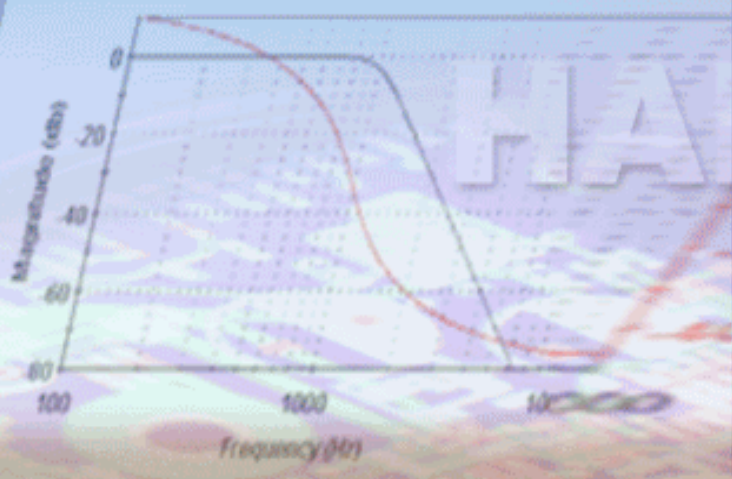


HANDS-ON



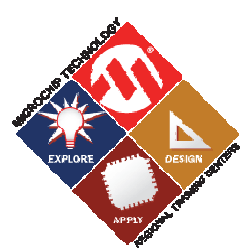
203 PRC 16-bit Standard Peripherals

Hand Out



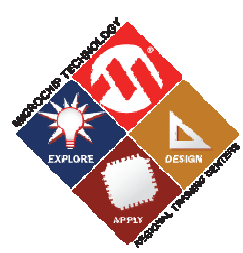
Training

v4.2 Oct 25, 2007



MPLAB Navigation

- **Quick ways to find functions or variables in MPLAB**
 - Source Locator
 - **To Enable**
 - Right-click on editor and go to “Properties...”
 - Check “Enable Source Locator”
 - On the Project window, click on the “Symbols” tab. Right click and check “Enable Tag Locators”
 - **Use this feature to quickly navigate through large applications**
 - Right-click on a function or variable in code and select “Goto Locator” to jump its definition
 - In the project window under the symbols tab, you can browse through and double click items to jump there in code
 - Edit->Find in Files (ctrl+shift+F)
 - **Use this to search all files in the project for a variable, function name, or anything else**

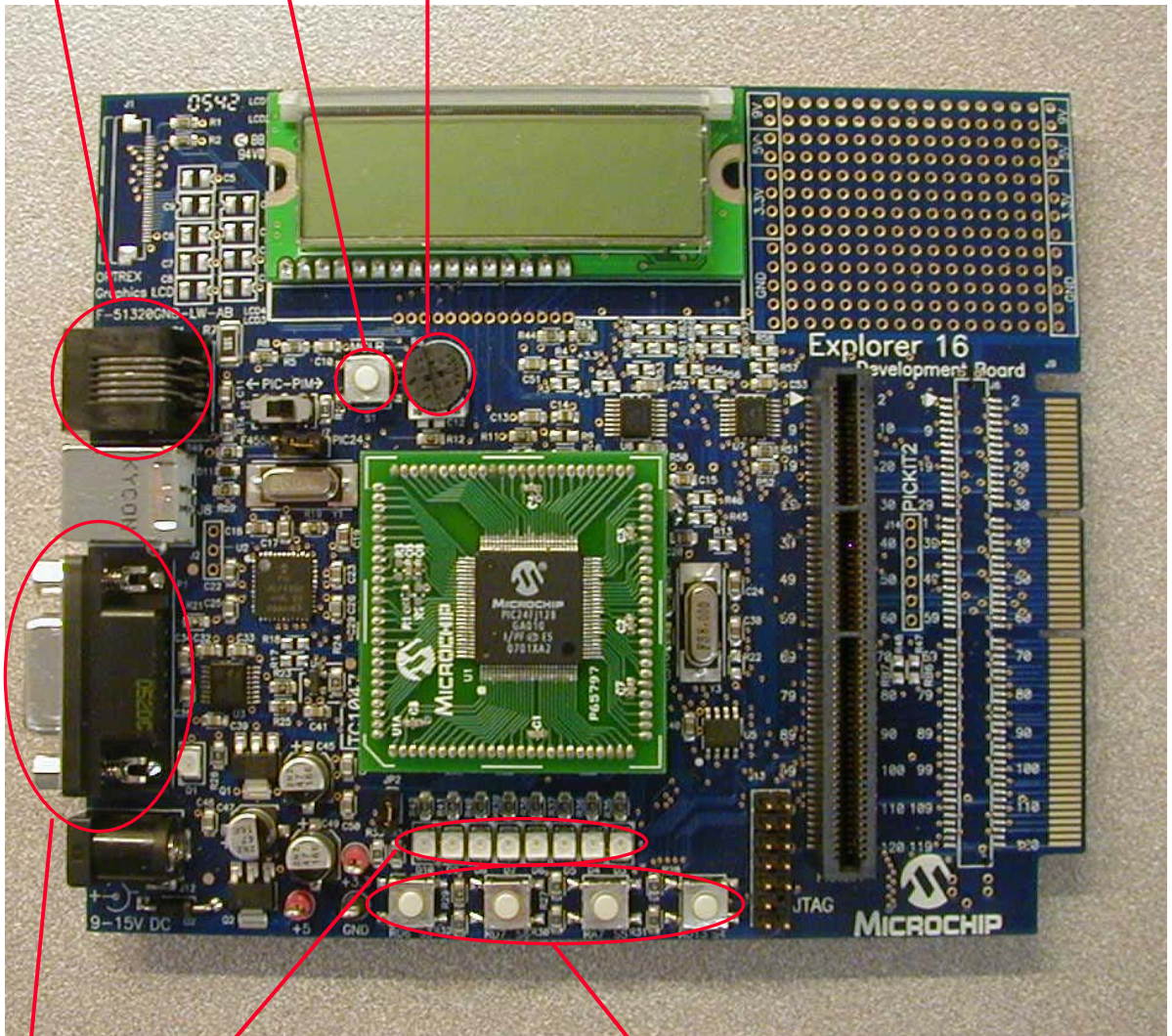


Explorer 16

RESET SWITCH

POT

ICD2 connector



LEDs

Switches

RS232 connector

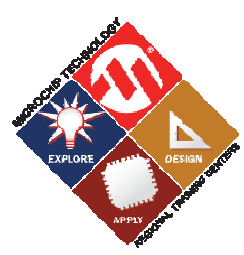
The Bode plot shows the magnitude response of a high-pass filter. The y-axis is labeled 'Magnitude (dB)' and ranges from 0 to -80. The x-axis is labeled 'Frequency (Hz)' and is on a logarithmic scale with major ticks at 100, 1000, and 10000. The magnitude is 0 dB for frequencies below 1000 Hz and drops to -80 dB for frequencies above 1000 Hz. The transition region is centered around 1000 Hz.

LAB 1

Working with C30 & MPLAB® IDE

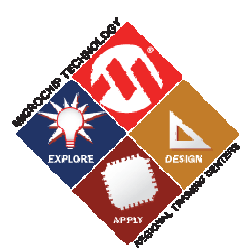


v4.2 Oct 25, 2007



LAB 1 Goals

- **To work with MPLAB[®] IDE environment**
- **To Do:**
 - Follow the presenter
- **Expected Result:**
 - Successfully build the project and program the device
 - LED D3 should blink

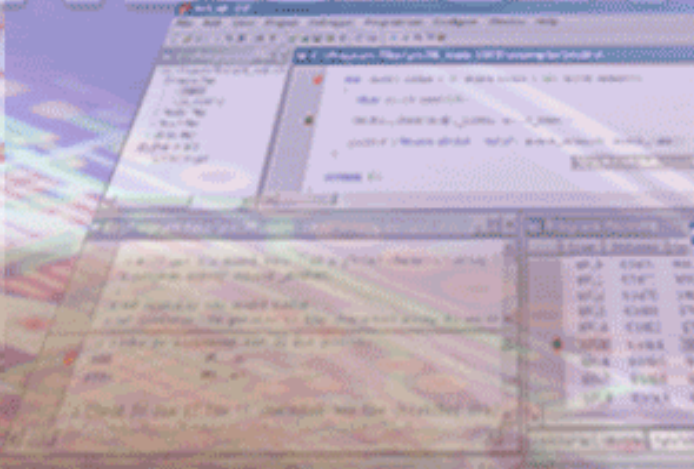
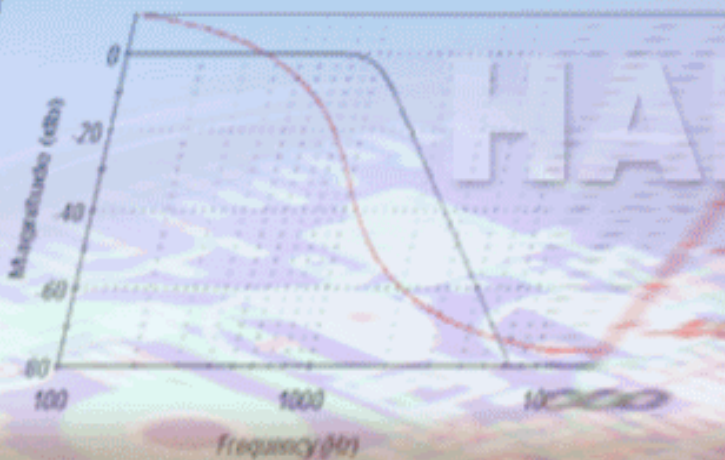


