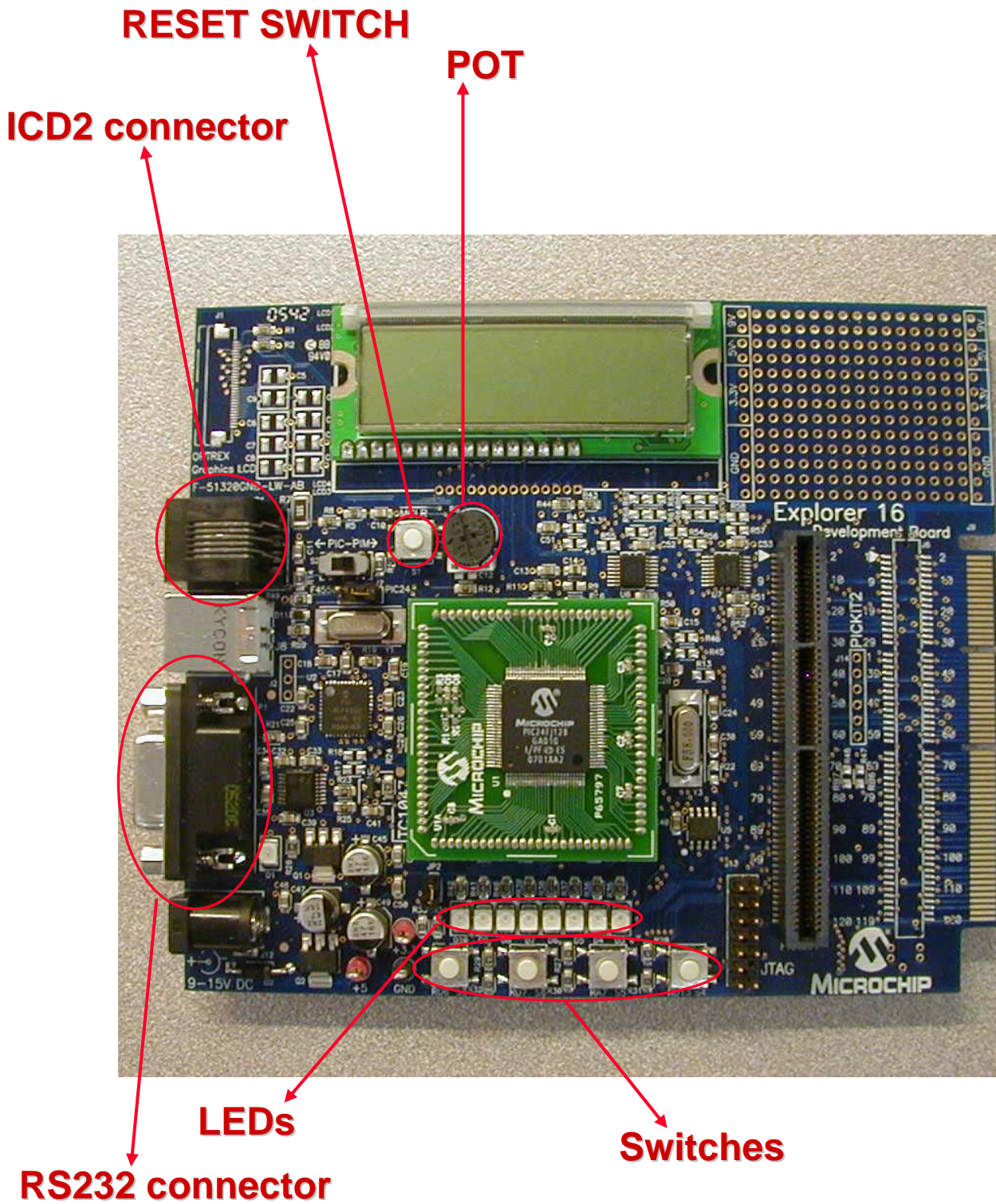




11011 EXP 16-bit Advanced Peripherals

Hand Out

Explorer 16





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**

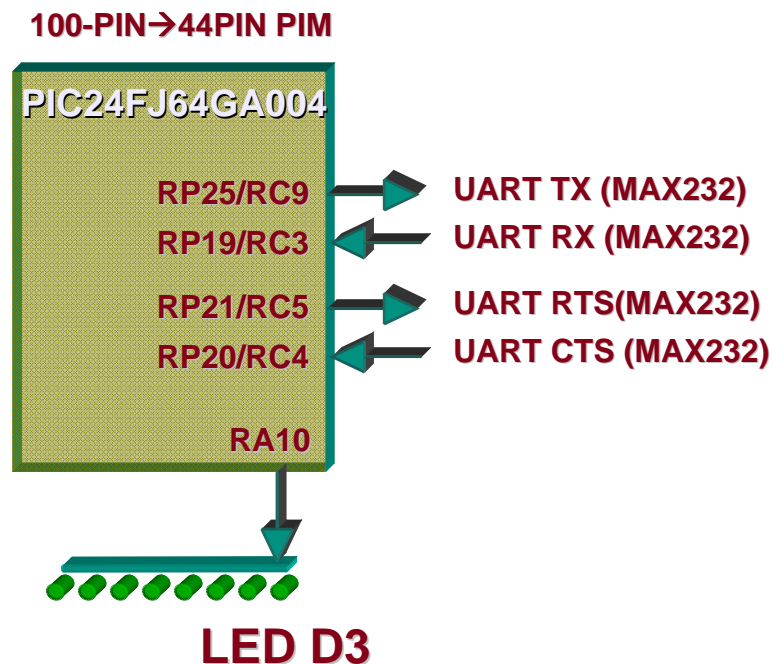


Lab 1

Peripheral Pin Select (PPS)

Lab 1 Goals

- To configure the PPS module with UART2 on the PIC24FJGA004 PIM on Explorer16
- To echo typed characters to HyperTerminal





Lab 1 To Do

- **Replace PIC24FJ128GA010 PIM with supplied PIC24FJ64GA004 PIM**
- **Open workspace C:\RTC\204_ADV\Lab1-PPS\PPS.mcw**
- **In “PPS.c”**
 - **STEP 1**
 - **Set up the Configuration Bits (via menu or using `_CONFIG` directive):**
 - JTAG = Disabled
 - OSC = Primary – HS
 - WDT = Disabled
 - IOLOCK = 1 Way
 - **STEP 2**
 - **In function “`ioMap()`”, issue the unlock sequence for the Reprogrammable Pin Mechanism (see function `ioLock()`)**
 - **STEP 3-6**
 - **In function “`ioMap()`”:**
 - Map RP19 pin to input function U2RX
 - Map RP20 pin to input function U2CTS
 - Map Output Function U2TX to RP25
 - Map Output Function U2RTS to RP21
 - **STEP 7**
 - **In function “`ioMap()`”, issue the lock sequence for the Reprogrammable Pin Mechanism (see function `ioLock()`)**
- **Open HyperTerminal file “Lab1.ht” (9600/8/N/1/Hardware Flow Control)**



