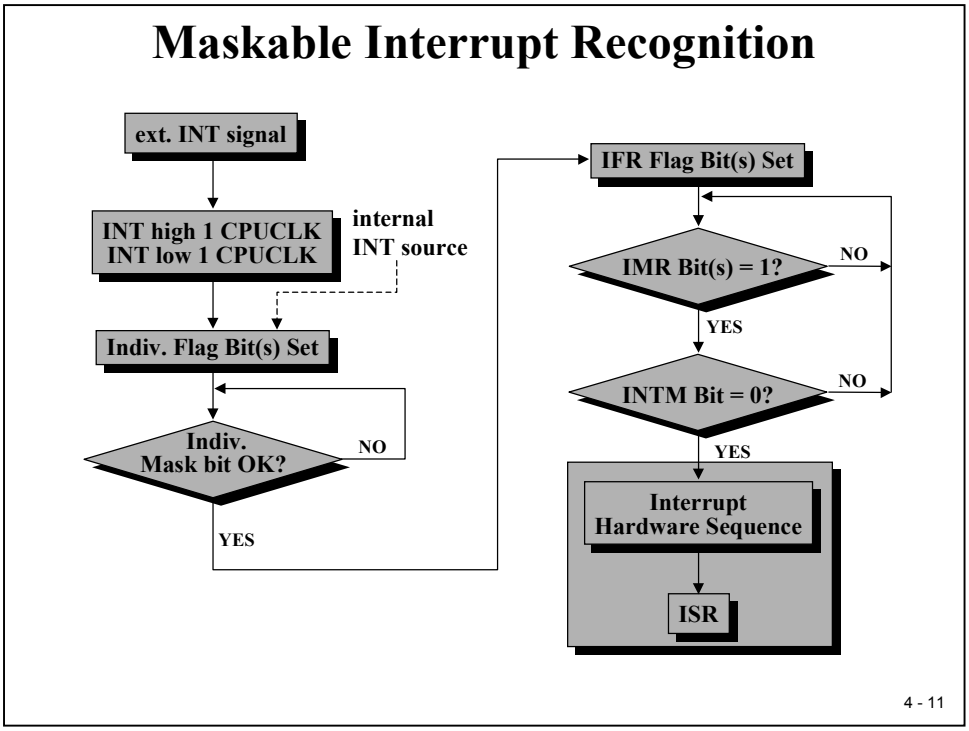
In this case, XINT1 is the highest priority interrupt in its arbitration group, so the arbitration logic will always send an interrupt request to the CPU regardless of what other (lower priority) interrupts are pending in its grouping:

- IFR (bit 0) = 1

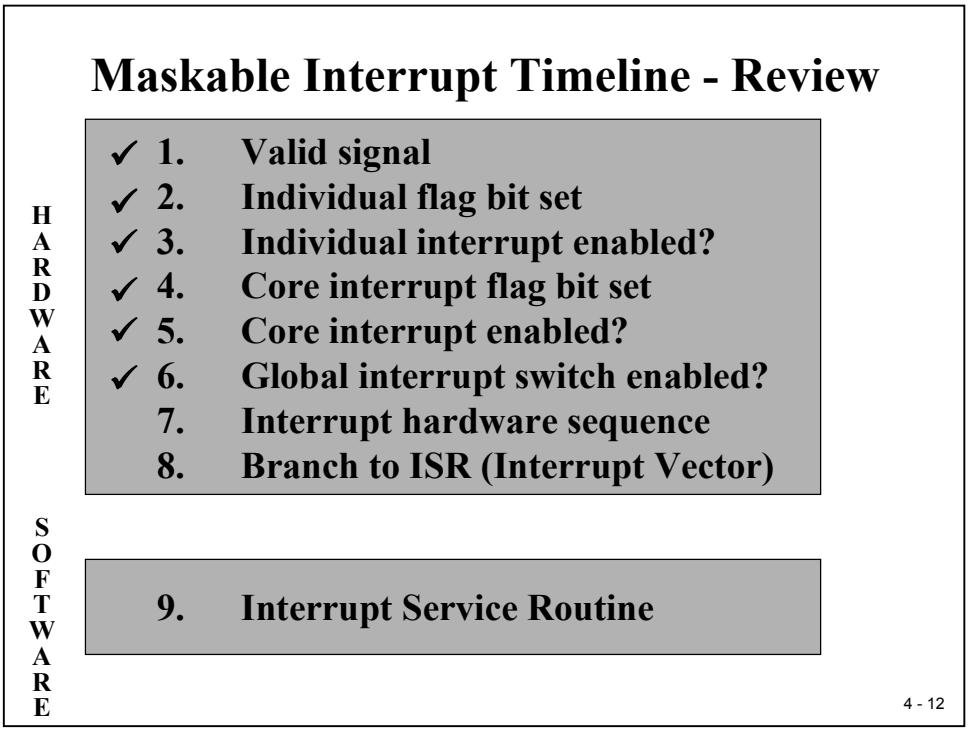
Finally, if the following conditions were met, the processor would automatically respond to the interrupt:

- IMR (bit 0) = 1
- ST0 (bit 9) = 0





Interrupt processing can be broken down into a series of steps which occur in a serial fashion (that is, if the interrupt is not interrupted!) The figure below shows a conceptual view of how interrupts are processed. The next figure shows a more detailed timeline of events.



Core Interrupt Vectors

After the interrupt has been recognized, the program counter (PC) is loaded with the corresponding address of the interrupt vector as shown in the table below. The user must map a set of branch instructions to the appropriate ISRs at address 0000h in program space using the .cmd file of the linker. Typically, a .sect directive is used:

```
.sect "vectors"
B reset ;reset vector
B INT1_ISR ;INT1 ISR
B INT2_ISR ;INT2 ISR
```

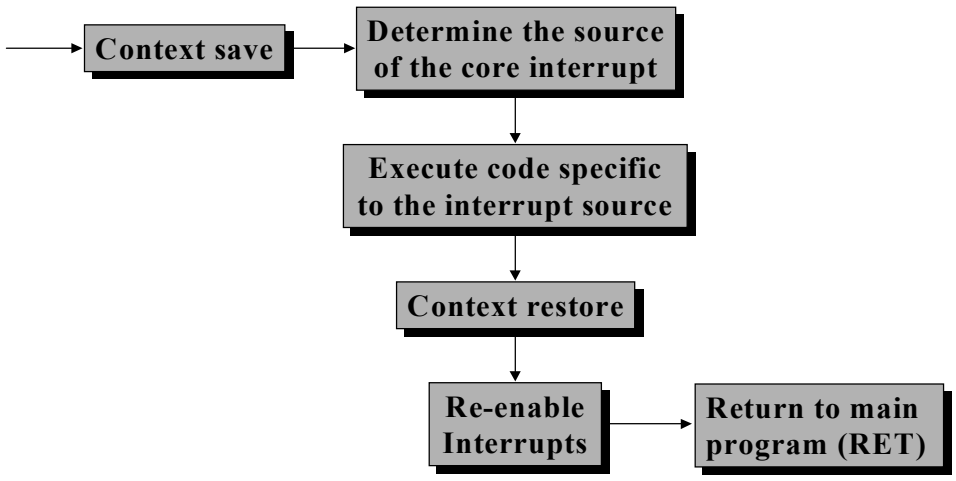
Interrupt Vector Locations

Interrupt	Location (Hex)	Description	Priority †
$\overline{RS}$	0	Reset	1
INT1	2	Core Interrupt #1	4
INT2	4	Core Interrupt #2	5
INT3	6	Core Interrupt #3	6
INT4	8	Core Interrupt #4	7
INT5	A	Core Interrupt #5	8
INT6	C	Core Interrupt #6	9
TRAP	22	Trap Instruction Vector	–
NMI	24	Nonmaskable Interrupt	3

† Software interrupts such as TRAP do not have a hardware priority

4 - 13

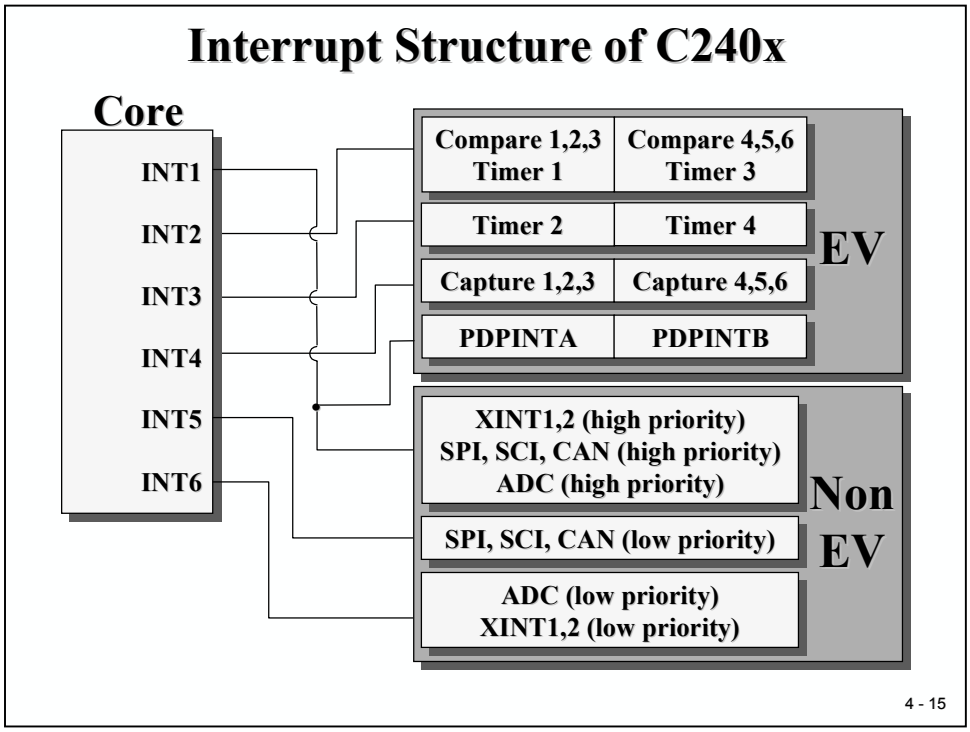
Interrupt Service Routine

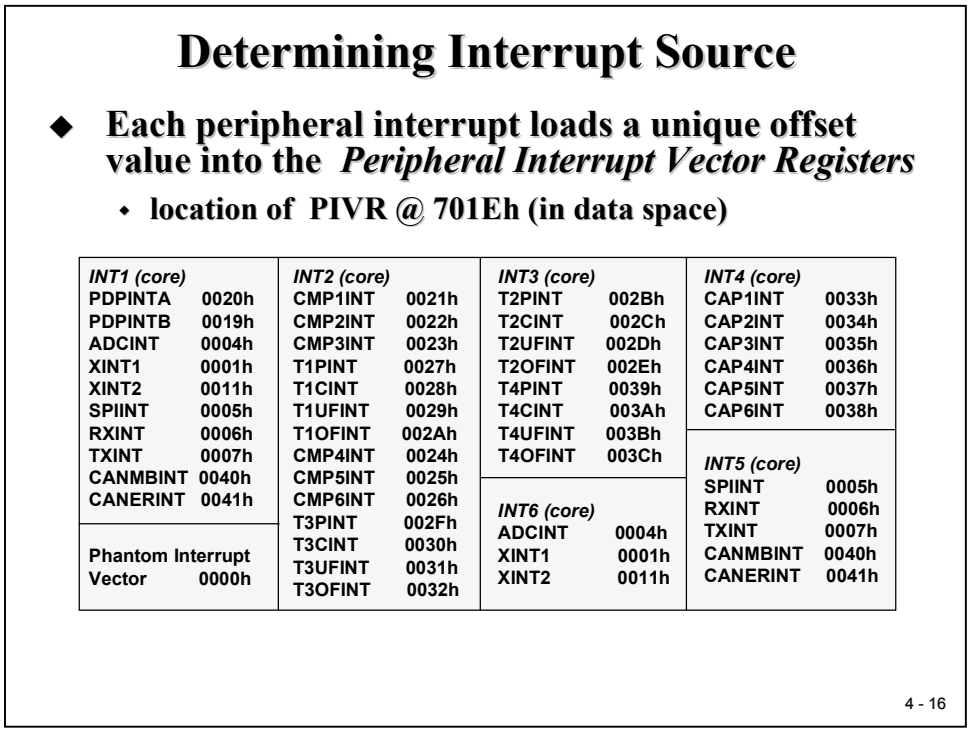


4 - 14

Interrupt Sources

The ‘LF2407 core, as previously stated, provides for only six maskable interrupts (INT1 - INT6)
plus the reset and NMI interrupts which are non-maskable. The number of peripheral and external interrupts on the ‘LF2407 number more than six, and therefore the maskable core interrupts must be shared among these sources. Three core interrupts have been dedicated to event-manager peripherals, with the remaining three shared among non-event manager peripherals, the PDPINT (part of event-manager), and all other external interrupt pins.





Nesting Interrupts

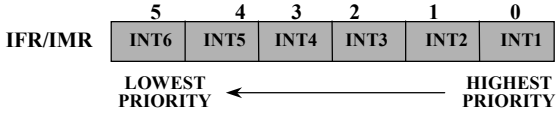
There may be some instance when the CPU is handling an ISR when a higher priority interrupt occurs and needs to be serviced before completion of the current ISR. Provisions have been made on the ‘LF2407 to handle nested interrupts, when necessary, via software control.

Only two additional items must be added to your ISR code to enable nesting of interrupts:

1. Re-prioritizing interrupts via IMR (remember to context save and restore the IMR)
2. Re-enabling INTs via INTM

Multiple Interrupts / Priority

- **Which interrupt is serviced first when multiple interrupts occur?**



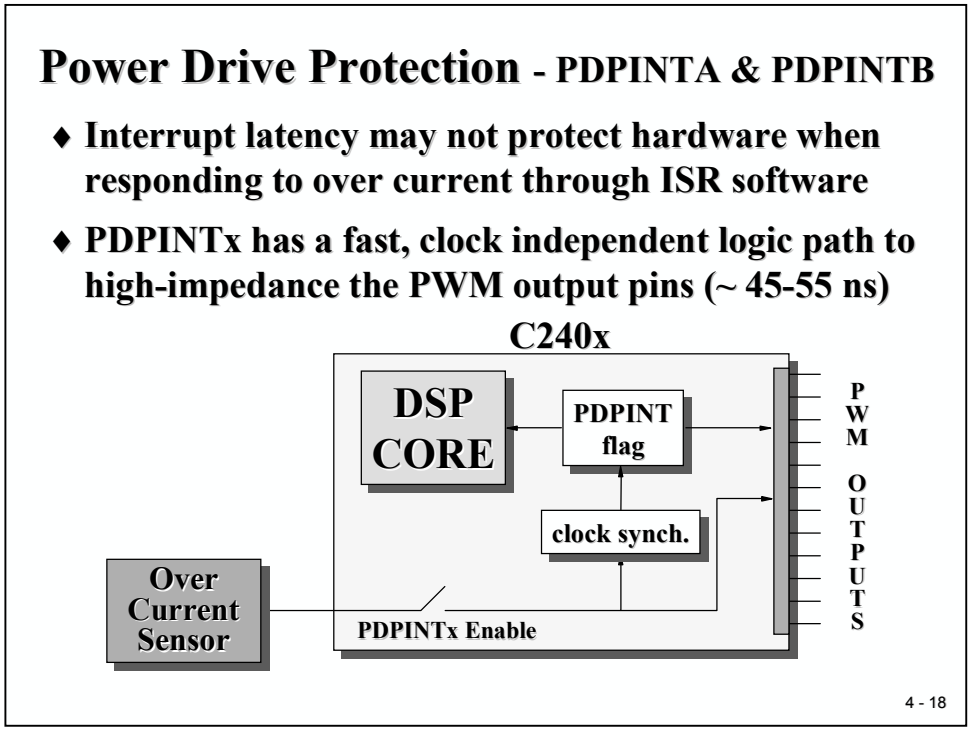
- **Priority is only associated with multiple interrupts**
- **If any interrupt is being serviced, and global interrupts are turned on ($\overline{\text{INTM}}$ = 0), and any enabled interrupt occurs, it will be serviced immediately**
- **Each ISR is responsible for modifying the IMR in software to control interrupt nesting**

4 - 17

Power-Drive Protection Interrupt

The PDPINT pin provides a special interrupt function for implementing over current protection that differs from using the other external interrupt sources (e.g. XINT1, XINT2). When a valid interrupt signal occurs on the PDPINT pin, two parallel actions take place. The first action is by dedicated logic that automatically high-impedances all PWM outputs in approximately 45-55 ns. This special hardware path avoids the latency that would occur if the high-impedance function were implemented by software in an interrupt service routine. In addition, the hardware path functions independent of the CPU clock, and therefore operates even during a clock failure (e.g. external oscillator disconnect). The second action taken by PDPINT is a regular interrupt request to the CPU through the normal channels (i.e. event manager interrupt arbitration logic, IFR, IMR,INTM).

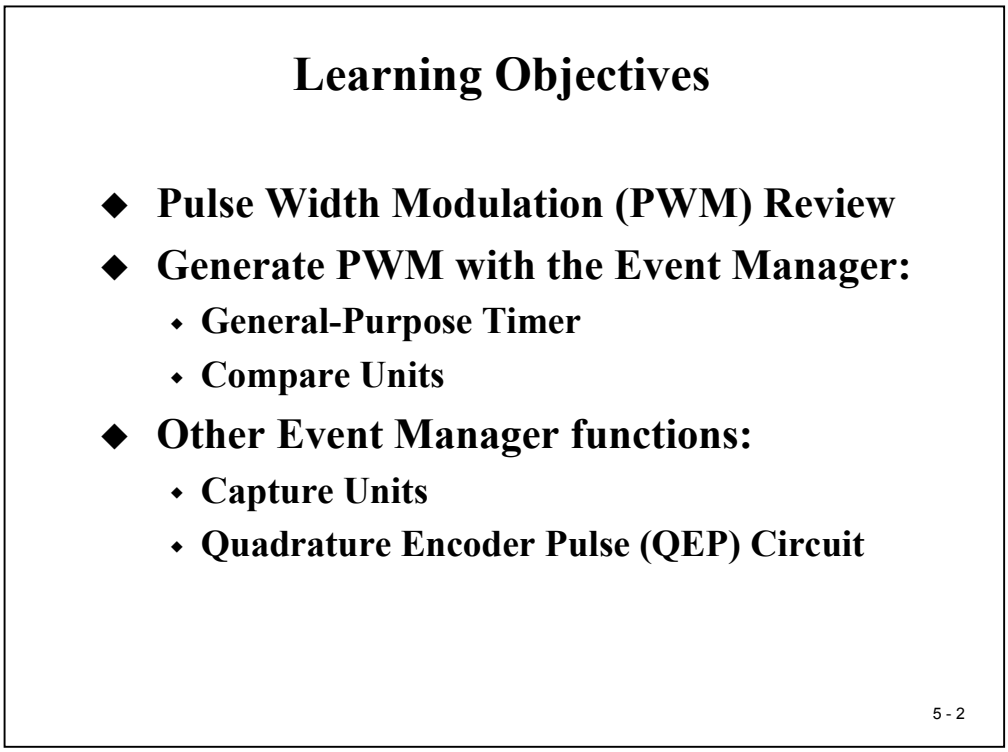
It is important to note that both PDPINT actions can be jointly disabled using the *PDPINT Enable* bit in the *EV Interrupt Mask Register A* (EVIMRA@742Ch). However, only the regular interrupt request action is affected by the IMR and INTM, the PWM high-impedance action will take place regardless of their settings.



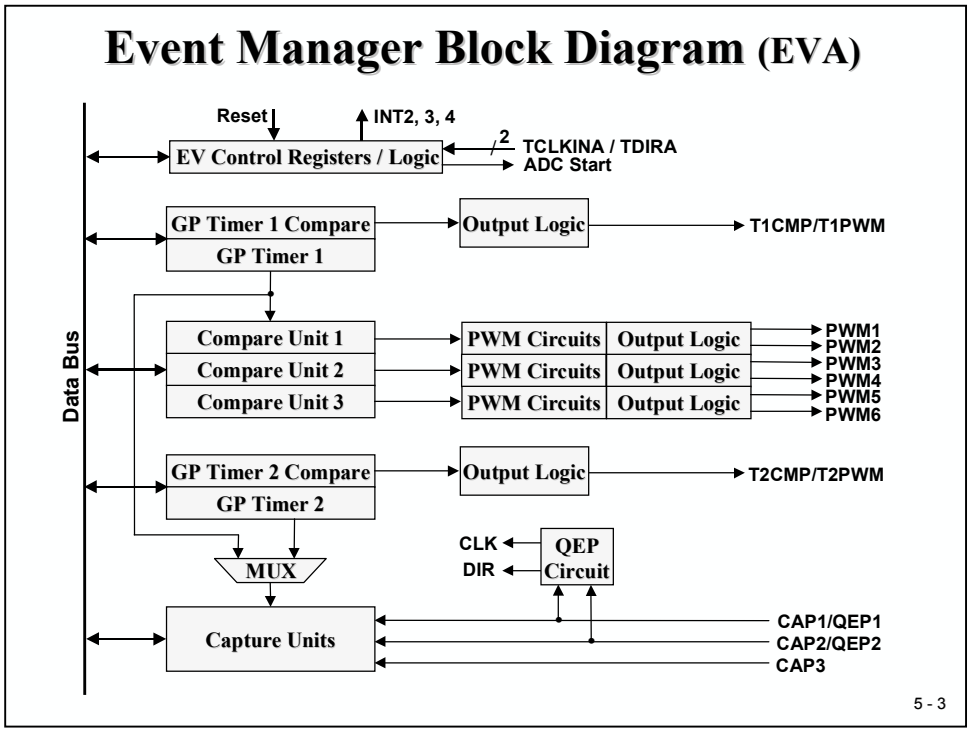
Introduction

This module explains how to generate PWM waveforms using the timers and compare units. Also, the capture units, quadrature encoder pulse circuit, and the hardware deadband units will be discussed.

Learning Objectives



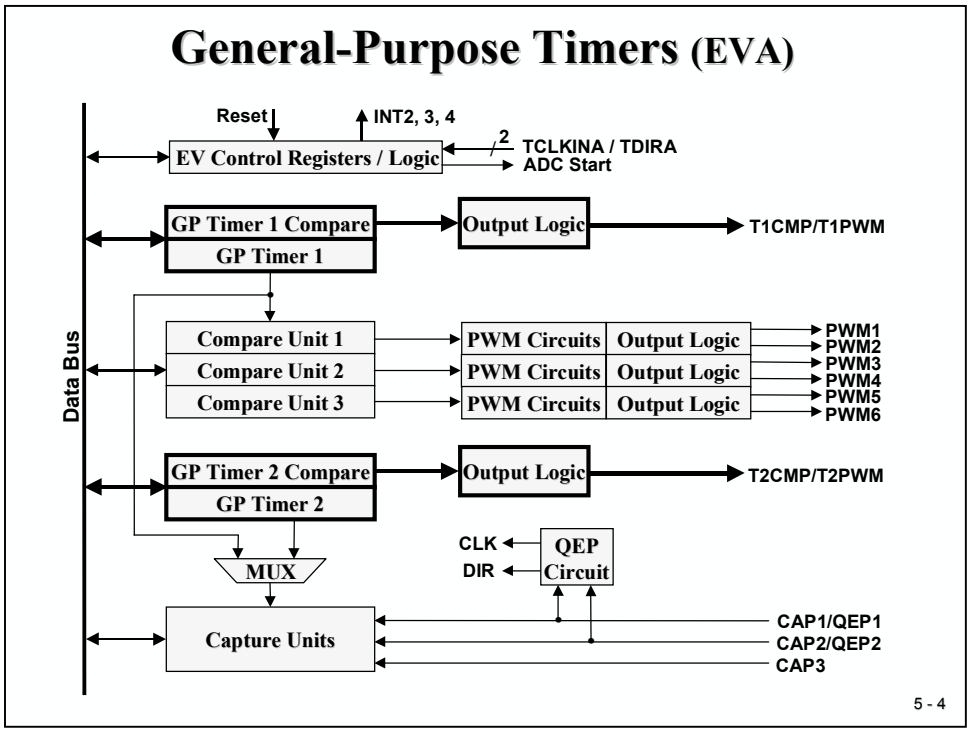
Event Manager



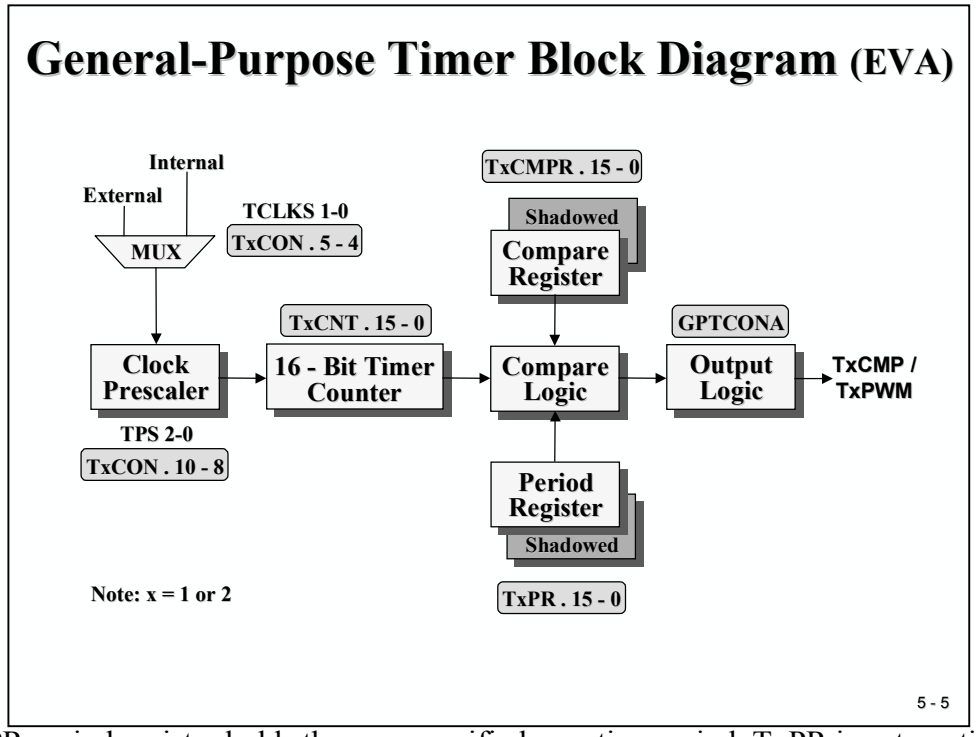
The Event Manager consist of the following blocks:

- General-Purpose Timers
- Full Compare Units
- Capture Units
- Quadrature Encoder Pulse (QEP) circuit

General-Purpose Timers

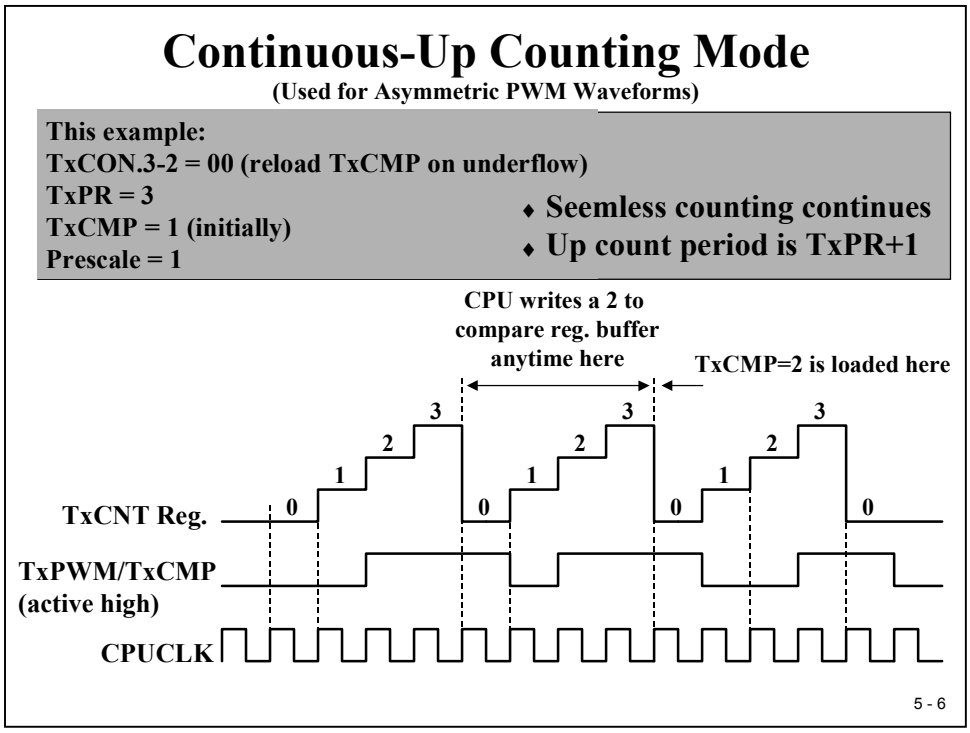


The GP Timers provide a time base for the operation of the compare units, and associated PWM circuits to generate PWM outputs. Additionally, they provide a time base for the operation of the quadrature encoder pulse (QEP) circuit (GP Timer 2 only) and the capture units. The GP Timers can also be used to generate a sampling period in a control system.



