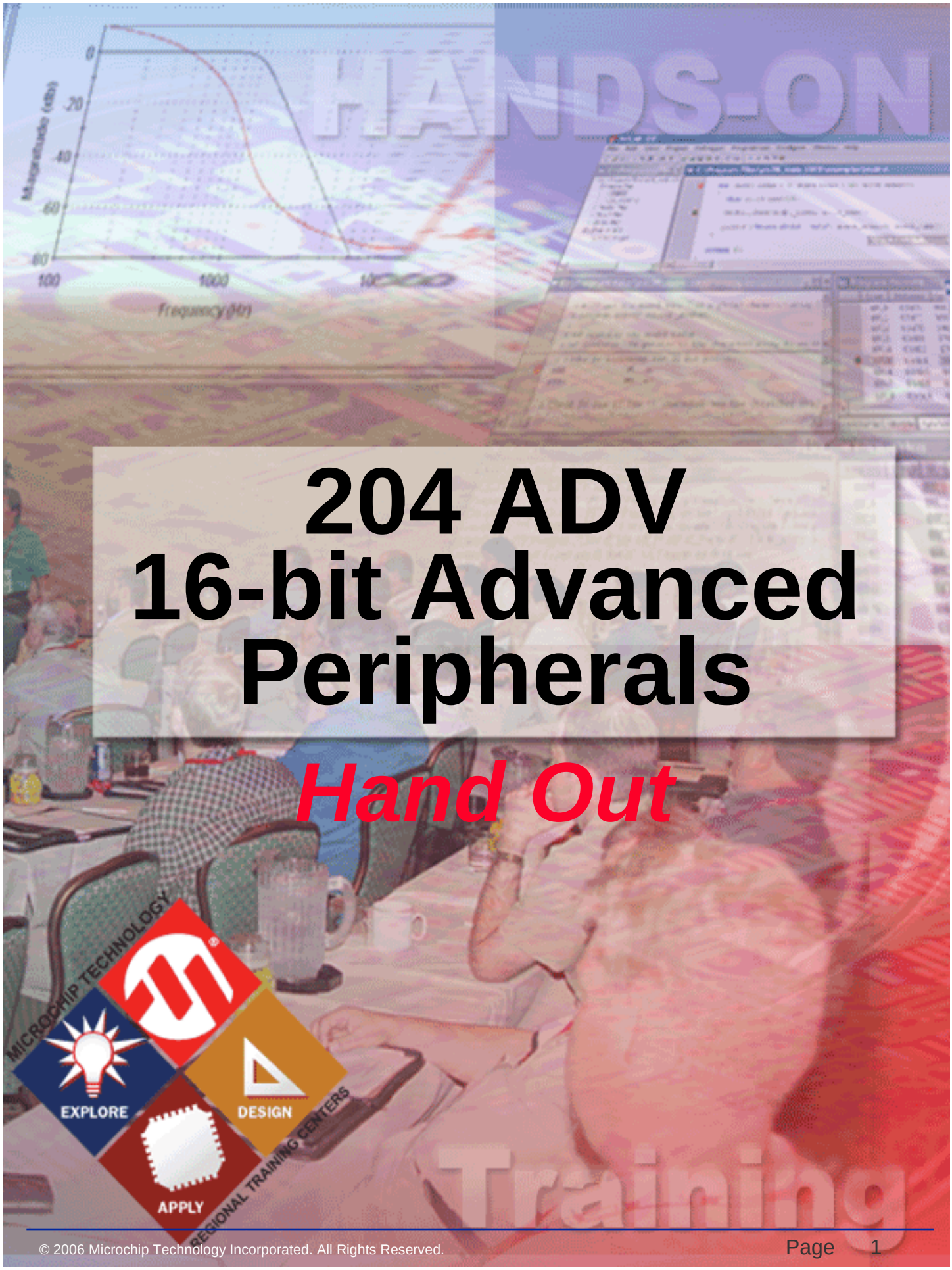