Lab 1

RPINRx

TABLE 9-1: SELECTABLE INPUT SOURCES (MAPS INPUT TO FUNCTION)⁽¹⁾

Input Name	Function Name	Register	Configuration Bits
External Interrupt 1	INT1	RPINR0	INT1R[4:0]
External Interrupt 2	INT2	RPINR1	INT2R[4:0]
Timer 2 External Clock	T2CK	RPINR3	T2CKR[4:0]
Timer 3 External Clock	T3CK	RPINR3	T3CKR[4:0]
Timer 4 External Clock	T4CK	RPINR4	T4CKR[4:0]
Timer 5 External Clock	T5CK	RPINR4	T5CKR[4:0]
Input Capture 1	IC1	RPINR7	IC1R[4:0]
Input Capture 2	IC2	RPINR7	IC2R[4:0]
Input Capture 3	IC3	RPINR8	IC3R[4:0]
Input Capture 4	IC4	RPINR8	IC4R[4:0]
Input Capture 5	IC5	RPINR9	IC5R[4:0]
Output Compare Fault A	OCFA	RPINR11	OCFAR[4:0]
Output Compare Fault B	OCFB	RPINR11	OCFBR[4:0]
UART 1 Receive	U1RX	RPINR18	U1RXR[4:0]
UART 1 Clear To Send	U1CTS	RPINR18	U1CTSR[4:0]
UART 2 Receive	U2RX	RPINR19	U2RXR[4:0]
UART 2 Clear To Send	U2CTS	RPINR19	U2CTSR[4:0]
SPI 1 Data Input	SDI1	RPINR20	SDI1R[4:0]
SPI 1 Clock Input	SCK1IN	RPINR20	SCK1R[4:0]
SPI 1 Slave Select Input	SS1IN	RPINR21	SS1R[4:0]
SPI 2 Data Input	SDI2	RPINR22	SDI2R[4:0]
SPI 2 Clock Input	SCK2IN	RPINR22	SCK2R[4:0]
SPI 2 Slave Select Input	SS2IN	RPINR23	SS2R[4:0]

Note 1: Unless otherwise noted, all inputs use the Schmitt input buffers

Refer to PIC24FJ64GA004 Data Sheet Section 9.4, page 100



Lab 1

Peripheral Output Function Numbers

TABLE 9-2: SELECTABLE OUTPUT SOURCES (MAPS FUNCTION TO OUTPUT)

Function	Output Function Number ⁽¹⁾	Output Name
NULL ⁽²⁾	0	NULL
C1OUT	1	Comparator 1 Output
C2OUT	2	Comparator 2 Output
U1TX	3	UART 1 Transmit
U1RTS	4	UART 1 Ready To Send
U2TX	5	UART 2 Transmit
U2RTS	6	UART 2 Ready To Send
SDO1	7	SPI 1 Data Output
SCK1OUT	8	SPI 1 Clock Output
SS1OUT	9	SPI 1 Slave Select Output
SDO2	10	SPI 2 Data Output
SCK2OUT	11	SPI 2 Clock Output
SS2OUT	12	SPI 2 Slave Select Output
OC1	18	Output Compare 1
OC2	19	Output Compare 2
OC3	20	Output Compare 3
OC4	21	Output Compare 4
OC5	22	Output Compare 5

- Note 1:** Value assigned to the RPn[4:0] registers corresponds to the peripheral output function number.
- 2:** The NULL function is assigned to all RPn outputs at device Reset and disables the RPn output function.

Refer to PIC24FJ64GA004 Data Sheet Section 9.4, page 101



Lab 1

Expected Result

- **Typed keys echoed back to HyperTerminal**



Optional Lab Parallel Master Port (PMP)



Opt. Lab Goals

- **To configure the PMP module**
- **To understand the signals required to interface with a LCD**
- **To display a string on LCD**



Opt. Lab To Do

- **In LCD.c**
 - **STEP 1**
 - **Configure PMCON: PMP on, address/data not multiplexed, PMBE active high,**
 - **PMWR I/O, PMRD I/O, 8-bit data, PMENB and PMRD/~PMWR active high.**

 - **STEP 2**
 - **Configure PMMODE: Interrupts, stall, buffers, inc/dec off, 8 bit mode,**
 - **combined read/write with byte enable signals, and max the 3 wait delays.**

 - **STEP 3**
 - **Configure PMAEN: Enable A0 function to control RS and disable all other PMP address pins.**

 - **STEP 4**
 - **Configure PMADDR: A0 selects type of instruction, either command or data.**
 - **This is a command so A0 should be low.**