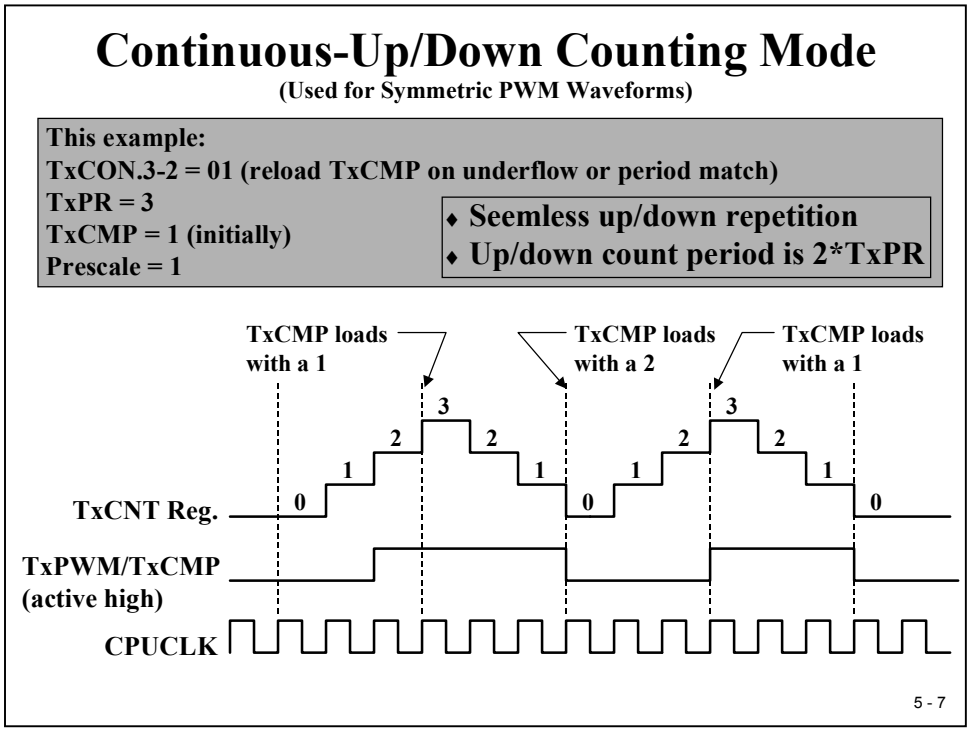
The TxPR period register holds the user specified counting period. TxPR is automatically loaded from the *period register buffer* on a counter underflow, which is defined as TxCNT=0. This allows for on-the-fly timer period changes. Note that the period register buffer is static in that if no change in the current period value is desired, one is not required to write the same value to the buffer on successive timer cycles.

GP Timer Modes of Operation

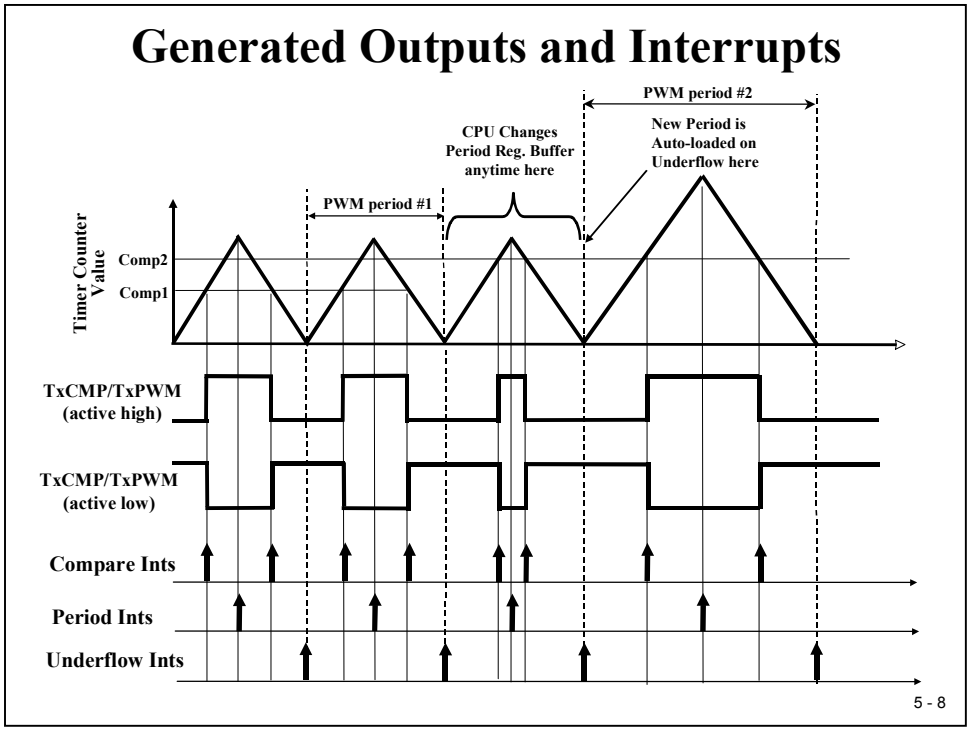


The procedure for GP Timer Continuous-Up Counting is as follows:
User sets bit 6 of TxCON register high to initiate counting
Counting begins on next rising clock edge

- 1st count is a "Zero" (no increment)
- Count up until match with period register



PWM Outputs and Interrupts



GP Timer Registers

As was the case with the period register, buffering is present for each timer compare register. Software writes a value to the *compare register buffe*r, from which the TxCMP register is automatically loaded on one of three user selected events:

1. timer underflow (TxCNT = 0)
2. timer underflow or period match
3. immediately

The event selection is made using bits 2 and 3 of the TxCON register, and allows for on-the-fly compare value changes. Note that the compare register buffer is static in that if no change in the current compare value is desired, one is not required to write the same value to the buffer on successive timer cycles.

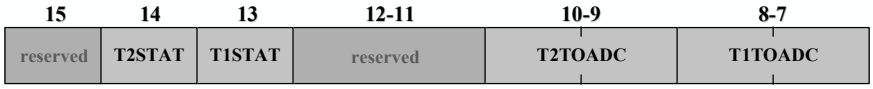
GP Timer Registers

	Register	Address	Description
EVA	GPTCONA	7400h	General Purpose Timer Control Register A
	T1CNT	7401h	GP Timer 1 Counter Register
	T1CMPR	7402h	GP Timer 1 Compare Register Buffer
	T1PR	7403h	GP Timer 1 Period Register Buffer
	T1CON	7404h	GP Timer 1 Control Register
	T2CNT	7405h	GP Timer 2 Counter Register
	T2CMPR	7406h	GP Timer 2 Compare Register Buffer
	T2PR	7407h	GP Timer 2 Period Register Buffer
	T2CON	7408h	GP Timer 2 Control Register
EVB	GPTCONB	7500h	General Purpose Timer Control Register B
	T3CNT	7501h	GP Timer 3 Counter Register
	T3CMPR	7502h	GP Timer 3 Compare Register Buffer
	T3PR	7503h	GP Timer 3 Period Register Buffer
	T3CON	7504h	GP Timer 3 Control Register
	T4CNT	7505h	GP Timer 4 Counter Register
	T4CMPR	7506h	GP Timer 4 Compare Register Buffer
	T4PR	7507h	GP Timer 4 Period Register Buffer
	T4CON	7508h	GP Timer 4 Control Register

5 - 9

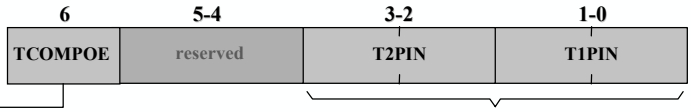
GPTCONA Register (EVA)

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

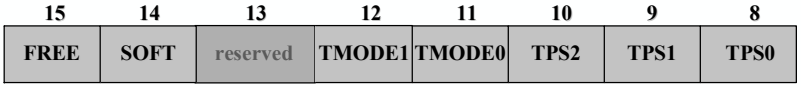
TxPWM/TxCMP Output Pin Conditioning
00: forced low
01: active low
10: active high
11: forced high

5 - 10

TxCON Register (EVA)

T1CON @ 7404h / T2CON @ 7408h

Upper Byte:



Emulation Halt Behavior
00 = stop immediately
01 = stop at end of period
1x = free run (do not stop)

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

5 - 11

TxCON Register (EVA)

T1CON @ 7404h / T2CON @ 7408h

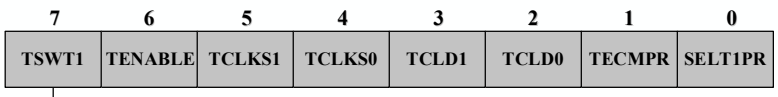
Lower Byte:

Timer Compare Operation Enable
0 = disabled
1 = enabled

Timer Enable
0 = timer disabled
1 = timer enabled

Timer Clock Source
00 = internal (CPUCLK)
01 = external TMRCLK pin
10 = internal (CPUCLK)
11 = QEP (Timer 2 only)

Period Register Select
0 = use own per. reg.
1 = use Timer 1 per. reg
(bit reserved in T1CON)



Start with Timer 1
0 = use own TENABLE
1 = use Timer 1 TENABLE
(bit reserved in T1CON)

Compare Register Reload Condition
00 = when counter equals zero (underflow)
01 = when counter equals zero or period reg
10 = immediately
11 = reserved

5 - 12

Lab 4: Digital Output at Port E (IOPE0..D7) plus GPT2

Aim:

- Advanced version of Lab2 ('Knight Rider')
- GPT2- runs as Timer , period = 0.1 sec
- Period - Interrupt - Service Routine to perform the next step
- GPT2- signal is active at pin #128 (EVM- P1 - pin 13)

Files :

- LED2407C2.c - source file main c program
- vectors.asm - jump table Interrupt Service Routine
- regs2407.h - pointer definition to peripherals
- LED2407C2.cmd - linker Command file
- rts2xx.lib - Runtime Library

Project -Directory:

- all files are included in : \Labs\lab4
- the source code files are printed in the student guide

5 - 13

New Registers involved in Lab 4:

- General Purpose Timer Control A : GPTCONA
- Timer 2 Control : T2CON
- Timer 2 Period : T2PR
- Timer 2 Compare : T2CMPR
- Timer 2 Counter : T2CNT
- EV-Manager A Interrupt Flag A : EVAIFRA
- EV-Manager A Interrupt Flag B : EVAIFRB
- EV-Manager A Interrupt Flag C : EVAIFRC
- EV-Manager A Interrupt Mask A : EVAIMRA
- EV-Manager AInterrupt Mask B : EVAIMRB
- EV-Manager A Interrupt Mask C : EVAIMRC
- Interrupt Flag : IFR
- Interrupt Mask : IMR
- Peripheral Interrupt Vector : PIVR

5 - 14

Lab 4: Digital Output with a GPT2 -period

- Objective

The objective of this lab is to practice and verify the mechanics of digital output together with a first interrupt service procedure. For this purpose we will combine Lab exercise 2 with a hardware timer, generated by GP-Timer T2. This timer initiates periodically every 100ms the start of its Interrupt Service Routine, which is used to update the 'Knight Rider'. The registers, involved in this lab, are :

MCRA	- Multiplex Control Register A
MCRB	- Multiplex Control Register B
PEDATDIR	- Port E Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1
GPTCONA	- EVA -General Purpose Timer Control Register
T2CON	- Timer 2 Control Register
T2PR	- Timer 2 Period Register
T2CMPR	- Timer 2 Compare Register
T2CNT	- Timer 2 Counter Register
EVAIFRA	- Event Manager A Interrupt Flag Group A
EVAIFRB	- Event Manager A Interrupt Flag Group B
EVAIFRC	- Event Manager A Interrupt Flag Group C
EVAIMRA	- Event Manager A Interrupt Mask Group A
EVAIMRB	- Event Manager A Interrupt Mask Group B
EVAIMRC	- Event Manager A Interrupt Mask Group C
IFR	- Global Interrupt Flag Register
IMR	- Global Interrupt Mask Register

- Procedure

Open Files, Create Project File

1. Using Code Composer, create a new project , called myLab4.MAK in C:\One2407\Labs\Lab4. **NOTE** : There is a project file Lab4.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

2. Add the Source Code Files: LED2407C2.C and vectors.asm as well as the Linker Command File: LED2407C2.cmd from C:\One2407\Labs\Lab4\ to your project by clicking : Project → Add Files to project

3. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

4. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

5. Close the Build Options Menu by clicking OK

Build and Load

6. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

7. Load the output file onto the EVM. Click : File → Load Program and choose the output file you generated. Note: The binary output file has the same name as the project with the extension .out, so you have to load either lab4.out or mylab4.out depending on your project name.

Test

8. Reset the DSP by clicking on Debug → Reset DSP

9. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "LED2407C2.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

10. Perform a real time run by clicking : Debug → Run

11. Verify the program running on the external LED's as supposed. Also note that the bottom left corner of the screen is marked "DSP Running" to sign the real time run.

12. Halt the program by clicking : Debug → Halt , reset the DSP (Debug → Reset DSP) and go again until main (Debug → Go main)

13. Place a Breakpoint in the LED2407C2.c – window inside the interrupt service routine at line "EVAIFRB=T2PINT;" . Do this by placing the cursor at this line ,click right mouse and select : "Toggle Breakpoint". The line is highlighted with a pink bar to mark an active breakpoint.

14. Perform a real time run by clicking : Debug → Run or Key F5. Verify that the program halts when it reaches the Breakpoint line. Watch the LED's on Port E. Continue the program execution by clicking : Debug → Run or Key F5 and watch the next cycle on the LED's.

15. Open the Watch-Window by clicking : View → Watch Window. Add the Register PEDATDIR to the watch window by clicking right mouse inside the watch window and select : " Insert New Expression" . Type in the Expression Window : *(int *)0x7095@data,x ; PEDATDIR

16. Replace the Breakpoint by a Probe Point. Click right mouse at the line "EVAIFRB=T2PINT;" , select 'Toggle Breakpoint' and 'Toggle Probe point'. The line should then be highlighted in blue instead of pink. Connect the Probe Point to the watch window . Click Debug → Probe Points... , and connect the probe point to the watch window.

17. Perform again a real time run by clicking : Debug → Run or Key F5. Verify that the watch window is updated with each interrupt service.

18. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab4.wks"

19. Delete the active probe point by clicking on the button "Remove all Probe Points", close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab4

Initializing Sequence for LAB4 :

1. Set Waitstates, Watchdog and System Control as in Lab2
2. In Multiplex Control (MCRA change bit 13 to drive T2PWM as output (MCRA.13 = 1)
3. Set **GPTCONA** to enable outputs (bit 6 = 1) and polarity of T2 to active low (bit 3-2 = 01b)
4. Load **T2PR** with the period and **T2CMPR** with period/2
5. Set **T2CON** to
   - continuous up counting mode (bit 12-11 = 10b)
   - Input Clock Prescaler CPUCLK/64 (bit 10-8 = 110b)
   - Clock Source Internal (bit 5-4 = 00b)
   - use T2 Period and T2Enable (bit 7 = 0 , bit 0 = 0)
6. Clear all EVA-Interrupt Flags (**EVAIFRx** = 0xFFFF)
7. Enable T2-Period -Interrupt (**EVAIMRB**-Bit 0 = 1)
8. Clear all global Interrupt Flags (**IFR** = 0xFFFF)
9. Unmask (enable) Global Interrupt Level 3 (**IMR**-Bit 3 = 1)
10. Enable all Interrupts (**INTM** = 0)
11. Enable Timer T2 (**T2CON** - Bit 6 = 1)

5 - 15

```
/*********************************************************************/
/* Testprogram for digital I/O on Port E                             */
/* running on TMS320LF2407 EVAL -Board                               */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/* date :  03.07.2000  ,  (C) Frank.Bormann@fh-zwickau.de            */
/*********************************************************************/
/* digital I/O on Port E0...E7                                       */
/* 8 LED's connected to Port E0...E7 ; LED-on : 1  LED off : 0       */
/* Knight - rider :  1 of 8 LED switched on, then go from left to right */
/* Time delay 0.1 sec made by use of Timer 2 Interrupt service       */
/* Pin T2CMP/IOPB5 ( Pin 128) on Connector P1-13 shows T2 -period    */
/* make sure that external Ground-Switch on IOPB5 is off             */
/* program-name :  LED2407C2.c  / project : LAB4                     */
/* NEW :  SCSR1 - clock enable EVA , PLL, ILLADR                     */
/*********************************************************************/
#include "regs2407.h"
/*************    SETUP for the MCRA - Register   **************/
#define MCRA15          0       /* 0 : IOPB7      1 : TCLKIN       */
#define MCRA14          0       /* 0 : IOPB6      1 : TDIR         */
#define MCRA13          1       /* 0 : IOPB5      1 : T2PWM        */
#define MCRA12          0       /* 0 : IOPB4      1 : T1PWM        */
#define MCRA11          0       /* 0 : IOPB3      1 : PWM6         */
#define MCRA10          0       /* 0 : IOPB2      1 : PWM5         */
#define MCRA9           0       /* 0 : IOPB1      1 : PWM4         */
#define MCRA8           0       /* 0 : IOPB0      1 : PWM3         */
#define MCRA7           0       /* 0 : IOPA7      1 : PWM2         */
#define MCRA6           0       /* 0 : IOPA6      1 : PWM1         */
#define MCRA5           0       /* 0 : IOPA5      1 : CAP3         */
#define MCRA4           0       /* 0 : IOPA4      1 : CAP2/QEP2    */
#define MCRA3           0       /* 0 : IOPA3      1 : CAP1/QEP1    */
#define MCRA2           0       /* 0 : IOPA2      1 : XINT1        */
#define MCRA1           0       /* 0 : IOPA1      1 : SCIRXD       */
#define MCRA0           0       /* 0 : IOPA0      1 : SCITXD       */
/*************************************************************/
/*************    SETUP for the MCRB - Register   ******************/
#define MCRB9           0       /* 0 : IOPD1      1 : XINT2/EXTSOC */
#define MCRB8           1       /* 0 : CKLKOUT  1 : IOPD0          */
#define MCRB7           0       /* 0 : IOPC7      1 : CANRX        */
#define MCRB6           0       /* 0 : IOPC6      1 : CANTX        */
#define MCRB5           0       /* 0 : IOPC5      1 : SPISTE       */
#define MCRB4           0       /* 0 : IOPC4      1 : SPICLK       */
#define MCRB3           0       /* 0 : IOPC3      1 : SPISOMI      */
#define MCRB2           0       /* 0 : IOPC2      1 : SPISIMO      */
#define MCRB1           1       /* 0 : BIO        1 : IOPC1        */
#define MCRB0           1       /* 0 : XF         1 : IOPC0        */
/*************    SETUP for the MCRC - Register   ****************************/
#define MCRC13          0       /* 0 : IOPF5      1 : TCLKIN2      */
#define MCRC12          0       /* 0 : IOPF4      1 : TDIR2        */
#define MCRC11          0       /* 0 : IOPF3      1 : T4PWM/T4CMP  */
#define MCRC10          0       /* 0 : IOPF2      1 : T3PWM/T3CMP  */
#define MCRC9           0       /* 0 : IOPF1      1 : CAP6         */
#define MCRC8           0       /* 0 : IOPF0      1 : CAP5/QEP3    */
#define MCRC7           0       /* 0 : IOPE7      1 : CAP4/QEP2    */
#define MCRC6           0       /* 0 : IOPE6      1 : PWM12        */
#define MCRC5           0       /* 0 : IOPE5      1 : PWM11        */
#define MCRC4           0       /* 0 : IOPE4      1 : PWM10        */
#define MCRC3           0       /* 0 : IOPE3      1 : PWM9         */
#define MCRC2           0       /* 0 : IOPE2      1 : PWM8         */
#define MCRC1           0       /* 0 : IOPE1      1 : PWM7         */
#define MCRC0           0       /* 0 : IOPE0      1 : CLKOUT       */
```

```
/*************     SETUP for the WDCR - Register   *************/
#define WDDIS           1       /* 0 : Watchdog enabled   1: disabled          */
#define WDCHK2          1       /* 0 : System reset       1: Normal OP         */
#define WDCHK1          0       /* 0 : Normal Oper.       1: sys reset         */
#define WDCHK0          1       /* 0 : System reset       1: Normal OP         */
#define WDSP            7       /* Watchdog prescaler 7 : div 64               */
/*****************************************************************/
/*************     SETUP for the SCSR1 - Register  *************/
#define CLKSRC          0       /* 0 : intern(20MHz)                           */
#define LPM             0       /* 0 : Low power mode 0 if idle                */
#define CLK_PS          1       /* 001 : PLL multiply by 2                     */
#define ADC_CLKEN       0       /* 0 : No ADC-service in this test             */
#define SCI_CLKEN       0       /* 0 : No SCI-service in this test             */
#define SPI_CLKEN       0       /* 0 : No SPI-servide in this test             */
#define CAN_CLKEN       0       /* 0 : No CAN-service in this test             */
#define EVB_CLKEN       0       /* 0 : No EVB-Service in this test             */
#define EVA_CLKEN       1       /* 1 : Enable Clock for EVA-unit               */
#define ILLADR          1       /* 1 : Clear ILLADR during startup             */
/*****************************************************************/
/*************     SETUP for the WSGR - Register   *************/
#define BVIS            0       /* 10-9 : 00 Bus visibility OFF                */
#define ISWS            0       /* 8 -6 : 000 0 Waitstates for IO              */
#define DSWS            0       /* 5 -3 : 000 0 Waitstates  data               */
#define PSWS            0       /* 2 -0 : 000 0 Waitstaes code                 */
/*****************************************************************/
/*************     SETUP for the GPTCONA - Register  ************/
#define GPTCON_T2TOADC      0
        /* 10-9  : T2TOADC = 00 : no ADC-Start by any GPT2-Event               */
#define GPTCON_T1TOADC      0
        /*  8-7  : T1TOADC = 00 : no ADC-Start by any GPT1-Event               */
#define GPTCON_TCOMPOE  1
        /*   6   : TCOMPOE = 1  : enable all 2 GPT compare outputs             */
#define GPTCON_T2PIN 1
        /*  3-2  : T2PIN = 01 : Pol. of GPT2 comp out=active low               */
#define GPTCON_T1PIN 0
        /*  1-0  : T1PIN = 00 : Pol. of GPT1 comp out=forced low               */
/*****************************************************************/
/**************     SETUP for the T2CON - Register  *************/
#define T2CON_FREESOFT      0
        /* 15-14 FREE, SOFT : 00 stop on JTAG-emulation suspend               */
#define T2CON_TMODE         2
         /* 12-11 : TMODE1,0   : 10 Contiuous up counting mode                */
#define T2CON_TPS           6
        /* 10-8  : TPS2-0     : 110   Input clock prescaler CPUCLK/64         */
#define T2CON_TSWT1         0
        /* 7         : TSWT1          0       use own TENABLE bit             */
#define T2CON_TENABLE       1
        /* 6     : TENABLE    : 1 enable GPT2,  in first instruction disable GPT2 !!  */
#define T2CON_TCLKS         0
        /* 5-4   : TCLKS1,0   : 00  Clock source select : internal             */
#define T2CON_TCLD          1
        /* 3-2   : TCLD1,0    : 01  Timer compare(active) register reload condition
                                 when counter value is 0 or equal to period register*/
#define T2CON_TECMPR        1
        /* 1     :  TECMPR        : 1 enable timer compare operation           */
#define T2CON_SELT1PR       0
        /* 0         :  SELT1PR    :0   use own period register                */
/*****************************************************************/
/*************     SETUP for the EVIMRA - Register  *************/
#define T1OFINT         0       /* 10 : Timer 1 overflow interrupt             */
#define T1UFINT         0       /*  9 : Timer 1 underflow interrupt            */
```

```
#define T1CINT          0       /*  8 : Timer 1 compare interrupt           */
#define T1PINT          0       /*  7 : Timer 1 period interrupt            */
#define  CMP3INT        0       /*  3 : Compare 3 interrupt                 */
#define CMP2INT         0       /*  2 : Compare 2 interrupt                 */
#define CMP1INT         0       /*  1 : Compare 1 interrupt                 */
#define PDPINT          0       /*  0 : Power Drive Protect Interrupt       */
/*****************************************************************/
/*************    SETUP for the EVIMRB - Register  ******************/
#define T2OFINT         0       /*  3 : Timer 2 overflow interrupt          */
#define T2UFINT         0       /*  2 : Timer 2 underflow interrupt         */
#define T2CINT          0       /*  1 : Timer 2 compare interrupt           */
#define T2PINT          1       /*  0 : Timer 2 period interrupt            */
/*****************************************************************/
/*************    SETUP for the EVIMRC- Register  ******************/
#define CAP3INT         0       /*  2 : Capture Unit 3 interrupt            */
#define CAP2INT         0       /*  1 : Capture Unit 2 Interrupt            */
#define CAP1INT         0       /*  0 : Capture unit 1 interrupt            */
/*****************************************************************/
/*************    SETUP for the IMR - Register    ******************/
#define INT6            0       /*  5 : Level INT6  is masked               */
#define INT5            0       /*  4 : Level INT5  is masked               */
#define INT4            0       /*  3 : Level INT4  is masked               */
#define INT3            1       /*  2 : Level INT3  is unmasked             */
#define INT2            0       /*  1 : Level INT2  is masked               */
#define INT1            0       /*  0 : Level INT1  is masked               */
/*****************************************************************/
#define PERIOD 47348    /* T2PERIOD = 0,1s
                           T2PR = 0,1s / ( 33ns * T2CON_TPS )           */
unsigned int LED[8]={0xFF01,0xFF02,0xFF04,0xFF08,
                     0xFF10,0xFF20,0xFF40,0xFF80};
                                /* lookup table for Port E              */
void c_dummy1(void)
{
        while(1);       /* Dummy ISR used to trap spurious interrupts   */
}

interrupt void T2PER_ISR(void)
{
 static unsigned char i=0;
 if ((PIVR-0x002B)==0)   /* Verify interrupt-Nr. 002B = T2PINT           */
 {
  if(i<7)PEDATDIR=LED[i++];
  else PEDATDIR=LED[14-i++];
  if (i>=14) i=0;
  EVAIFRB=T2PINT;
 }
}

void main(void)
{
 asm (" setc INTM");      /*Disable all interrupts                        */
 asm (" clrc SXM");       /*Clear Sign Extension Mode bit                 */
 asm (" clrc OVM");       /*Reset Overflow Mode bit                       */
 asm (" clrc CNF");       /*Configure block B0 to data mem.               */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                          /* setup waitstates                             */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                          /* Initialize Watchdog-timer                    */
```

```
SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
        (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
        (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                        /* Initialize SCSR                                  */

MCRB = ((MCRB9<<9)+(MCRB8<<8)+
       (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
       (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                        /* Initialize master control register B             */


MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
       (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
       (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
       (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                        /* Initialize master control register A             */

MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)+
        (MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)+
        (MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)+
        (MCRC1<<1)+MCRC0);
                        /* Initialize master control register C             */

GPTCONA=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
       (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                        /*      Initialize GP Timer Control A               */

PEDATDIR = 0xFF00;  /* Clear Port E                                         */
T2PR = PERIOD;      /* initialize T2-period                                 */
T2CMPR = PERIOD/2;  /* T2-signal with duty 1:1                              */
T2CNT=0x0000;       /* start value for T2-counter                           */

T2CON=((T2CON_FREESOFT<<14)+(T2CON_TMODE<<11)+ (T2CON_TPS<<8)+
       (T2CON_TSWT1<<7)+(T2CON_TCLKS<<4)+(T2CON_TCLD<<2)+
       (T2CON_TECMPR<<1)+T2CON_SELT1PR);

EVAIFRA=0xFFFF;     /* clear EV Interrupt Flag Register Group A             */

EVAIFRB=0xFFFF;     /* clear EV Interrupt Flag Register Group B             */

EVAIFRC=0xFFFF;     /* clear EV Interrupt Flag Register Group C             */

EVAIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
       (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                        /* EV Interrupt Mask Register Group A               */

EVAIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                        /* EV Interrupt Mask Register Group B               */

EVAIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                        /* EV Interrupt Mask Register Group C               */

IFR=0xFFFF;             /* Interrupt Flag Register , address 0x0006         */
                        /* Reset  all core interrupts                       */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                        /* Interrupt Mask Register                          */
asm (" clrc INTM");     /* Enable all unmasked interrupts                   */
T2CON=T2CON+(T2CON_TENABLE<<6);     /* enable GPT2 now                      */
while(1);               /* endless loop, action done by ISR                 */
}
```

```
        .title    "vectors.asm"
        .ref      _c_int0,_c_dummy1,_T2PER_ISR
        .sect     ".vectors"

reset:      b     _c_int0
int1:       b     _c_dummy1
int2:       b     _c_dummy1
int3:       b     _T2PER_ISR
int4:       b     _c_dummy1
int5:       b     _c_dummy1
int6:       b     _c_dummy1
reserved:   b     _c_dummy1
sw_int8:    b     _c_dummy1
sw_int9:    b     _c_dummy1
sw_int10:   b     _c_dummy1
sw_int11:   b     _c_dummy1
sw_int12:   b     _c_dummy1
sw_int13:   b     _c_dummy1
sw_int14:   b     _c_dummy1
sw_int15:   b     _c_dummy1
sw_int16:   b     _c_dummy1
trap:       b     _c_dummy1
nmint:      b     _c_dummy1
emu_trap:   b     _c_dummy1
sw_int20:   b     _c_dummy1
sw_int21:   b     _c_dummy1
sw_int22:   b     _c_dummy1
sw_int23:   b     _c_dummy1
```

Optional Lab - Exercise 4 - A

- **Modify Lab Exercise 4 ('Knight Rider')**
- **Modify the C-Source :**
  - **include the DIP-switches**
  - **read the status of the DIP-switches**
  - **modify the frequency subject to the DIP-switch - status**
  - **use precise frequenies :**
    - **F = 20Hz, 10Hz, 5Hz, 2Hz, 1Hz, 0.5Hz, 0.2Hz**
    - **and 0.1Hz**
- **optional : use Timer T1 instead of T2**

5 - 18

Optional Lab 4A:
Digital Output with a variable GPT2 –period , controlled by Digital Input

- **Objective**

The objective of this lab is to combine Lab4 (Digital Output 'Knight Rider' with a fixed period generated by GPT-ISR) with reading of the Digital Inputs at GPIO-B. The Aim is to modify the frequency of the 'Knight Rider' LED's depending on the value read from the DIP-Switches. The goal is to modify the interval between 20Hz (0.05s) and 0.1Hz(10s).