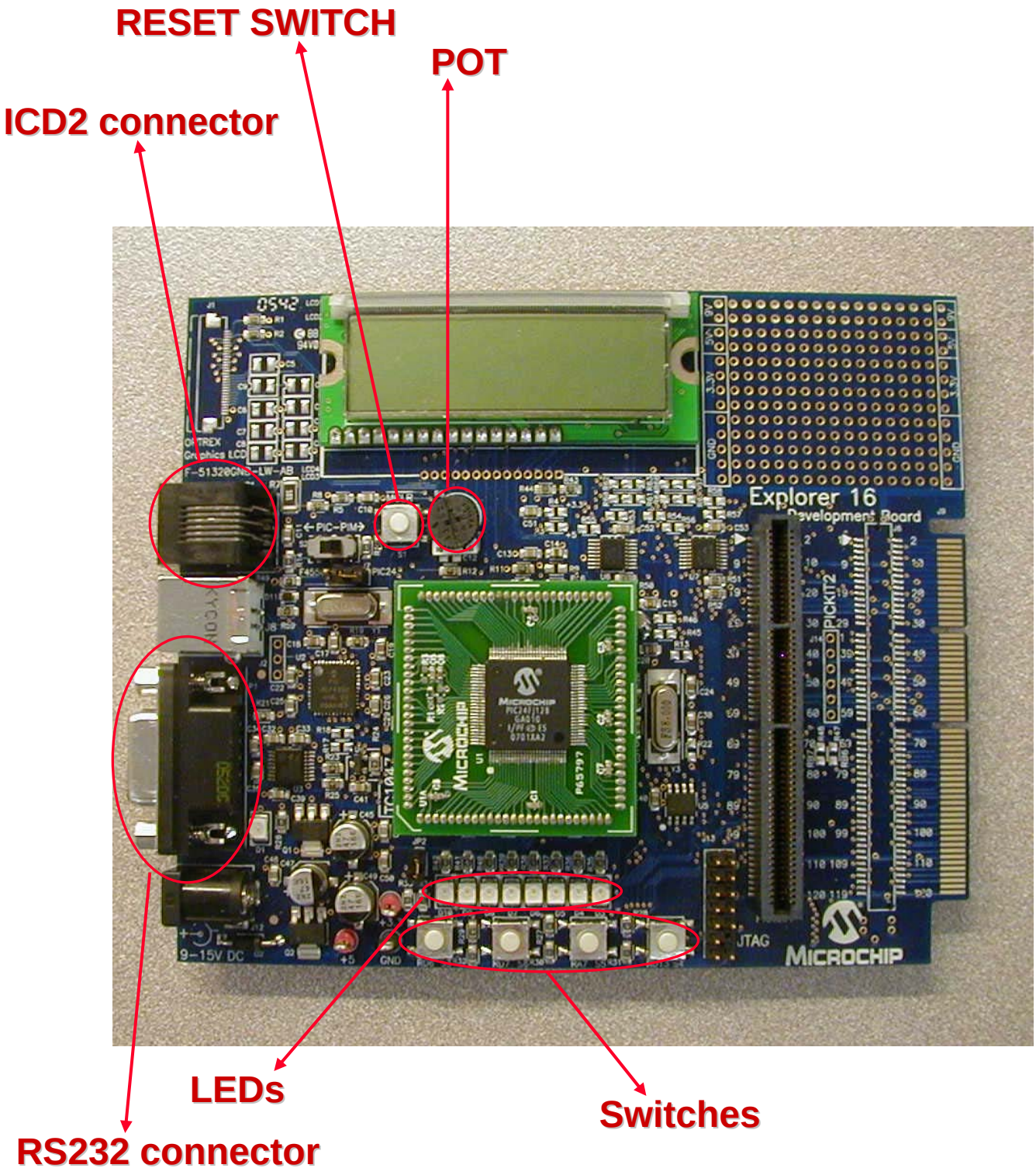