LAB 1 Solution

```
MPLAB IDE Editor
Lab1_solution.c

136  /***** START OF MAIN FUNCTION *****/
137
138  int main ( void )
139  {
140
141      TRISAbits.TRISA0 = 0; //setup for LED output
142
143      while(1){
144
145          __builtin_btg((__builtin_int*)LATA,0x0); //toggle the LED pin
146
147          delay(); //wait for some time
148
149      }
150  }
151
152
153  void delay(void)
154  {
155      unsigned int i;
156
157      //delay for a while
158      for(i = 0; i < 0xFFFF; i++);
159
160  }
161
162  /***** END OF MAIN FUNCTION *****/
```


HANDS-ON

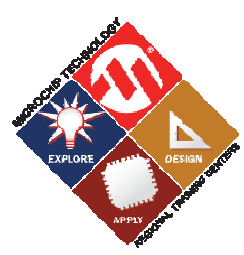


LAB 2

Working with PSV



v4.2 Oct 25, 2007



LAB 2 Goals

- **To initialize PSV**
- **To store a “Hello World” string in PSV space**
- **To read and display this string on the LCD**



LAB 2 To Do

- **Open the project**

- C:\RTC\203_PRC\Lab2\Lab2.mcp

- **Open the file**

- C:\RTC\203_PRC\Lab2\Lab2.c

- **Step 1**

- Use Space attribute and define a array “MyString” and place it in PSV space.

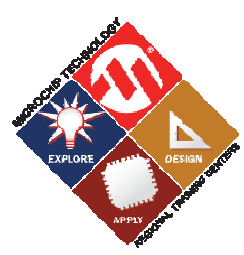
- `__attribute__((space(psv)))`

- **Step 2**

- In the PSVInit() function use the built-in function of C30 to load PSVPAG register with the page of PSV space where the array is placed.

- `__builtin_psvpage(variable)`

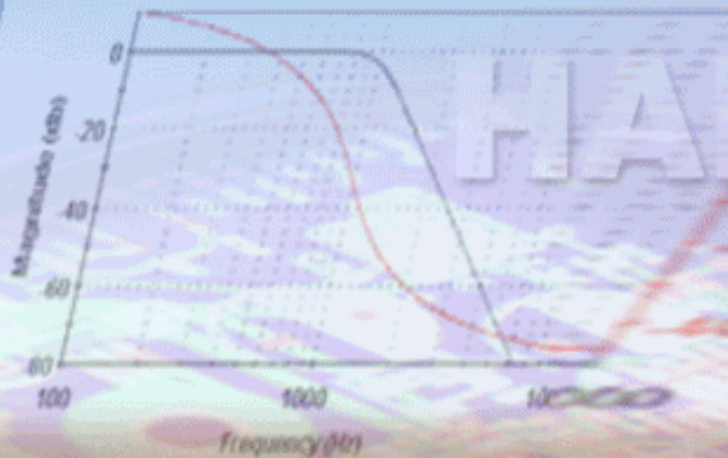
- **Build the Project and program the device**



LAB 2 Expected Result

- **Compare the content of the LCD display with the array defined.**
- **Both should match**

HANDS-ON



LAB 3

Interrupt Handling

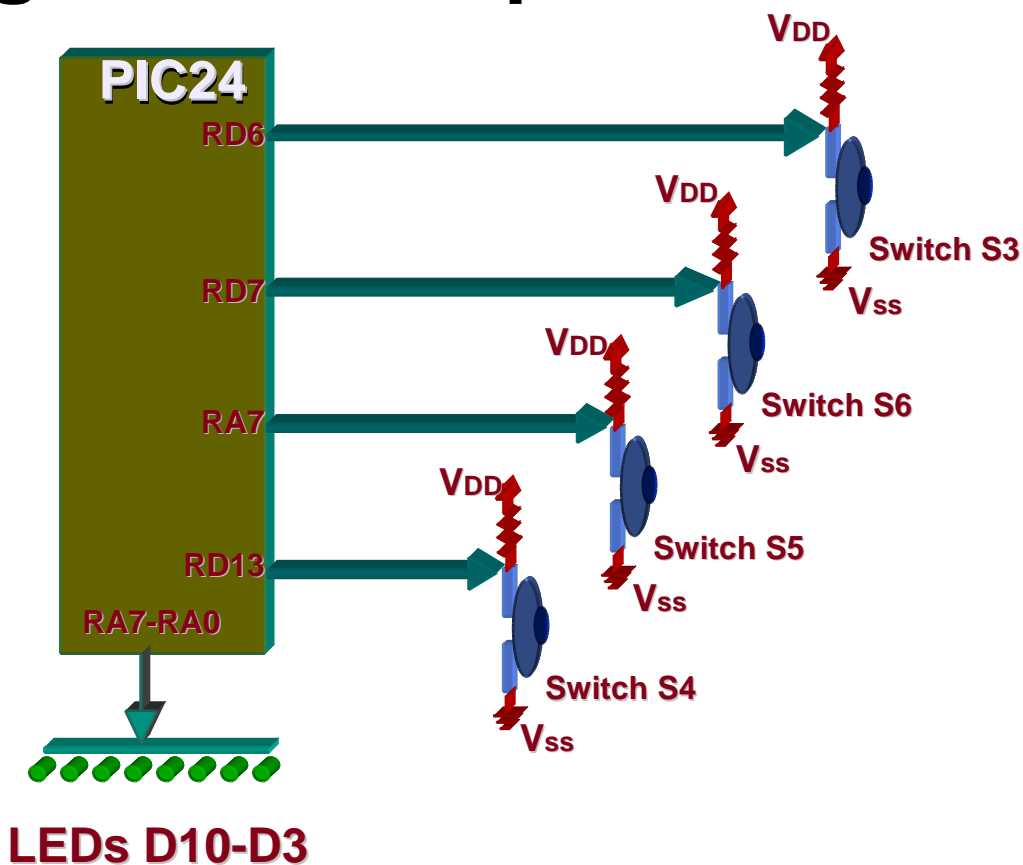


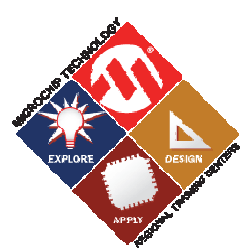
v4.2 Oct 25, 2007



LAB 3 Goals

- Understand Interrupt configuration
- Understand Interrupt priority
- Writing Interrupt handler for a given Interrupt vector





LAB 3 To Do

- **Open the project**
 - C:\RTC\203_PRC\Lab3\Lab3.mcp
- **Open the file**
 - C:\RTC\203_PRC\Lab3\Lab3.c
- **Here Timer 3 is configured to give .5 sec ticks and Timer 5 is configured to give .25 Sec ticks.**
- **LED D3 indicates CPU is in T3 ISR and LED D7 indicates CPU is in T5 ISR.**
- **Look at TMR3Init and TMR5Init to see how timer interrupts are enabled.**



LAB 3 To Do

- **Step 1: Case 1 – S3**

- Look for Switch Case1 and configure T3 priority to be higher than T5 priority. **NOTE: T3 & T5 interrupts must be disabled before modifying IPR bits, then re-enabled afterwards!**
- By pressing S3 Case 1 will be executed.

- **Step 2: Case 2 – S6**

- Look for Switch Case2 and configure T5 priority to be higher than T3 priority. **NOTE: T3 & T5 interrupts must be disabled before modifying IPR bits, then re-enabled afterwards!**
- By pressing S6 Case 2 will be executed.