The maximum value for T2PERIOD = interval time / (CPUCLK * Pre-Scale)
= 10s /(33,9084 ns * 128)
= 2,304,000

which does not fit into the 16-bit Register T2PR. Therefore a software counter (36) has to be used to count 36 Interrupt Services before the next output state is given to the LED's:

T2PR	Counter	Cycles	Interval
64,000	36	2,304,000	10s
32,000	36	1,152,000	5s
12,800	36	460,800	2s
6,400	36	230,400	1s
3200	36	115,200	0.5s
1.280	36	46,080	0.2s
640	36	23,040	0.1s
320	36	11,520	0.05s

The registers, involved in this lab, are :

MCRA - Multiplex Control Register A
MCRB - Multiplex Control Register B
MCRC - Multiplex Control Register C
PEDATDIR - Port E Data & Direction Register
PBDATDIR - Port B Data & Direction Register
WDCR - Watchdog Control Register
WSGR - Wait State Generator Register
SCSR1 - System Control and Status Register
GPTCON - General Purpose Timer Control Register 1
T2CON - Timer 2 Control Register
T2PR - Timer 2 Period Register
T2CMPR - Timer 2 Compare Register
T2CNT - Timer 2 Counter Register
EVAIFRA - Event Manager A Interrupt Flag Group A
EVAIFRB - Event Manager AInterrupt Flag Group B
EVAIFRC - Event Manager A Interrupt Flag Group C
EVAIMRA - Event Manager A Interrupt Mask Group A
EVAIMRB - Event Manager A Interrupt Mask Group B
EVAIMRC - Event Manager A Interrupt Mask Group C
IFR - Global Interrupt Flag Register
IMR - Global Interrupt Mask Register

- **Procedure**

Open Files, Create Project File

1. Using Code Composer, create a new project, called Lab4A.mak in C:\One2407Labs\Lab4.

2. Open the source-file LED2407C2.c in c:\One2407\labs\lab4 and save it as LED2407C3.c in c:\One2407\labs\lab4 by clicking File → Open and File → Save as..

3. Modify the program LED2407C3.C. Take into account these modifications :
    1. create a new array PERIOD[8] for the 8 values to be loaded into T2PR (see table from previous page)
    2. Modify the Interrupt Service Routine T2PER_ISR. Add a counter (#36) into the function to count 365 calls of the ISR, before any action takes place. Make sure to count only those calls, that belong to T2-Period-Interrupt (PIVR = 0x002B). After 36 calls serve the LED-output (PEDATDIR = LED[i]), read the DIP-Switches (x = PBDATDIR & 0x00FF) and modify T2PR depending on the PBDATDIR-value.
    3. DON'T forget to switch MCRA13 back to IOPB5 (= 0) ! It was set in Lab4 to make the frequency measurement at T2PWM-output. See the #define-block for MCRA.
    4. Set the Prescaler for T2 to 128 (T2CON_TPS = 111).

Note : A solutions directory c:\one2407\solutions\lab4A contains a solution for this modifications and for all other lab exercises. If you can't solve the task now or your own solution does not work you can copy the source file from there.

4. Add the Source Code Files: LED2407C3.C and vectors.asm as well as the Linker Command File: LED2407C2.cmd from C:\One2407\Labs\Lab4\ to your project by clicking : Project → Add Files to project

5. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

6. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

7. Close the Build Options Menu by clicking OK

Build and Load

8. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. Load the program into the DSP.

Test

9. Verify the program running on the external LED's as supposed. Change the DIP-switches and watch the changing frequency of the LED's. Debug as necessary.

End of Exercise Lab4A

PWM Review

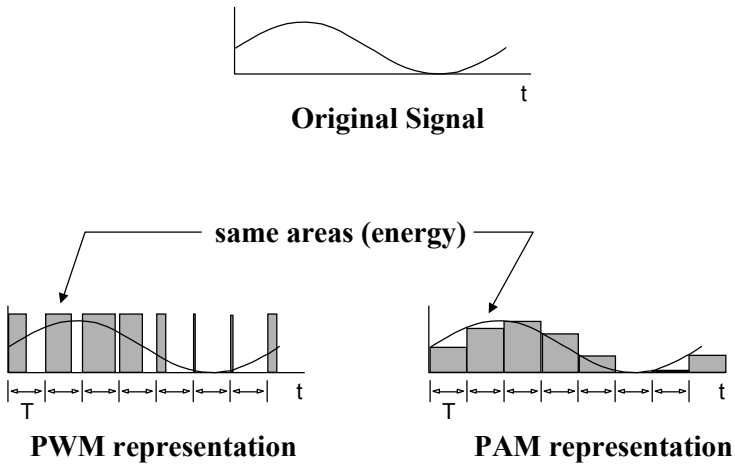
What is Pulse Width Modulation?

- **PWM is a scheme to represent a signal as a sequence of pulses**
  - **fixed carrier frequency**
  - **fixed pulse amplitude**
  - **pulse width proportional to instantaneous signal amplitude**
  - **PWM energy ≈ original signal energy**
- **Differs from PAM (Pulse Amplitude Mod.)**
  - **fixed width, variable amplitude**

5 - 19

Pulse width modulation (PWM) is a method for representing an analog signal with a digital approximation. The PWM signal consists of a sequence of variable width, constant amplitude pulses which contain the same total energy as the original analog signal. This property is valuable in digital motor control as sinusoidal current (energy) can be delivered to the motor using PWM signals applied to the power converter. Although energy is input to the motor in discrete packets, the mechanical inertia of the rotor acts as a smoothing filter. Dynamic motor motion is therefore similar to having applied the sinusoidal currents directly.

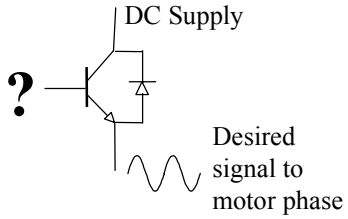
PWM Signal Representation



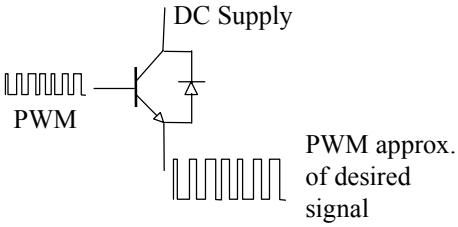
5 - 20

Why Use PWM in Digital Motor Control?

- **Desired motor phase currents or voltages are known**
- **Power switching devices are transistors**
  - **Difficult to control in proportional region**
  - **Easy to control in saturated region**
- **PWM is a digital signal ⇒ easy for DSP to output**



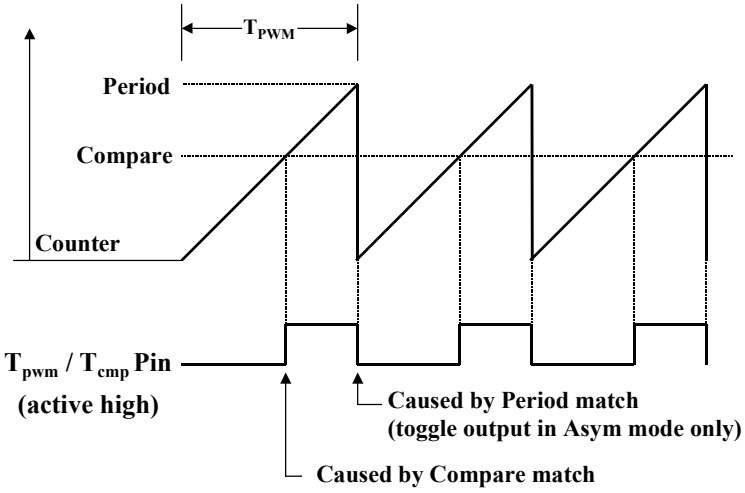
Unknown Gate Signal



Gate Signal Known with PWM

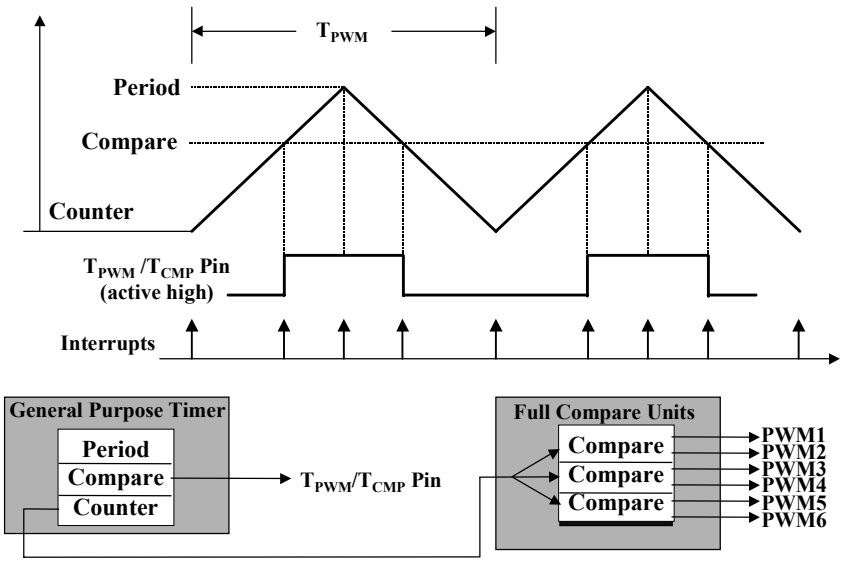
5 - 21

Asymmetric PWM Waveform



5 - 22

Symmetric PWM Waveform



5 - 23

Lab 5: PWM signal with variable duty-cycle

AIM :

- generates a PWM-signal at pin T1PWM (#130 , I/O-connector P1 - pin #12) in asymmetric mode
- uses T1-Compare Interrupt Service Routine to change the duty-cycle 'on the fly'
- Polarity of GPT1-Output : active low

Files :

PWM01.c	-	source file main c program
vectors.asm	-	jump table Interrupt Service Routine
regs2407.h	-	pointer definition to peripherals
PWM01.cmd	-	linker command file
rts2xx.lib	-	Runtime Library

5 - 24

New Registers involved in Lab 5:

- General Purpose Timer Control A : GPTCONA
- Timer 1 Control : T1CON
- Timer 1 Period : T1PR
- Timer 1 Compare : T1CMPR
- Timer 1 Counter : T1CNT
- EV-Manager A Interrupt Flag A: EVAIFRA
- EV-Manager A Interrupt Flag B: EVAIFRB
- EV-Manager A Interrupt Flag C: EVAIFRC
- EV-Manager A Interrupt Mask A: EVAIMRA
- EV-Manager A Interrupt Mask B: EVAIMRB
- EV-Manager A Interrupt Mask C: EVAIMRC
- Interrupt Flag : IFR
- Interrupt Mask : IMR
- Peripheral Interrupt Vector : PIVR

5 - 25

Initializing Sequence for LAB 5 :

1. Set Waitstates, Watchdog and System Control as in Lab2
2. In Output Control (MCRA) change bit 12 to drive T1PWM as output (MCRA.12 = 1)
3. Set **GPTCONA** to enable outputs (bit 6 = 1) and polarity of T1 to active low (bit 1-0 = 01b)
4. Load **T1PR** with the period and **T1CMPR** with the first duty #
5. Set **T1CON** to
   - continuous up counting mode (bit 12-11 = 10b)
   - Input Clock Prescaler CPUCLK/1 (bit 10-8 = 000b)
   - Clock Source Internal (bit 5-4 = 00b)
   - enable T1 Compare Operation (bit 1 = 1)
6. Clear all EV-Interrupt Flags (**EVAIFRx** = 0xFFFF)
7. Unmask (enable) T1-Compare -Interrupt (**EVAIMRA**-Bit 8 = 1)
8. Clear all global Interrupt Flags (**IFR** = 0xFFFF)
9. Unmask (enable) Global Interrupt Level 2 (**IMR**-Bit 2 = 1)
10. Enable all Interrupts (**INTM** = 0)
11. Enable Timer T1 (**T1CON** - Bit 6 = 1)

5 - 26

Lab 5: PWM Signal at T1PWM with variable duty-cycle

- **Objective**

  The objective of this lab is to practice the generation of a PWM-Signal. The PWM-signal's frequency is fixed to 50µs but the duty-cycle is varies with each interval ('change the duty cycle on the fly'). This method is often used to synthesise harmonic signals. The Timer T1 runs in asymmetric mode, the T1-Compare – Interrupt is used to load the next value into the compare-register.

  The registers, involved in this lab, are :

WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register
GPTCONA	- General Purpose Timer Control Register
T1CON	- Timer 2 Control Register
T1PR	- Timer 2 Period Register
T1CMPR	- Timer 2 Compare Register
T1CNT	- Timer 2 Counter Register
EVAIFRA	- Event Manager Interrupt Flag Group A
EVAIFRB	- Event Manager Interrupt Flag Group B
EVAIFRC	- Event Manager Interrupt Flag Group C
EVAIMRA	- Event Manager Interrupt Mask Group A
EVAIMRB	- Event Manager Interrupt Mask Group B
EVAIMRC	- Event Manager Interrupt Mask Group C
IFR	- Global Interrupt Flag Register
IMR	- Global Interrupt Mask Register
PIVR	- Peripheral Interrupt Vector Register

- Procedure

Open Files, Create Project File

1. Using Code Composer, create a new project , called myLab5.MAK in C:\One2407\Labs\Lab5. **NOTE** : There is a project file Lab5.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

2. Add the Source Code Files: PWM01.C and vectors.asm as well as the Linker Command File: PWM01.cmd from C:\One2407\Labs\Lab5\ to your project by clicking : Project → Add Files to project

3. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

4. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

5. Close the Build Options Menu by clicking OK

Build and Load

6. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. To open up more space, close any open files or windows that you do not need at this time.

7. Load the output file down to the EVM. Click : File → Load Program and choose the output file.

Test

8. Reset the DSP by clicking on Debug → Reset DSP

9. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "PWM01.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

10. Place a Breakpoint in the PWM01.c – window inside the interrupt service routine at line " EVAIFRA=(T1CINT<<8);" . Do this by placing the cursor at this line ,click right mouse and select : "Toggle Breakpoint". The line is highlighted with a pink bar to mark an active breakpoint.

11. Perform a real time run by clicking : Debug → Run or Key F5. Verify that the program halts when it reaches the Breakpoint line.

12. Open the Watch-Window by clicking : View → Watch Window. Add the Register T1CMPR to the watch window by clicking right mouse inside the watch window and select : " Insert New Expression" . Type in the Expression Window :

    *(int *)0x7402@data ; T1CMPR

13. Perform again a real time run by clicking : Debug → Run or Key F5. Verify that the watch window is updated with each interrupt service. The content of T1CMPR should toggle between 1 and 400 with each run. If you'd like to use other duty cycles : change the arrays table1[16] and table2[16], perform a new build and test.

14. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab5.wks"

15. Delete the active breakpoint by clicking on the button "Remove all Breakpoints", close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab5

```
/*********************************************************************/
/* Testprogram for PWM-generation at PIN T1PWM                       */
/* running on TMS320LF2407 EVAL -Board,                              */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/* date :  03.07.2000  ,  (C) Frank.Bormann@fh-zwickau.de            */
/*********************************************************************/
/* Testprogram for PWM-generation at PIN T1PWM                       */
/* polarity of T1PWM -Output : active low                            */
/* Interrupt to change the duty-cycle on the fly using               */
/* the T1-Compare Interrupt                                          */
/* make sure that external Ground-Switch on IOPB5 is off             */
/* program-name :  PWM01.c       / project : LAB5                    */
/*********************************************************************/
#include "regs2407.h"

#define LENGTH 16               /* length of lookup tables for duty-cycles   */
#define PERIOD 1474             /* period PWM-signal in 33ns steps = 50µs    */

/*************   SETUP for the MCRA - Register   ***************************/
#define MCRA15          0       /* 0 : IOPB7      1 : TCLKIN                 */
#define MCRA14          0       /* 0 : IOPB6      1 : TDIR                   */
#define MCRA13          0       /* 0 : IOPB5      1 : T2PWM                  */
#define MCRA12          1       /* 0 : IOPB4      1 : T1PWM                  */
#define MCRA11          0       /* 0 : IOPB3      1 : PWM6                   */
#define MCRA10          0       /* 0 : IOPB2      1 : PWM5                   */
#define MCRA9           0       /* 0 : IOPB1      1 : PWM4                   */
#define MCRA8           0       /* 0 : IOPB0      1 : PWM3                   */
#define MCRA7           0       /* 0 : IOPA7      1 : PWM2                   */
#define MCRA6           0       /* 0 : IOPA6      1 : PWM1                   */
#define MCRA5           0       /* 0 : IOPA5      1 : CAP3                   */
#define MCRA4           0       /* 0 : IOPA4      1 : CAP2/QEP2              */
#define MCRA3           0       /* 0 : IOPA3      1 : CAP1/QEP1              */
#define MCRA2           0       /* 0 : IOPA2      1 : XINT1                  */
#define MCRA1           0       /* 0 : IOPA1      1 : SCIRXD                 */
#define MCRA0           0       /* 0 : IOPA0      1 : SCITXD                 */
/*************   SETUP for the MCRB - Register   ***************************/
#define MCRB9           0       /* 0 : IOPD1      1 : XINT2/EXTSOC           */
#define MCRB8           1       /* 0 : CKLKOUT    1 : IOPD0                  */
#define MCRB7           0       /* 0 : IOPC7      1 : CANRX                  */
#define MCRB6           0       /* 0 : IOPC6      1 : CANTX                  */
#define MCRB5           0       /* 0 : IOPC5      1 : SPISTE                 */
#define MCRB4           0       /* 0 : IOPC4      1 : SPICLK                 */
#define MCRB3           0       /* 0 : IOPC3      1 : SPISOMI                */
#define MCRB2           0       /* 0 : IOPC2      1 : SPISIMO                */
#define MCRB1           1       /* 0 : BIO        1 : IOPC1                  */
#define MCRB0           1       /* 0 : XF         1 : IOPC0                  */
/*************   SETUP for the MCRC - Register   **************/
#define MCRC13          0       /* 0 : IOPF5      1 : TCLKIN2                */
#define MCRC12          0       /* 0 : IOPF4      1 : TDIR2                  */
#define MCRC11          0       /* 0 : IOPF3      1 : T4PWM/T4CMP            */
#define MCRC10          0       /* 0 : IOPF2      1 : T3PWM/T3CMP            */
#define MCRC9           0       /* 0 : IOPF1      1 : CAP6                   */
#define MCRC8           0       /* 0 : IOPF0      1 : CAP5/QEP3              */
#define MCRC7           0       /* 0 : IOPE7      1 : CAP4/QEP2              */
#define MCRC6           0       /* 0 : IOPE6      1 : PWM12                  */
#define MCRC5           0       /* 0 : IOPE5      1 : PWM11                  */
#define MCRC4           0       /* 0 : IOPE4      1 : PWM10                  */
#define MCRC3           0       /* 0 : IOPE3      1 : PWM9                   */
#define MCRC2           0       /* 0 : IOPE2      1 : PWM8                   */
#define MCRC1           0       /* 0 : IOPE1      1 : PWM7                   */
#define MCRC0           0       /* 0 : IOPE0      1 : CLKOUT                 */
```

```
/*************     SETUP for the WDCR - Register   ***************************/
#define WDDIS           1       /* 0 : Watchdog enabled   1: disabled           */
#define WDCHK2          1       /* 0 : System reset       1: Normal operation   */
#define WDCHK1          0       /* 0 : Normal operation   1: system  reset      */
#define WDCHK0          1       /* 0 : System reset       1: Normal operation   */
#define WDSP            7       /* Watchdog prescaler 111 : divide by 64        */
/*************************************************************************/
/*************     SETUP for the SCSR1 - Register  ***************************/
#define CLKSRC                  0       /* 0 : intern(30MHz)                    */
#define LPM                     0       /* 0 : Low power mode 0 if idle         */
#define CLK_PS                  1       /* 001 : PLL multiply by 2              */
#define ADC_CLKEN               0       /* 0 : No ADC-service in this test      */
#define SCI_CLKEN               0       /* 0 : No SCI-service in this test      */
#define SPI_CLKEN               0       /* 0 : No SPI-servide in this test      */
#define CAN_CLKEN               0       /* 0 : No CAN-service in this test      */
#define EVB_CLKEN               0       /* 0 : No EVB-Service in this test      */
#define EVA_CLKEN               1       /* 1 : Enable Clock for EVA-unit        */
#define ILLADR                  1       /* 1 : Clear ILLADR during startup      */
/*************************************************************************/
/*************     SETUP for the WSGR - Register   ***************************/
#define BVIS                    0       /* 10-9 : 00 Bus visibility OFF         */
#define ISWS                    0       /* 8 -6 : 000 0 Waitstates for IO       */
#define DSWS                    0       /* 5 -3 : 000 0 Waitstates  data        */
#define PSWS                    0       /* 2 -0 : 000 0 Waitstaes code          */
/*************************************************************************/
/*************     SETUP for the GPTCONA - Register  *********************/
#define GPTCON_T2TOADC      0
        /* 10-9  : T2TOADC = 00 : no ADC-Start by any GPT2-Event                */
#define GPTCON_T1TOADC      0
        /*  8-7  : T1TOADC = 00 : no ADC-Start by any GPT1-Event                */
#define GPTCON_TCOMPOE  1
        /*   6   : TCOMPOE = 1  : enable all 2 GPT compare outputs              */
#define GPTCON_T2PIN 0
        /*  3-2  : T2PIN = 01 : Pol. of GPT2 comp out=forced low                */
#define GPTCON_T1PIN 1
        /*  1-0  : T1PIN = 00 : Pol. of GPT1 comp out=active low                */
/*************************************************************************/
/*************     SETUP for the T1CON - Register  ***********************/
#define T1CON_FREESOFT      0
                /* 15-14 FREE, SOFT : 00 stop on JTAG-emulation suspend         */
#define T1CON_TMODE             2
                /* 12-11 : TMODE1,0   : 10
                        Count mode selection: Continuous up counting mode       */
#define T1CON_TPS               0
                /* 10-8  : TPS2-0     : 000
                        Input clock prescaler CPUCLK/1                          */
#define T1CON_TENABLE           1
                /* 6     : TENABLE    : 1
                        enable GPT1                                             */
#define T1CON_TCLKS             0
                /* 5-4   : TCLKS1,0   : 00
                        Clock source select : internal                          */
#define T1CON_TCLD              1
                /* 3-2   : TCLD1,0    : 01
                        Timer compare(active) register reload condition
                        when counter value is 0 or equal to period register     */
#define T1CON_TECMPR            1
                /* 1     :  TECMPR            : 1
                        enable timer compare operation                          */
/*************************************************************************/
```

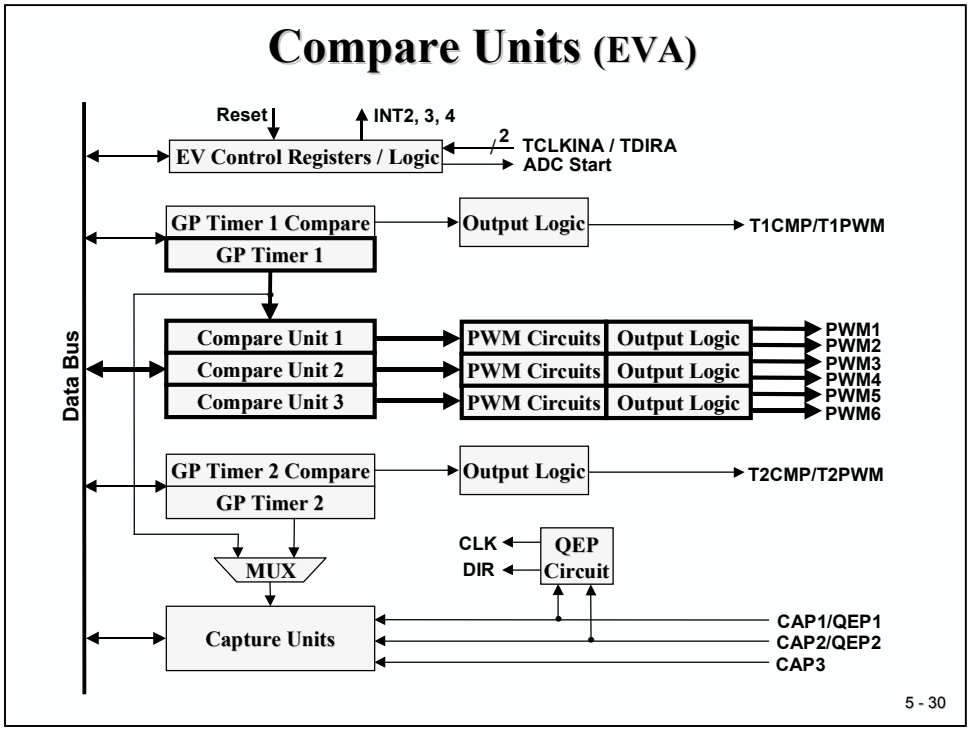
```
/*************    SETUP for the EVAIMRA - Register  ************************/
#define T1OFINT         0       /* 10 : Timer 1 overflow interrupt                  */
#define T1UFINT         0       /*  9 : Timer 1 underflow interrupt                 */
#define T1CINT          1       /*  8 : Timer 1 compare interrupt                   */
#define T1PINT          0       /*  7 : Timer 1 period interrupt                    */
#define  CMP3INT        0       /*  3 : Compare 3 interrupt                         */
#define CMP2INT         0       /*  2 : Compare 2 interrupt                         */
#define CMP1INT         0       /*  1 : Compare 1 interrupt                         */
#define PDPINT          0       /*  0 : Power Drive Protect Interrupt               */
/*****************************************************************************/
/*************    SETUP for the EVAIMRB - Register  ************************/
#define T2OFINT         0       /*  3 : Timer 2 overflow interrupt                  */
#define T2UFINT         0       /*  2 : Timer 2 underflow interrupt                 */
#define T2CINT          0       /*  1 : Timer 2 compare interrupt                   */
#define T2PINT          0       /*  0 : Timer 2 period interrupt                    */
/*****************************************************************************/
/*************    SETUP for the EVAMRC- Register  **************************/
#define CAP3INT         0       /*  2 : Capture Unit 3 interrupt                    */
#define CAP2INT         0       /*  1 : Capture Unit 2 Interrupt                    */
#define CAP1INT         0       /*  0 : Capture unit 1 interrupt                    */
/*****************************************************************************/
/*************    SETUP for the IMR - Register    ******************************/
#define INT6            0       /*  5 : Level INT6  is masked                       */
#define INT5            0       /*  4 : Level INT5  is masked                       */
#define INT4            0       /*  3 : Level INT4  is masked                       */
#define INT3            0       /*  2 : Level INT3  is masked                       */
#define INT2            1       /*  1 : Level INT2  is unmasked                     */
#define INT1            0       /*  0 : Level INT1  is masked                       */
/*****************************************************************************/
unsigned char RepIsrNo;         /* index into the actual lookup-table               */
unsigned char lookupTable;      /* number of  the actual lookup-table               */
unsigned int *pt;               /* pointer into the actual lookup-table             */
                /* for test purposes tables for duty-cycle's with alternating value's      */
unsigned int table1[16]={400,1,400,1,400,1,400,1,400,1,400,1,400,1,400,1};
unsigned int table2[16]={400,1,400,1,400,1,400,1,400,1,400,1,400,1,400,1};

interrupt void T1CMP_ISR(void)
{
 if((PIVR-0x0028)==0)           /*Verify interrupt-No. (28=T1CINT, Compare)       */
        {
        if(lookupTable==0 && RepIsrNo==LENGTH)
                {                               /* if end of table1 change to table2 */
                pt=table2;
                lookupTable=1;
                RepIsrNo=0;
                }

        if(lookupTable==1 && RepIsrNo==LENGTH)
                {                               /* if end of table2 change to table1 */
                pt=table1;
                lookupTable=0;
                RepIsrNo=0;
                }
        T1CMPR=*pt++;           /*      load next value from table to T1CMPR       */
        RepIsrNo++;             /*      increment table index                      */
        EVAIFRA=(T1CINT<<8);    /*      Clear only TCINT1-Interrupt        */
        }
}
```

```
void c_dummy1(void)
{
        while(1);                   /* Dummy ISR used to trap spurious interrupts      */
}

void main(void)
{
 asm (" setc INTM");       /*Disable all interrupts                                    */
 asm (" clrc SXM");        /*Clear Sign Extension Mode bit                             */
 asm (" clrc OVM");        /*Reset Overflow Mode bit                                   */
 asm (" clrc CNF");        /*Configure block B0 to data mem.                           */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                           /* setup waitstates                                        */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                           /* Initialize Watchdog-timer                               */

 SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                    (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                    (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                           /* Initialize SCSR1                                        */

 MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                   (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                   (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                           /* Initialize master control register B                    */


 MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
       (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
       (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
       (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                           /* Initialize master control register A                    */

 MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                   +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                   +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                   +(MCRC1<<1)+MCRC0);
                           /* Initialize master control register C                    */

 GPTCONA=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
       (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+
       (GPTCON_T1PIN)); /*       Initialize GP Timer Control                       */

 T1PR=PERIOD;              /*Initialize T1 period                                    */
 lookupTable=0;
 pt=table1;                /* Pointer into lookup table 1                            */
 T1CMPR=*pt++;
 RepIsrNo=1;               /* first load of T1CMPR outside of the ISR                */

 T1CNT=0x0000;             /* set start value for the counter T1                     */

 T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+
             (T1CON_TPS<<8)+(T1CON_TCLKS<<4)+
             (T1CON_TCLD<<2)+(T1CON_TECMPR<<1));

 EVAIFRA=0xFFFF;           /* clear EV Interrupt Flag Register Group A          */

 EVAIFRB=0xFFFF;           /* clear EV Interrupt Flag Register Group B          */
```