204 ADV 16-bit Advanced Peripherals

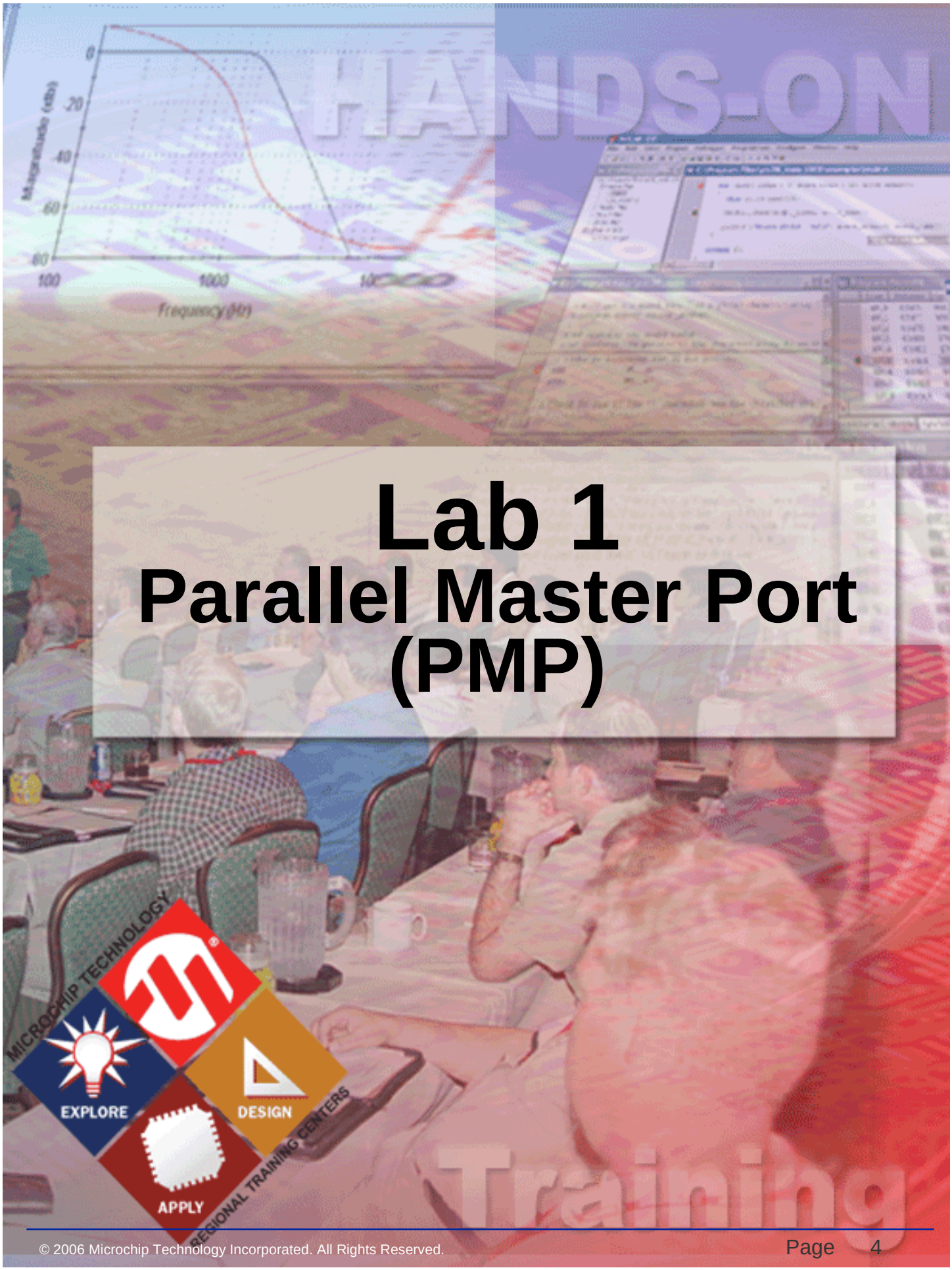
Hand Out

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
Parallel Master Port (PMP)

Lab 1 Goals

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

Lab 1 To Do

- **In LCD.c**
  - STEP 1
    - **Configure PMPCON: PMP on, address/data not multiplexed, PMPBE active high,**
    - **PMPWR I/O, PMPRD I/O, 8-bit data, PMPENB and PMPRD/~PMPWR active high.**
  - STEP 2
    - **Configure PMPMODE: 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 PMPEN: Enable A0 function to control RS and disable all other PMP address pins.**
  - STEP 4
    - **Configure PMPADDR: A0 selects type of instruction, either command or data.**
    - **This is a command so A0 should be low.**