- **Step 3: Case 3 – S5**

- Look for Switch Case3 and configure CPU priority to be higher than T5 priority and lower than T3 priority. **NOTE: T3 & T5 interrupts must be disabled before modifying IPR bits, then re-enabled afterwards!**
- By pressing S5 Case 3 will be executed.

- **Step 4: Case 4 – S4**

- Look for Switch Case4 and configure CPU priority to be higher than both T3 and T5 priority. **NOTE: T3 & T5 interrupts must be disabled before modifying IPR bits, then re-enabled afterwards!**
- By pressing S4 Case 4 will be executed.

- **Watch what switch you are pressing!**

- They are not in ascending order on the board

- **To configure the priority levels you require to write into some registers, IPC2, IPC7 and SR. The details are given in following pages**

- **Build the project and program the device**



LAB 3 Interrupt Registers

IPC2: Interrupt priority control Register 2

Bit:15

Bit:8

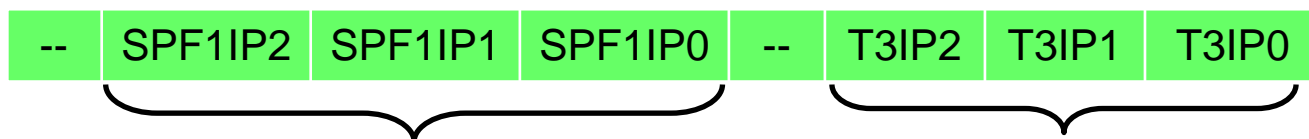


UART 1 Receive
Interrupt Priority
Level

SPI 1 Event
Interrupt Priority
Level

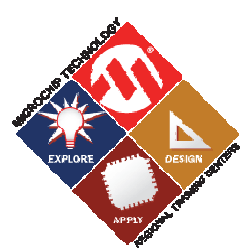
Bit:7

Bit:0



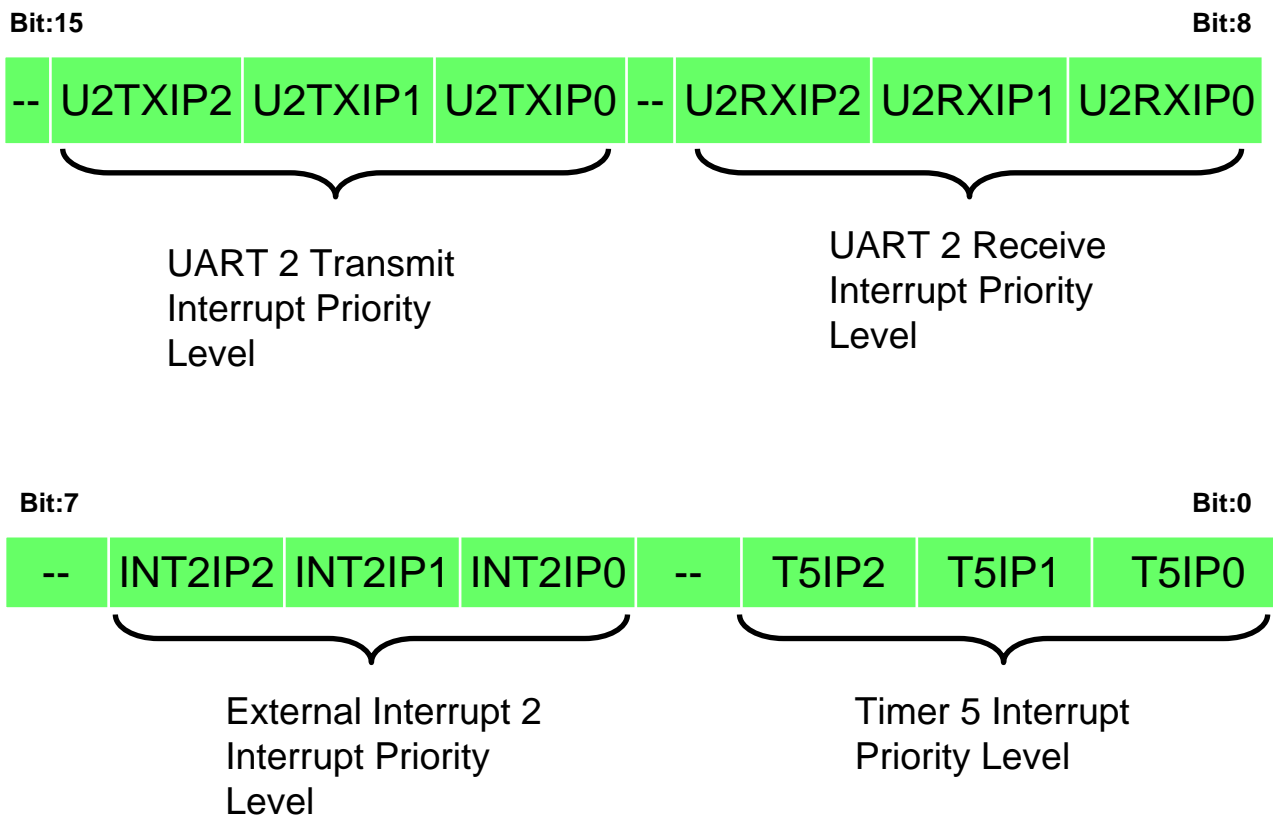
SPI 1 Error
Interrupt Priority
Level

Timer 3 Interrupt
Priority Level



LAB 3 Interrupt Registers

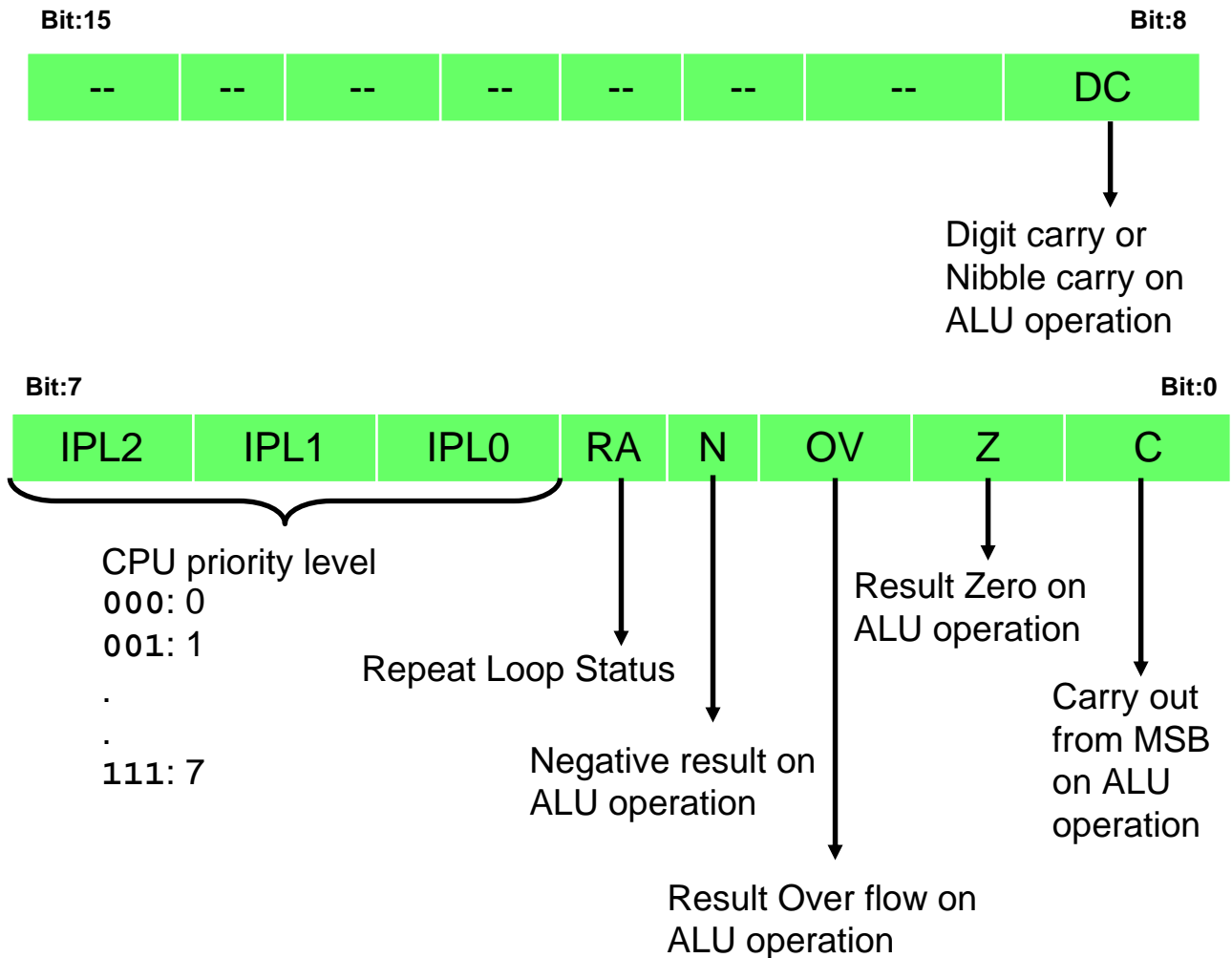
IPC7: Interrupt priority control Register 7





LAB 3 CPU Registers

SR: CPU Status Register





LAB 3 Expected Result

..... T5 ISR
- - - T3 ISR
— CPU Idle

● Case 1

- The LEDs D3 and D7 will be flashing but one after the other
- D7 will never be ON when D3 is ON as T5 cannot preempt T3 ISR.