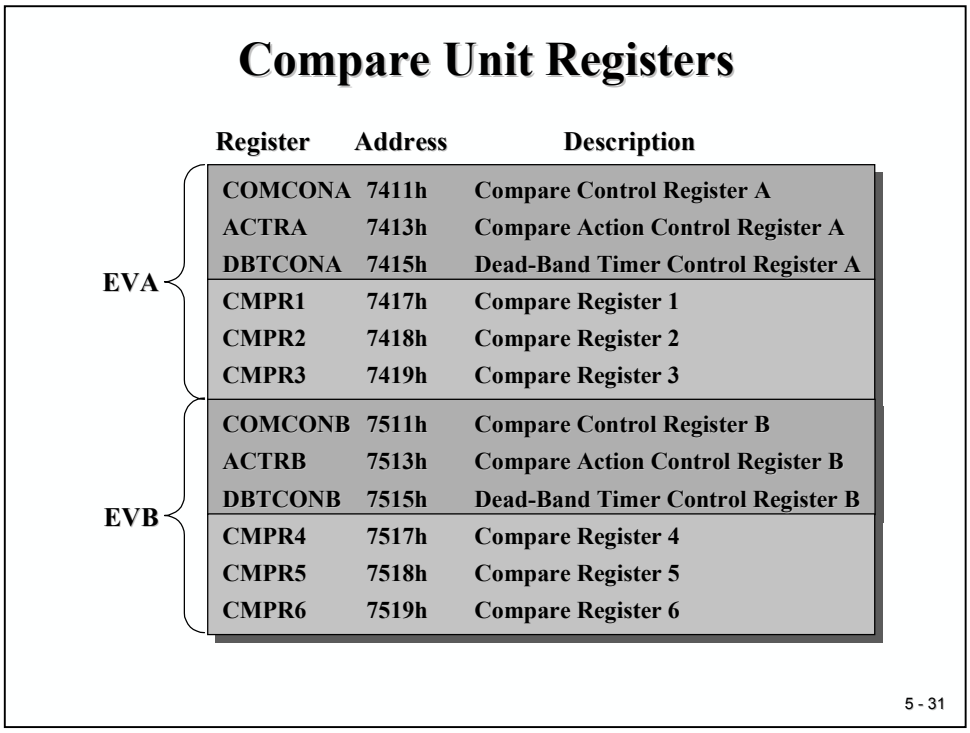
```
EVAIFRC=0xFFFF;       /* clear EV Interrupt Flag Register Group C          */

EVAIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
          (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                      /* EV Interrupt Mask Register Group A                */

EVAIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                      /* EV Interrupt Mask Register Group B                */

EVAIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                      /* EV Interrupt Mask Register Group C                */

IFR=0xFFFF;           /* Interrupt Flag Register , address 0x0006          */
                      /* Reset  all core interrupts                        */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                      /* Interrupt Mask Register                           */

asm (" clrc INTM");   /* Enable all unmasked interrupts                    */

T1CON=T1CON+(T1CON_TENABLE<<6); /* enable GPT1 now                         */

while(1);             /* endless loop, ISR's to change the duty-cycle      */
}
```

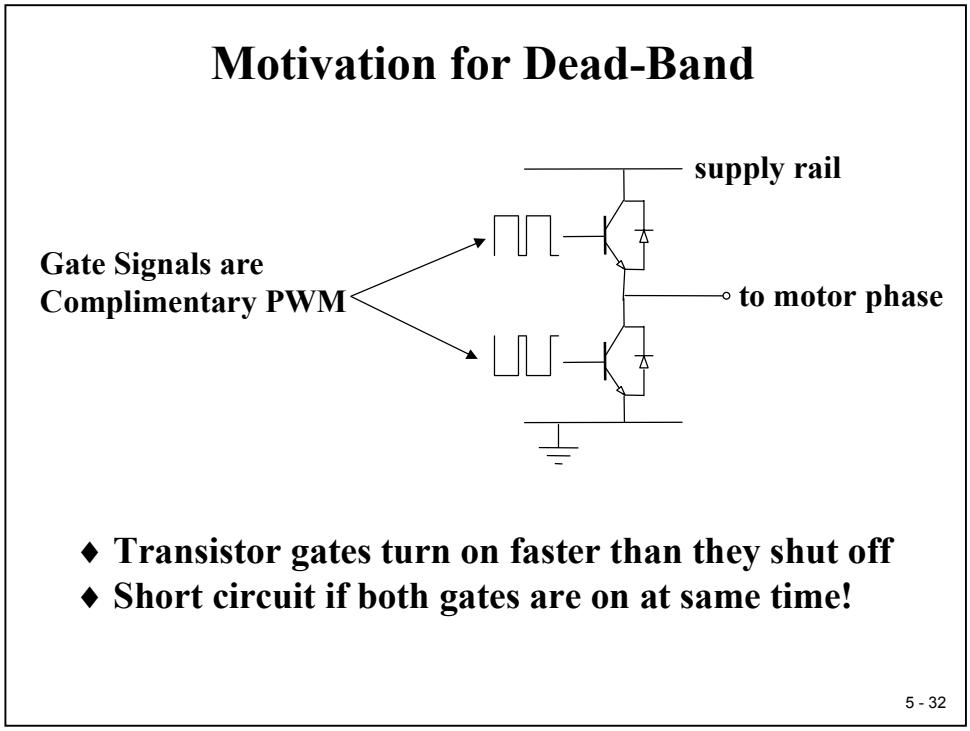
```
        .title    "vectors.asm"
        .ref      _c_int0,_c_dummy1,_T1CMP_ISR
        .sect     ".vectors"

reset:          b       _c_int0
int1:           b       _c_dummy1
int2:           b       _T1CMP_ISR
int3:           b       _c_dummy1
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
sw_int12:       b       _c_dummy1
sw_int13:       b       _c_dummy1
sw_int14:       b       _c_dummy1
sw_int15:       b       _c_dummy1
sw_int16:       b       _c_dummy1
trap:           b       _c_dummy1
nmint:          b       _c_dummy1
emu_trap:       b       _c_dummy1
sw_int20:       b       _c_dummy1
sw_int21:       b       _c_dummy1
sw_int22:       b       _c_dummy1
sw_int23:       b       _c_dummy1
```

Compare Units



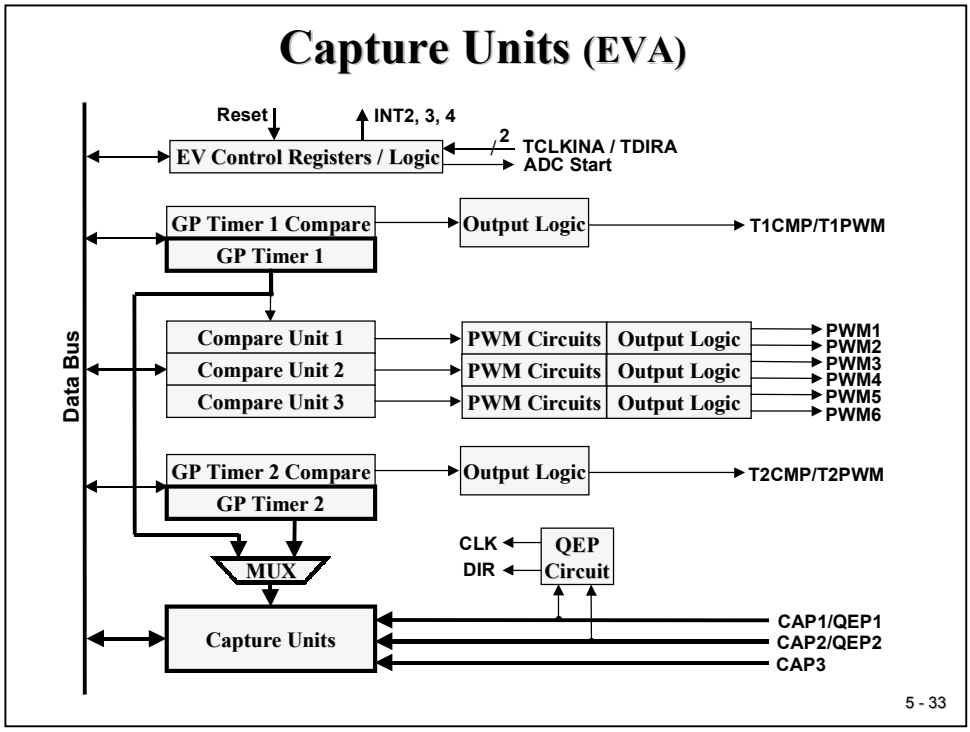
There are three compare units. Each compare unit has two associated PWM outputs. They have capabilities beyond the GP timer compares, and feature programmable hardware deadband. The time base for the compare units is provided by GP timer 1.





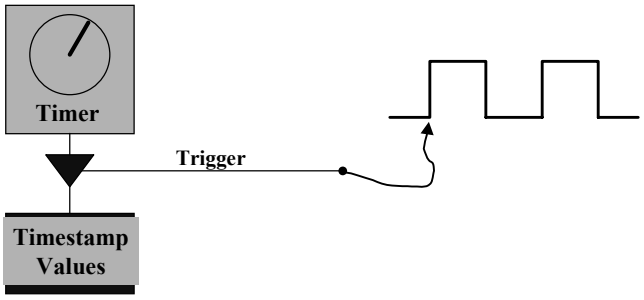
Dead-band control provides a convenient means of combating current shoot-through problems in a power converter. Shoot-through occurs when both the upper and lower gates in the same phase of a power converter are open simultaneously. This condition shorts the power supply and results in a large current draw. Shoot-through problems occur because transistors open faster than they close, and because high-side and low-side power converter gates are typically switched in a complimentary fashion. Although the duration of the shoot-through current path is finite during PWM cycling, (i.e. the closing gate will eventually shut), even brief periods of a short circuit condition can produce excessive heating and over stress in the power converter and power supply.

Capture Units



There are three capture units, and each is associated with a capture input pin. Each capture unit can choose GP timer 1 or 2 as its time base. The value of GP timer 1 or 2 is captured and stored in the corresponding 2-level-deep FIFO stack when a specified transition is detected on a capture input pin.

Capture Units



- **Capture units timestamp transitions on capture input pins**
- **Three capture units - each associated with a capture input pin**

5 - 34

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:**

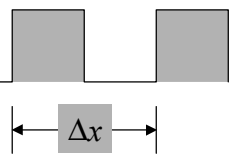
Problem: At low speeds, calculation of speed based on a measured position change at fixed time intervals produces large estimate errors

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

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



5 - 35

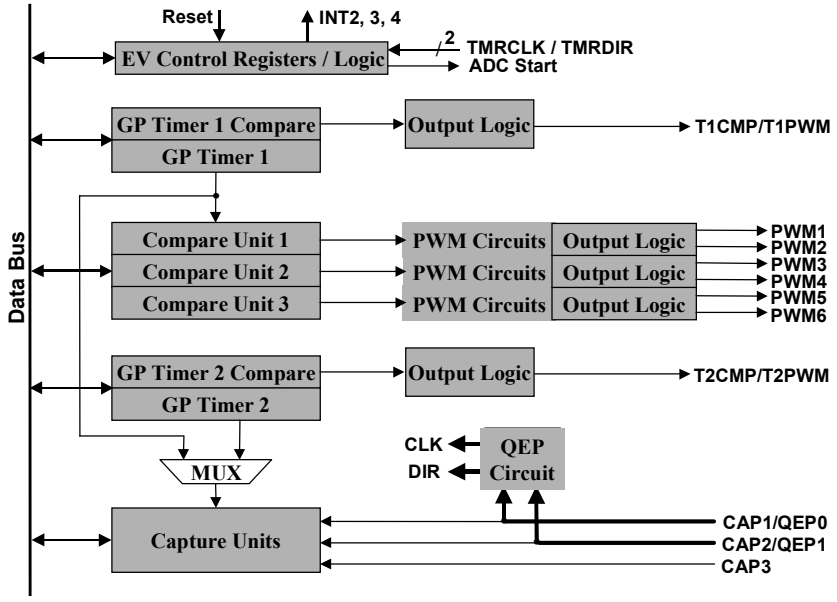
Capture Units Registers

Register	Address	Description
CAPCON	7420h	Capture Control Register
CAPFIFO	7422h	Capture FIFO Status Register
CAP1FIFO	7423h	Two-Level Deep FIFO 1 Stack
CAP2FIFO	7424h	Two-Level Deep FIFO 2 Stack
CAP3FIFO	7425h	Two-Level Deep FIFO 3 Stack
CAP1FBOT	7427h	Bottom Register of FIFO 1
CAP2FBOT	7428h	Bottom Register of FIFO 2
CAP3FBOT	7429h	Bottom Register of FIFO 3

5 - 36

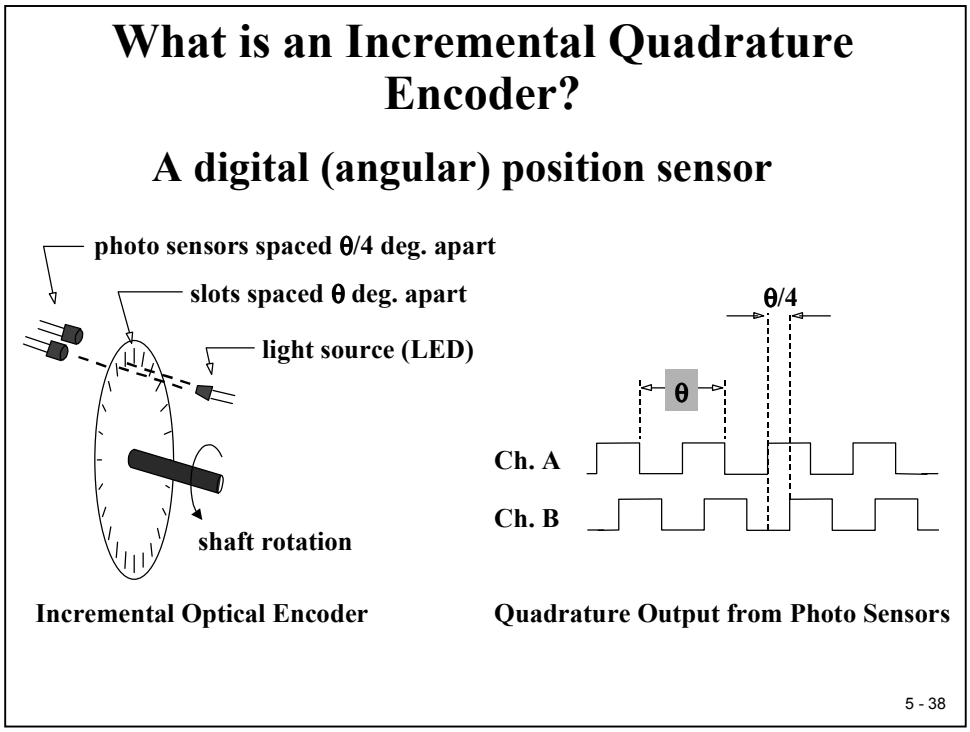
Quadrature Encoder Pulse (QEP)

Quadrature Encoder Pulse



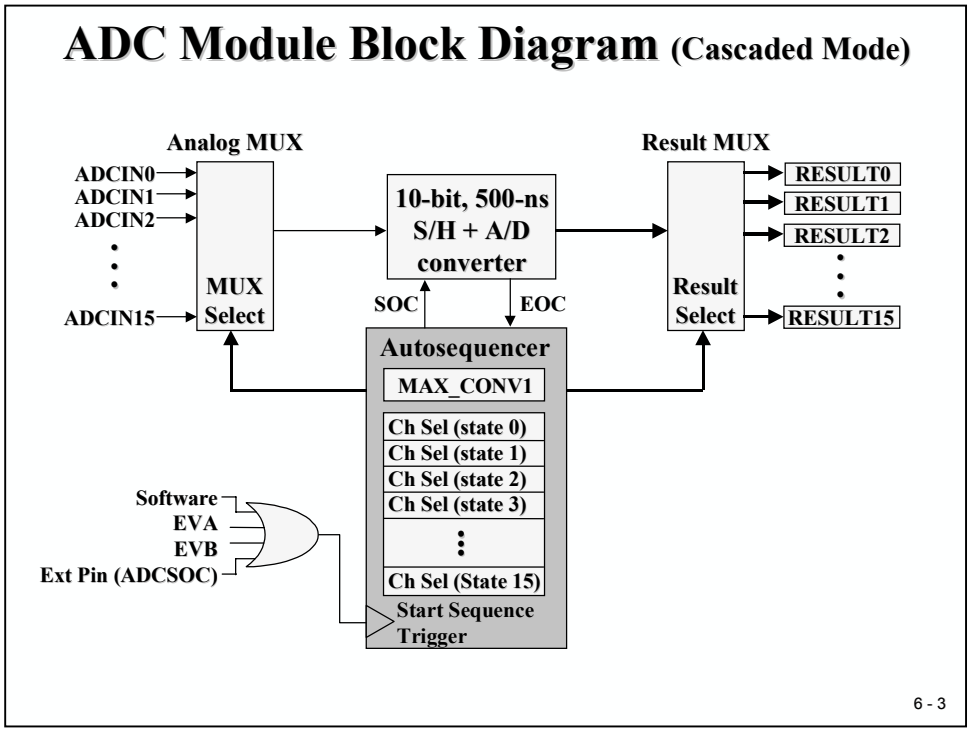
5 - 37

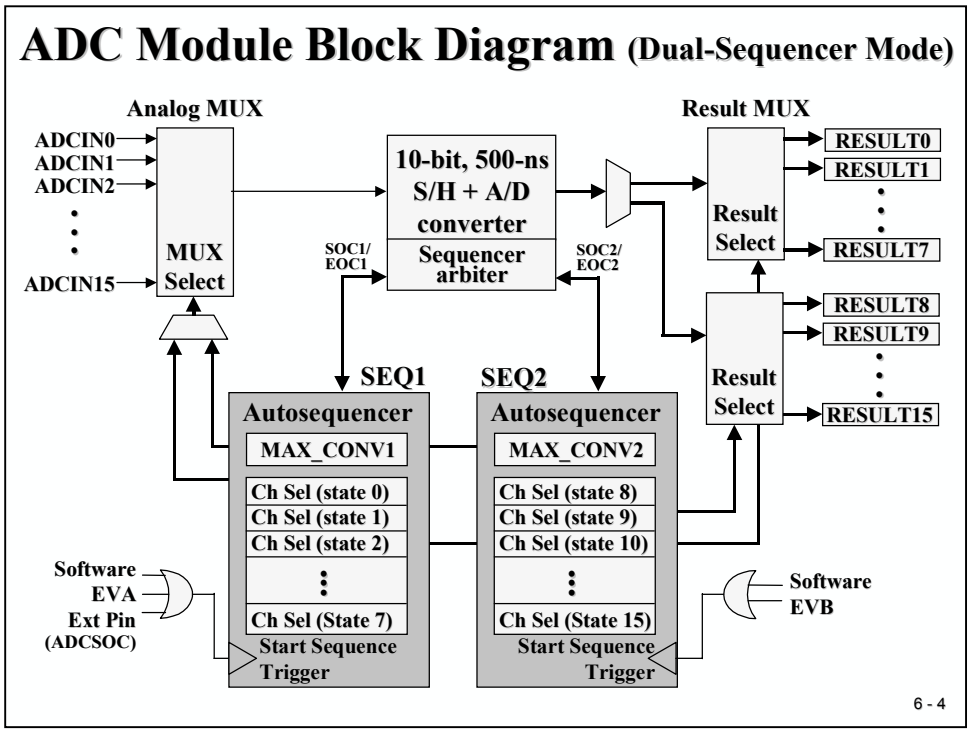
The QEP circuit, when enabled, decodes and counts the quadrature encoded input pulses on pins CAP1/QEP0 and CAP2/QEP1. The QEP circuit can be used to interface with an optical encoder to get position and speed information from a rotating machine. When the QEP circuit is enabled, the capture function on CAP1 and CAP2 pins is disabled. The QEP time base is provided by GP timer 2.



Introduction

This module discusses the operation of the analog-to-digital converter.





The LF2407 has a successive approximation 10-bit analogue-to-digital converter on-chip with a built-in sample-and-hold circuit. A total of 16 analogue multiplexed input channels are available. The fastest conversion time is 500ns with a 30 MHz clock and a prescale of 1. The converter is also capable of up to sixteen consecutive conversions performed 500ns apart. There are two operating modes, the cascaded mode and the dual sequencer mode. In cascaded mode the two sequencers are operated in a single chain of up to sixteen consecutive conversions. Each step of this chain can be performed with another AD-input or with the same input. This is done with the help of four registers for channel selection. In dual sequencer mode there are two independent sequencers with 8 states each.

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**

6 - 5

Analog-to-Digital Converter Registers

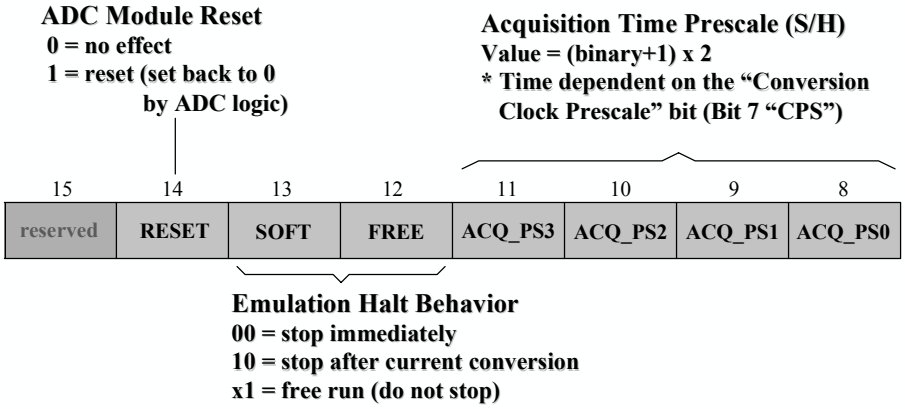
Register	Address	Description
ADCTRL1	70A0h	ADC Control Register 1
ADCTRL2	70A1h	ADC Control Register 2
MAX_CONV	70A2h	Maximum Conversion Channels Register
CHSELSEQ1	70A3h	Channel Select Sequencing Control Register 1
CHSELSEQ2	70A4h	Channel Select Sequencing Control Register 2
CHSELSEQ3	70A5h	Channel Select Sequencing Control Register 3
CHSELSEQ4	70A6h	Channel Select Sequencing Control Register 4
AUTO_SEQ_SR	70A7h	Autosequence Status Register
RESULT0	70A8h	Conversion Result Buffer Register 0
RESULT1	70A9h	Conversion Result Buffer Register 1
RESULT2	70AAh	Conversion Result Buffer Register 2
: :	: :	: : : : :
RESULT14	70B6h	Conversion Result Buffer Register 14
RESULT15	70B7h	Conversion Result Buffer Register 15
CALIBRATION	70B8h	Calibration Result (next conversion correction)

6 - 6

Analog-to-Digital Converter Registers

ADC Control Register 1 - Upper Byte

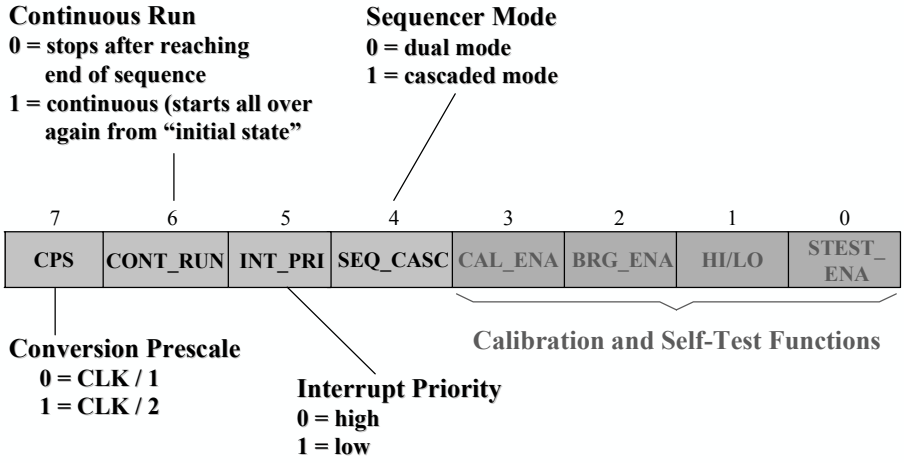
ADCTRL1 @ 70A0h



6 - 7

ADC Control Register 1 - Lower Byte

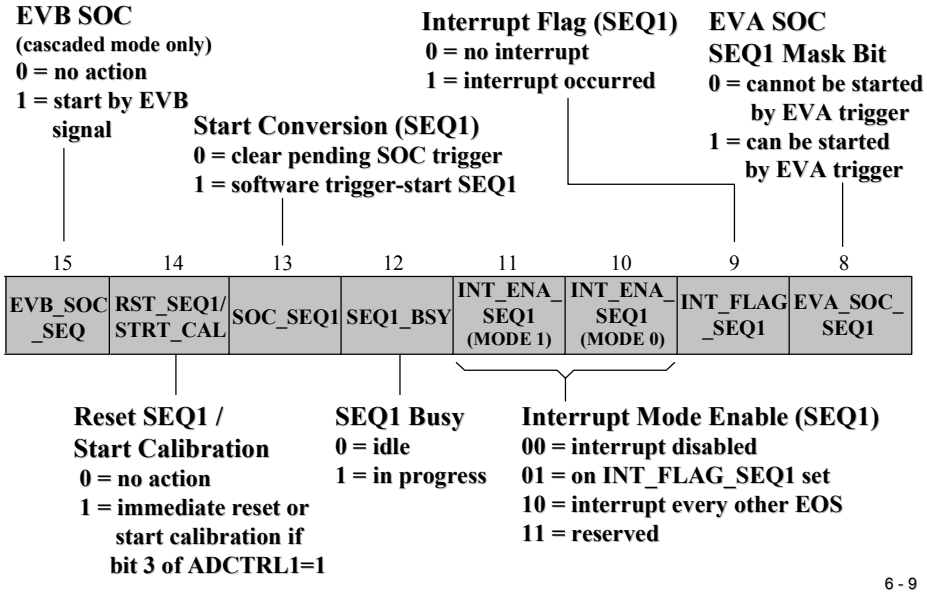
ADCTRL1 @ 70A0h



6 - 8

ADC Control Register 2 - Upper Byte

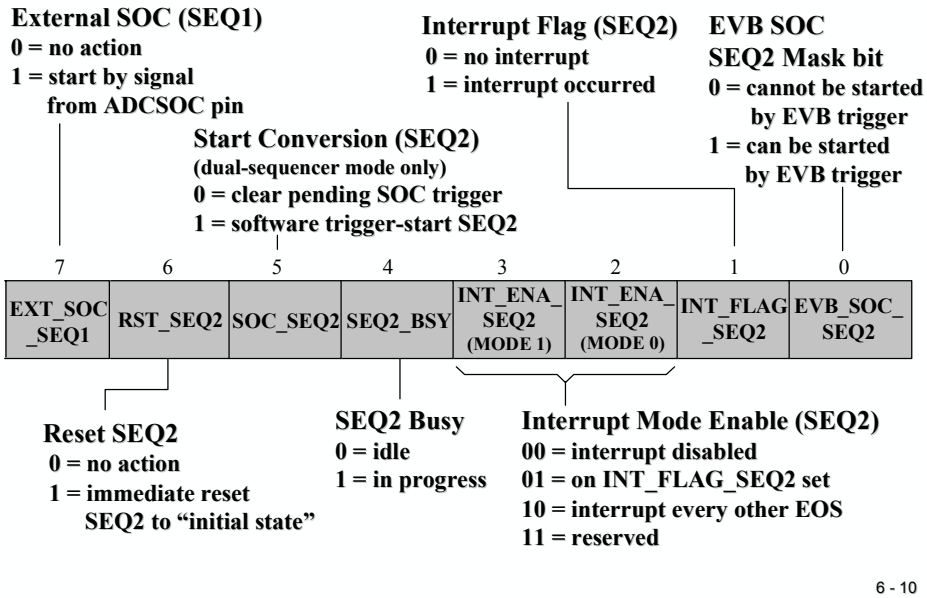
ADCTRL2 @ 70A1h



6 - 9

ADC Control Register 2 - Lower Byte

ADCTRL2 @ 70A1h

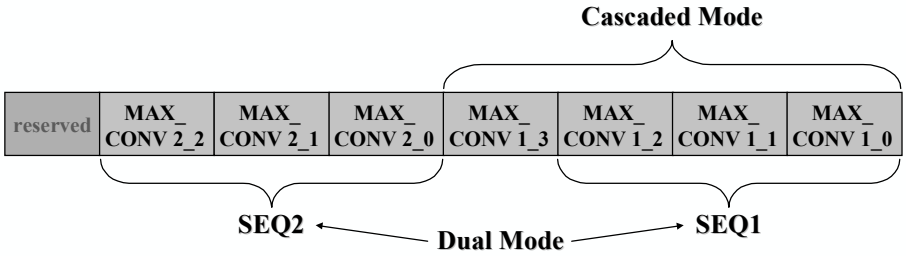


6 - 10

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

6 - 11

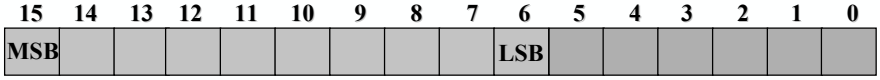
ADC Input Channel Select Sequencing Control Register