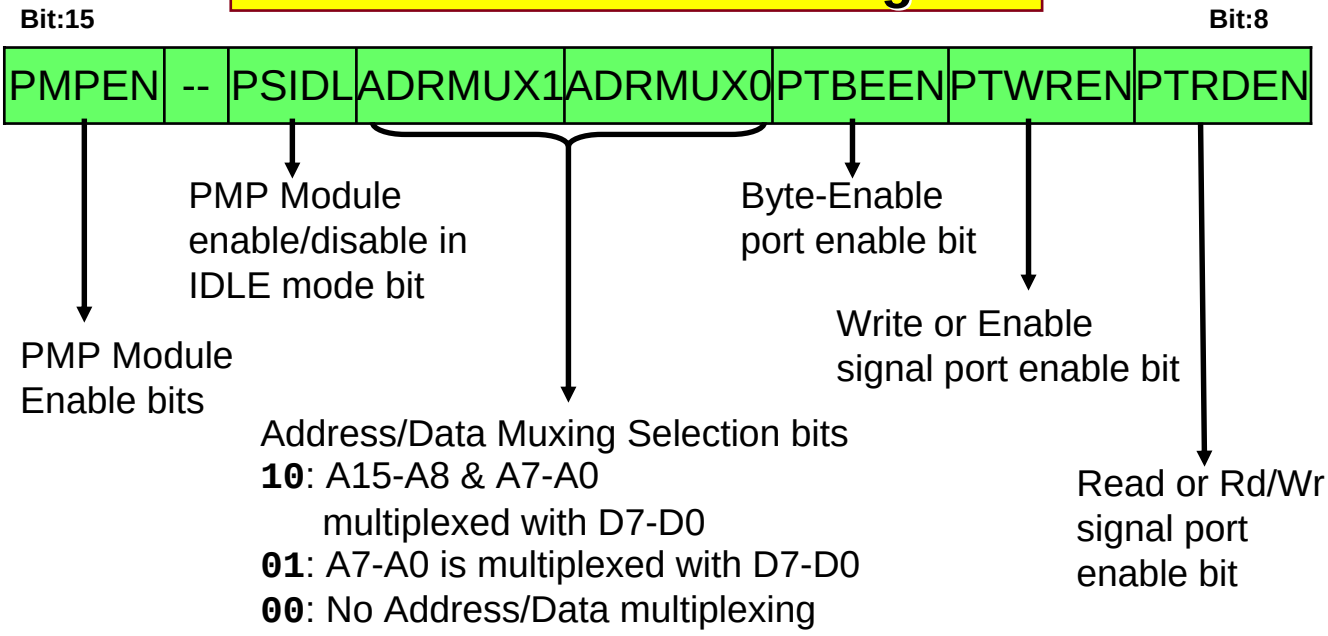
Lab 1 To Do

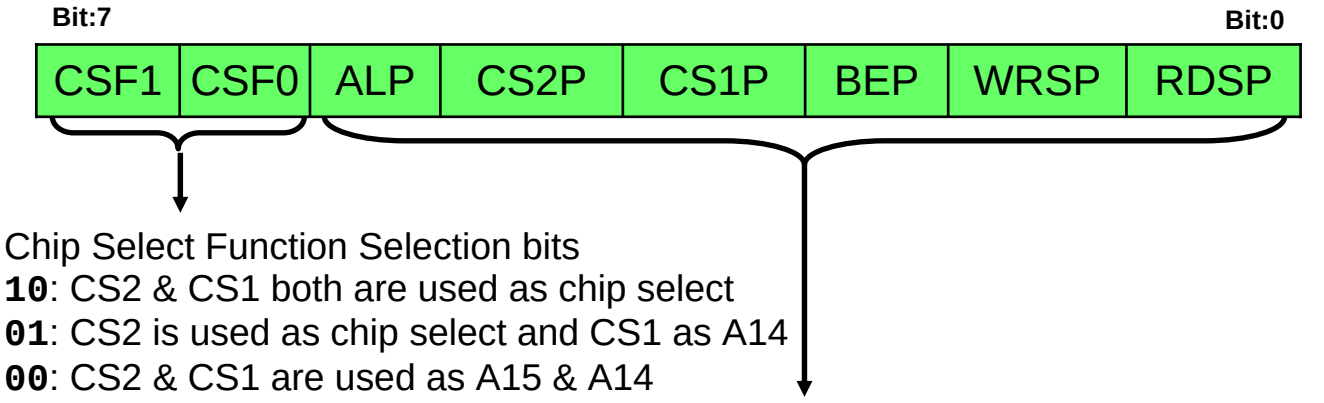
- **In main.c**
  - STEP 5:
    - **Change text to your name**