● Case 2

- The LEDs D3 and D7 will be flashing at the same time
- D7 becomes ON even when D3 is ON as T5 can preempt T3 ISR



● Case 3

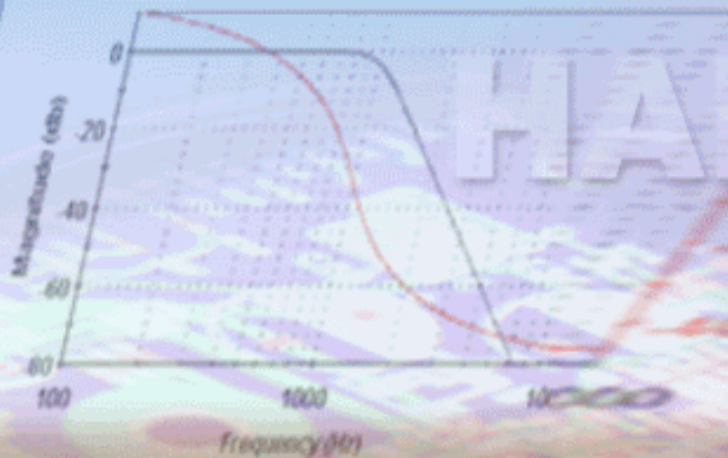
- The LED D3 will be flashing but not D7 as T5 cannot interrupt the CPU



● Case 4

- Both the LEDs D3 and D7 stops flashing as both T3 and T5 cannot interrupt the CPU

HANDS-ON

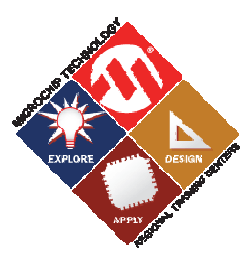


LAB 4

Working with ADC

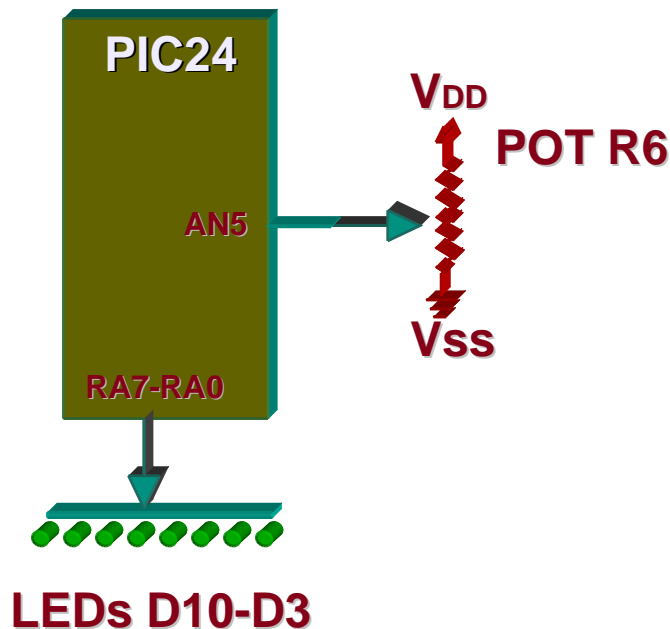


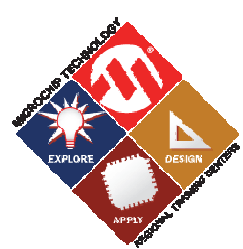
v4.2 Oct 25, 2007



LAB 4 Goals

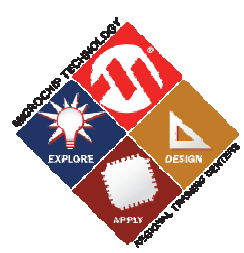
- To configure ADC
- To configure I/O ports
- To read ADC and display on LEDs





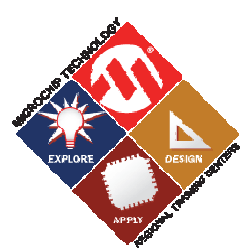
LAB 4 To Do

- **Open the project**
 - C:\RTC\203_PRC\Lab4\Lab4.mcp
- **Open the file**
 - C:\RTC\203_PRC\Lab4\Lab4.c
- **Look for ADCInit() function and configure ADC by initializing the registers AD1CON1, AD1CON2, and AD1CON3 looking into the Register details on the next few pages.**
 - ▶ **STEP 1: AD1CON1**
 - ▶ **Select Integer Format Result**
 - ▶ **Auto Conversion Start**
 - ▶ **Sample after conversion**
 - ▶ **STEP 2: AD1CON2**
 - ▶ **Select AVDD and AVss as references**
 - ▶ **Disable Scan mode**
 - ▶ **Interrupt at 16th sample/Convert sequence**
 - ▶ **16*1 level buffer**
 - ▶ **Always use Mux A**
 - ▶ **STEP 3: AD1CON3**
 - ▶ **Select Sample Time = 13TAD**
 - ▶ **Conversion Time is always 12TAD**
 - ▶ **Select AD Clock Source such that you get 16 samples in around 1 mSec (16 ksp/s)**
 - ▶ **Assume 1TCY = .25 μ S (FCY = 4 MHz)**



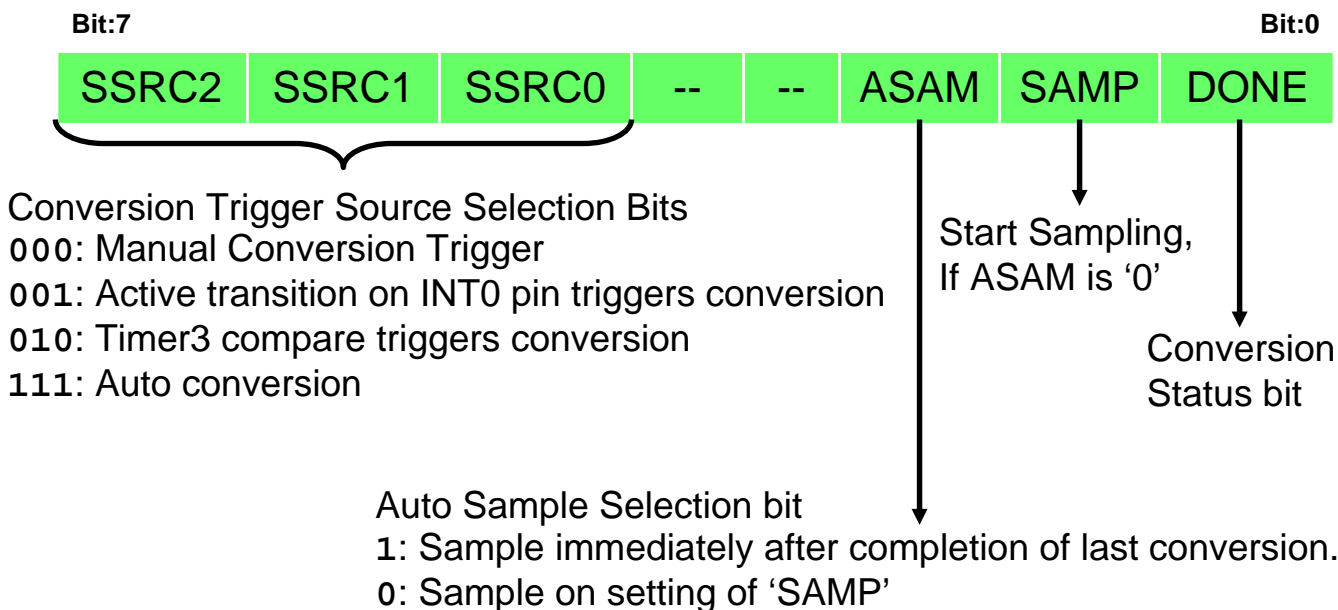
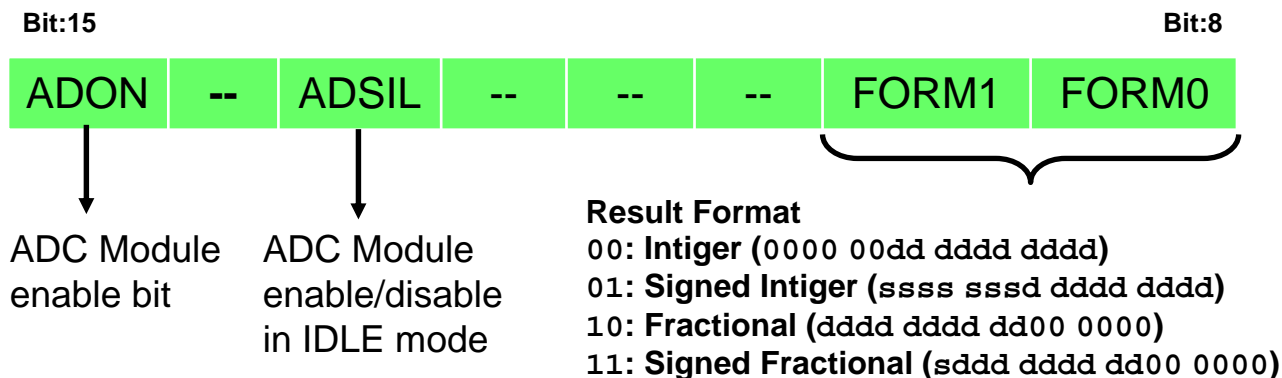
LAB 4 To Do