	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

6 - 12

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

6 - 13

Lab 6: Two Channel Analogue Conversion initiated by Timer GPT1

AIM :

- perform an AD-conversion of ADC0 and ADC8 initiated by GPT1-period of 0.2 sec
- ADC0 and ADC8 are connected to two potentiometers (0 ... 3,3V)
- no GPT1-interrupt-service -> Auto-start of ADC with T1TOADC-bit !!
- ADC-Interrupt Service reads the results
- main loop shows alternately the result as light-bar on LED's

Files :

F2407ADC.c	- source file main C-Program
vectors.asm	- jump table Interrupt Service Routine
regs2407.h	- pointer definition to peripherals
F2407ADC.cmd	- memory map definition
rts2xx.lib	- Runtime Library

6 - 14

New Registers involved in Lab 6:

General Purpose Timer Control :	:	GPTCONA
Timer 1 Control	:	T1CON
Timer 1 Period	:	T1PR
Timer 1 Compare	:	T1CMPR
Timer 1 Counter	:	T1CNT
Interrupt Flag	:	IFR
Interrupt Mask	:	IMR
Peripheral Interrupt Vector	:	PIVR
AD-Control 1	:	ADCTRL1
AD-Control 2	:	ADCTRL2
Calibration-Register	:	CALIBRATION
Channel Select Sequencer 1	:	CHSELSEQ1
Max. number of conversions	:	MAXCONV
ADC - Result 0	:	RESULT0
ADC - Result 1	:	RESULT1

6 - 15

Initializing Sequence for LAB 6 :

1. Set Waitstates, Watchdog and System Control as before
2. In Output Control (MCRx) set all bits to I/O
3. Set **GPTCONA** to disable outputs (bit 6 = 0) ,polarity of T2 & T1 to forced low (bit 3-0 = 0000b) , enable **ADC-Start by Timer-Period - Event** (bits 8-7 = 10b)
4. Set Timer T1 as before (**T1PR, T1CMPR, T1CON**) to generate a time base of 0.2 sec

5. Init **ADCTRL1** :
   - reset entire ADC module (bit 15 = 0)
   - acquisition time 16 x TCLK (bits 11-8 = 111)
   - ADC-Clock prescaler = CLK/2 (bit 7= 1)
   - no continuous mode (bit 6 = 0)
   - ADC interrupt with high priority (INT1) (bit 5 = 0)
   - dual sequencer mode (bit 4 = 0)
   - no calibration mode (bits 3-0 = 0010)

6 - 16

Initializing Sequence for LAB 6 :

6. Init **CHSELSEQ1:**
   - CONV00 (bit 3-0) = ADCIN0 (0000)
   - CONV01 (bit 7-4) = ADCIN8 (1000)
7. Init **ADCTRL2** :
   - EVA starts conversion sequencer 1(bit 8 = 1)
   - Clear pending SOC trigger (bit 13 = 0)
   - Interrupt Mode 1 - immediately (bit11,10 = 01)
   - disable sequencer 2 ant its interrupts
8. Init **MAXCONV**
   - 2 subsequent conversions (= 1)
9. Clear all global Interrupt Flags (**IFR** = 0xFFFF)
10. Unmask (enable) Global Interrupt Level 1 (**INT1** = 1)
11. Enable all Interrupts (**INTM** = 0)
12. Enable Timer T1 (**T1CON** - Bit 6 = 1)

6 - 17

Lab 6: Two Channel ADC started by GP Timer1

▪ Objective

The objective of this lab is to practice the use of the ADC-unit. We will use the dual sequencer mode and start the conversions automatically by the period-event of Timer T1 without interrupting the main-control loop ! The start of the ADC does not depend on an interrupt-software-routine, this guarantees a synchronized behaviour of the Conversion ! Timer T1 is initialized for a period of 0.2s . ADCIN0 and ADCIN8 are connected to two potentiometers, therefore the input analogue voltages on both channels can vary between 0 and +3.3V. We will use the sequencer 1 only to perform two continous conversions. On completion of the conversion a interrupt - service will take out the data's from the ADC registers RESULT0 ans RESULT1. This is not time critical, because it can be done at some point before the next Interrupt occurs.

The result of the AD-Conversions is then converted into an output for the LED's connected to GPIO-Port D. A value of 3.3V will switch on all 8 LED's while a value of 0V will switch off all LED's.

The registers, involved in this lab, are :

WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1
GPTCONA	- General Purpose Timer Control Register
T1CON	- Timer 1 Control Register
T1PR	- Timer 1 Period Register
T1CMPR	- Timer 1 Compare Register
T1CNT	- Timer 1 Counter Register
IFR	- Global Interrupt Flag Register
IMR	- Global Interrupt Mask Register
PIVR	- Peripheral Interrupt Vector Register
ADCTRL1	- AD - Control Register 1
ADCTRL2	- AD -Control Register 2
CALIBRATION	- AD-Calibration Control Register
CHSELSEQ1	- AD-Channel Select Sequencer1 Register
MAXCONV	- AD-Maximum Conversions Number Register
RESULT0	- AD Result of Sequencer 1 State 0
RESULT1	- AD Result of Sequencer 1 State 1

▪ Procedure

Open Files, Create Project File

1. Using Code Composer, create a new project , called myLab6.MAK in C:\One2407\Labs\Lab6. **NOTE** : There is a project file Lab6.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

2. Add the Source Code Files: F2407ADC.C and vectors.asm as well as the Linker Command File: F2407ADC.cmd from C:\One2407\Labs\Lab6\ to your project by clicking : Project → Add Files to project

3. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

4. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

5. Close the Build Options Menu by clicking OK

Build and Load

6. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary.

7. Load the output file down to the EVM. Click : File → Load Program and choose the desired output file.

Test

8. Reset the DSP by clicking on Debug → Reset DSP

9. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407AD2.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

10. Perform a real time run. The LED's should show two light-bar's alternately. Turn one of the two potentiometers and watch the changing LED-pattern.

11. Exercise to place a probe point ; watch the registers RESULT0, RESULT1 and the global variables ADC0_result , ADC1_result as in previous Labs.

12. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab6.wks"

13. In case you placed breakpoints or probe points delete them by clicking on the particular buttons, close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab6

Source Code Files Lab Exercise 6

F2407ADC.c

```
/*********************************************************************/
/* Testprogram for internal ADC                                      */
/* running on TMS320LF2407 EVM                                       */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/* date :  17.08.2000                                                */
/*********************************************************************/
/* Potentiometers ( 0 to 3,3 V ) connected to ADCIN00 and ADCIN08    */
/* 8 LED's connected to Port E0...E7 ; LED-on : 1  LED off : 0       */
/* Timer GPT1 generates a 0.2second - period                         */
/* the period-match causes the start of the Sequencer1               */
/* ADConverter#1 :  ADCin00   ADConverter#2 : ADCin08                */
/* With the EOC the ADC_ISR interrupt service routine is entered     */
/* The results are read from the RESULT0 and RESULT1 registers       */
/* the function show_ADC shows the last result as a light-beam on LED's */
/* program-name :  F2407ADC2.c  / project : Lab6                     */
/*********************************************************************/

#include "regs2407.h"

/*************    SETUP for the MCRA - Register   ***************************/
#define MCRA15          0       /* 0 : IOPB7       1 : TCLKIN              */
#define MCRA14          0       /* 0 : IOPB6       1 : TDIR                */
#define MCRA13          0       /* 0 : IOPB5       1 : T2PWM               */
#define MCRA12          0       /* 0 : IOPB4       1 : T1PWM               */
#define MCRA11          0       /* 0 : IOPB3       1 : PWM6                */
#define MCRA10          0       /* 0 : IOPB2       1 : PWM5                */
#define MCRA9           0       /* 0 : IOPB1       1 : PWM4                */
#define MCRA8           0       /* 0 : IOPB0       1 : PWM3                */
#define MCRA7           0       /* 0 : IOPA7       1 : PWM2                */
#define MCRA6           0       /* 0 : IOPA6       1 : PWM1                */
#define MCRA5           0       /* 0 : IOPA5       1 : CAP3                */
#define MCRA4           0       /* 0 : IOPA4       1 : CAP2/QEP2           */
#define MCRA3           0       /* 0 : IOPA3       1 : CAP1/QEP1           */
#define MCRA2           0       /* 0 : IOPA2       1 : XINT1               */
#define MCRA1           0       /* 0 : IOPA1       1 : SCIRXD              */
#define MCRA0           0       /* 0 : IOPA0       1 : SCITXD              */
/*********************************************************************/
/*************    SETUP for the MCRB - Register   ***************************/
#define MCRB9           0       /* 0 : IOPD1       1 : XINT2/EXTSOC        */
#define MCRB8           1       /* 0 : CKLKOUT  1 : IOPD0                  */
#define MCRB7           0       /* 0 : IOPC7       1 : CANRX               */
#define MCRB6           0       /* 0 : IOPC6       1 : CANTX               */
#define MCRB5           0       /* 0 : IOPC5       1 : SPISTE              */
#define MCRB4           0       /* 0 : IOPC4       1 : SPICLK              */
#define MCRB3           0       /* 0 : IOPC3       1 : SPISOMI             */
#define MCRB2           0       /* 0 : IOPC2       1 : SPISIMO             */
#define MCRB1           1       /* 0 : BIO         1 : IOPC1               */
#define MCRB0           1       /* 0 : XF          1 : IOPC0               */
/*********************************************************************/
/*************    SETUP for the MCRC - Register   ***************************/
#define MCRC13          0       /* 0 : IOPF5       1 : TCLKIN2             */
#define MCRC12          0       /* 0 : IOPF4       1 : TDIR2               */
#define MCRC11          0       /* 0 : IOPF3       1 : T4PWM/T4CMP         */
#define MCRC10          0       /* 0 : IOPF2       1 : T3PWM/T3CMP         */
#define MCRC9           0       /* 0 : IOPF1       1 : CAP6                */
```

```
#define MCRC8           0       /* 0 : IOPF0      1 : CAP5/QEP3                */
#define MCRC7           0       /* 0 : IOPE7      1 : CAP4/QEP2                */
#define MCRC6           0       /* 0 : IOPE6      1 : PWM12                    */
#define MCRC5           0       /* 0 : IOPE5      1 : PWM11                    */
#define MCRC4           0       /* 0 : IOPE4      1 : PWM10                    */
#define MCRC3           0       /* 0 : IOPE3      1 : PWM9                     */
#define MCRC2           0       /* 0 : IOPE2      1 : PWM8                     */
#define MCRC1           0       /* 0 : IOPE1      1 : PWM7                     */
#define MCRC0           0       /* 0 : IOPE0      1 : CLKOUT                   */
/*********************************************************************/
/*************    SETUP for the WDCR - Register   *********************/
#define WDDIS           1       /* 0 : Watchdog enabled   1: disabled          */
#define WDCHK2          1       /* 0 : System reset       1: Normal OP         */
#define WDCHK1          0       /* 0 : Normal Oper.       1: sys reset         */
#define WDCHK0          1       /* 0 : System reset       1: Normal OP         */
#define WDSP            7       /* Watchdog prescaler 7 : div 64               */
/*********************************************************************/
/*************    SETUP for the SCSR1 - Register  *********************/
#define CLKSRC          0       /* 0 : intern(30MHz)                           */
#define LPM             0       /* 0 : Low power mode 0 if idle                */
#define CLK_PS          1       /* 001 : PLL multiply by 2                     */
#define ADC_CLKEN       1       /* 1 : Use ADC-service in this test            */
#define SCI_CLKEN       0       /* 0 : No SCI-service in this test             */
#define SPI_CLKEN       0       /* 0 : No SPI-servide in this test             */
#define CAN_CLKEN       0       /* 0 : No CAN-service in this test             */
#define EVB_CLKEN       0       /* 0 : No EVB-Service in this test             */
#define EVA_CLKEN       1       /* 1 : Use EVA-Service in this test            */
#define ILLADR          1       /* 1 : Clear ILLADR during startup             */
/*********************************************************************/
/*************    SETUP for the ADCTRL1 - Register  *******************/
#define RESET           0       /* 15 : 1 Resets entire ADC Module             */
#define SOFTFREE        2       /* 13-12 : 10 complete ADC before halt         */
#define ACQ_PS          7       /* 11-8 : Acquisition time 16 x TClk           */
#define CPS             1       /* 7 : 1 ADC logic Clock = CLK/2               */
#define CONT_RUN        0       /* 6 : 0 No Continuous Run                     */
#define INT_PRI         0       /* 5 : 0 High ADC interrupt priority           */
#define SEQ_CASC        0       /* 4 : 0 Dual-Sequencer mode                   */
#define CAL_ENA         0       /* 3 : 0 Calibration mode disabled             */
#define BRG_ENA         0       /* 2 : 0 Full reference Volt. to ADC           */
#define HILO            1       /* 1 : 0 VREFLO as Test Voltage                */
#define STEST_ENA       0       /* 0 : 0 Self-Test mode disabled               */
/*********************************************************************/
/*************    SETUP for the ADCTRL2 - Register  *******************/
#define EVB_SOC_SEQ             0       /* 15 : 1 EVB starts Cascaded Sequ.    */
#define RST_SEQ1                0       /* 14 : 1 Reset Sequencer 1            */
#define  SOC_SEQ1               0       /* 13 : 0 Clears a pending SOC trig    */
#define  INT_ENA_SEQ1           1       /* 11-10 : 1 Interrupt Mode 1          */
#define  EVA_SOC_SEQ1           1       /* 8: 1 EVA starts Sequencer1          */
#define EXT_SOC_SEQ1            0       /* 7: 1 ADCSOC Pin starts Sequencer1   */
#define RST_SEQ2                0       /* 6 : 1 Reset Sequencer 2             */
#define  SOC_SEQ2               0       /* 5 : 0 Clears a pending SOC trig     */
#define  INT_ENA_SEQ2           0       /* 3-2 : 0 Interrupt SEQ2 disabled     */
#define  EVB_SOC_SEQ2           0       /* 8: 1 EVB starts Sequencer1          */
/*************    SETUP for the GPTCONA - Register  *******************/
#define GPTCONA_T2TOADC 0       /* 10-9 : 0 no ADC-Start by GPT2-Event     */
#define GPTCONA_T1TOADC 2       /* 8-7  : 2 ADC-Start by GPT1-PeriodEvent*/
#define GPTCONA_TCOMPOE 0       /* 6 : 0 disable all 2 GPT compare outputs  */
#define GPTCONA_T2PIN   0       /* 3-2 : 00 Pol.  GPT2 comp out=forced low*/
#define GPTCONA_T1PIN   0       /* 1-0 : 00 Pol.  GPT1 comp out=forced low*/
/*********************************************************************/
```

```
/**************     SETUP for the T1CON - Register  ***************************/
#define T1CON_FREESOFT      0       /* 15-14 : 0 stop on JTAG suspend          */
#define T1CON_TMODE         2       /* 12-11 : 2 Continuous up counting        */
#define T1CON_TPS           7       /* 10-8  : 7 CPUCLK/128                    */
#define T1CON_TENABLE       1       /* 6     : 1 Timer1 enable                 */
#define T1CON_TCLKS         0       /* 5-4   : 0 Clock source: internal        */
#define T1CON_TCLD          1       /* 3-2 : 1 reload when 0 or = T1PR         */
#define T1CON_TECMPR        0       /* 1 : disable timer compare               */
/*****************************************************************************/
/**************     SETUP for the WSGR - Register   ***************************/
#define BVIS                0       /* 10-9 : 00 Bus visibility OFF            */
#define ISWS                0       /* 8 -6 : 000 0 Waitstates for IO          */
#define DSWS                0       /* 5 -3 : 000 0 Waitstates  data           */
#define PSWS                0       /* 2 -0 : 000 0 Waitstaes code             */
/**************     SETUP for the IMR - Register    ***************************/
#define INT6                0       /*  5 : Level INT6  is masked              */
#define INT5                0       /*  4 : Level INT5  is masked              */
#define INT4                0       /*  3 : Level INT4  is masked              */
#define INT3                0       /*  2 : Level INT3  is masked              */
#define INT2                0       /*  1 : Level INT2  is masked              */
#define INT1                1       /*  0 : Level INT1  is unmasked            */
/*****************************************************************************/

#define PERIOD 46080            /* T1PERIOD= 33,9084ns * 128 * 46080 = 0.2s    */

unsigned int ADC0_result,ADC1_result;

void c_dummy1(void)
{
        while(1);               /*Dummy ISR used to trap spurious interrupts*/
}

void show_ADC(unsigned int result)
/* show the result of the AD-conversion on 8 LED's on Port E0-E7               */
/* the result will be show as light-beam                                       */
/* 00 0000 0000  =  all LED'S off                                              */
/* 00 01xx xxxx   =  LED 1 on                                                  */
/* 00 1xxx xxxx  =  LED 1+2 on                                                 */
/*     ...                                                                     */
/* 11 1xxx xxxx  =  all LED's on                                               */
{
        result>>=6;
        switch(result) {
                case 0 : PEDATDIR=0xFF00;break;
                case 1 : PEDATDIR=0xFF01;break;
                }
        result>>=1;
        switch(result) {
                case 1 : PEDATDIR=0xFF03;break;
                case 2 : PEDATDIR=0xFF07;break;
                case 3 : PEDATDIR=0xFF0F;break;
                case 4 : PEDATDIR=0xFF1F;break;
                case 5 : PEDATDIR=0xFF3F;break;
                case 6 : PEDATDIR=0xFF7F;break;
                case 7 : PEDATDIR=0xFFFF;break;
                }
}
```

```
interrupt void ADC_ISR(void)
{
        if((PIVR-0x0004)==0)    /*Verify type of interrupt ( 4 = ADC )      */
        {
                ADC0_result=RESULT0>>6;
                ADC1_result=RESULT1>>6;
        ADCTRL2 |= 0x4200;      /* clear ADC-Sequencer1-Interrupt flag     */
                                /* and reset Sequencer 1                   */
        }
}

void main(void)
{
        asm (" setc INTM");     /*Disable all interrupts                    */
        asm (" clrc SXM");      /*Clear Sign Extension Mode bit             */
        asm (" clrc OVM");      /*Reset Overflow Mode bit                   */
        asm (" clrc CNF");      /*Configure block B0 to data mem.           */

        WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                /* set the external waitstates WSGR        */

        WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                /* Initialize Watchdog-timer               */

        SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                        (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                        (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                                /* Initialize SCSR1                        */

        MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                        +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                        +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                        +(MCRC1<<1)+MCRC0);
                                /* Initialize multiplex control register C */

        MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                        (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                        (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                                /* Initialize multiplex control register B */


        MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
        (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
        (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
        (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                                /* Initialize multiplex control register A */


        GPTCONA=((GPTCONA_T2TOADC<<9)+(GPTCONA_T1TOADC<<7)+
        (GPTCONA_TCOMPOE<<6)+(GPTCONA_T2PIN<<2)+
        (GPTCONA_T1PIN));       /*      Initialize GP Timer Control        */

        T1PR = PERIOD;          /* initialize T1-period                    */
        T1CNT= 0x0000;          /* start value for T1-counter              */

        T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
                (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                                /* initialize Timer1                       */

        CALIBRATION = 0;        /* ADC-offset is not corrected             */
```

```
        ADCTRL1 = 0x4000;       /* Master reset on the entire ADC-module   */

        ADCTRL1 = ((RESET<<15)+(SOFTFREE<<12)+(ACQ_PS<<8)+
                                (CPS<<7)+(CONT_RUN<<6)+(INT_PRI<<5)+
                                (SEQ_CASC<<4)+(CAL_ENA<<3)+(BRG_ENA<<2)+
                                (HILO<<1)+STEST_ENA);
                                * Initialize ADC-module                 */

        CHSELSEQ1 = 0x0080;   /* Channel 0 and Channel 8 are scanned       */

        ADCTRL2 |= 0x5000;      /* clear ADC-Sequencer1-Interrupt flag     */

        ADCTRL2 = ((EVB_SOC_SEQ<<15)+(RST_SEQ1<<14)+(SOC_SEQ1<<13)+
                        (INT_ENA_SEQ1<<10)+(EVA_SOC_SEQ1<<8)+(EXT_SOC_SEQ1<<7)+
                        (RST_SEQ2<<6)+(SOC_SEQ2<<5)+(INT_ENA_SEQ1<<2)+
                                EVB_SOC_SEQ2);
                                /* Initialize ADC-sequencer                */

        MAXCONV = 1;            /* 2 subsequent conversions are performed  */
                                /* in 1 session                            */


        PEDATDIR = 0xFF00;    /* Clear Port E                              */

        IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+
                (INT3<<2)+(INT2<<1)+(INT1));    /* Interrupt Mask Register*/

        IFR=0xFFFF;     /* Interrupt Flag Register , address 0x0006        */
                                /* Reset  all core interrupts              */

        asm (" clrc INTM");  /* Enable all unmasked interrupts             */

        T1CON=T1CON+(T1CON_TENABLE<<6); /* enable GPT1 now     */

        while(1){
                unsigned int i;
        for(i=0;i<65000;i++)
                show_ADC(ADC0_result);          /* displays the latest result on LED's */
        for(i=0;i<65000;i++)
                show_ADC(ADC1_result);
        }
}
```

Source Code Files Lab Exercise 2

Vectors.asm

```
.title    "vectors.asm"
          .ref    _c_int0,_c_dummy1,_ADC_ISR
          .sect   ".vectors"

reset:          b       _c_int0
int1:           b       _c_dummy1
int2:           b       _c_dummy1
int3:           b       _c_dummy1
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _ADC_ISR
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
sw_int12:       b       _c_dummy1
sw_int13:       b       _c_dummy1
sw_int14:       b       _c_dummy1
sw_int15:       b       _c_dummy1
sw_int16:       b       _c_dummy1
trap:           b       _c_dummy1
nmint:          b       _c_dummy1
emu_trap:       b       _c_dummy1
sw_int20:       b       _c_dummy1
sw_int21:       b       _c_dummy1
sw_int22:       b       _c_dummy1
sw_int23:       b       _c_dummy1
```

6.4. Optional Lab-Exercise 6 - A

Modify Lab-Exercise 4 ('Knight-Rider') :

- **use the Analogue Input ADCIN0 to change the frequency for the LED's**
- **for adding the ADC-instructions use Exercise 6**
- **use a LED-frequency range between 50Hz and 1 Hz**
- **use (1) a linear or (2) a logarithm scale between Fmin and Fmax.**

6 - 19

Optional Lab 6A: ADC-Input at ADCIN0 , two Timers GPT1 and GPT2 and GPIO-Port E

- Objective

The objective of this lab is to combine Lab4 (Digital Output 'Knight Rider' with a time base generated by GPT2-ISR) with the reading of the ADCIN0-channel. ADCIN0 is converted every 0.2s , auto started by GPT1-period. The result gained by the ADC's interrupt service routine is then used to modify the frequency of the LED-outputs (time base generated by GPT2).
The Aim is to modify the frequency of the 'Knight Rider' LED's depending on the value taken from the AD-Converter. One step of the night rider LED should last between 0.02s and 1s.
This program involves :

- ADCIN0 - Analogue input 0
- GPIO -Port E
- GPT1 and GPT2
- Interrupt Service ADC : read the latest result of conversion
- Interrupt Service GPT2 : put the next pattern onto GPIO-Port E

The maximum value for T2PERIOD = interval time / (CPUCLK * Pre-Scale)
= 1s / (33,9084 ns * 128)
= 230.400
which does not fit into the 16-bit Register T2PR. Unlike Lab4A we will now fix the value for T2PR to 4.608 and modify the number of T2-ISR's that must pass before the next pattern is copied to GPIO-E. We will also take into account that the ADC delivers a result between 0 and 1023 that determines the number of T2-ISR's.

The registers, involved in this lab, are :

MCRA	- Multiplex Control Register Group A
MCRB	- Multiplex Control Register Group B
MCRC	- Multiplex Control Register Group C
PEDATDIR	- Port E Data & Direction Register
WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register1
GPTCONA	- General Purpose Timer Control Register EVA
T2CON	- Timer 2 Control Register
T2PR	- Timer 2 Period Register
T1CON	- Timer 1 Control Register
T1PR	- Timer 1 Period Register
EVAIFRA	- Event Manager A Interrupt Flag Group A
EVAIFRB	- Event Manager A Interrupt Flag Group B
EVAIFRC	- Event Manager A Interrupt Flag Group C
EVAIMRA	- Event Manager A Interrupt Mask Group A
EVAIMRB	- Event Manager A Interrupt Mask Group B
EVAIMRC	- Event Manager A Interrupt Mask Group C

IFR - Global Interrupt Flag Register
IMR - Global Interrupt Mask Register
ADCCTRL1 - ADC Control Register 1
ADCCTRL2 - ADC Control Register 2
RESULT0 - ADC Result Register 0
CALIBRATION - ADC Calibration Register
CHSELSEQ1 - ADC Channel Select Register Sequencer 1
MAXCONV - ADC Maximum Conversion Number Register

- Procedure

Open Files, Create Project File

1. Using Code Composer, create a new project , called Lab6A.MAK in C:\One2407\Labs\Lab6.

2. Open the source-file F2407ADC.c in c:\One2407\labs\lab6 and save it as F2407ADCA.c in c:\One2407\labs\lab6 by clicking File → Open and File → Save as..

3. Modify the program F2407ADCA.C. Take into account these modifications :
    1. Define the period for T1 : T1_PERIOD = 46080 and the prescaler to 128(46080 * 33,9084ns * 128 = 0.2s). Initialize GPT1 for continuous up-mode, disable its interrupt and enable auto-start of ADC
    2. Define the period for T2 : T2_PERIOD = 4608 and the prescaler to 128 (4608 * 33,9084ns * 128 = 0.02s). Initialize GPT2 for continuous up-mode, enable its period interrupt.
    3. Define a variable T2_big_period to count the number of 0.02s-intervals
    4. Modify the Interrupt Service Routine T2PER_ISR. Count the number of calls until T2_big_period is reached. Then put the next LED-pattern out to Port E and modify T2_big_period depending on the latest AD-conversion result (the latest ADC-Result should be stored in a global variable). Take into account to scale T2_big_period, e.g. if ADC = 0 then generate an LED-interval of 0.02s, if ADC=1023 then generate 1sec.
    5. Add a ADC-ISR to copy RESULT0 into the global variable for the ADC-Result.

Note : A solutions directory c:\one2407\solutions\lab6A contains a solution for this modifications. If you can't solve the task now or your own solution does not work you can copy the source file from there.

4. Modify the file vectors.asm to vector the two interrupt services for INT1 and INT3

5. Add the Source Code Files: F2407ADCA.C and vectors.asm as well as the Linker Command File: ADC.cmd from C:\One2407\Labs\Lab6\ to your project by clicking : Project → Add Files to project

6. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

7. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

8. Close the Build Options Menu by clicking OK

Build and Load

9. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary. Load the program into the DSP.