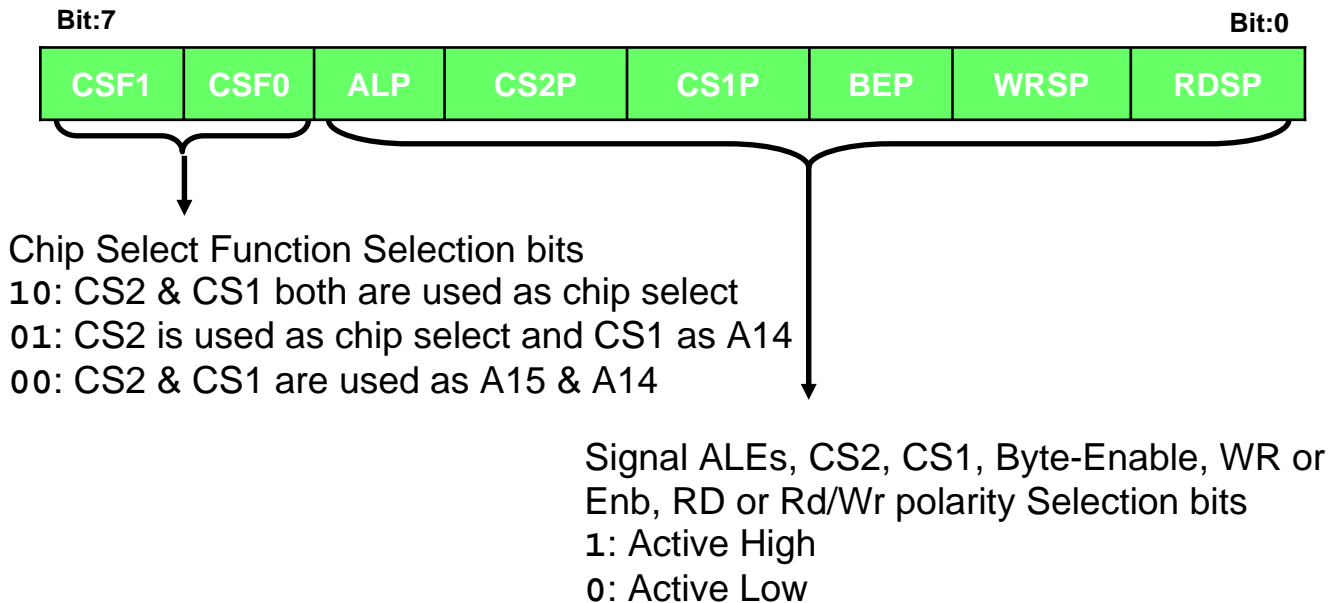
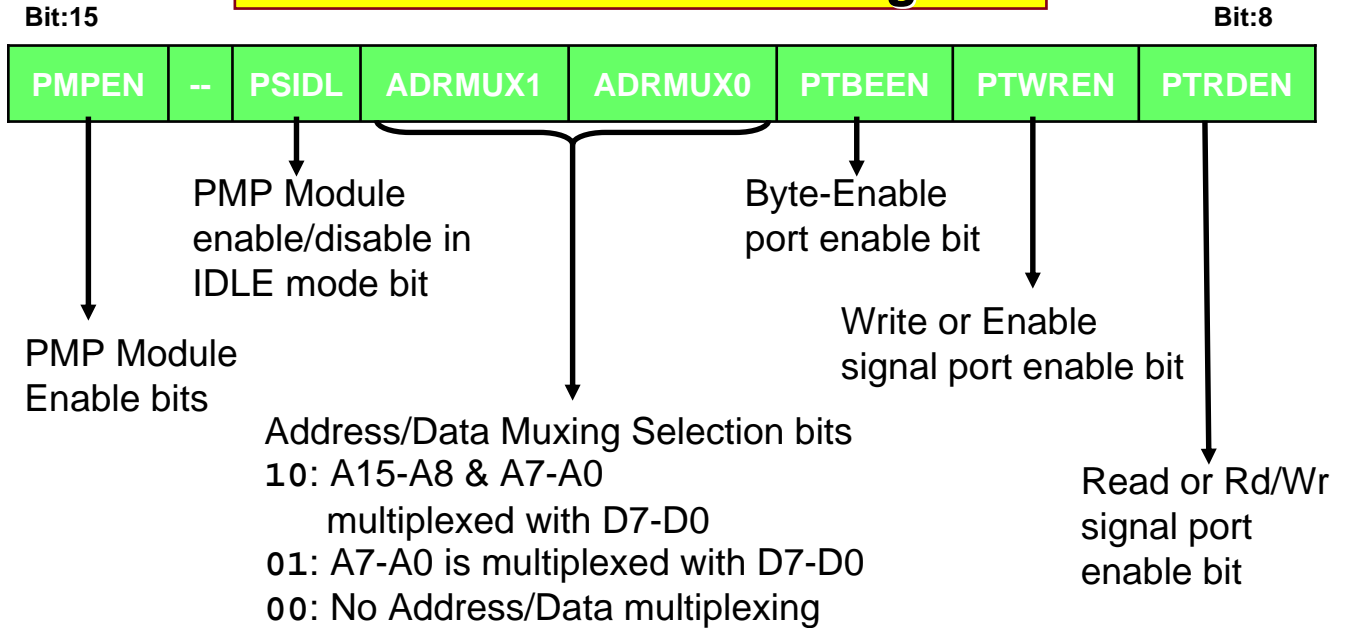


Opt. Lab To Do

- **In main.c**
 - STEP 5:
 - **Change text to your name**
- **Extra Credit for advanced users**
 - Modify code provided to display the rotating banners in my_banner
 - Useful variables and functions: pban, num_banners, wait_time, mLCDPutChar(char), and mLCDClear()
 - Refer to comments in code for explanation of functions