- **Continue to configure ADC by initializing the registers AD1CHS, AD1PCFG, and AD1CSSL looking into the Register details on the next few pages.**
 - ▶ **STEP 4: AD1CHS**
 - ▶ **Set the positive sample input channel for MUX A to use AN5**
 - ▶ **Set the negative input channel for MUX A to use VR-**
 - ▶ **STEP 5: AD1PCFG**
 - ▶ **Set AD1PCFG so that the only pin using analog functionality is AN5**
 - ▶ **STEP 6: AD1CSSL**
 - ▶ **Channel scanning is not enabled, so no input channels should be selected for scanning**
- **Build the project and program the device**
- **Procedure to Test**
 - **Vary the POT and observe LEDs**



LAB 4 ADC Registers

AD1CON1: A/D CONTROL REGISTER 1





LAB 4 ADC Registers

AD1CON2: A/D CONTROL REGISTER 2

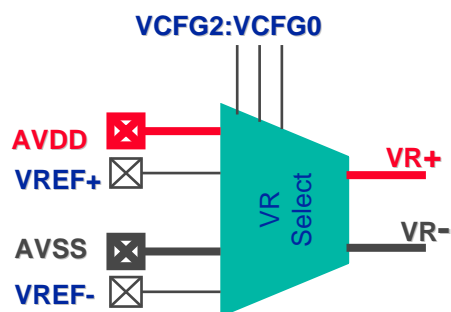
Bit:15

Bit:8



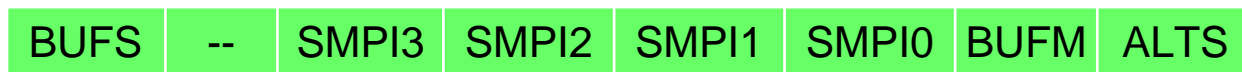
VCFG2:VCFG0	VR+	VR-
000	AVDD	AVSS
001	VREF+	AVSS
010	AVDD	VREF-
011	VREF+	VREF-
1xx	AVDD	AVSS

Scan CH0 Mux A Input



Bit:7

Bit:0



SMPI3:SMPI0	Interrupt Event (Sample/convert sequence)
0000	each
0001	alternate
...	...
1110	Every 15th
1111	Every 16th

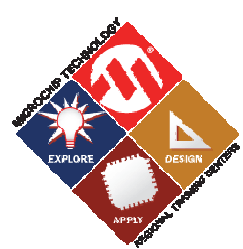
Sample alternatively MUX-A & MUX-B

Buffer Status bit, is valid only when BUFM = '1'

- 1: Buffer 8-F is being filled, can access Buffer 0-7
- 0: Buffer 0-7 is being filled, can access Buffer 8-F

Buffer Mode Select bit

- 1: Buffer configured as two 8-words buffers
- 0: Buffer configured as one 16-words buffers



LAB 4 ADC Registers

AD1CON3: A/D CONTROL REGISTER 3

Bit:15

Bit:8



A/D conversion Clock Source

1: ADRC is used

0: System clock is used

A/D Sample Time Selection bits

SAMC4:SAMC0	Sampling Time
00000	0 T_{AD}
00001	1 T_{AD}
....
11110	30 T_{AD}
11111	31 T_{AD}

Bit:7

Bit:0



A/D Conversion Clock Selection bits

$$ADCS = (T_{AD}/T_{CY}) - 1$$

ADCS7:ADCS0	Conversion Clock
00000000	$T_{CY} (F_{CY})$
00000001	$2 * T_{CY} (F_{CY} / 2)$
....
11111110	$255 * T_{CY} (F_{CY} / 255)$
11111111	$256 * T_{CY} (F_{CY} / 256)$



LAB 4 ADC Registers

AD1CHS : A/D Input Select Register

Bit:15

Bit:8

CH0NB	--	--	--	CH0SB3	CH0SB2	CH0SB1	CH0SB0
-------	----	----	----	--------	--------	--------	--------

CH0 Negative input for
MUX B
1: AN1
0: VR-

CH0SB3:CH0SB0	CH0 Positive Input for MUX B
0000	AN0
0001	AN1
....
1110	AN14
1111	AN15

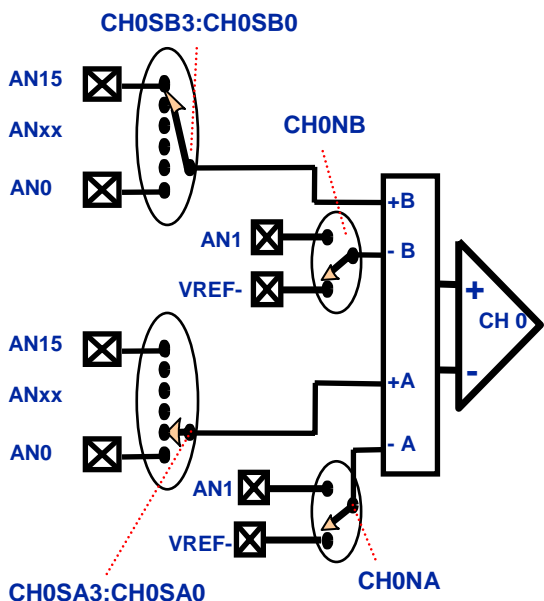
Bit:7

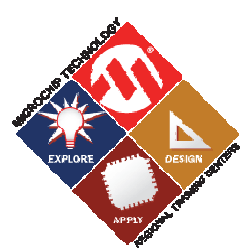
Bit:0

CH0NA	--	--	--	CH0SA3	CH0SA2	CH0SA1	CH0SA0
-------	----	----	----	--------	--------	--------	--------

CH0 Negative input for
MUX A
1: AN1
0: VR-

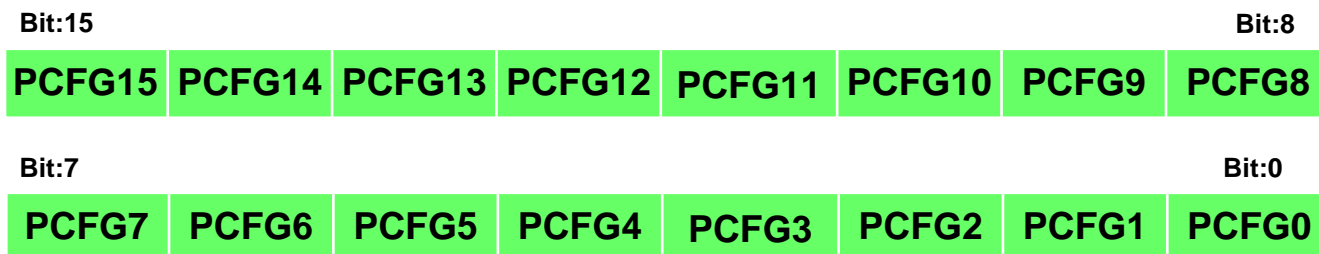
CH0SA3:CH0SA0	CH0 Positive Input for MUX A
0000	AN0
0001	AN1
....
1110	AN14
1111	AN15