Test

10. Verify the program running on the external LED's as supposed. Change the potentiometer on ADCIN0 and watch the modification of the LED's frequency. Debug as necessary.

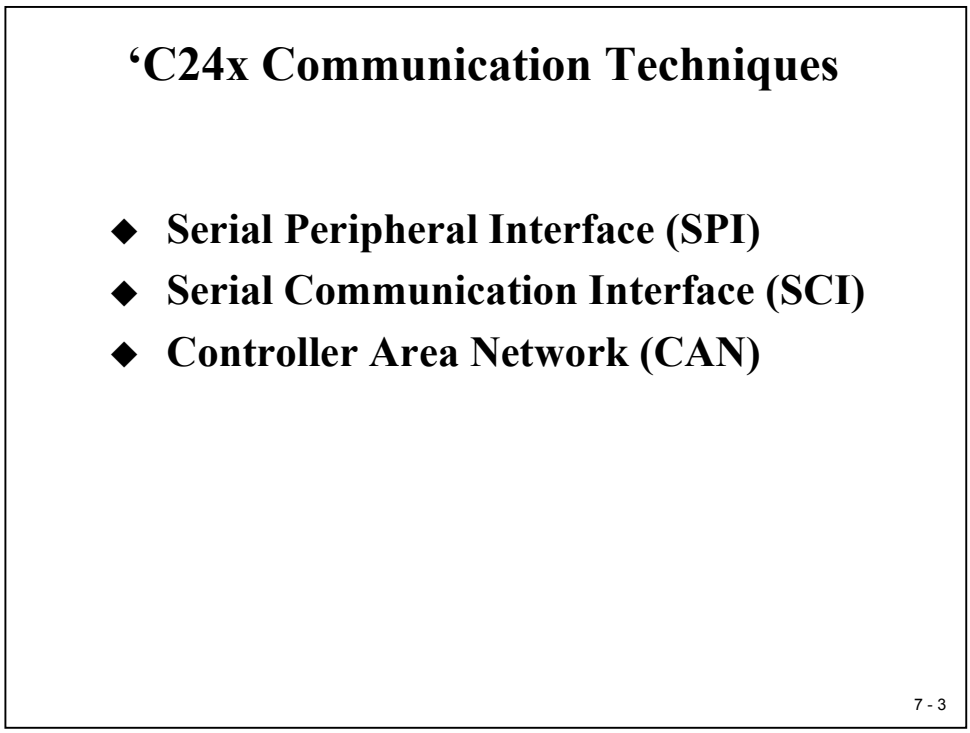
End of Exercise Lab6A

Introduction

The TMS320LF2407 contains features that allow several methods of communication and data exchange between the ‘LF2407 and other devices. Many of the most commonly used communications techniques are presented in this module.

Communications Techniques

Several methods of implementing a TMS320LF2407 communications system are possible. The method selected for a particular design should reflect the method that meets the required data rate at the lowest cost. Various categories of interface are available and are summarized in the following figure. Each will be described in the following sections.

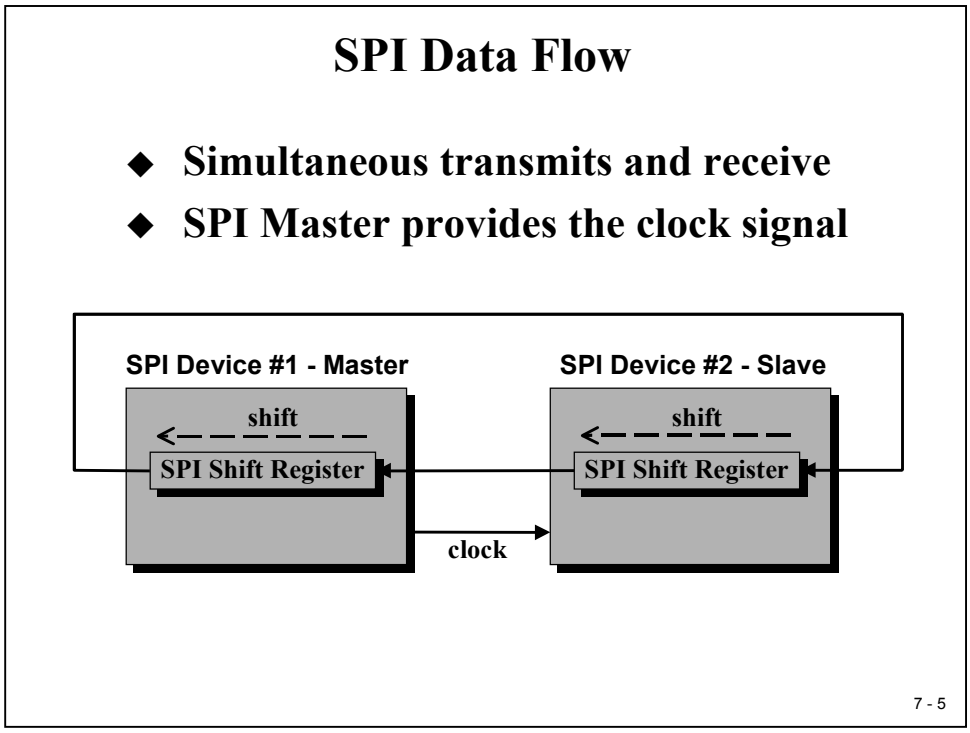


Serial Peripheral Interface (SPI)

The SPI module is a synchronous serial I/O port that shifts a serial bit stream of variable length and data rate between the ‘LF2407 and other peripheral devices. During data transfers, one SPI device must be configured as the transfer MASTER, and all other devices configured as SLAVES. The master drives the transfer clock signal for all SLAVES on the bus. SPI communications can be implemented in any of three different modes:

- MASTER sends data, SLAVES send dummy data
- MASTER sends data, one SLAVE sends data
- MASTER sends dummy data, one SLAVE sends data

In its simplest form, the SPI can be thought of as a programmable shift register. Data is shifted in and out of the SPI through the SPIDAT register. Data to be transmitted is written directly to the SPIDAT register, and received data is latched into the SPIBUF register for reading by the CPU. This allows for double-buffered receive operation, in that the CPU need not read the current received data from SPIBUF before a new receive operation can be started. However, the CPU must read SPIBUF before the new operation is complete of a receiver overrun error will occur. In addition, double-buffered transmit is not supported: the current transmission must be complete before the next data character is written to SPIDAT or the current transmission will be corrupted. The Master can initiate a data transfer at any time because it controls the SPICLK signal. The software, however, determines how the Master detects when the Slave is ready to broadcast.

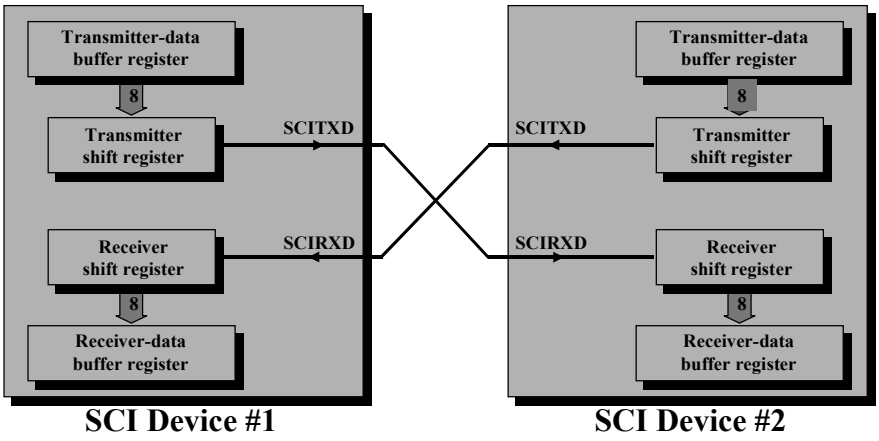


Serial Communications Interface (SCI)

The SCI module is a serial I/O port that permits Asynchronous communication between the ‘LF2407 and other peripheral devices. The SCI transmit and receive registers are both double-buffered to prevent data collisions and allow for efficient CPU usage. In addition, the ‘LF2407 SCI is a full duplex interface which provides for simultaneous data transmit and receive. Parity checking and data formatting is also designed to be done by the port hardware, further reducing software overhead.

SCI Pin Connections

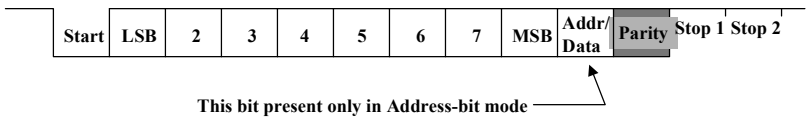
(Full Duplex Shown)



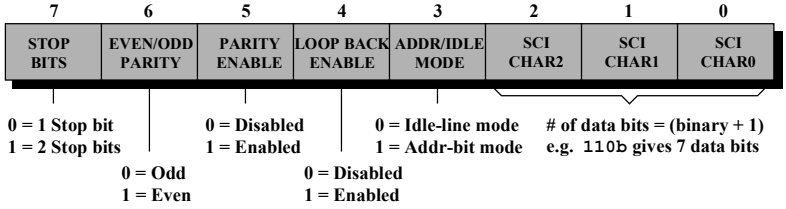
7 - 7

SCI Programmable Data Format

NRZ (nonreturn to zero) format



Communications Control Register (SCICCR) - 7050h



7 - 8

SCI Baud Rate

$$\text{SCI baud rate} = \begin{cases} \dfrac{\text{CLKOUT}}{(\text{BRR} + 1) \times 8}, & \text{BRR} = 1 \text{ to } 65535 \\ \dfrac{\text{CLKOUT}}{16}, & \text{BRR} = 0 \end{cases}$$

Baud-Select MSbyte Register (SCIHBAUD) - 7052h

7	6	5	4	3	2	1	0
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8

Baud-Select LSbyte Register (SCILBAUD) - 7053h

7	6	5	4	3	2	1	0
BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUD1	BAUD0 (LSB)

7 - 9

Baud Rate Determination: The values in the baud-select registers (SCIHBAUD and SCILBAUD) concatenate to form a 16 bit number that specifies the baud rate for the SCI.

- SCI Asynchronous Baud Rate = *SYSCLK / ((BRR +1)*8)* bits/sec

Note that the SYSCLK for the SCI module is one-half the CPU clock rate (e.g. a 'C240 running at 20 MHz has SYSCLK = 10 MHz for the SCI module). For SYSCLK = 10 MHz, one can compute the maximum baud rates as 625 Kbps and 5.0 Mbps respectively for Asynchronous and Isosynchronous communication.

SCI Module Summary

- **Asynchronous communications format**
- **65,000+ different programmable baud rates**
  - **Maximum Baud rate of 1.25 Mbps @20 MHz CPUCLK**
- **Two wake-up multiprocessor modes**
  - **Idle-line wake-up & Address-bit wake-up**
- **Programmable data word format**
  - **1 to 8 bit data word length**
  - **1 or 2 stop bits**
  - **even/odd/no parity**
- **Error Detection Flags**
  - **Parity error; Framing error; Overrun error; Break detection**
- **Double-buffered transmit and receive**
- **Individual interrupts for transmit and receive**

7 - 10

LAB 7. SCI - Transmission to PC's COM

AIM :

- ◆SCI-communication RS232 from EVA-board to PC- COM2
- ◆Sending a String "2407 UART is fine !"
  to Windows-95/98 Hyper-Terminal-program
- ◆9600 Baud , 8bit, no parity , 1 Stopbit , no protocol
- ◆Timer GPT2 -ISR starts a transmission every 2.0 seconds

Files :

F2407SCI.c	- source file main c program
vectors.asm	- jump table Interrupt Service Routine
regs2407.h	- pointer definition to peripherals
F2407SCI.cmd	- memory map definition
rts2xx.lib	- Runtime Library

7 - 11

New Registers involved in Lab 7:

SCI Command & Control	:	SCICCR
SCI Control Register 1	:	SCICTL1
SCI Control Register 2	:	SCICTL2
SCI Priority Register	:	SCIPRI
SCI Transmit Buffer	:	SCITXBUF
SCI Baud Rate High Byte	:	SCIHBAUD
SCI Baud Rate Low Byte	:	SCILBAUD

7 - 12

Setup for SCI -Registers (1 of 2)

```
/*************   SETUP for the SCICCR - Register **************/
#define STOPBITS        0       /* bit 7 =  0 :  one stop bit                  */
#define EVENODD         0       /*  bit 6 = 0 :  odd parity                    */
#define PARITY          0       /*  bit 5 = 0 :  no parity                     */
#define LOOPBACKENA     0       /*  bit 4 = 0 :  Loop Back Test Mode disabled  */
#define ADDRIDLE        0       /*  bit 3 = 0 :  IDLE-line for multi-proccessor*/
#define SCICHAR         7       /*  bit 2-0 = 111b  : 8-bit data transmission  */
/*************   SETUP for the SCICTL1 - Register **************/
#define RXERRINT        0       /*  bit 6 = 0 : disable Rx Error Interrupts    */
#define SWRESET         0       /*  bit 5 = 0 : apply reset state              */
                                /*         = 1 : re-enable  SCI after setup    */
#define TXWAKE          0       /*  bit 3 = 0 : no wakeup function             */
#define SLEEP           0       /*  bit 2 = 0 : sleep mode disabled            */
#define TXENA           1       /*  bit 1 = 1 : Transmitter enabled            */
#define RXENA           0       /*  bit 0 = 0 : Receiver  disabled             */
```

7 - 13

Setup for SCI-Registers (2 of 2)

```
/*************   SETUP for the SCICTL2 - Register **************/
#define RXBKINT         0       /* bit 1= 0 : disable RX and Break interrupt  */
#define TXINT           1       /* bit 0 = 1 : enable TXRDY-interrupt         */
/*************   SETUP for the SCIPRI - Register **************/
#define TXPRIORITY      0       /* bit 6 = 0 : TXD-int on high Priority (INT1)*/
#define RXPRIORITY      0       /* bit 5 = 0 : RXD-int on high Priority (INT1)*/
#define SCISOFTFREE     2       /* bit 4,3 = 10b :on JTAG suspend complete SCI*/


#define BRR         383         /* Baud-rate Register Constant:               */
                                /* 383 = (29,4912e+6/(9600Baud *8)) -1        */
```

7 - 14

Lab 7: SCI Transmission to PC's serial Com-Port (COM1 / COM2)

- Objective

  The objective of this lab is to include a serial communication between the DSP and the PC. For this purpose we will connect the DSP's RS232-connector with a simple DB9-cable to the PC's COM1 or COM2. The aim is to continuously transmit a string from the DSP to the PC. The transmission of the first character is initiated by GPT-Timer 2 period interrupt service. The remaining characters are transmitted by means of the SCI's transmitter interrupt service routine. After the last character has been transmitted the transmission stops until T2 starts again the whole procedure.

  The Windows-Hyperterminal program is the counterpart from the PC's-side and must be adapted (Baudrate, Parity,...) for correct function.

  The registers, involved in this lab, are :

WDCR	- Watchdog Control Register
WSGR	- Wait State Generator Register
SCSR1	- System Control and Status Register 1
GPTCONA	- General Purpose Timer Control Register
T2CON	- Timer 2 Control Register
T2PR	- Timer 2 Period Register
EVAIFRA	- Event Manager A Interrupt Flag Group A
EVAIFRB	- Event Manager A Interrupt Flag Group B
EVAIFRC	- Event Manager A Interrupt Flag Group C
EVAIMRA	- Event Manager AInterrupt Mask Group A
EVAIMRB	- Event Manager A Interrupt Mask Group B
EVAIMRC	- Event Manager A Interrupt Mask Group C
IFR	- Global Interrupt Flag Register
IMR	- Global Interrupt Mask Register
PIVR	- Peripheral Interrupt Vector Register
SCICCR	- SCI Control & Command Register
SCICTL1	- SCI Control Register 1
SCICTL2	- SCI Control Register 2
SCIPRI	- SCI Priority Register
SCIHBAUD	- SCI Baud Rate High Byte
SCILBAUD	- SCI Baud Rate Low Byte
SCITXBUF	- SCI Transmit Buffer

- Procedure

  ### Open W95 / W98 – Hyperterminal

  1. Prepare a Hyperterminal Session : From Windows – Menubar choose : Start → Programs → Accessories → Communication → Hyper Terminal

  2. Click on "hypertrm.exe"

3. A new connection will be opened. Give this connection a name and select a symbol, then click 'OK'

4. In the connect-to-window select "direct connection to COM1"

5. A new window with COM1-properties opens. Modify the properties of COM1 to :
   - 9600 Bit per second
   - 8 data bits
   - 1 stop bit
   - no protocol

6. Click on 'OK' . The Connection is then immediately active. Leave this session active and go back to Code Composer.

Open Files, Create Project File

7. Using Code Composer, create a new project , called myLab7.MAK in C:\One2407\Labs\Lab7. **NOTE** : There is a project file Lab7.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 6.

8. Add the Source Code Files: F2407SCI.c and vectors.asm as well as the Linker Command File: F2407SCI.cmd from C:\One2407\Labs\Lab7\ to your project by clicking : Project → Add Files to project

9. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

10. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

11. Close the Build Options Menu by clicking OK

Build and Load

12. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary.

13. Load the output file down to the EVM. Click : File → Load Program and choose the desired output file.

Test

14. Reset the DSP by clicking on Debug → Reset DSP

15. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407SCI.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

16. Perform a real time run. Switch back to your hyper terminal session, that should still be active. You should see the string "The 2407 UART is fine !" in the hyper terminal message window, repeated with a period of 2 seconds.

17. Exercise to place a breakpoint or a probe point inside the SCI-Interrupt service routine; watch the registers SCITXBUF and the variable 'index'.

18. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab7.wks"

19. In case you placed breakpoints or probe points delete them by clicking on the particular buttons, close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab7

Initializing Sequence for LAB 7 :

1. Set Waitstates, Watchdog and System Control as before
2. In Multiplex Control M**CRA** set bits 1,0 to 11b (**SCIRXD | TXD**)
3. Set **GPTCONA** to disable outputs (bit 6 = 0) ,polarity of T2,T1 to forced low (bit 3-0 = 0000b) , disable ADC-Start by Timer-Period - Event (bits 8-7 = 00b)
4. Set Tiner T2 as before (**T2PR, T2CMPR, T2CON**) to generate a time base
5. Setup the SCI-Interface (**SCICCR, SCICTL1, SCICTL2, SCIHBAUD, SCILBAUD, SCIPRI**)
6. Re-enable SCI (**SCICTL1**- bit 5= 1)
7. Clear all EV-Interrupt Flags (**EVAIFRx** = 0xFFFF)
8. Enable T2-Period Interrupt (**EVAIMRB** - bit 0 = 1)
9. Clear all global Interrupt Flags (**IFR** = 0xFFFF)
10. Enable Interrupt Level 3 (T2PR) and 1 (SCI) (**IMR**-Bit 2,0 = 1)
11. Global Interrupt Enable (**INTM** = 0)
12. Enable Timer T2 (**T2CON** - bit 6 = 1)

7 - 15

Source Code Files Lab Exercise 7
F2407SCI.C

```
/*********************************************************************/
/* Testprogram for serial communication SCI                          */
/* running on TMS320LF2407 EVM                                       */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/*********************************************************************/
/* SCI-communication RS232 from EVA-board to PC- COM2                 */
/* Sending a String "2407 UART is fine !" to Windows-95 Hyperterminal*/
/* 9600 Baud , 8bit, no parity , 1 Stopbit , no protocol             */
/* Timer GPT2 starts transmission every 2.0 second                   */
/* program-name :  F2407SCI.c     / project : lab7.mak               */
/* date : 07/17/2001   (c) Frank.Bormann@fh-zwickau.de               */
/*********************************************************************/

#include "regs2407.h"
/*************    SETUP for the MCRA - Register   ***************************/
#define MCRA15          0       /* 0 : IOPB7       1 : TCLKIN              */
#define MCRA14          0       /* 0 : IOPB6       1 : TDIR                */
#define MCRA13          0       /* 0 : IOPB5       1 : T2PWM               */
#define MCRA12          0       /* 0 : IOPB4       1 : T1PWM               */
#define MCRA11          0       /* 0 : IOPB3       1 : PWM6                */
#define MCRA10          0       /* 0 : IOPB2       1 : PWM5                */
#define MCRA9           0       /* 0 : IOPB1       1 : PWM4                */
#define MCRA8           0       /* 0 : IOPB0       1 : PWM3                */
#define MCRA7           0       /* 0 : IOPA7       1 : PWM2                */
#define MCRA6           0       /* 0 : IOPA6       1 : PWM1                */
#define MCRA5           0       /* 0 : IOPA5       1 : CAP3                */
#define MCRA4           0       /* 0 : IOPA4       1 : CAP2/QEP2           */
#define MCRA3           0       /* 0 : IOPA3       1 : CAP1/QEP1           */
#define MCRA2           0       /* 0 : IOPA2       1 : XINT1               */
#define MCRA1           1       /* 0 : IOPA1       1 : SCIRXD              */
#define MCRA0           1       /* 0 : IOPA0       1 : SCITXD              */
/*************    SETUP for the MCRB - Register   ***************************/
#define MCRB9           0       /* 0 : IOPD1       1 : XINT2/EXTSOC        */
#define MCRB8           1       /* 0 : CKLKOUT  1 : IOPD0                  */
#define MCRB7           0       /* 0 : IOPC7       1 : CANRX               */
#define MCRB6           0       /* 0 : IOPC6       1 : CANTX               */
#define MCRB5           0       /* 0 : IOPC5       1 : SPISTE              */
#define MCRB4           0       /* 0 : IOPC4       1 : SPICLK              */
#define MCRB3           0       /* 0 : IOPC3       1 : SPISOMI             */
#define MCRB2           0       /* 0 : IOPC2       1 : SPISIMO             */
#define MCRB1           1       /* 0 : BIO         1 : IOPC1               */
#define MCRB0           1       /* 0 : XF          1 : IOPC0               */
/*************    SETUP for the MCRC - Register   ***************************/
#define MCRC13          0       /* 0 : IOPF5       1 : TCLKIN2             */
#define MCRC12          0       /* 0 : IOPF4       1 : TDIR2               */
#define MCRC11          0       /* 0 : IOPF3       1 : T4PWM/T4CMP         */
#define MCRC10          0       /* 0 : IOPF2       1 : T3PWM/T3CMP         */
#define MCRC9           0       /* 0 : IOPF1       1 : CAP6                */
#define MCRC8           0       /* 0 : IOPF0       1 : CAP5/QEP3           */
#define MCRC7           0       /* 0 : IOPE7       1 : CAP4/QEP2           */
#define MCRC6           0       /* 0 : IOPE6       1 : PWM12               */
#define MCRC5           0       /* 0 : IOPE5       1 : PWM11               */
#define MCRC4           0       /* 0 : IOPE4       1 : PWM10               */
#define MCRC3           0       /* 0 : IOPE3       1 : PWM9                */
#define MCRC2           0       /* 0 : IOPE2       1 : PWM8                */
#define MCRC1           0       /* 0 : IOPE1       1 : PWM7                */
#define MCRC0           0       /* 0 : IOPE0       1 : CLKOUT              */
```

```
/*************    SETUP for the WDCR - Register   ***************************/
#define WDDIS          1       /* 0 : Watchdog enabled   1: disabled          */
#define WDCHK2         1       /* 0 : System reset       1: Normal OP         */
#define WDCHK1         0       /* 0 : Normal Oper.       1: sys reset         */
#define WDCHK0         1       /* 0 : System reset       1: Normal OP         */
#define WDSP           7       /* Watchdog prescaler 7 : div 64               */
/*************    SETUP for the SCSR1 - Register  ****************************/
#define CLKSRC         0       /* 0 : intern(30MHz)                           */
#define LPM            0       /* 0 : Low power mode 0 if idle                */
#define CLK_PS         1       /* 001 : PLL multiply by 2                     */
#define ADC_CLKEN      0       /* 0 : No ADC-service in this test             */
#define SCI_CLKEN      1       /* 1 : Use SCI-service in this test            */
#define SPI_CLKEN      0       /* 0 : No SPI-servide in this test             */
#define CAN_CLKEN      0       /* 0 : No CAN-service in this test             */
#define EVB_CLKEN      0       /* 0 : No EVB-Service in this test             */
#define EVA_CLKEN      1       /* 1 : Use EVA-Service in this test            */
#define ILLADR         1       /* 1 : Clear ILLADR during startup             */
/*************    SETUP for the WSGR - Register   ****************************/
#define BVIS           0       /* 10-9 : 00 Bus visibility OFF                */
#define ISWS           0       /* 8 -6 : 000 0 Waitstates for IO              */
#define DSWS           0       /* 5 -3 : 000 0 Waitstates  data               */
#define PSWS           0       /* 2 -0 : 000 0 Waitstaes code                 */
/*************    SETUP for the EVAIMRA - Register  **************************/
#define T1OFINT        0       /* 10 : Timer 1 overflow interrupt             */
#define T1UFINT        0       /*  9 : Timer 1 underflow interrupt            */
#define T1CINT         0       /*  8 : Timer 1 compare interrupt              */
#define T1PINT         0       /*  7 : Timer 1 period interrupt               */
#define CMP3INT        0       /*  3 : Compare 3 interrupt                    */
#define CMP2INT        0       /*  2 : Compare 2 interrupt                    */
#define CMP1INT        0       /*  1 : Compare 1 interrupt                    */
#define PDPINT         0       /*  0 : Power Drive Protect Interrupt          */
/*************    SETUP for the EVAIMRB - Register  **************************/
#define T2OFINT        0       /*  3 : Timer 2 overflow interrupt             */
#define T2UFINT        0       /*  2 : Timer 2 underflow interrupt            */
#define T2CINT         0       /*  1 : Timer 2 compare interrupt              */
#define T2PINT         1       /*  0 : Timer 2 period interrupt               */
/*************    SETUP for the EVAIMRC- Register  ***************************/
#define CAP3INT        0       /*  2 : Capture Unit 3 interrupt               */
#define CAP2INT        0       /*  1 : Capture Unit 2 Interrupt               */
#define CAP1INT        0       /*  0 : Capture unit 1 interrupt               */
/*************    SETUP for the IMR - Register    ****************************/
#define INT6           0       /*  5 : Level INT6  is masked                  */
#define INT5           0       /*  4 : Level INT5  is masked                  */
#define INT4           0       /*  3 : Level INT4  is masked                  */
#define INT3           1       /*  2 : Level INT3  is unmasked                */
#define INT2           0       /*  1 : Level INT2  is masked                  */
#define INT1           1       /*  0 : Level INT1  is unmasked                */
/*************    SETUP for the GPTCONA - Register  **************************/
#define GPTCONA_T2TOADC   0        /* 10-9 : 0 no ADC-Start by GPT2-Event     */
#define GPTCONA_T1TOADC   0        /*  8-7  : 0 no ADC-Start by GPT1-Event    */
#define GPTCONA_TCOMPOE   0        /*  6   : 0 disable all 2 GPT compare outputs*/
#define GPTCONA_T2PIN     0        /* 3-2 : 00 GPT2 comp out=forced low       */
#define GPTCONA_T1PIN     0        /* 1-0 : 00 GPT1 comp out=forced low       */
/***************    SETUP for the T2CON - Register  ***************************/
#define T2CON_FREESOFT    0        /* 15-14 : 00 stop on JTAG suspend         */
#define T2CON_TMODE       2        /* 12-11 : 10 Contiuous up counting        */
#define T2CON_TPS         7        /* 10-8  :111 CPUCLK/128                   */
#define T2CON_TSWT1       0         /* 7       :  0 use own TENABLE bit       */
#define T2CON_TENABLE     1        /* 6     :  1 Timer 2 enable               */
#define T2CON_TCLKS       0        /* 5-4   : 00 Clock source internal        */
#define T2CON_TCLD        1        /* 3-2   : 01 reload when 0 or T2PR        */
```

```
#define T2CON_TECMPR        1       /* 1    :  1 enable timer compare          */
#define T2CON_SELT1PR       0       /* 0    :  0 use T2 period register        */
/*************    SETUP for the SCICCR - Register ***************************/
#define STOPBITS            0       /*  0 :  one stop bit                      */
#define EVENODD             0       /*  0 :  odd parity                        */
#define PARITY              0       /*  0 :  no parity                         */
#define LOOPBACKENA         0       /*  0 :  Loop Back Test Mode disabled      */
#define ADDRIDLE            0       /*  0 :  IDLE-line mode for multipr.       */
#define SCICHAR             7       /*  111 : 8-bit data transmission          */
/*************    SETUP for the SCICTL1 - Register **************************/
#define RXERRINT            0       /*  0 : disable Rx Error Interrupts        */
#define SWRESET             0       /*  0 : apply reset state                  */
                                    /*  1 : reenabling SCI, after config.      */
#define TXWAKE              0       /*  0 : no wakeupfunction now              */
#define SLEEP               0       /*  0 : sleep mode disabled                */
#define TXENA               1       /*  1 : Transmitter enabled                */
#define RXENA               0       /*  1 : Receiver  disabled                 */
/*************    SETUP for the SCICTL2 - Register **************************/
#define RXBKINT             0       /* 0 : disable RX and Break inter.         */
#define TXINT               1       /* 1 : enable TXRDY-interrupt              */
/*************    SETUP for the SCIPRI - Register ***************************/
#define TXPRIORITY          0       /*  TXD-int on high Priority (INT1)        */
#define RXPRIORITY          0       /* RXD-int on high Priority (INT1)         */
#define SCISOFTFREE         2       /* on emulator suspend complete SCI        */
/*************************************************************************/
#define PERIOD 46080                /* T2PER = 33,9084ns * 128 * 31250 =0.2s   */
#define BRR  383                    /* baudrate register constant              */
                                    /* 383 = (29,4912e+6/(9600Baud *8)) -1     */
static char index=0;
const char message[25]="The 2407 UART is fine !\n\r";

void c_dummy1(void)
{
            while(1);               /* Dummy ISR used to trap spurious interrupts  */
}

interrupt void T2PER_ISR(void)
{
            static unsigned char i=0;
            if((PIVR-0x002B)==0)    /* Verify T2PINT ( 0x002B)                 */
                {
                i++;
                if(i>=10){          /* wait for 10 * 0.2 s before start new transmission  */
                        i=0;
                        SCITXBUF=message[index++];
                        }
                EVAIFRB=T2PINT;
                }
}


interrupt void SCI_ISR(void)
{
             if((PIVR-0x0007)==0)   /*Verify type of interrupt ( 7 = TxD )      */
                {
                if(index<=24) SCITXBUF=message[index++]; /*transmit             */
                else index=0;       /* end of transmission                     */
                }
}
```

```
void main(void)
{
 asm (" setc INTM");       /*Disable all interrupts                                  */
 asm (" clrc SXM");        /*Clear Sign Extension Mode bit                           */
 asm (" clrc OVM");        /*Reset Overflow Mode bit                                 */
 asm (" clrc CNF");        /*Configure block B0 to data mem.                         */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                           /* set the external waitstates          WSGR             */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                           /* Initialize Watchdog-timer                               */

 SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                    (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                    (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                           /* Initialize SCSR1                                        */

 MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                    +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                    +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                    +(MCRC1<<1)+MCRC0);
                           /* Initialize multiplex control register C                 */

 MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                     (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                     (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                           /* Initialize multiplex control register B                 */


 MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
       (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
       (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
       (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                           /* Initialize multiplex control register A                 */

 GPTCONA=((GPTCONA_T2TOADC<<9)+(GPTCONA_T1TOADC<<7)+
       (GPTCONA_TCOMPOE<<6)+(GPTCONA_T2PIN<<2)+
       (GPTCONA_T1PIN));    /*      Initialize GP Timer Control                     */

 T2PR = PERIOD;            /* initialize T2-period                                   */
 T2CNT= 0x0000;            /* start value for T2-counter                             */

 T2CON=((T2CON_FREESOFT<<14)+(T2CON_TMODE<<11)+(T2CON_TPS<<8)+
                    (T2CON_TSWT1<<7)+(T2CON_TCLKS<<4)+(T2CON_TCLD<<2)+
                    (T2CON_TECMPR<<1)+T2CON_SELT1PR);

 SCICCR=((STOPBITS<<7)+(EVENODD<<6)+(PARITY<<5)+
      (LOOPBACKENA<<4)+(ADDRIDLE<<3)+SCICHAR);
                           /* Initialize SCI Control Register                        */

 SCICTL1=((RXERRINT<<6)+(SWRESET<<5)+
                    (TXWAKE<<3)+(SLEEP<<2)+(TXENA<<1)+RXENA);
                           /* initialize SCI Control Register 1                      */

 SCICTL2=((RXBKINT<<1)+TXINT);
                           /* initialize SCI Control Register 2                      */

 SCIHBAUD=BRR>>8;   /* load the Baud-Rate Register                                   */
 SCILBAUD=BRR & 0x00FF;
```

```
SCIPRI= ((TXPRIORITY<<6)+(RXPRIORITY<<5)+(SCISOFTFREE<<3));
                        /* Initialize SCI Priority control register          */

SCICTL1|=0x0020;        /* reenable SCI ,  SWRESET=1                         */

EVAIFRA=0xFFFF;         /* clear EVA Interrupt Flags Group A                 */

EVAIFRB=0xFFFF;         /* clear EVA Interrupt Flags Group B                 */

EVAIFRC=0xFFFF;         /* clear EVA Interrupt Flags Group C                 */

EVAIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+
                (T1PINT<<7)+(CMP3INT<<3)+(CMP2INT<<2)+
                (CMP1INT<<1)+PDPINT);
                        /* enable specific EVA Interrupts Group A            */

EVAIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+T2PINT);
                        /* enable specific EVA Interrupts Group B            */

EVAIMRC=((CAP3INT<<2)+(CAP2INT<<1)+CAP1INT);
                        /* enable specific EVA Interrupts Group C            */

IFR=0xFFFF;             /* Reset  all core interrupts                        */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+INT1);
                        /* enable specific core Interrupts                   */

asm (" clrc INTM");     /* Enable all unmasked interrupts                    */

T2CON=T2CON+(T2CON_TENABLE<<6);         /* enable GPT2 now               */

while(1);               /* endless loop ; action done by ISR                 */

}
```