- **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

Lab 1
PMP Registers

PMCON: PMP Control Register

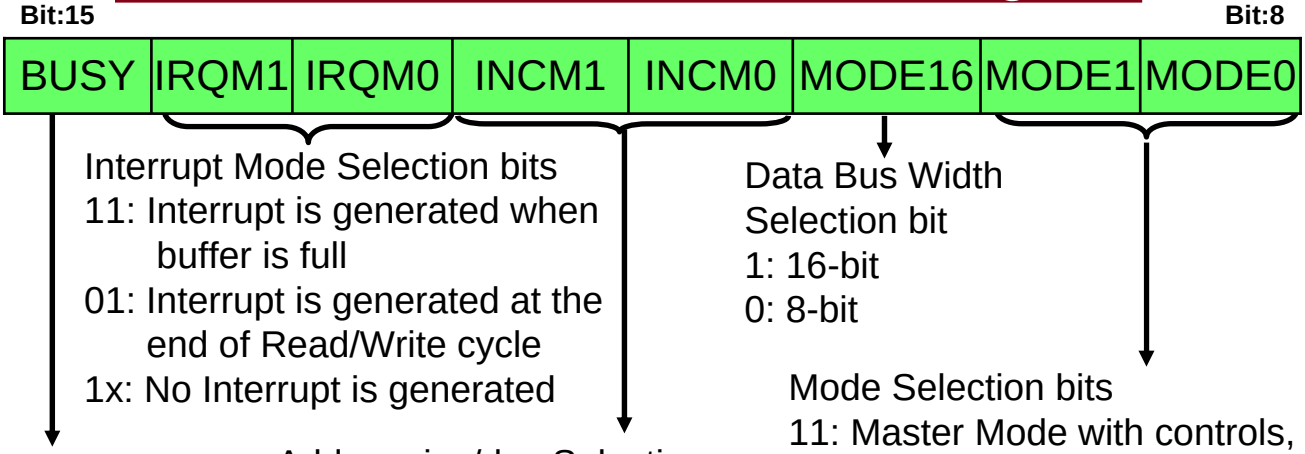




Refer to PIC24FJ128GA010 Data Sheet, page 140

Lab 1
PMP Registers

PMMODE: PMP Mode Selection Register



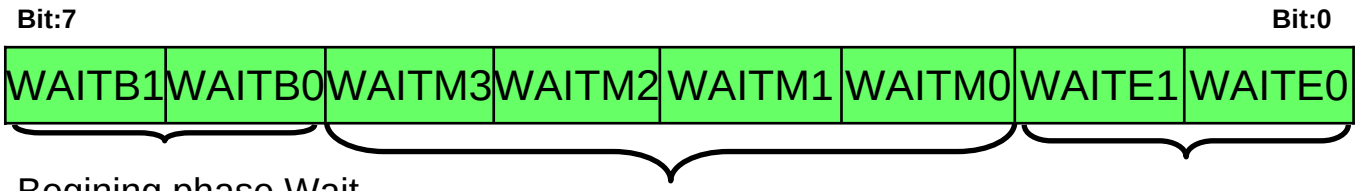
Interrupt Mode Selection bits
11: Interrupt is generated when buffer is full
01: Interrupt is generated at the end of Read/Write cycle
1x: No Interrupt is generated

Data Bus Width Selection bit
1: 16-bit
0: 8-bit

PMP Module Status bits

Address inc/dec Selection bits
11: Buffer mode
10: Auto address decrement
01: Auto address increment
00: No Address inc/dec

Mode Selection bits
11: Master Mode with controls, CSs, Rd/Wr, Enb, BE, ALEs
10: Master Mode with controls, CSs, Read, Write, BE, ALEs
01: Enhanced Slave Mode
00: Legacy Slave Mode



Begining phase Wait Selection bits
11: 4 Tcycle
10: 3 Tcycle
01: 2 Tcycle
00: 1 Tcycle