LAB 4 ADC Registers

AD1PCFG : A/D Port Configuration Register

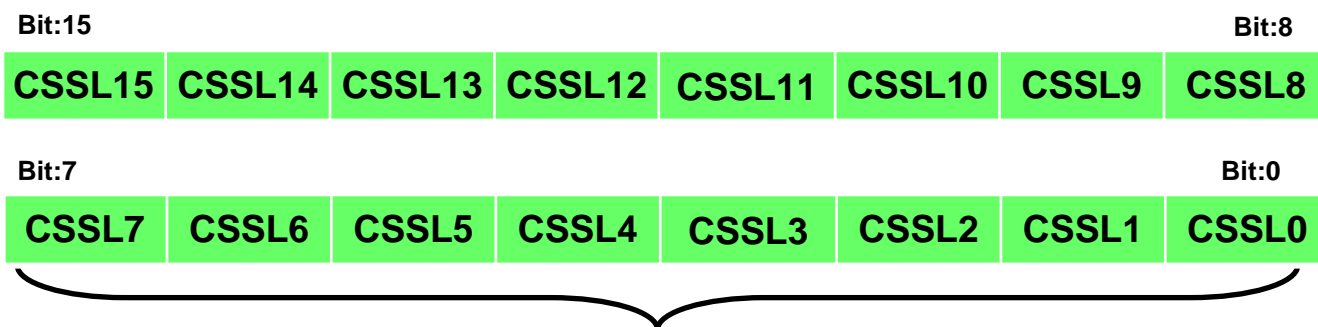


Analog Input Pin Configuration Control bits 0 to 15

1: Pin for corresponding analog channel (ANxx) is in digital mode

0: Pin for corresponding analog channel (ANxx) is in analog mode

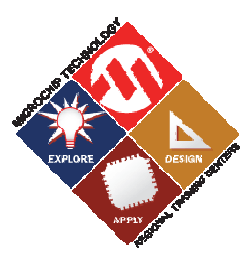
AD1CSSL : A/D Input Scan Select Register



A/D Input Channel Scan Selection bits 0 to 15

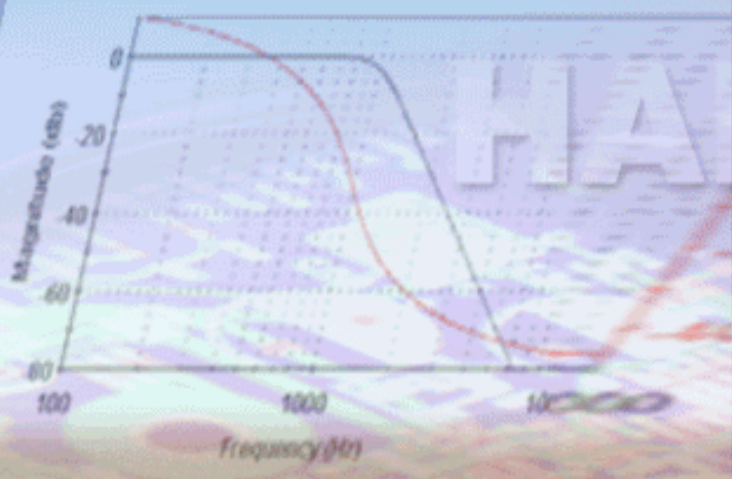
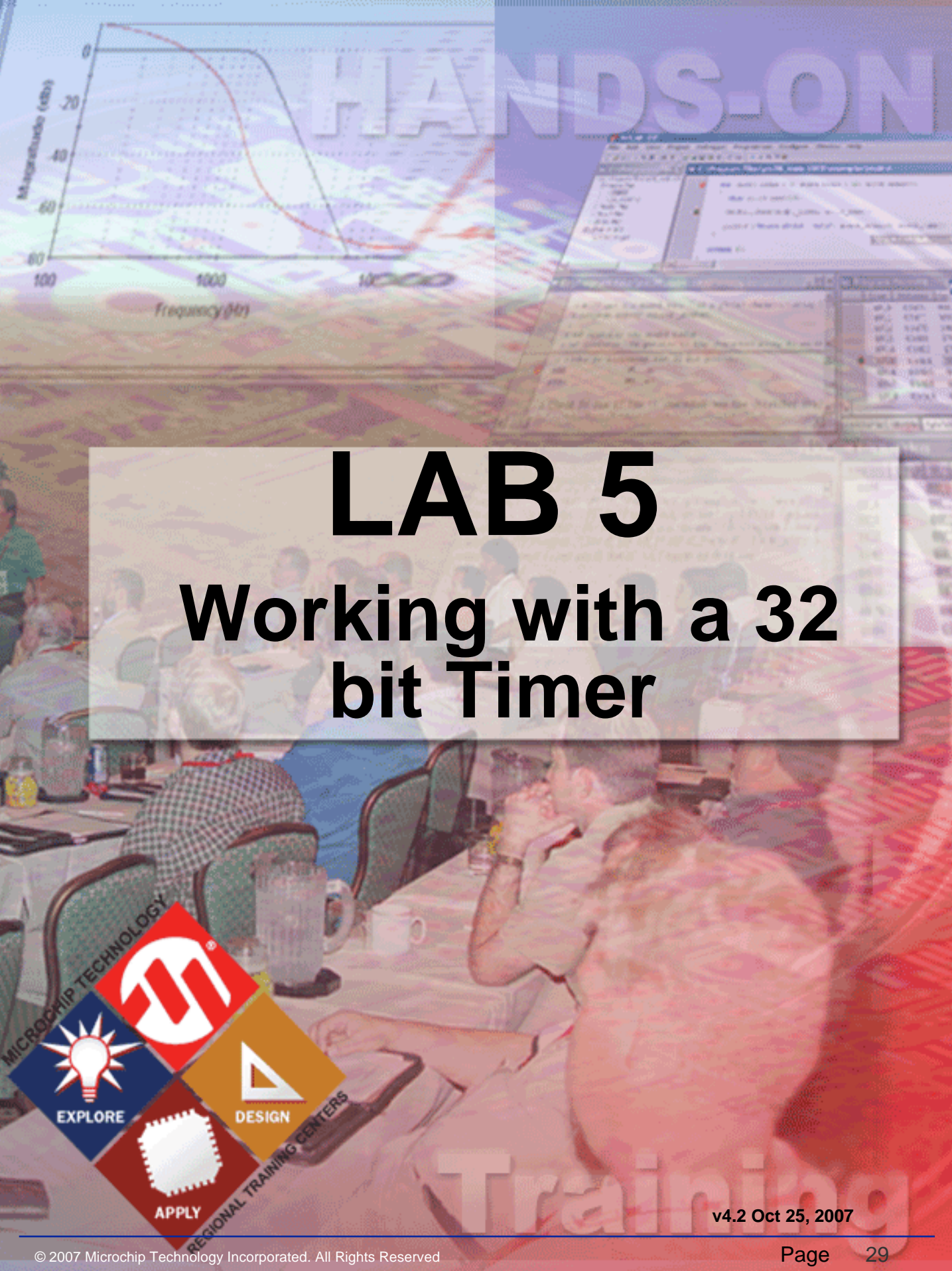
1: Corresponding analog channel (ANxx) is selected for sequential scanning

0: Corresponding analog channel (ANxx) is ignored for sequential scanning



LAB 4 Expected Result

- **POT value is averaged for 16 samples over 1 ms.**
- **POT value is displayed on LEDs as a binary value from 0 to 255**
- **Pin RB2 toggles each time 16 samples are taken (a frequency of 500 Hz)**



LAB 5

Working with a 32 bit Timer

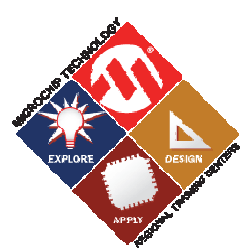


Training



LAB 5 Goals

- **Understand working of Timers in 32bit mode**
- **Configure the Timer 2/3 pair for 32 bit mode**
- **Implement a stop watch**