vectors.asm

```
        .title   "vectors.asm"
        .ref     _c_int0,_c_dummy1,_SCI_ISR,_T2PER_ISR
        .sect    ".vectors"

reset:          b       _c_int0
int1:           b       _SCI_ISR
int2:           b       _c_dummy1
int3:           b       _T2PER_ISR
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
sw_int12:       b       _c_dummy1
sw_int13:       b       _c_dummy1
sw_int14:       b       _c_dummy1
sw_int15:       b       _c_dummy1
sw_int16:       b       _c_dummy1
trap:           b       _c_dummy1
nmint:          b       _c_dummy1
emu_trap:       b       _c_dummy1
sw_int20:       b       _c_dummy1
sw_int21:       b       _c_dummy1
sw_int22:       b       _c_dummy1
sw_int23:       b       _c_dummy1
```

Optional Lab - Exercise 7-A

- **Use Exercise 7 as starting point**
- **Transmit the status of the DIP-switches to Windows - Hyper-Terminal every 2 seconds**

- Note : transform the hexadecimal value taken from Port B first into an ASCII-string and than transmit the string

- Note : the function “ltoa” converts a long integer into an ASCII-string [int ltoa(long val, char *buffer);]
  - see Code Composers online help for details

7 - 19

Optional Exercise Lab 7A: SCI Transmission to PC's serial Com-Port (COM1 / COM2) ; transmit the status of GPIO-Port B

- Objective

  The objective aim is to continuously transmit the actual value from the DIP-switches on Port GPIO-B as a numerical value to PC's Hyper Terminal. The transmission should be initiated by GPT-Timer 2 period interrupt service every 2.0 seconds. The numerical value taken from Port B must be first converted into a string and then transmitted to COM1. The transmitter-interrupt should be used to transmit the next characters of the string. For the conversion of the value into a string the function "ltoa" could be utilised ("Long Integer to ASCII"). If you are not familiar with such functions this is a good opportunity to use Code Composers Online Help to find the syntax of this function.

  The registers, involved in this lab, are the same as in Lab 7

- Procedure

Open W95 / W98 – Hyper terminal

1. Open again the Hyper terminal Session from Lab7 .

Open Files, Create Project File

2. Using Code Composer, create a new project , called Lab7A.MAK in C:\One2407\Labs\Lab7.

3. Open the source-file F2407SCI.c in c:\One2407\labs\lab7 and save it as F2407SCIA.c in c:\One2407\labs\lab7 by clicking File → Open and File → Save as..

4. Modify the program F2407SCIA.C. Take into account these modifications :
   1. Modify the T2PER_ISR-routine:
      - Read PBDATDIR
      - Use the ltoa-function : int ltoa(long val, char *buffer) to convert the value into a string buffer
      - Add the characters '\n' and '\r' at the end of the string
      - Send the first character
   2. Modify the SCI_ISR-routine :
      Adapt the number of characters to be sent.

Note : A solutions directory c:\one2407\solutions\lab7A contains a solution for this modifications. If you can't solve the task now or your own solution does not work you can copy the source file from there.

5. Add the Source Code Files: F2407SCIA.c and vectors.asm as well as the Linker Command File: F2407SCI.cmd from C:\One2407\Labs\Lab7\ to your project by clicking : Project → Add Files to project

6. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

7. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

8. Close the Build Options Menu by clicking OK

Build and Load

9. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary.

10. Load the output file down to the EVM. Click : File → Load Program and choose the desired output file.

Test

11. Reset the DSP by clicking on Debug → Reset DSP

12. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407SCIA.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

13. Perform a real time run. Switch back to your hyper terminal session, that should still be active. In the hyper terminal message window you should see a string representing the value taken from the DIP-switches (Port B). Change some of the switches and watch the result in the message window.

14. You might want to use the workspace environment in further sessions. For this purpose it could be helpful to store the current workspace. To do so, click : File → Workspace → Save Workspace and save it as "Lab7A.wks"

15. In case you placed breakpoints or probe points delete them by clicking on the particular buttons, close the project by Clicking Project → Close Project and close all open windows that you do not need any further.

End of Exercise Lab7A

LAB 8: SCI - Receive from and Transmit to PC's serial COM

AIM :

- SCI-communication RS232 from EVA-board to PC- COM2
- 9600 Baud , 8bit, no parity , 1 Stopbit , no protocol
- PC has to send a string with a <CR> at the end (W98 - Hyper-Terminal - function : "send a textfile")
- after receiving the <CR> the LF2407 sends the string "READY<CR>" back to PC

Files :

F2407SCI2.c	- main source file (C)
vectors.asm	- jump table Interrupt Service Routine
regs2407.h	- pointer definition to peripherals
F2407SCI.cmd	- memory map definition
rts2xx.lib	- Runtime Library

7 - 20

New Registers involved in Lab 8:

SCI Command & Control	:	SCICCR
SCI Control Register 1	:	SCICTL1
SCI Control Register 2	:	SCICTL2
SCI Priority Register	:	SCIPRI
SCI Transmit Buffer	:	SCITXBUF
SCI Receive Buffer	:	SCIRXBUF
SCI Baud Rate High Byte	:	SCIHBAUD
SCI Baud Rate Low Byte	:	SCILBAUD

7 - 21

Setup for SCI -Registers (1 of 2)

```
/*************   SETUP for the SCICCR - Register **************/
#define STOPBITS      0     /* bit 7 =  0 :  one stop bit                    */
#define EVENODD       0     /*  bit 6 = 0 :  odd parity                      */
#define PARITY        0     /*  bit 5 = 0 :  no parity                       */
#define LOOPBACKENA   0     /*  bit 4 = 0 :  Loop Back Test Mode disabled    */
#define ADDRIDLE      0     /*  bit 3 = 0 :  IDLE-line for multi-proccessor  */
#define SCICHAR       7     /*  bit 2-0 = 111b  : 8-bit data transmission    */
/*************   SETUP for the SCICTL1 - Register *************/
#define RXERRINT      0     /*  bit 6 = 0 : disable Rx Error Interrupts      */
#define SWRESET       0     /*  bit 5 = 0 : apply reset state                */
                            /*         = 1 : re-enable  SCI after setup      */
#define TXWAKE        0     /*  bit 3 = 0 : no wakeup function               */
#define SLEEP         0     /*  bit 2 = 0 : sleep mode disabled              */
#define TXENA         1     /*  bit 1 = 1 : Transmitter enabled              */
#define RXENA         1     /*  bit 0 = 1 : Receiver  enabled                */
```

7 - 22

Setup for SCI-Registers (2 of 2)

```
/*************   SETUP for the SCICTL2 - Register *************/
#define RXBKINT       1     /* bit 1= 1  : enable RX and Break interrupt    */
#define TXINT         1     /* bit 0 = 1 : enable TXRDY-interrupt           */
/*************   SETUP for the SCIPRI - Register **************/
#define TXPRIORITY    0     /* bit 6 = 0 : TXD-int on high Priority (INT1)  */
#define RXPRIORITY    0     /* bit 5 = 0 : RXD-int on high Priority (INT1)  */
#define SCISOFTFREE   2     /* bit 4,3 = 10b :on JTAG suspend complete SCI*/


#define BRR       383       /* Baud-rate Register Constant:                 */
                            /* 383 = (29,4912e+6/(9600Baud *8)) -1          */
```

7 - 23

Lab 8: SCI Receive & Transmission from/to PC's serial Com-Port (COM1 / COM2)

- Objective

  The objective of this lab is to extend Lab7 into a bi-directional communication. The DSP is waiting for a string , received from the PC's-COM. After recognising a <CR> the DSP will transmit the string "READY" back to the PC. Again we will make use of Windows-Hyper terminal program as the counterpart.

  In addition to Lab7 there is one more registers involved :

  SCIRXBUF - SCI Receiver Buffer Register

- Procedure

Open W95 / W98 – Hyper terminal

1. Prepare or open a Hyper terminal session like in Lab 7.

Open Files, Create Project File

2. Using Code Composer, create a new project , called myLab8.MAK in C:\One2407\Labs\Lab8. **NOTE** : There is a project file Lab8.mak provided in this directory. So you can either make your own project or use the provided project file. In the last case proceed to step 7.

3. Add the Source Code Files: F2407SCI2.c and vectors.asm as well as the Linker Command File: F2407SCI.cmd from C:\One2407\Labs\Lab8\ to your project by clicking : Project → Add Files to project

4. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

5. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

6. Close the Build Options Menu by clicking OK

Build and Load

7. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary.

8. Load the output file down to the EVM. Click : File → Load Program and choose the desired output file.

Test

9. Reset the DSP by clicking on Debug → Reset DSP

10. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407SCI2.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

11. Perform a real time run. Switch back to your hyper terminal session, that should still be active. Now select "Send Textfile" from the hyper terminal menu bar and then select the file "Text.txt" from c:\One2407\\labs\lab8.

12. Immediately after you send the textfile the DSP should answer with the string "READY" , displayed in the message window.

End of Exercise Lab8

Initializing Sequence for LAB 8 :

1. Set Waitstates, Watchdog and System Control as before
2. In Multiplex Control **MCRA** set bits 1,0 to 11b (**SCIRXD | TXD**)

3. Setup the SCI-Interface (**SCICCR, SCICTL1, SCICTL2, SCIHBAUD, SCILBAUD, SCIPRI**)
4. Re-enable SCI (**SCICTL1**- bit 5= 1)
5. Clear all EV-Interrupt Flags (**EVAIFRx** = 0xFFFF)
6. Disable all EV-Interrupts (**EVAIMRx** = 0x0000)
7. Clear all global Interrupt Flags (**IFR** = 0xFFFF)
8. Enable Interrupt Level 1 (SCI) (**IMR**-Bit 0 = 1)
9. Global Interrupt Enable (**INTM** = 0)

Note : No GP-Timer involved in this Lab, only Rx- and Tx-Interrupt-Service

7 - 24

Source Code Files Lab Exercise 8

F2407SCI2.C

```
/*********************************************************************/
/* Testprogram for serial communication SCI                          */
/* running on TMS320LF2407 EVM                                       */
/* external clock is 14.7456 MHz, PLL * 2 , CPU-Clock then 29.49 MHz */
/*********************************************************************/
/* SCI-communication RS232 from EVA-board to PC- COM2                 */
/* Receiving from  Windows-95 Hyperterminal-Session                  */
/* write a textfile containing a one line text and a <CR> and send this */
/* file by Hyperterminal to the LF2407 ( "Transmission - Text file" ) */
/* 9600 Baud , 8bit, no parity , 1 Stopbit , no protocol             */
/* The F2407 monitors the receiver by its interrupt service routine, when */
/* the received character was a <CR> the string 'READY' is transmitted */
/* uses both TXD-interrupt- and RXD-interrupt-service                */
/* program-name :  F2407SCI2.c    / project : lab8.mak               */
/* date : 07/17/2001   (c) Frank.Bormann@fh-zwickau.de               */
/*********************************************************************/

#include "regs2407.h"

/*************    SETUP for the MCRA - Register   ***************************/
#define MCRA15          0       /* 0 : IOPB7      1 : TCLKIN              */
#define MCRA14          0       /* 0 : IOPB6      1 : TDIR                */
#define MCRA13          0       /* 0 : IOPB5      1 : T2PWM               */
#define MCRA12          0       /* 0 : IOPB4      1 : T1PWM               */
#define MCRA11          0       /* 0 : IOPB3      1 : PWM6                */
#define MCRA10          0       /* 0 : IOPB2      1 : PWM5                */
#define MCRA9           0       /* 0 : IOPB1      1 : PWM4                */
#define MCRA8           0       /* 0 : IOPB0      1 : PWM3                */
#define MCRA7           0       /* 0 : IOPA7      1 : PWM2                */
#define MCRA6           0       /* 0 : IOPA6      1 : PWM1                */
#define MCRA5           0       /* 0 : IOPA5      1 : CAP3                */
#define MCRA4           0       /* 0 : IOPA4      1 : CAP2/QEP2           */
#define MCRA3           0       /* 0 : IOPA3      1 : CAP1/QEP1           */
#define MCRA2           0       /* 0 : IOPA2      1 : XINT1               */
#define MCRA1           1       /* 0 : IOPA1      1 : SCIRXD              */
#define MCRA0           1       /* 0 : IOPA0      1 : SCITXD              */
/*************    SETUP for the MCRB - Register   ***************************/
#define MCRB9           0       /* 0 : IOPD1      1 : XINT2/EXTSOC        */
#define MCRB8           1       /* 0 : CKLKOUT  1 : IOPD0                 */
#define MCRB7           0       /* 0 : IOPC7      1 : CANRX               */
#define MCRB6           0       /* 0 : IOPC6      1 : CANTX               */
#define MCRB5           0       /* 0 : IOPC5      1 : SPISTE              */
#define MCRB4           0       /* 0 : IOPC4      1 : SPICLK              */
#define MCRB3           0       /* 0 : IOPC3      1 : SPISOMI             */
#define MCRB2           0       /* 0 : IOPC2      1 : SPISIMO             */
#define MCRB1           1       /* 0 : BIO        1 : IOPC1               */
#define MCRB0           1       /* 0 : XF         1 : IOPC0               */
/*************    SETUP for the MCRC - Register   ***************************/
#define MCRC13          0       /* 0 : IOPF5      1 : TCLKIN2             */
#define MCRC12          0       /* 0 : IOPF4      1 : TDIR2               */
#define MCRC11          0       /* 0 : IOPF3      1 : T4PWM/T4CMP         */
#define MCRC10          0       /* 0 : IOPF2      1 : T3PWM/T3CMP         */
#define MCRC9           0       /* 0 : IOPF1      1 : CAP6                */
#define MCRC8           0       /* 0 : IOPF0      1 : CAP5/QEP3           */
#define MCRC7           0       /* 0 : IOPE7      1 : CAP4/QEP2           */
```

```
#define MCRC6           0       /* 0 : IOPE6      1 : PWM12                    */
#define MCRC5           0       /* 0 : IOPE5      1 : PWM11                    */
#define MCRC4           0       /* 0 : IOPE4      1 : PWM10                    */
#define MCRC3           0       /* 0 : IOPE3      1 : PWM9                     */
#define MCRC2           0       /* 0 : IOPE2      1 : PWM8                     */
#define MCRC1           0       /* 0 : IOPE1      1 : PWM7                     */
#define MCRC0           0       /* 0 : IOPE0      1 : CLKOUT                   */
/*************   SETUP for the WDCR - Register   ****************************/
#define WDDIS           1       /* 0 : Watchdog enabled   1: disabled          */
#define WDCHK2          1       /* 0 : System reset       1: Normal OP         */
#define WDCHK1          0       /* 0 : Normal Oper.       1: sys reset         */
#define WDCHK0          1       /* 0 : System reset       1: Normal OP         */
#define WDSP            7       /* Watchdog prescaler 7 : div 64               */
/*************   SETUP for the SCSR1 - Register  ****************************/
#define CLKSRC          0       /* 0 : intern(30MHz)                           */
#define LPM             0       /* 0 : Low power mode 0 if idle                */
#define CLK_PS          1       /* 001 : PLL multiply by 2                     */
#define ADC_CLKEN       0       /* 0 : No ADC-service in this test             */
#define SCI_CLKEN       1       /* 1 : Use SCI-service in this test            */
#define SPI_CLKEN       0       /* 0 : No SPI-servide in this test             */
#define CAN_CLKEN       0       /* 0 : No CAN-service in this test             */
#define EVB_CLKEN       0       /* 0 : No EVB-Service in this test             */
#define EVA_CLKEN       0       /* 0 : No EVA-Service in this test             */
#define ILLADR          1       /* 1 : Clear ILLADR during startup             */
/*************   SETUP for the WSGR - Register   ****************************/
#define BVIS            0       /* 10-9 : 00 Bus visibility OFF                */
#define ISWS            0       /* 8 -6 : 000 0 Waitstates for IO              */
#define DSWS            0       /* 5 -3 : 000 0 Waitstates  data               */
#define PSWS            0       /* 2 -0 : 000 0 Waitstaes code                 */
/*************   SETUP for the IMR - Register    ****************************/
#define INT6            0       /*  5 : Level INT6  is masked                  */
#define INT5            0       /*  4 : Level INT5  is masked                  */
#define INT4            0       /*  3 : Level INT4  is masked                  */
#define INT3            0       /*  2 : Level INT3  is masked                  */
#define INT2            0       /*  1 : Level INT2  is masked                  */
#define INT1            1       /*  0 : SCI -Level INT1  is unmasked           */
/*************   SETUP for the SCICCR - Register ****************************/
#define STOPBITS        0       /*  0 :  one stop bit                          */
#define EVENODD         0       /*  0 :  odd parity                            */
#define PARITY          0       /*  0 :  no parity                             */
#define LOOPBACKENA     0       /*  0 :  Loop Back Test Mode disabled          */
#define ADDRIDLE        0       /*  0 :  IDLE-line mode for multipr.           */
#define SCICHAR         7       /*  111 : 8-bit data transmission              */
/*************   SETUP for the SCICTL1 - Register ****************************/
#define RXERRINT        0       /*  0 : disable Rx Error Interrupts            */
#define SWRESET         0       /*  0 : apply reset state                      */
                                /*  1 : reenabling SCI, after config.          */
#define TXWAKE          0       /*  0 : no wakeupfunction now                  */
#define SLEEP           0       /*  0 : sleep mode disabled                    */
#define TXENA           1       /*  1 : Transmitter enabled                    */
#define RXENA           1       /*  1 : Receiver  enabled                      */
/*************   SETUP for the SCICTL2 - Register ****************************/
#define RXBKINT         1       /* 0 : enable RX and Break inter.              */
#define TXINT           1       /* 1 : enable TXRDY-interrupt                  */
/*************   SETUP for the SCIPRI - Register ****************************/
#define TXPRIORITY      0       /* TXD-int on high Priority (INT1)             */
#define RXPRIORITY      0       /* RXD-int on high Priority (INT1)             */
#define SCISOFTFREE     2       /* on emulator suspend complete SCI            */
/*****************************************************************************/
```

```
#define BRR  383                    /* baudrate register constant                    */
                                    /* 383 = (29,4912e+6/(9600Baud *8)) -1           */

static char index=0;
const char message[7]="READY\n\r";

void c_dummy1(void)
{
                while(1);           /*Dummy ISR used to trap spurious interrupts    */
}

interrupt       void SCI_ISR(void)
{
 if((PIVR-0x0007)==0)               /*Verify type of interrupt ( 7 = TxD )          */
                {
                if(index<=6) SCITXBUF=message[index++];   /* Transmit next byte   */
                else index=0;       /* reset message pointer                         */
                }
  if((PIVR-0x0006)==0)              /* 6 = receiver interrupt                       */
                {
                if(SCIRXBUF==0x0D) SCITXBUF=message[index++];
                                    /* when <CR> received then start transmission    */
                }
}

void main(void)
{
 asm (" setc INTM");        /*Disable all interrupts                                 */
 asm (" clrc SXM");         /*Clear Sign Extension Mode bit                          */
 asm (" clrc OVM");         /*Reset Overflow Mode bit                                */
 asm (" clrc CNF");         /*Configure block B0 to data mem.                        */

 WSGR=((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                            /* set the external waitstates          WSGR          */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                            /* Initialize Watchdog-timer                          */

 SCSR1= ((CLKSRC<<14)+(LPM<<12)+(CLK_PS<<9)+(ADC_CLKEN<<7)+
                    (SCI_CLKEN<<6)+(SPI_CLKEN<<5)+(CAN_CLKEN<<4)+
                    (EVB_CLKEN<<3)+(EVA_CLKEN<<2)+ILLADR);
                            /* Initialize SCSR1                                   */

 MCRC = ((MCRC13<<13)+(MCRC12<<12)+(MCRC11<<11)+(MCRC10<<10)
                    +(MCRC9<<9)+(MCRC8<<8)+(MCRC7<<7)+(MCRC6<<6)
                    +(MCRC5<<5)+(MCRC4<<4)+(MCRC3<<3)+(MCRC2<<2)
                    +(MCRC1<<1)+MCRC0);
                            /* Initialize multiplex control register C            */

 MCRB = ((MCRB9<<9)+(MCRB8<<8)+
                     (MCRB7<<7)+(MCRB6<<6)+(MCRB5<<5)+(MCRB4<<4)+
                     (MCRB3<<3)+(MCRB2<<2)+(MCRB1<<1)+MCRB0);
                            /* Initialize multiplex control register B            */

  MCRA = ((MCRA15<<15)+(MCRA14<<14)+(MCRA13<<13)+(MCRA12<<12)+
        (MCRA11<<11)+(MCRA10<<10)+(MCRA9<<9)+(MCRA8<<8)+
        (MCRA7<<7)+(MCRA6<<6)+(MCRA5<<5)+(MCRA4<<4)+
        (MCRA3<<3)+(MCRA2<<2)+(MCRA1<<1)+MCRA0);
                            /* Initialize multiplex control register A            */
```

```
SCICCR=((STOPBITS<<7)+(EVENODD<<6)+(PARITY<<5)+
      (LOOPBACKENA<<4)+(ADDRIDLE<<3)+SCICHAR);
                        /* Initialize SCI Control Register                  */

SCICTL1=((RXERRINT<<6)+(SWRESET<<5)+
                  (TXWAKE<<3)+(SLEEP<<2)+(TXENA<<1)+RXENA);
                        /* initialize SCI Control Register 1                */

SCICTL2=((RXBKINT<<1)+TXINT);      /* initialize SCI Control Register 2     */

SCIHBAUD=BRR>>8;   /* load the Baud-Rate Register                           */
SCILBAUD=BRR & 0x00FF;

SCIPRI= ((TXPRIORITY<<6)+(RXPRIORITY<<5)+(SCISOFTFREE<<3));
                        /* Initialize SCI Priority control register         */

SCICTL1|=0x0020;        /* reenable SCI ,  SWRESET=1                        */

IFR=0xFFFF;             /* Reset  all core interrupts                       */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+INT1);
                        /* enable specific core Interrupts                  */

asm (" clrc INTM");     /* Enable all unmasked interrupts                   */

while(1);               /* endless loop ; action done by ISR                */

}
```

Source file : vectors.asm

```
.title    "vectors.asm"
          .ref    _c_int0,_c_dummy1,_SCI_ISR
          .sect   ".vectors"

reset:          b       _c_int0
int1:           b       _SCI_ISR
int2:           b       _c_dummy1
int3:           b       _c_dummy1
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
sw_int12:       b       _c_dummy1
sw_int13:       b       _c_dummy1
sw_int14:       b       _c_dummy1
sw_int15:       b       _c_dummy1
sw_int16:       b       _c_dummy1
trap:           b       _c_dummy1
nmint:          b       _c_dummy1
emu_trap:       b       _c_dummy1
sw_int20:       b       _c_dummy1
sw_int21:       b       _c_dummy1
sw_int22:       b       _c_dummy1
sw_int23:       b       _c_dummy1
```

Optional Exercise LAB 8-A

AIM :
Modify the frequency of the 'Knight Rider LED'
by a value received with SCI !

Start : use a 'mixture' of LAB 8 and LAB 4 .

- Let Windows-Hyper-Terminal send a text-file, that contains only one number , e.g.3000, and a <CR>
- use the function 'atoi' (see manual) to convert the string into an integer (don't forget to include "stdlib.h")
- put this converted value into T2's period register
- send back to Hyper-Terminal your 'READY' when the DSP has received the <CR>
- enable T2-period-interrupt, SCI-receiver-interrupt & SCI-transmit-interrupt
- Write a second text-file with another number , e.g. 50000 , and send the two files alternately to the DSP

7 - 27

Optional Exercise Lab 8A : SCI Receive & Transmission from/to PC's serial Com-Port (COM1 / COM2) and Digital Output at GPIO-Port E (LED's)

- Objective

The objective of this lab is to extend Lab8 into a control system. The PC has to send a text file through hyper terminal. This text file contains only one line with a numerical value, coded as ASCII-string with an ending <CR>. The DSP waits for this string, sends back the string "READY". The last value received by SCI should then be used as the next period for GPT2, which controls the frequency of the 'Knight Rider' LED's (see Lab 4). The function 'atoi' (ASCII to integer) , part of stdlib.h, converts the ASCII-string into an integer.