Opt. Lab PMP Registers

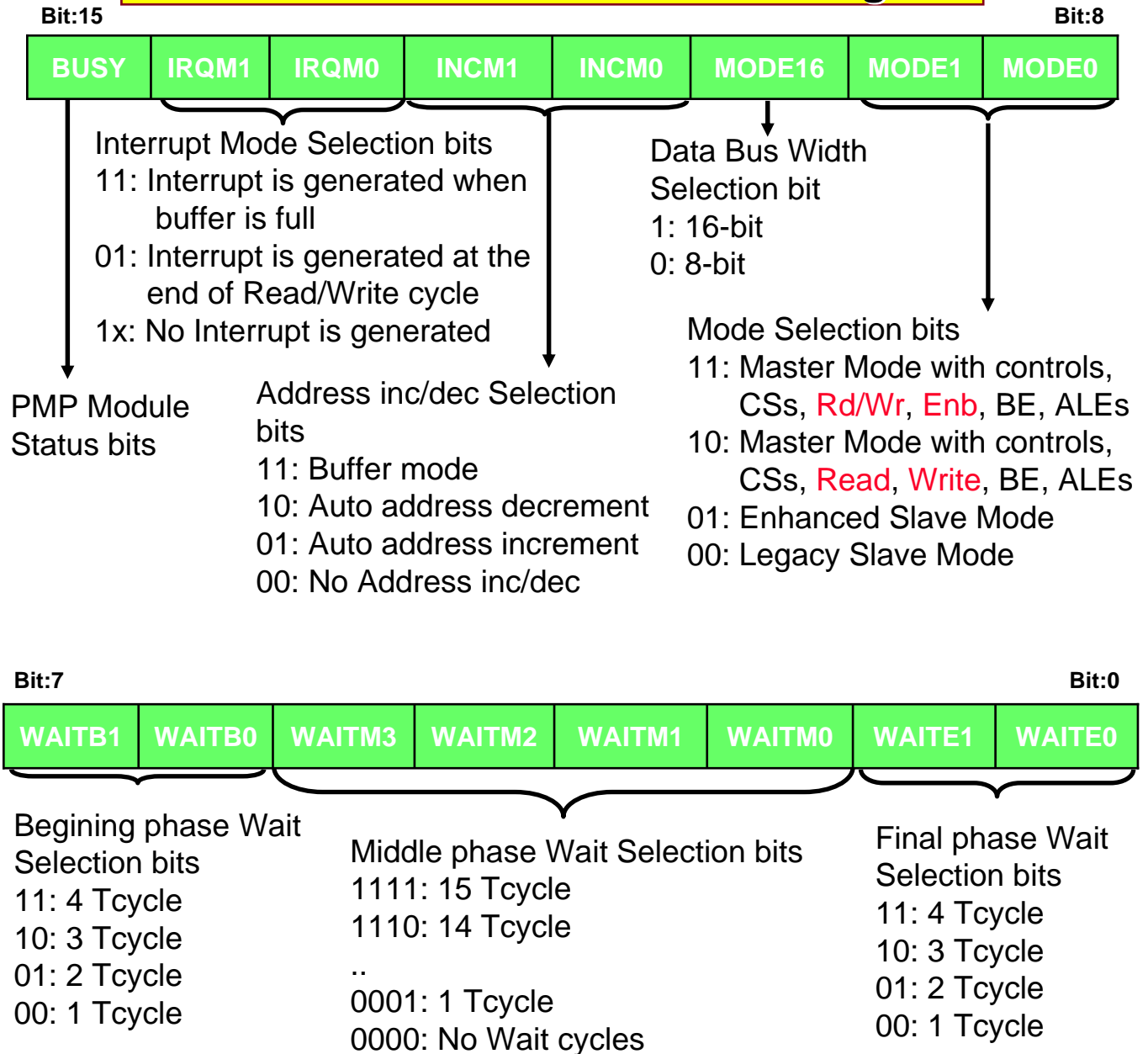
PMCON: PMP Control Register



Refer to PIC24FJ128GA010 Data Sheet Section 17, page 162

Opt. Lab PMP Registers

PMMODE: PMP Mode Selection Register

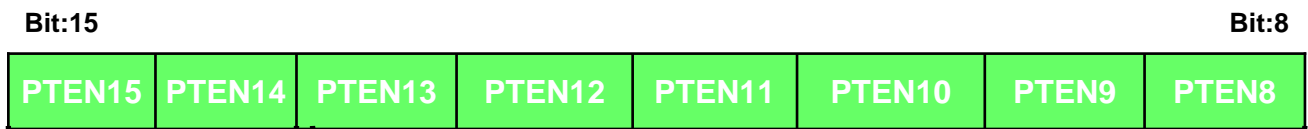


Refer to PIC24FJ128GA010 Data Sheet Section 17, page 164



Opt. Lab PMP Registers

PMAEN: ADDRESS ENABLE REGISTER



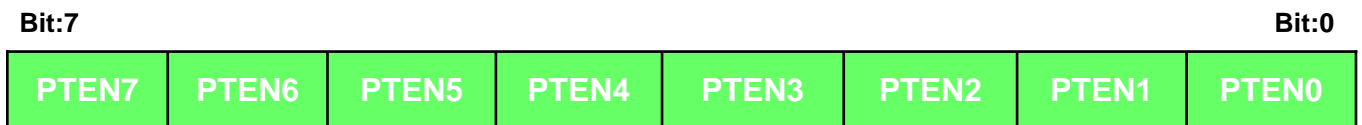
PMP Address Port Enable

1: PMA<13:8> functions as PMP address lines
0: PMA<13:8> functions as port I/O

PMP Chip Select Strobe Enable

1: PMA15 and PMA14 function as either PMA<15:14> or as PMCS1/2

0: PMA<15:14> functions as port I/O



PMP Address Port Enable

1: PMA<7:2> functions as PMP address lines
0: PMA<7:2> functions as port I/O

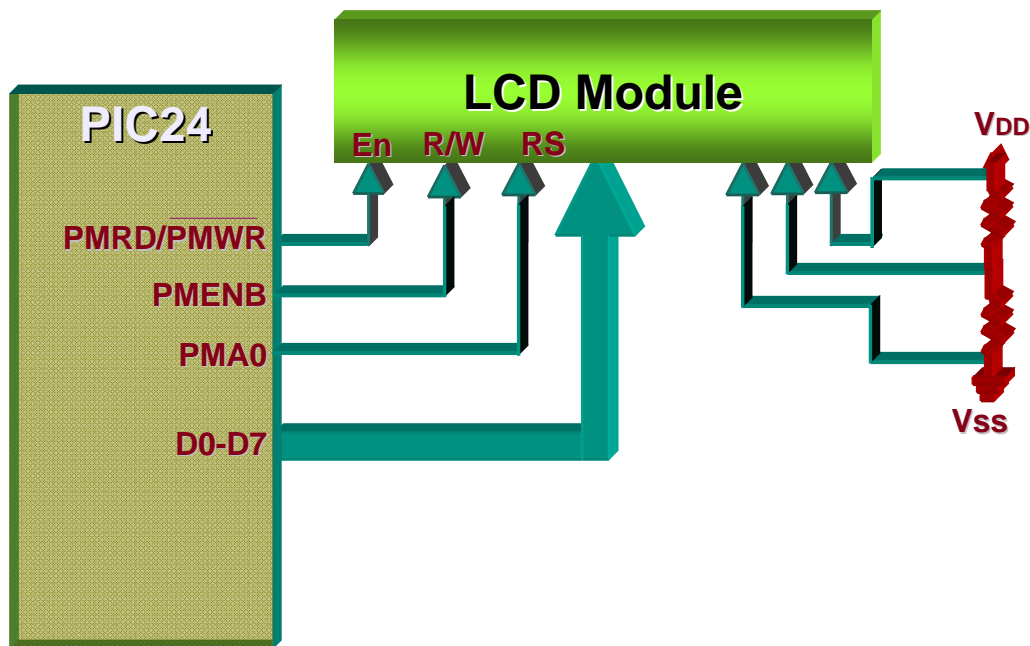
PMALH/PMALL Strobe Enable

1: PMA1 and PMA0 pads function as either PMA<1:0> or PMALH/L
0: PMA1 and PMA0 pads function as port I/O

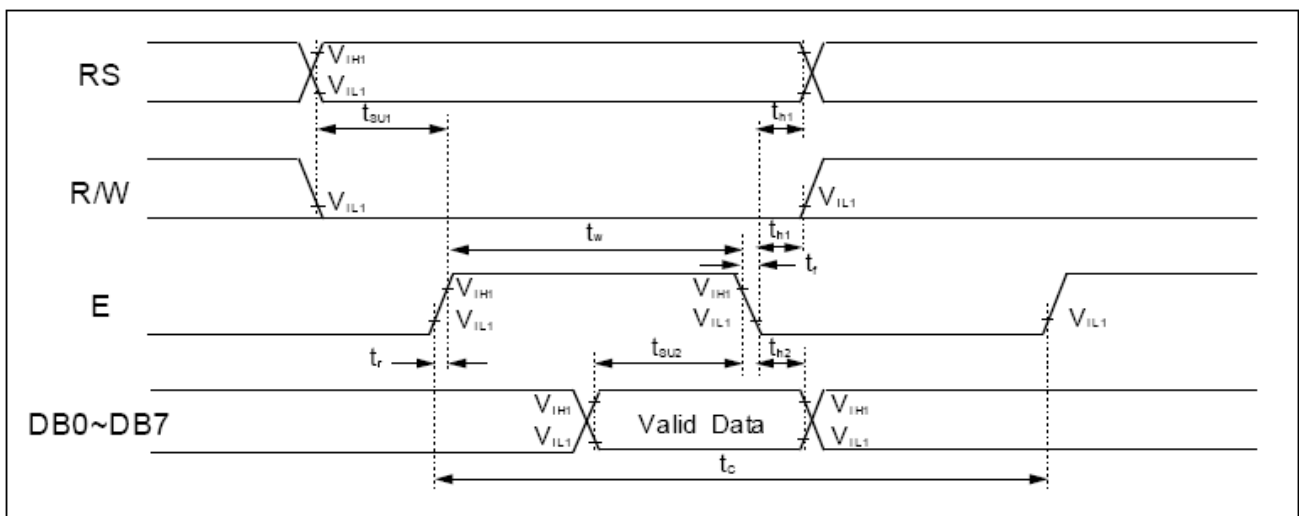
Refer to PIC24FJ128GA010 Data Sheet Section 17, page 165