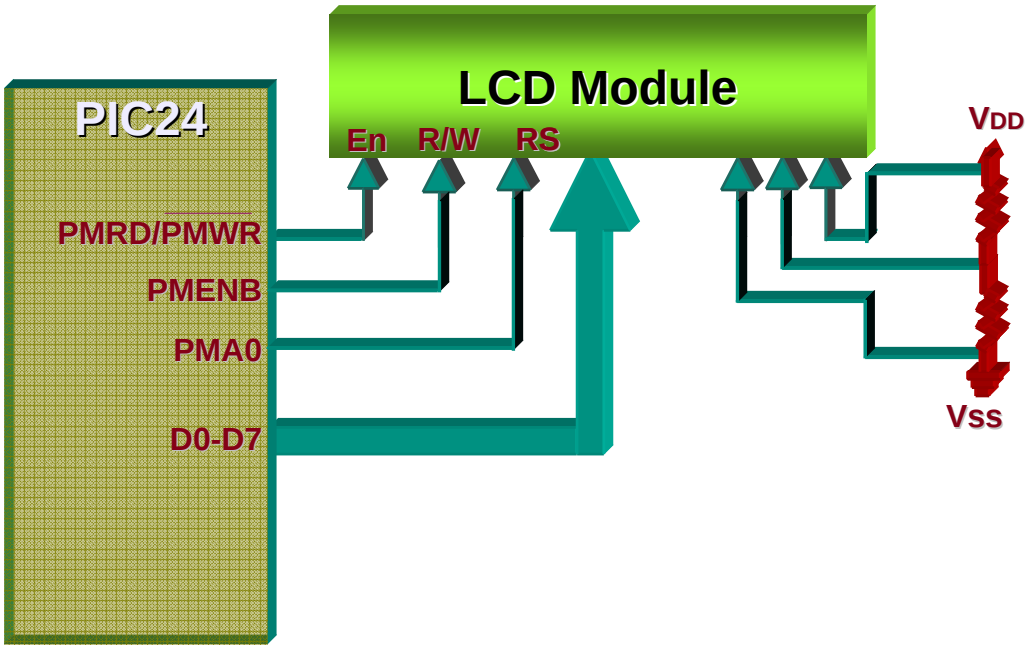
Middle phase Wait Selection bits
1111: 15 Tcycle
1110: 14 Tcycle
..
0001: 1 Tcycle
0000: No Wait cycles

Final phase Wait Selection bits
11: 4 Tcycle
10: 3 Tcycle
01: 2 Tcycle
00: 1 Tcycle

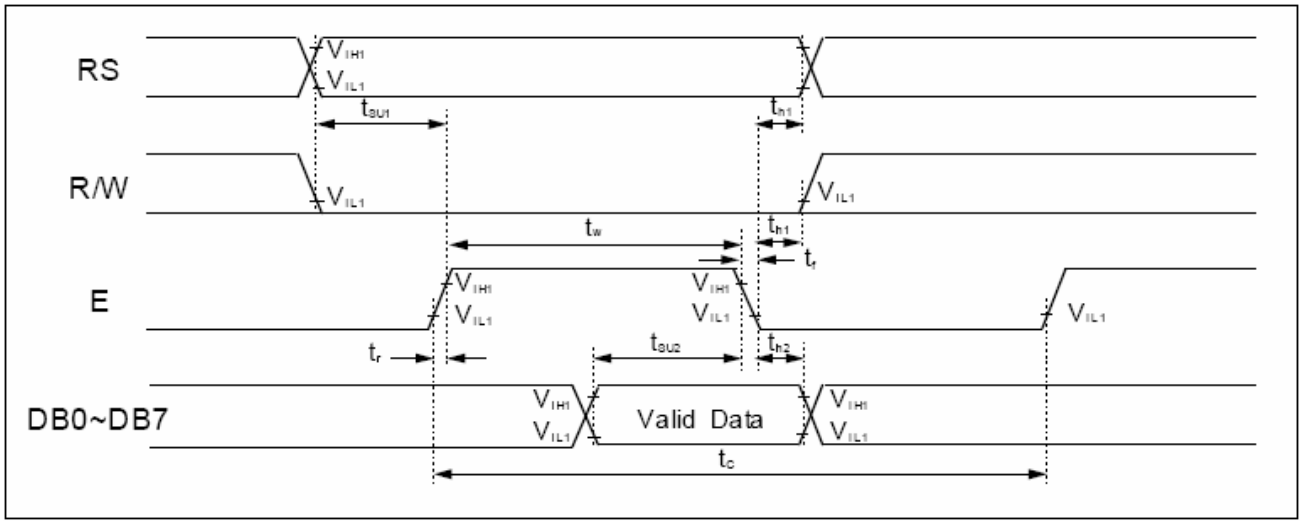
Refer to PIC24FJ128GA010 Data Sheet, page 142