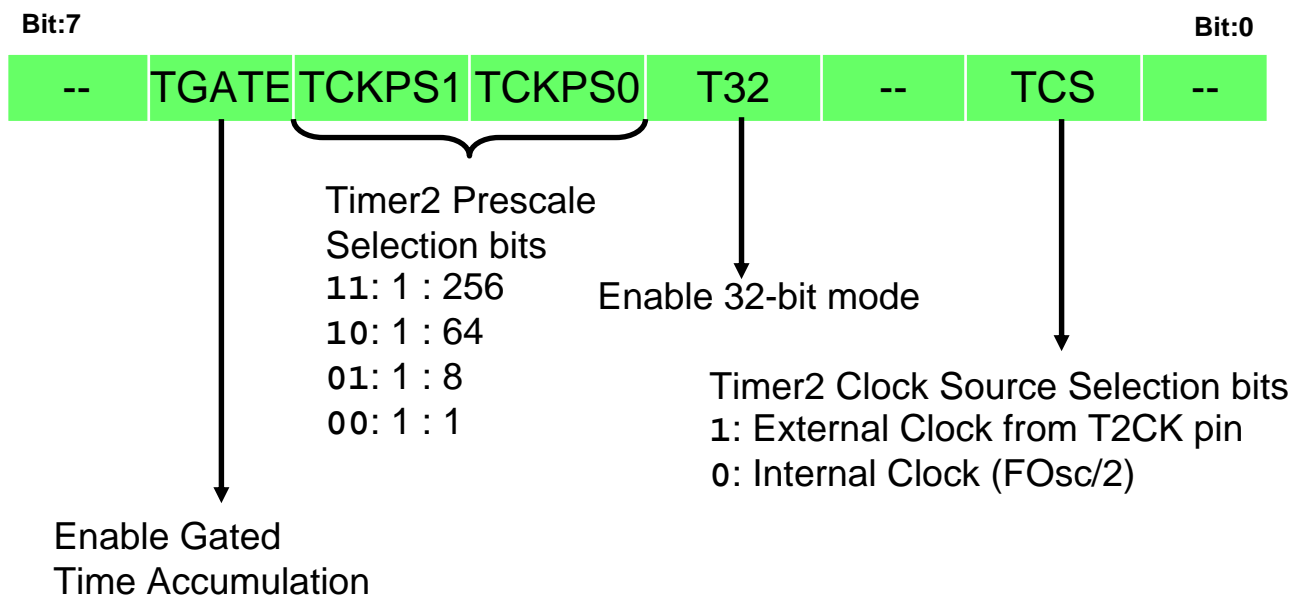
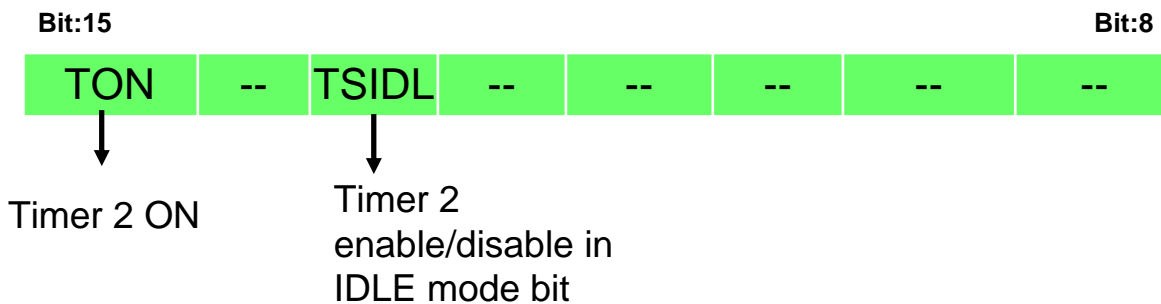
LAB 5 To Do

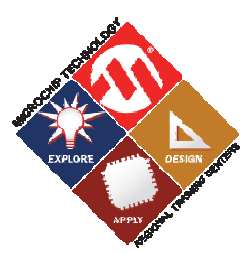
- **Open the project**
 - C:\RTC\203_PRC\Lab5\Lab5.mcp
- **Open the file**
 - C:\RTC\203_PRC\Lab5\Lab5.c
- **Look for Timer23Init() function and configure Timer 2 & 3 by initializing the registers T2CON, PR2 and PR3.**
 - ▶ **STEP 1: T2CON**
 - ▶ **Select Internal clock as clock source (Fosc/2)**
 - ▶ **1:1 Pre-scale**
 - ▶ **No Gated time accumulation**
 - ▶ **32-bit Timer mode**
 - ▶ **STEP 2: PR3 & PR2**
 - ▶ **Load the count to get 1 tenth of a second ticks**
 - ▶ **$PR = (Fosc/2) / (\# \text{ Ticks per second})$**
- **Build the project and program the device**



LAB 5 Timer Registers

T2CON: Timer 2 Control Register

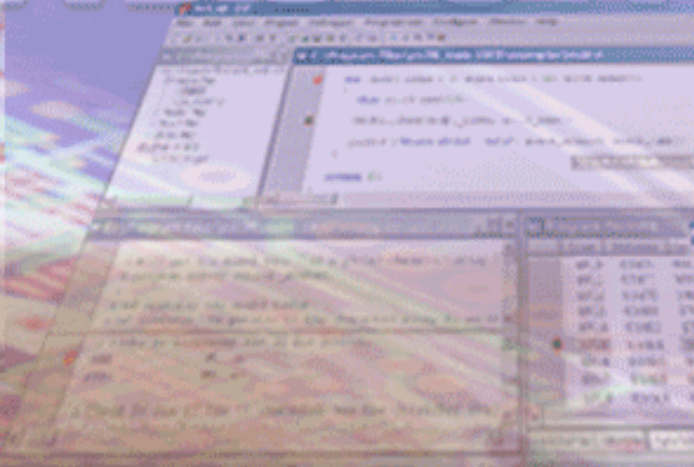
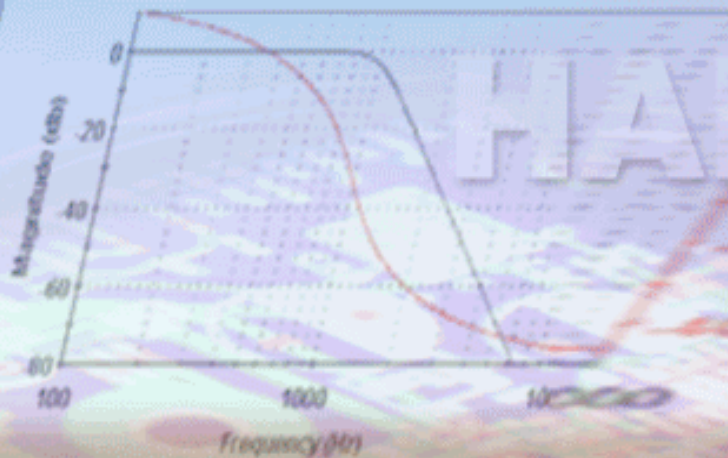




LAB 5 Expected Result

- **The On-Board LCD will be Displaying**
 - **Press S3 - Start**
- **Press the Switch S3 to start timer**
- **Again press the Switch S3 to stop timer and LCD displays the Time elapsed between the start and stop**

HANDS-ON

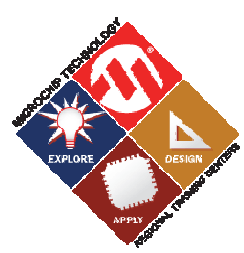


LAB 6

Working with UART



v4.2 Oct 25, 2007



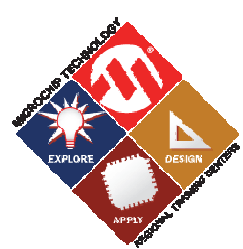
LAB 6 Goals

- **Understand Configuration of UART module**
- **Understand Transmit and Receive interrupts**
- **Understand the advantages of the FIFO**
 - A slow baud rate is used to make this more easily visible
- **Write a software to Transmit and Receive data using UART**