Opt. Lab LCD Operation

- PMP to LCD Connections



- LCD write timing





Opt. Lab Expected Result

- **Name is displayed on the LCD**
- **For extra credit:
Rotate banner
displayed on LCD
once approximately
every 2 seconds.**



LAB 2

Real Time Clock and Calendar (RTCC)



Lab 2 Goals

- **Configure RTCC**
- **Set RTCC Time and Alarm**



Lab 2 To Do

- **In rtcc.c**
 - **STEP 1:**
 - **Unlock RTCC Registers**
 - Hint: look for mRTCCUnlock macro
 - **STEP 2:**
 - **Configure RCFGCAL, RTCPTR Auto-decrementing pointer**
 - **STEP 3:**
 - **Write Year To RTCVAL**
 - **Write Month & Day To RTCVAL**
 - **Write Weekday & Hour To RTCVAL**
 - **Write Minutes & Seconds To RTCVAL**



Lab 2 To Do

- STEP 4:
 - **Enable RTCC**

- STEP 5:
 - **Lock RTCC Registers**
 - Hint: look for mRTCCLock macro

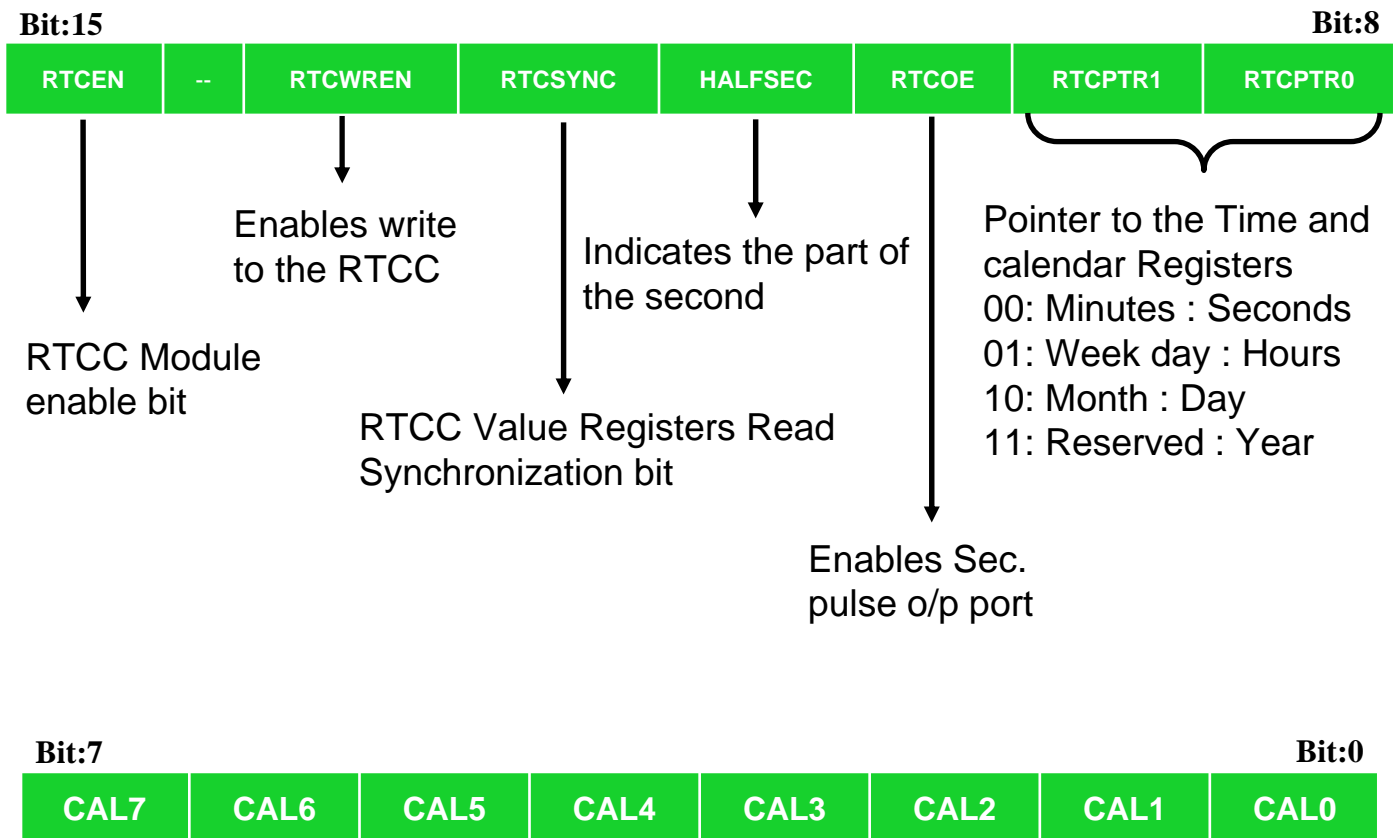
- STEP 6:
 - **In ALCFGRPT, Configure Alarm Frequency Every 10 seconds**
 - **In ALCFGRPT, Configure Alarm To Repeat 3 Times**