- Procedure

Open W95 / W98 – Hyper Terminal

1. Prepare or open a Hyper terminal session as done during Lab 8.

Open Files, Create Project File

2. Using Code Composer, create a new project , called Lab8A.MAK in C:\One2407\Labs\Lab8.

3. Open the source-file F2407SCI2.c in c:\One2407\labs\lab8 and save it as F2407SCIA.c in c:\One2407\labs\lab8 by clicking File → Open and File → Save as..

4. Modify the program F2407SCIA.C. Take into account these modifications :
    1. Look into Lab4 and add the code-lines for the LED's with GPT2-Interrupt (see Lab 4).
    2. Inside the SCI_ISR-routine modify the part for the receiver:
        - Put the incoming characters into a string array "string_received", increment the array-pointer
        - In case the last character was a <CR>, then transform the "string_received" into an integer (use function 'atoi' , for syntax see Code Composer Help).
        - Put this integer into T2PR to modify the frequency of the LED's.

    Note : A solutions directory c:\one2407\solutions\lab8A contains a solution for this modifications. If you can't solve the task now or your own solution does not work you can copy the source file from there.

5. Add the Source Code Files: F2407SCIA.c and vectors.asm as well as the Linker Command File: F2407SCI.cmd from C:\One2407\Labs\Lab8\ to your project by clicking : Project → Add Files to project

6. Add the C-runtime library to your project by clicking : Project → Options → Linker → Library Search Path : 'c:\tic2xx\c2000\cgtools\lib'. Then Add the library by clicking : Project → Options → Linker → Include Libraries : 'rts2xx.lib'

7. Verify that in Project → Options → Linker the C-Initialization-Field contains : 'ROM-Autoinitialisation Model [-c]

8. Close the Build Options Menu by clicking OK

Build and Load

9. Click the "Rebuild All" button or perform : Project → Build and watch the tools run in the build window. Debug as necessary.

10. Load the output file down to the EVM. Click : File → Load Program and choose the desired output file.

Test

11. Reset the DSP by clicking on Debug → Reset DSP

12. Run the program until the first line of your C-code by clicking : Debug → Go main. Verify that in the working area the window of the source code "F2407SCIA.c" is highlighted and that the yellow bar for the current Program Counter is placed on the line 'void main(void)'.

13. Perform a real time run. The LED-program should run as in Lab4.

14. Now switch back to the hyper terminal session. Send a textfile containing only one line : '2000' and <CR> (Note : in c:\one2407\solutions\lab8a\ there are three ASCII-files '2000.txt', '30000.txt' and '50000.txt' . You couls take one of them and send it to the DSP) Immediately after you send the textfile the DSP should modify its frequency for the LED-outputs. The valid range for the control value is between 1 and 65536 (due to the 16-bit register T2PR).

15. Repeat step 14 by sending another numerical string to the DSP.

End of Exercise Lab8A

Controller Area Network (CAN) ‘X241/3

What is CAN?

- CAN: Controller Area Network
- Serial bus system
- Broadcast type of bus:
  - Each node can transmit to all CAN nodes.
- Each message contains an identifier:
  - Message filtering
  - Message priority

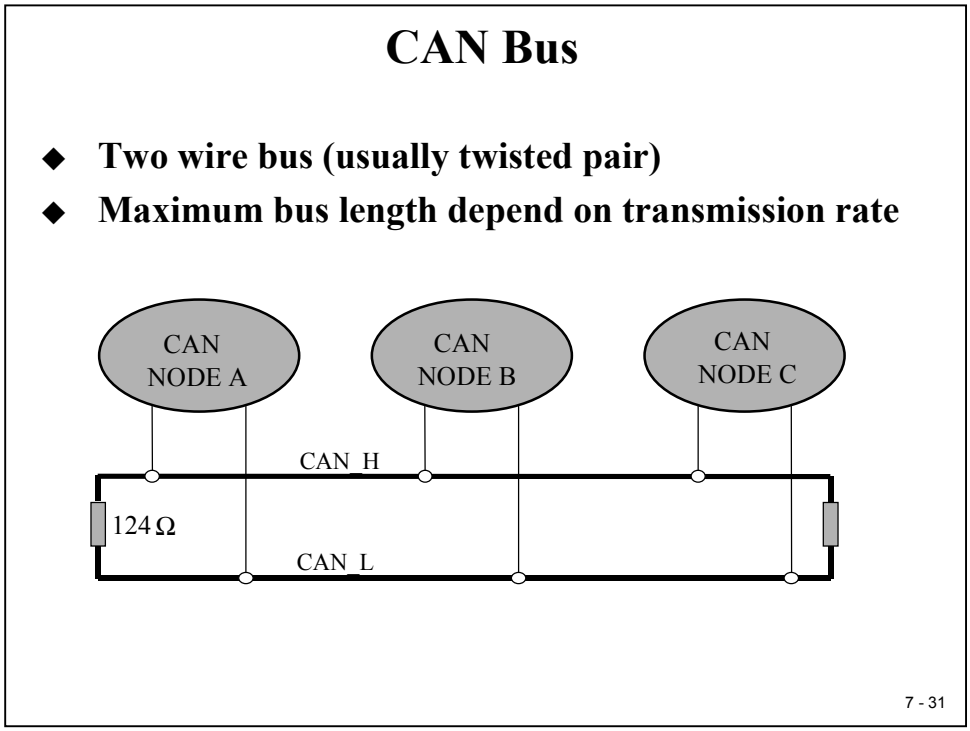
7 - 29

Why CAN?

- Low cost
- Real time capabilities
- Multi-master network
- Up to 1Mbit/sec (40 m bus length)
- High reliability (error detection and handling)
- International standard (ISO 11898)
- Easy to learn

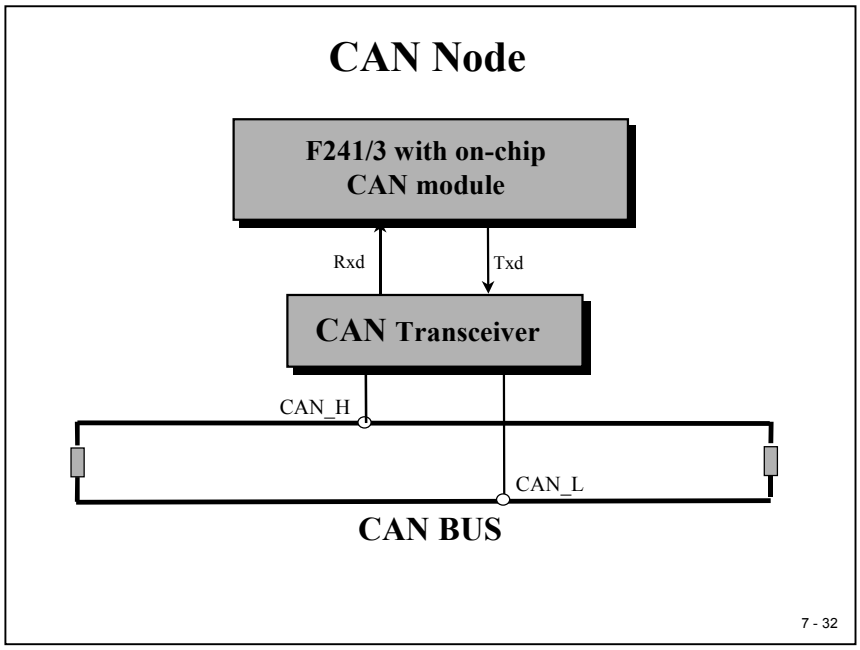
7 - 30

CAN does not use physical adresses to address stations. Each message is sent with an identifier that is recognized by the different nodes. The identifier has two functions – it is used for message filtering and for message priority. The identifier determines if a transmitted message will be received by CAN modules and determines the priority of the message when two or more nodes want to transmit at the same time.

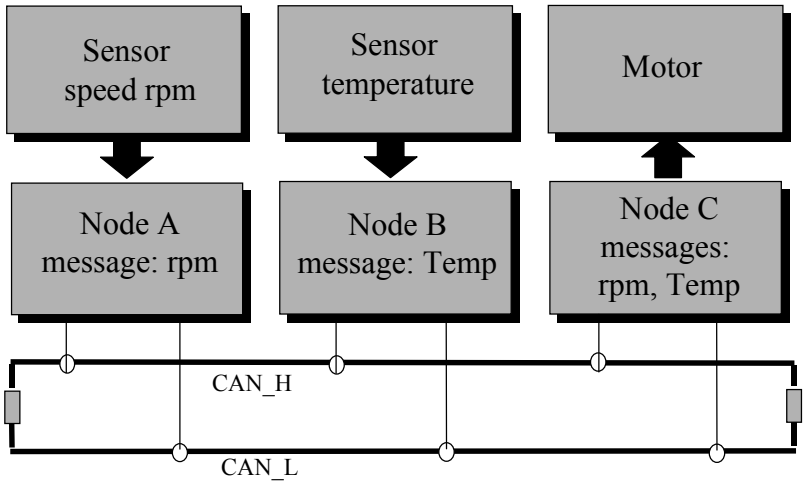


The ‘X241/3 communicates to the CAN Bus using a transceiver. The CAN bus is a twisted pair wire, and the transmission rate depends on the bus length. If the bus is less than 40 meters the transmission rate is capable up to 1 Mbit/second.

DSP24 14 DSP24 14 - 29 29



CAN Application



7 - 33

The ‘F241/3, LF2407 and LF2406 CAN module is a FullCAN Controller. It contains a message handler for transmission and reception management, and frame storage. The specification is CAN 2.0B Active – that is, the module can send and accept standard (11-bit identifier) and extended frames (29-bit identifier).

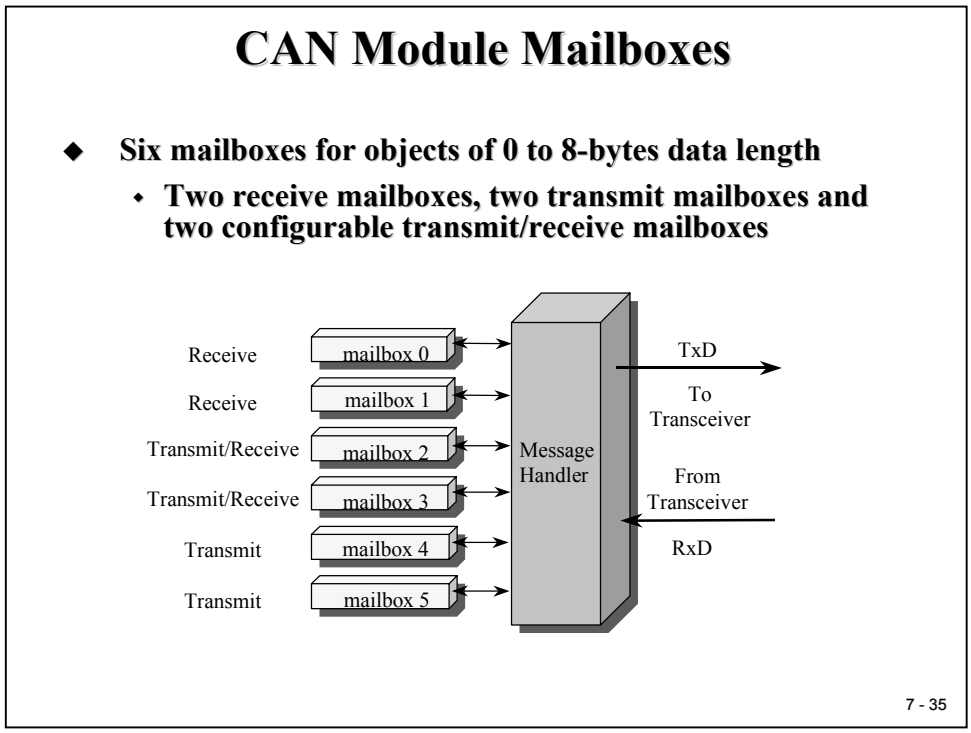
CAN Module

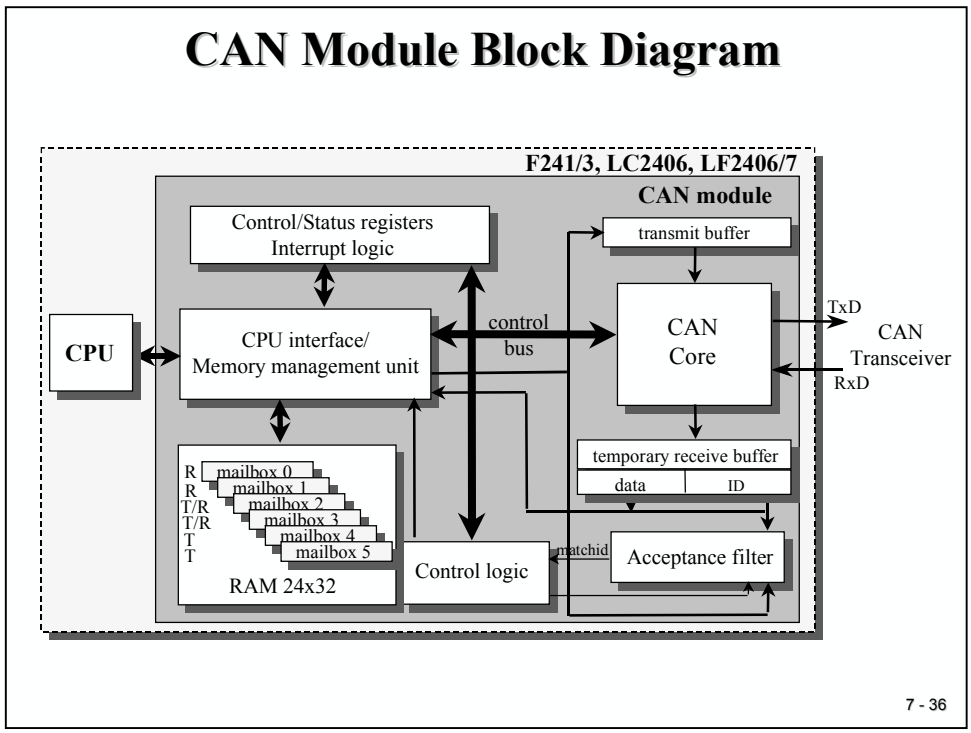
- **FullCAN Controller**
  - **Contains a message handler (for transmission and reception management and frames storage)**
  - **Less CPU overhead needed than with BasicCAN Controller**
- **16-bit peripheral**
  - **16-bit access to control/status registers and to CAN mailboxes**
- **CAN 2.0B Active**
  - **Send and accept standard and extended frames**
  - **Compatible with CAN 2.0A, CAN 2.0B Passive**
- **Self test mode: can operate in loop back mode, receiving its own transmitted message**

7 - 34

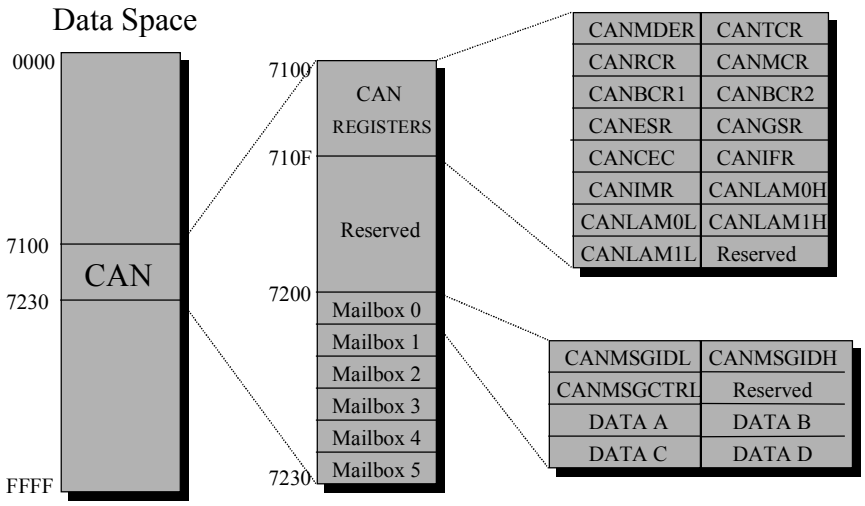
The CAN controller module contains six mailboxes for objects of 0 to 8-byte data lengths:

- two receive mailboxes (mailboxes 0 & 1)
- two transmit mailboxes (mailboxes 4 & 5)
- two configurable transmit/receive mailboxes (mailboxes 2 & 3)





CAN Memory



7 - 37

Optional : CAN -Exercises

- **Can - Transmission & Receive**
- **see Examples / Exercises at http://www.fh-zwickau.de/tutorial/dsp**
- **from download-area load the *.zip - Files**
- **Lab10 : CAN-Receiver**
- **Lab11 : CAN-Transmission**
- **Lab12 : CAN-Receive &Transmit**

7 - 38

Introduction

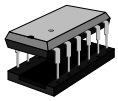
The Code Composer Version 4.12 includes a feature to perform a real time debug mode. This is done by adding a real time monitor code into the application code. This monitor guarantees the service of active interrupt-sources while the debug system is halted or single stepped. This is vital for e.g. power electronics connected to the PWM's .

1. Execution Control Modes

The JTAG-Emulator supports two debug modes :

- ***Stop mode***
- ***Real-time mode***

Both modes can suspend program execution at break events
(software breakpoints , access to specified memory space)



University of Applied Sciences Zwickau Date: 24.08.00 10:27
(c) Frank.Bormann@FH-zwickau.de page 2

1.1. Stop mode

• provides complete control of program execution

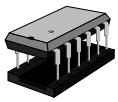
• stop mode disables all interrupts

• is based on JTAG accessible scan-paths into DSP

• HLL-Debugger stops the DSP and uses the scan-paths to execute special instructions to view / modify DSP-Registers

• during Debug - operations the DSP is always halted and unable to respond to interrupts

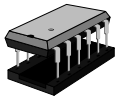
• DSP is therefore unable to execute interrupt - driven real - time tasks



University of Applied Sciences Zwickau Date: 24.08.00 10:27
(c) Frank.Bormann@FH-zwickau.de page 3

1.2. Stop mode states

- **Debug - Halt**
  - can observe the DSP
  - cannot service interrupts , including NMI or RS
- **Single-Instruction state**
  - cannot observe the DSP
  - cannot service interrupts, if STEP 1 is used
  - can service interrupts, if RUN 1 is used
- **Run state**
  - cannot observe CPU
  - can service interrupts



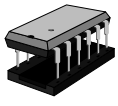
1.3. Real-time mode

• allows interrupt-service-routines to be performed while execution of background code (e.g. main or a function of main) is halted.

• DSP is never halted and therefore able to respond to interrupts and to execute interrupt-driven real-time tasks

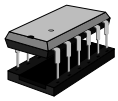
• is based on a JTAG message-passing scan-path that is independent of the DSP

• a small monitor program, embedded into the application code is needed to allow message-passing



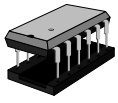
1.3. Real-time mode

- **the DSP calls the monitor program to respond to an outstanding message whenever it does not have higher priority tasks to execute (e.g. ISR's)**
- **The JTAG-Debugger does not stop the DSP, instead it uses the independent scan path to pass messages to the monitor program.**
- **JTAG-Debugger is able to view or modify memory / registers or have the DSP to step through code that is not interrupt - callable at that time**
- **The DSP views the monitor as an interrupt-driven device driver for the message passing scan - path**
- **c200mnrt.asm is provided for this purpose**



1.4. Real-time mode states

- **Debug - Halt**
  - can observe the DSP
  - can service interrupts , including NMI or RS
- **Single-Instruction state**
  - cannot observe the DSP
  - cannot service interrupts, if STEP 1 is used
  - can service interrupts, if RUN 1 is used
- **Run state**
  - can observe CPU
  - can service interrupts

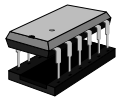


1.5. Setup the Real-time mode

- **Modify your application project :**
  - add the monitor ‘c200mnrt.asm’ to your project
  - modify the interrupt vector table (‘vectors.asm’)
  - modify the linker command file
  - inside your application call MON_RT_CNFG()
- **Linker Command File**
  - reserve space for the monitor , typically B2

```
mon_main : {} > PROG PAGE 0   /* monitor code */
mon_pge0 : {} > B2      PAGE 1   /* monitor variables */
mon_rgst  : {} > B2      PAGE 1   /* monitor variables */
```

  - B2 is no longer available for user program

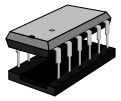


1.5. Setup the Real-time mode (cont.)

- Interrupt Vector Table (‘vectors.asm’)
  - **add in line 7 the monitor-drive vectors , e. g. :**

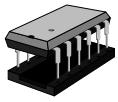
```
reset:      b      _c_int0
int1:       b      _c_dummy1
int2:       b      _c_dummy1
int3:       b      _T2PER_ISR
int4:       b      _c_dummy1
int5:       b      _c_dummy1
int6:       b      _c_dummy1
reserved:   .include c200vecs.inc
```

  - **the file c200vecs.inc includes itself the real-time configuration settings (file : c200mnrt.i)**



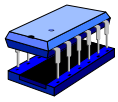
1.6. Resource Usage for Real-time mode

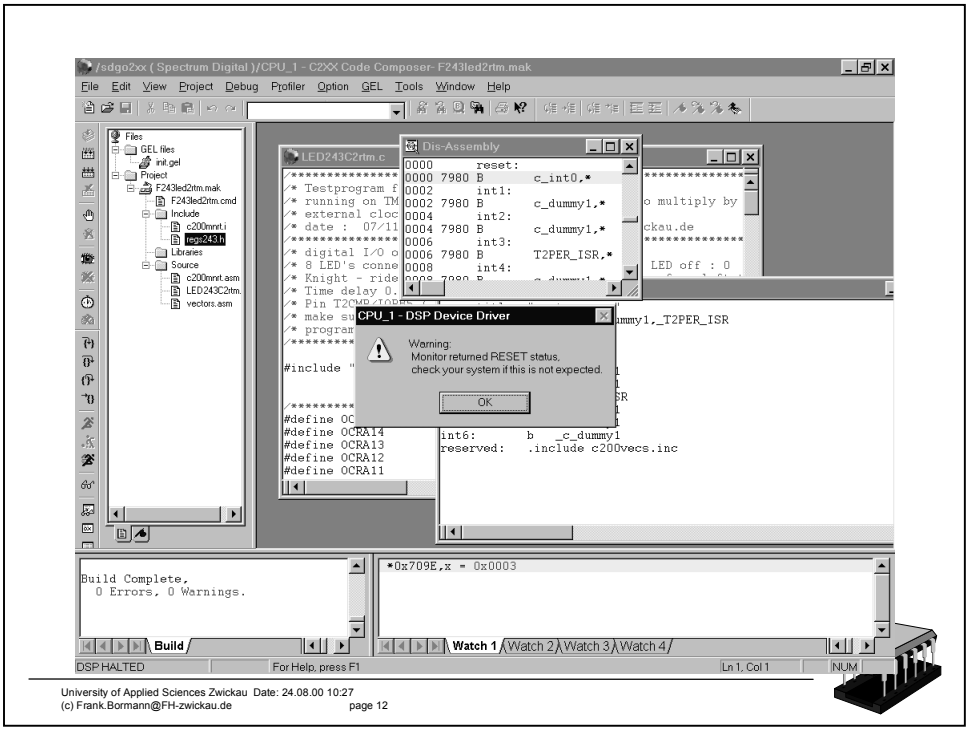
- Program memory:
  - **mon_main :** **220 words**
  - **mon_eintr_vecs , mon_etrap_vecs :** **24 words**
- Data Memory :
  - **mon_pge0 :** **13 words in page zero**
  - **mon_rgst :** **19 words anywhere**
- Stack :
  - **1 level of stack**
- Register usage :
  - **ST0, ST1, ACCL, ACCH (saved & restored by monitor)**
- AUX-Registers
  - **one, defined by MON_AR_VALUE**



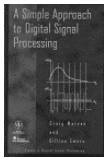
1.7. Example for Real-Time-Debug:

- **An Example for a real-time debug adapted session is included in the directory c:\One2407\labs\rtm**
  - **F2407RTM.c : main-source code**
  - **vectors.asm : Interrupt vector table**
  - **C200mnrt.asm : Real-Time Monitor source code**
  - **C200vecs.inc : Include file to expand vectors.asm**
  - **C200mnrt.i : Include Conditional assembly options (in C200vecs.inc)**
  - **F2407rtm.cmd : Linker Command File**
  - **c:\tic2xx\c2000\cgtools\include\regs243.h : Memory mapped register definitions**
- **Open the Project, Build it and Load the output into EVM**
- **Perform a reset**
- **Add: *(int*)0x7095,x to the Watch Window (PEDATDIR)**
- **Switch on the Real Time Mode :**
  ==> Debug ==> Real Time Mode (click OK inside the Warning Dialog box)
- **Notice the active LED's while the DSP is halted !**



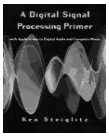

/sdgo2xx (Spectrum Digital)/CPU_1 - C2XX Code Composer- F243led2rtm.mak
File Edit View Project Debug Profiler Option GEL Tools Window Help
Files
GEL files
init.gel
Project
F243led2rtm.mak
F243led2rtm.cmd
Include
c200mnrt.i
regs243.h
Libraries
Source
c200mnrt.asm
LED243C2rtm.
vectors.asm
LED243C2rtm.c
Dis-Assembly
0000 reset:
0000 7980 B c_int0,*
0002 int1:
0002 7980 B c_dummy1,*
0004 int2:
0004 7980 B c_dummy1,*
0006 int3:
0006 7980 B T2PER_ISR,*
0008 int4:
CPU_1 - DSP Device Driver
Warning:
Monitor returned RESET status,
check your system if this is not expected.
OK
#define OCRA14
#define OCRA13
#define OCRA12
#define OCRA11
int6: b _c_dummy1
reserved: .include c200vecs.inc
Build Complete,
0 Errors, 0 Warnings.
*0x709E,x = 0x0003
Build
Watch 1
Watch 2
Watch 3
Watch 4
DSP HALTED
For Help, press F1
Ln 1, Col 1
NUM

Want To Learn More About DSP?



- "A Simple Approach to Digital Signal Processing" by Craig Marven and Gillian Ewers; ISBN 0-4711-5243-9

- "DSP Primer (Primer Series)" by C. Britton Rorabaugh; ISBN 0-0705-4004-7



- "A DSP Primer : With Applications to Digital Audio and Computer Music" by Ken Steiglitz; ISBN 0-8053-1684-1

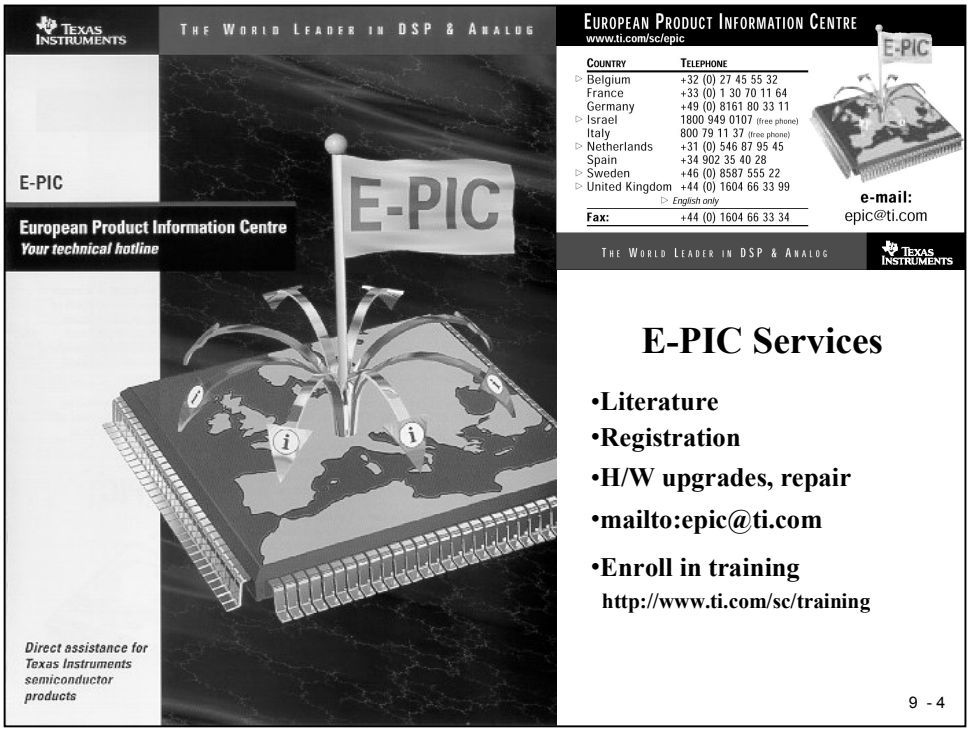
- "DSP First : A Multimedia Approach (Matlab Curriculum Series)" James H. McClellan; ISBN 0-1324-3171-8

9 - 1

World's Largest DSP Third-Party Network

www.ti.com/sc/3p

9 - 3



University of Applied Sciences Zwickau

- **http://www.fh-zwickau.de/tutorial/dsp**
- **TMS320F24x - web site**
  - **Application notes**
  - **Students exercises / lab sessions**
- **Consulting**
- **Student placements / graduates**
- **co-operation / prototype design**
- **e-mail : Frank.Bormann@fh-zwickau.de**

9 - 5