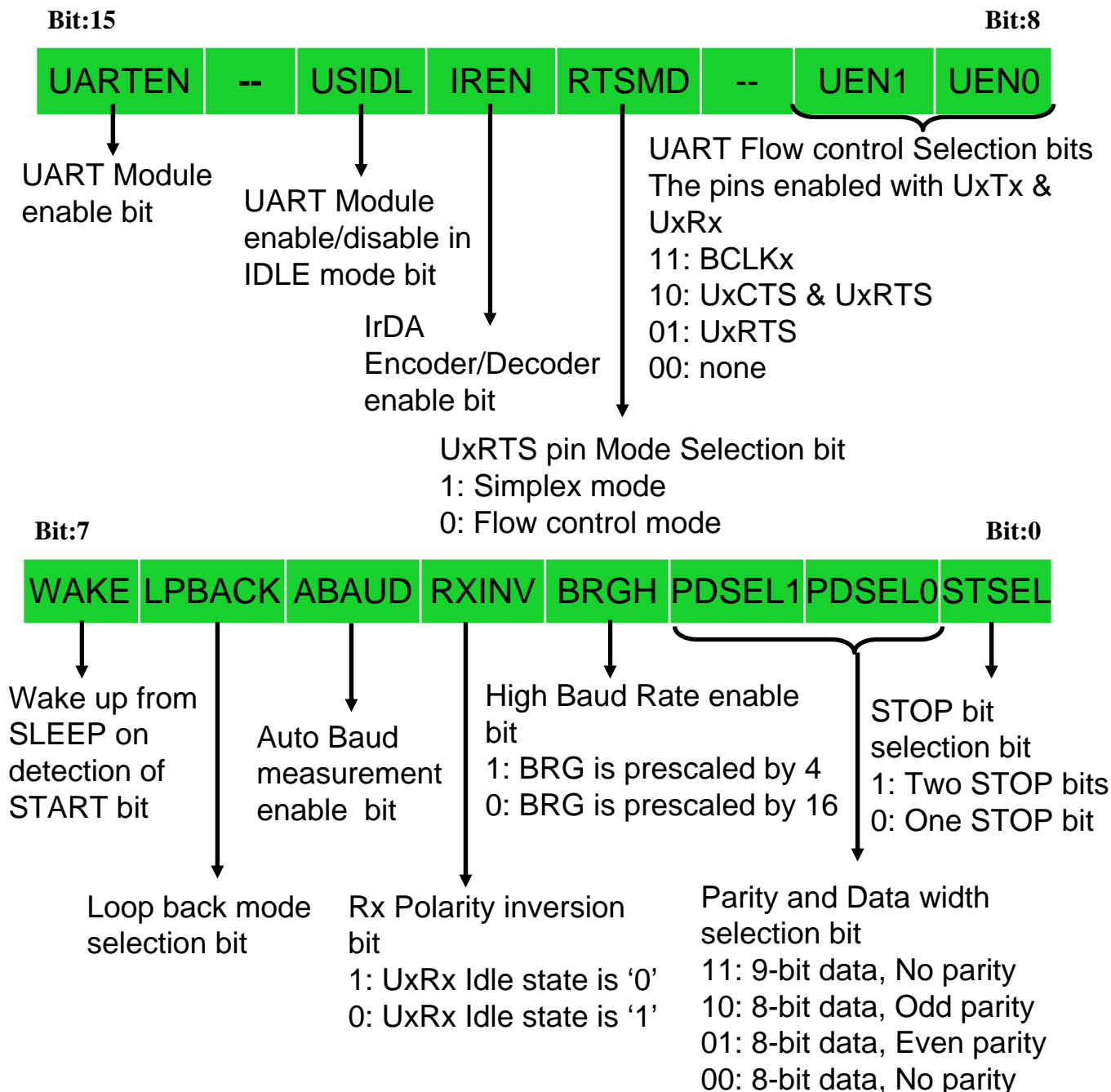
LAB 6 To Do

- **Open the project**
 - C:\RTC\203_PRC\Lab6\Lab6.mcp
- **Open the file**
 - C:\RTC\203_PRC\Lab6\Lab6.c
- **Look for UARTInit() function and configure UART by initializing the registers U2MODE, U2STA and U2BRG.**
 - ▶ **STEP 1: U2BRG**
 - ▶ Load the count to get 300 baudrate
 - ▶ $BRG = F_{cy}/(16 * BaudRate) - 1$ where $F_{cy} = 4MHz$
 - ▶ **STEP 2: U2MODE**
 - ▶ Select No IrDA
 - ▶ Select Flow Control Mode
 - ▶ Enable RTS and CTS
 - ▶ No Wake-Up
 - ▶ No Loop-back mode
 - ▶ No Auto baud
 - ▶ No Rx Inversion
 - ▶ Low Baudrate
 - ▶ 1 Stop-bit
 - ▶ 8-bit data with parity of your choice
 - ▶ Enable the UART
 - ▶ **STEP 3: U2STA**
 - ▶ No Tx Inversion
 - ▶ Disable Sync Break
 - ▶ Enable Transmit
 - ▶ Disable Address Detect
 - ▶ Interrupt on every receive.
- **Build the project and program the device**
- **Using HyperTerminal**
 - Transmit a text file C:\RTC\203_PRC\Lab6\TextFile.txt using the Hyper Terminal
 - OR**
 - Type characters into the HyperTerminal window
- **Make sure the settings of Hyper Terminal are: 300 baud, 8-bit data, Parity of your choice, 1 Stop bit, Hardware Flow Control**



LAB 6 UART Registers

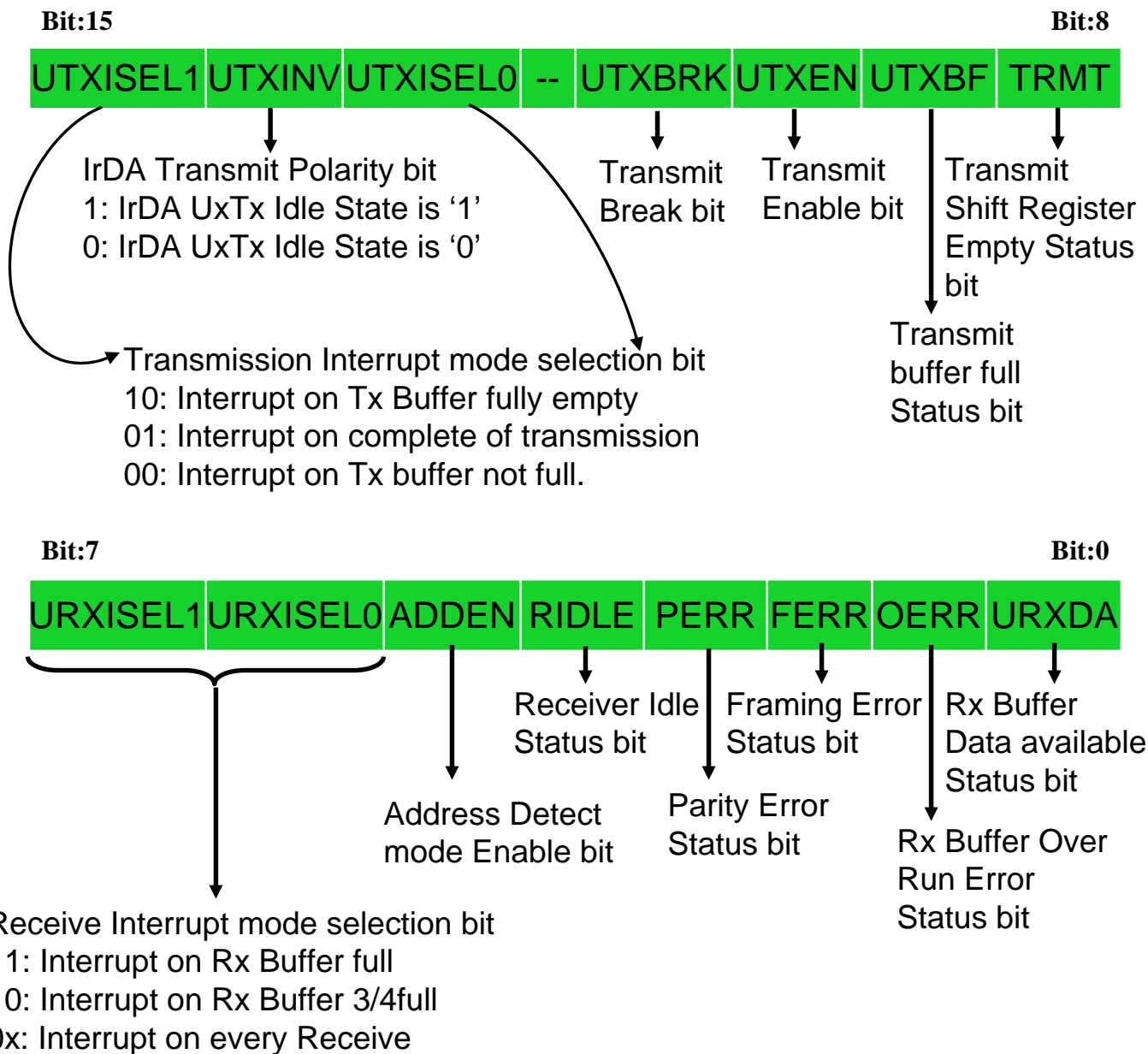
UxMODE: UART Mode register





LAB 6 UART Registers

UxSTA: UART Status and Control register





LAB 6 Expected Result

- **LED D3 indicates amount of CPU time spent processing UART data. Slower blinking means indicates less time spent servicing UART interrupt routine.**
- **Press Switch S4, the rate at which LED flashes decreases as now UART is using the RxBuffer and interrupting only on Buffer.**



Normal LED pattern – blink each receive



S4 Pressed – blink each full buffer, 4 characters received

- **The transmitted data can be observed on the screen as the received Data is transmitted back**
 - Without S4, the data comes back as you enter it
 - With S4 pressed, the data will come back in packets of 4