- STEP 7:
 - **Enable Alarm**



Lab 2 RTCC Registers

RCFGCAL: RTCC Calibration and Configuration Register

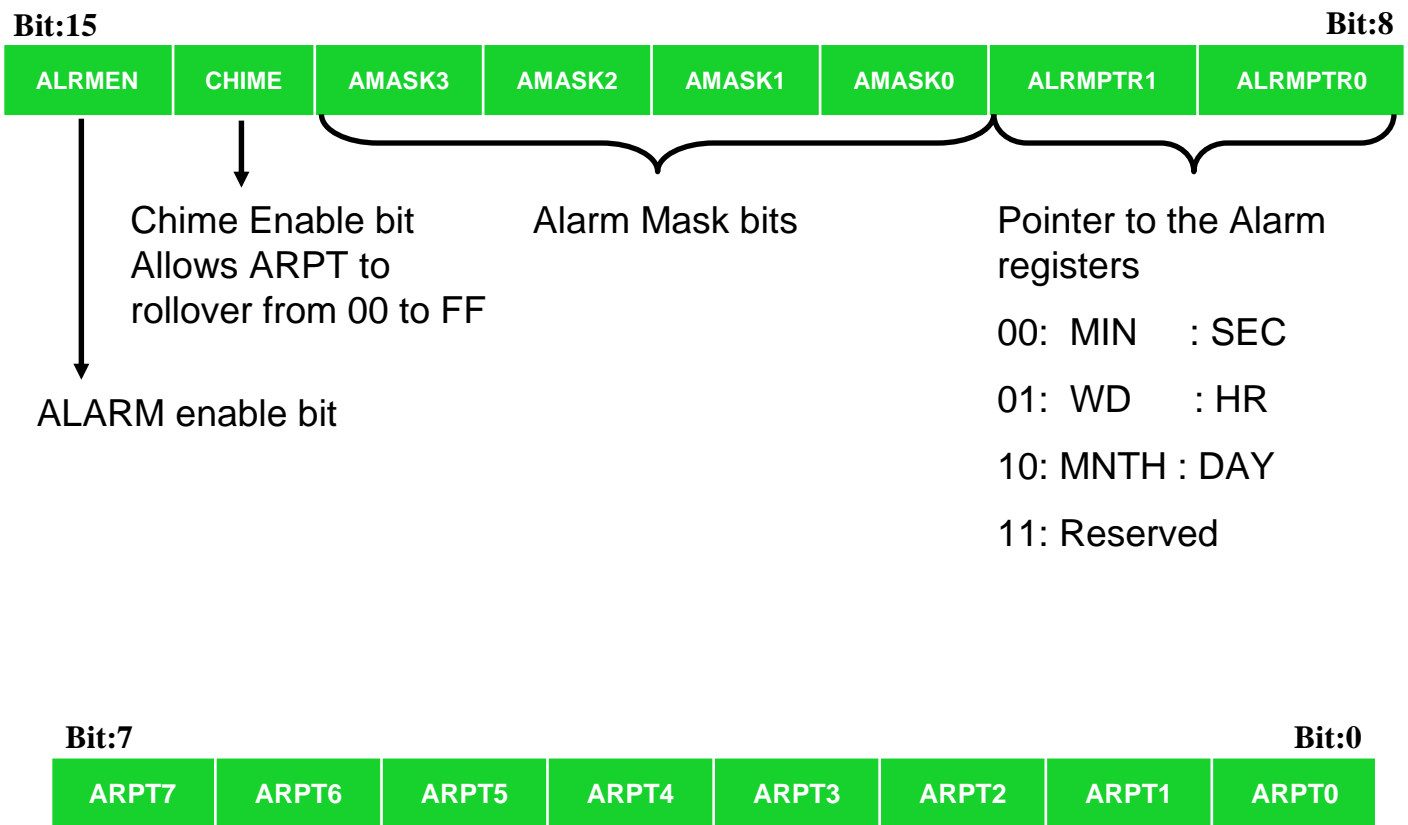


Crystal offset calibration bits (RTCC Drift calibration bits)

Refer to PIC24FJ128GA010 Data Sheet Section 18, page 173

Lab 2 RTCC Registers

ALCFGRPT: RTCC Alarm Configuration register



Alarm Repeat Counter Value bits (Repeat count = 2^n)

Refer to PIC24FJ128GA010 Data Sheet Section 18, page 175



Lab 2 RTCC Registers

RTCVAL: RTCC Value Register

- **RTCPTR<1:0>** auto decrements when **RTCVAL<15:8>** is read or written until it reaches '00'

RTCPTR<1:0>	RTCVAL<15:8>	RTCVAL<7:0>
11	---	YEAR
10	MONTH	DAY
01	WEEKDAY	HOURS
00	MINUTES	SECONDS

ALRMVAL: RTCC Alarm Value Register

- **ALRMPTR<1:0>** auto decrements when **ALRMVAL<15:8>** is read or written until it reaches '00'

ALRMPTR<1:0>	ALRMVAL<15:8>	ALRMVAL<7:0>
11	---	---
10	ALRMMNTH	ALRMDAY
01	ALRMWD	ALRMHR
00	ALRMMIN	ALRMSEC

Refer to PIC24FJ128GA010 Data Sheet Section 18, page 176-179



Lab 2 Expected Results

- **The time and date will be displayed on the LCD.**
- **An LED should blink once every 10 seconds for 3 blinks when the RTCC seconds value equals Alarm seconds value (5).**



LAB 3

Cyclic Redundancy Check Generator (CRC)



Lab 3 Goals

- **Understand Configuration of CRC module**
- **Understand CRC operation**
- **Find the CRC Result of a data transmission**



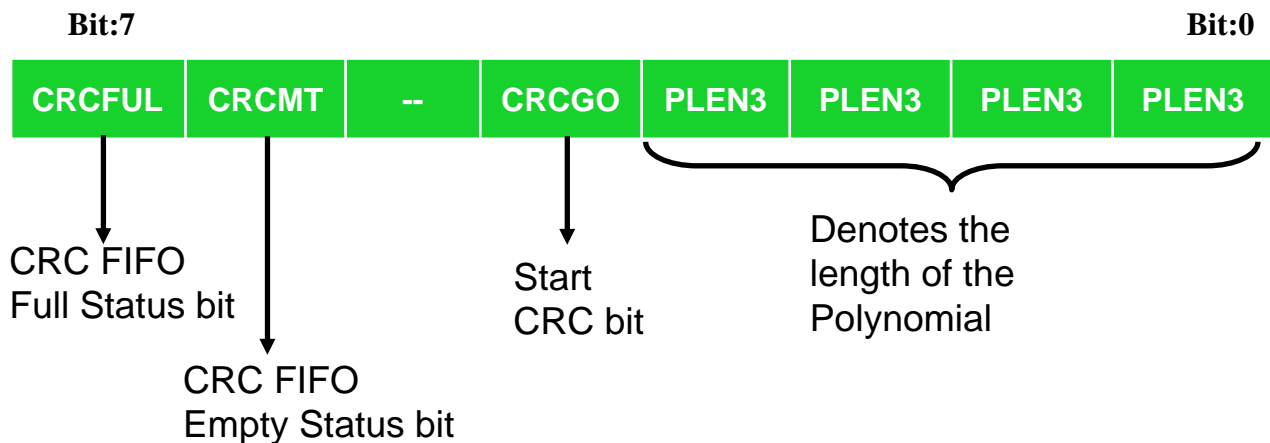
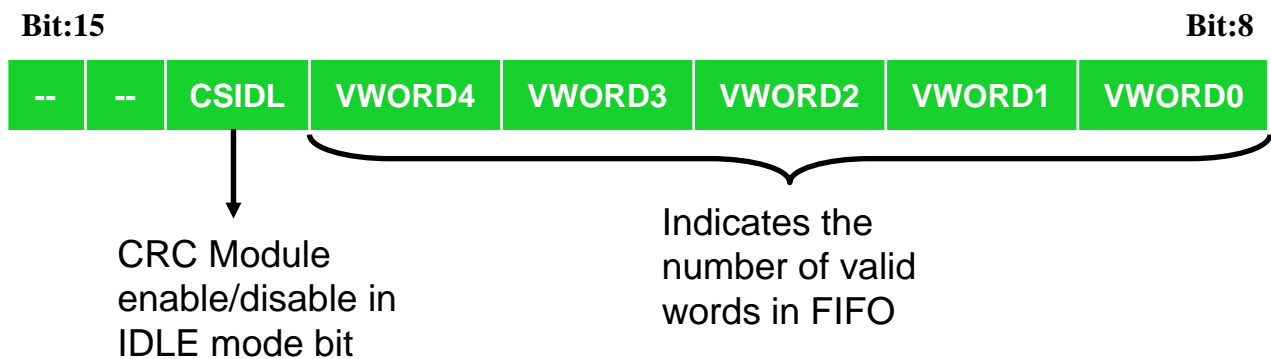
Lab 3 To Do

- **In main.c:**
 - **STEP 1:**
 - **In CRCCON, Configure The Polynomial Length (PLEN) for the Polynomial:**
 - $x^{16} + x^{15} + x^2 + 1$
 - **STEP 2:**
 - **In CRCXOR, Configure for the Polynomial $x^{16} + x^{15} + x^2 + 1$**
 - **STEP 3:**
 - **Clear CRCWDAT**
 - **STEP 4:**
 - **In CRCCON, Enable The CRC Generator**



Lab 3 CRC Registers

CRCCON: CRC Control register



Refer to PIC24FJ128GA010 Data Sheet Section 19, page 181



Lab 3 To Do

- Step 5 – Open Needed Files & Programs:
 - **Open HyperTerminal by, Lab5.ht, in the directory**
 - **Open CRC spreadsheet, CRCCalc.xls, in the directory**
 - If there are errors, go to Tools->Add-Ins and check “Analysis Toolpack” and “Analysis Toolpack – VBA”
 - **Open Lab5.txt in the directory**

- Step 6 – Calculate A Known Good CRC Value:
 - **Enter 10 words of data in the CRC spreadsheet in blue cells A4 to A13**
 - **Copy the green cell C13 Into The Lab5.txt file, This is your data message and CRC checksum**



Lab 3 To Do

- Step 7 – Transmit Data message + CRC value
 - **Compile and run the code**
 - **Send the data with HyperTerminal using copy then right click -> “Paste to host” or Transfer -> “Send text file...”**
 - Ctrl+V will not work correctly
 - **Check the LCD display and verify that “CRC Verified OK” is displayed**

- Step 8 – Corrupt the data message
 - **Change any value in the text file to corrupt the message**
 - **Send the data with HyperTerminal using copy then right click -> “Paste to host” or Transfer -> “Send text file...”**
 - Ctrl+V will not work correctly
 - **Check the LCD display and verify that “CRC Verified NOK” is displayed. This indicates that the CRC verification failed.**



Lab 3 Expected Results

- **With a correct data transmission the LCD displays “CRC verified OK”**
- **With a corrupted data transmission the LCD displays “CRC verified NOK”**
- **Try Both!**



Lab 4

Direct Memory Access (DMA)



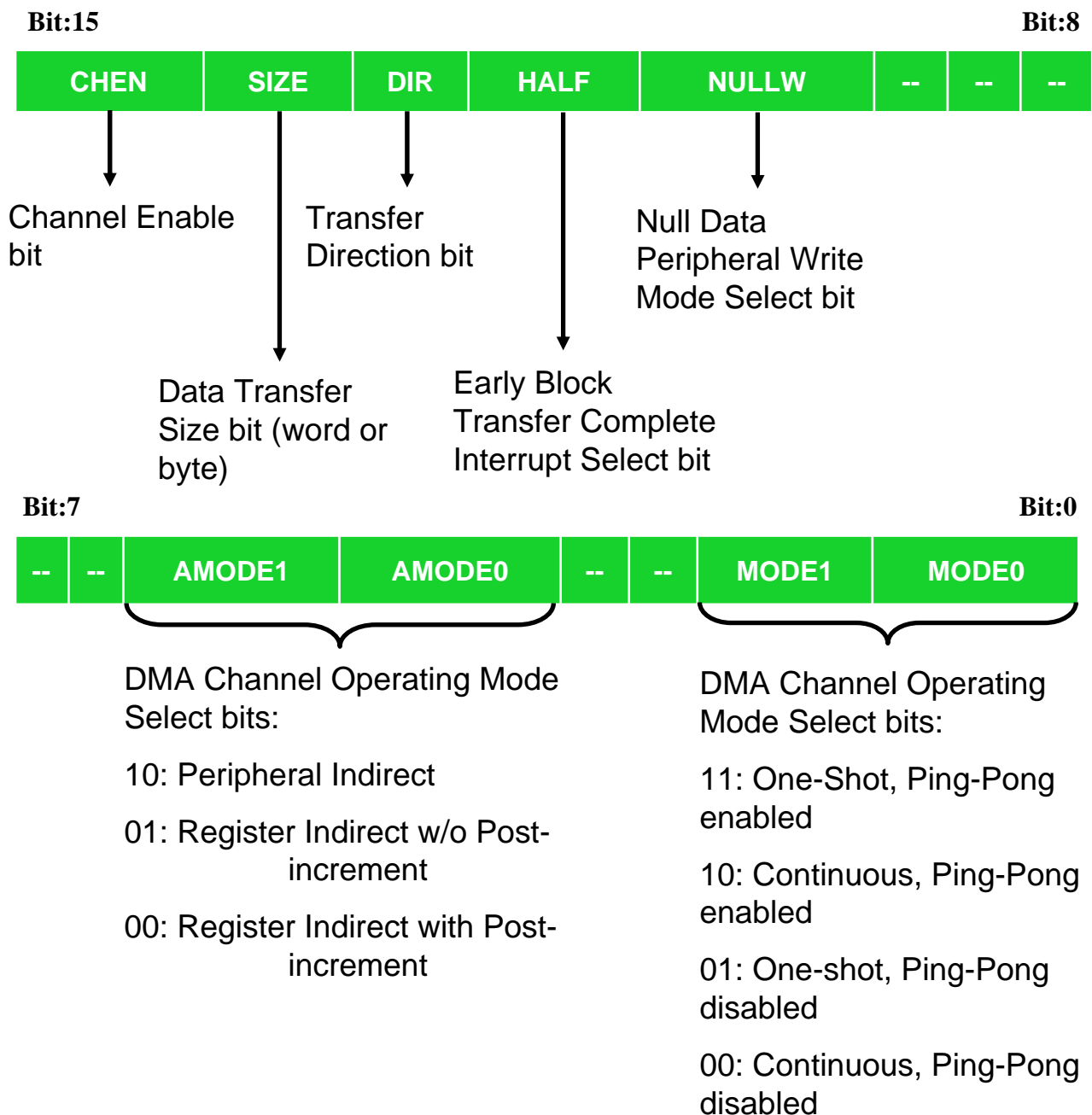
Lab 4 Goals

- **Implement UART loop back utilizing DMA for receiving and transmitting**
- **Receive and buffer 8 characters one at a time**
- **Transmit all 8 characters back**



Lab 4 DMA Registers

DMAxCON: DMA control register





Lab 4 To Do

● Step 1

- Configure UART for DMA transfers

```
// Interrupt after one Tx character is transmitted
U2STAbits.UTXISEL0 = 0;
U2STAbits.UTXISEL1 = 0;

// Interrupt after one RX character is received
U2STAbits.URXISEL = 0;
```

```
IEC4bits.U2EIE = 0;           // Enable UART2 Error Interrupt

void __attribute__((__interrupt__)) _U2ErrInterrupt(void)
{
    /* Process UART 2 Error Condition here */
    IFS4bits.U2EIF = 0; // Clear the UART2 Error Interrupt Flag
}
```

● Step 2

- Enable UART Rx and Tx

```
U2MODEbits.UARTEN = 1;       // Enable UART
U2STAbits.UTXEN   = 1;       // Enable UART Tx
```



Lab 4 To Do

● Step 3

- Associate DMA Channels
 - Channel 0 with UART Tx
 - Channel 1 with UART Rx

Desired Peripheral to DMA Association	DMAxREQ Register, Bits IRQSEL<6:0>	DMAxPAD Register Values to Read from Peripheral	DMAxPAD Register Values to Write to Peripheral
INT0 - External Interrupt 0	0000000	-	-
IC1 - Input Compare 1	0000001	0x0140 (IC1BUF)	-
IC2 - Input Capture 2	0000101	0x0144 (IC2BUF)	-
OC1 - Output Compare 1 Data	0000010	-	0x0182 (OC1R)
OC1 - Output Compare 1 Secondary Data	0000010	-	0x0180 (OC1RS)
OC2 - Output Compare 2 Data	0000110	-	0x0188 (OC2R)
OC2 - Output Compare 2 Secondary Data	0000110	-	0x0186 (OC2RS)
TMR2 - Timer 2	0000111	-	-
TMR3 - Timer 3	0001000	-	-
SPI1 - Transfer Done	0001010	0x0248 (SPI1BUF)	0x0248 (SPI1BUF)
SPI2 - Transfer Done	0100001	0x0268 (SPI2BUF)	0x0268 (SPI2BUF)
UART1RX - UART1 Receiver	0001011	0x0226 (U1RXREG)	-
UART1TX - UART1 Transmitter	0001100	-	0x0224 (U1TXREG)
UART2RX - UART2 Receiver	0011110	0x0236 (U2RXREG)	-
UART2TX - UART2 Transmitter	0011111	-	0x0234 (U2TXREG)
ECAN1 - RX Data Ready	0100010	0x0440 (C1RXD)	-
ECAN1 - TX Data Request	1000110	-	0x0442 (C1TXD)
ECAN2 - RX Data Ready	0110111	0x0540 (C2RXD)	-
ECAN2 - TX Data Request	1000111	-	0x0542 (C2TXD)
DCI - CODEC Transfer Done	0111100	0x0290 (RXBUF0)	0x0298 (TXBUF0)
ADC1 - ADC1 convert done	0001101	0x0300 (ADC1BUF0)	-
ADC2 - ADC2 Convert Done	0010101	0x0340 (ADC2BUF0)	-

```

DMA0REQbits.IRQSEL = 0x1F;
DMA0PAD = (volatile unsigned int) &U2TXREG;
DMA1REQbits.IRQSEL = 0x1E;
DMA1PAD = (volatile unsigned int) &U2RXREG;

```



Lab 4 To Do

● Step 4

- Configure DMA Channel 1 to:
 - Transfer data from UART to RAM Continuously
 - Register Indirect with Post-Increment
 - Using two 'ping-pong' buffers
 - 8 transfers per buffer
 - Transfer words

```
DMA1CONbits.AMODE = 0; // Register Indirect with Post-Increment
DMA1CONbits.MODE = 2; // Continuous, Ping-Pong
DMA1CONbits.DIR = 0; // Peripheral-to-RAM direction
DMA1CONbits.SIZE = 0; // Word transfers

DMA1CNT = 7; // 8 DMA Requests
```

● Step 5

- Configure DMA Channel 0 to:
 - Transfer data from RAM to UART
 - One-Shot mode
 - Register Indirect with Post-Increment
 - Using single buffer
 - 8 transfers per buffer
 - Transfer words

```
DMA0CONbits.AMODE = 0; // Register Indirect with Post-Increment
DMA0CONbits.MODE = 1; // One-Shot, Single Buffer
DMA0CONbits.DIR = 1; // RAM-to-Peripheral direction
DMA0CONbits.SIZE = 0; // Word transfers

DMA0CNT = 7; // 8 DMA Requests
```



Lab 4 To Do

● Step 6

- Allocate two buffers for DMA transfers
- Associate one buffer with Channel 0 for one-shot operation
- Associate two buffers with Channel 1 for 'Ping-Pong' operation

```
unsigned int BufferA[8] __attribute__(space(dma));  
unsigned int BufferB[8] __attribute__(space(dma));  
  
DMA1STA = __builtin_dmaoffset(BufferA);  
DMA1STB = __builtin_dmaoffset(BufferB);  
  
DMA0STA = __builtin_dmaoffset(BufferA);
```




Lab 4 To Do

● Step 7

- Setup DMA interrupt handlers
- Force transmit after 8 words are received

```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
    IFS0bits.DMA0IF = 0;        //Clear the DMA0 Interrupt Flag;
}

void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
    // Keep record of which buffer contains Rx Data
    static unsigned int BufferCount = 0;

    if(BufferCount == 0)
    {
        // Point DMA 0 to data to be transmitted
        DMA0STA = __builtin_dmaoffset(BufferA);
    }
    else
    {
        // Point DMA 0 to data to be transmitted
        DMA0STA = __builtin_dmaoffset(BufferB);
    }

    DMA0CONbits.CHEN = 1;        // Re-enable DMA0 Channel
    DMA0REQbits.FORCE = 1;       // Manual mode: Kick-start the
                                // 1st transfer

    BufferCount ^= 1;
    IFS0bits.DMA1IF = 0;        // Clear the DMA1 Interrupt Flag
}
```



Lab 4 To Do

- **Step 8**

- Enable DMA Interrupts

```
IFS0bits.DMA0IF = 0;           // Clear DMA 0 Interrupt Flag
IEC0bits.DMA0IE = 1;           // Enable DMA 0 interrupt
IFS0bits.DMA1IF = 0;           // Clear DMA 1 interrupt
IEC0bits.DMA1IE = 1;           // Enable DMA 1 interrupt
```

- **Step 9**

- Enable DMA Channel 1 to receive UART data

```
DMA1CONbits.CHEN = 1; // Enable DMA Channel 1
```



Lab 4 To Do

● Step 10

- Compile, download and run code
- Connect to HyperTerminal (9600 Baud, 8-N-1)
- Type characters into HyperTerminal



Lab 4 Expected Results

- **HyperTerminal should display all 8 typed characters when application transmits them back**