Lab 1
LCD Operation

- PMP to LCD Connections



- LCD write timing



Lab 1
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

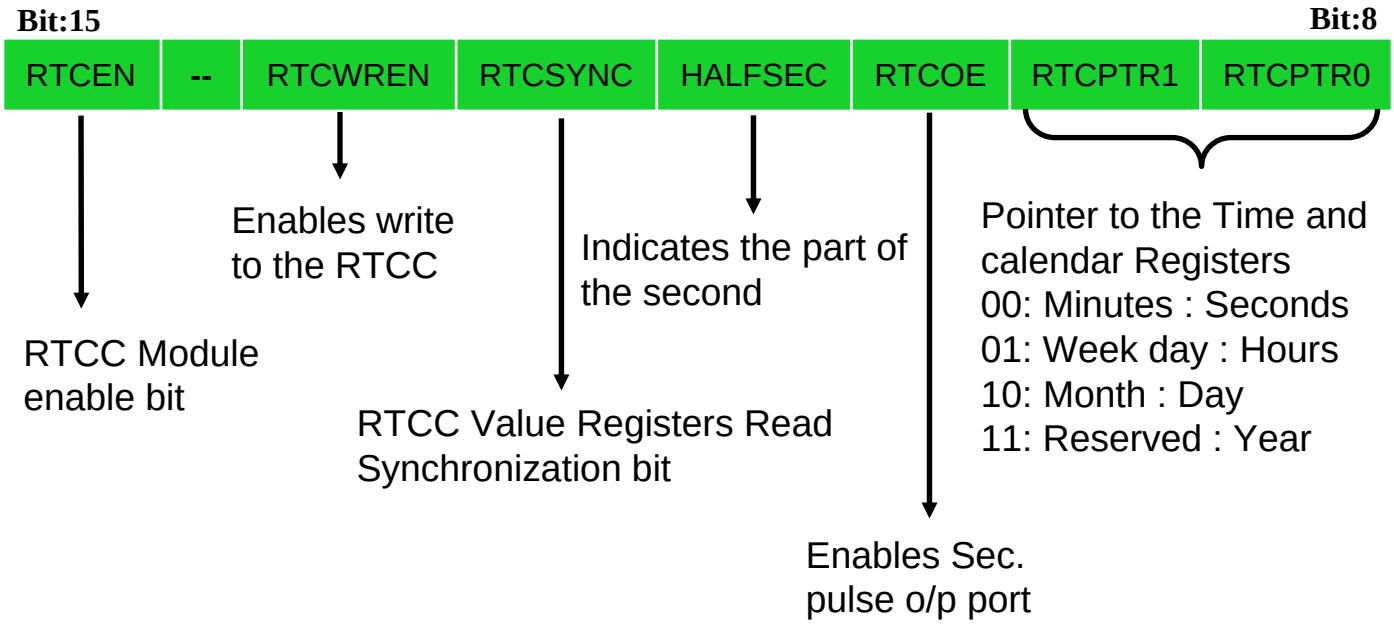
- **In rtcc.c**
  - STEP 1:
    - **Unlock RTCC Registers**
  - STEP 2:
    - **Configure RCFGCAL, RTCPTR Auto-decrementing pointer**
  - STEP 3:
    - **Write Year To RTCVAL**
    - **Write Month & Day To RTCVAL**
    - **Write Weekday & Hour To RTCVAL**
    - **Write Minutes & Seconds To RTCVAL**

Lab 2 To Do

- STEP 4:
  - **Enable RTCC**
- STEP 5:
  - **Lock RTCC Registers**
- STEP 6:
  - **In ALCFGRPT, Configure Alarm Frequency Every 10 seconds**
  - **In ALCFGRPT, Configure Alarm To Repeat 10 Times**
- STEP 7:
  - **Enable Alarm**

Lab 2 RTCC Registers

RCFGCAL: RTCC Calibration and Configuration Register

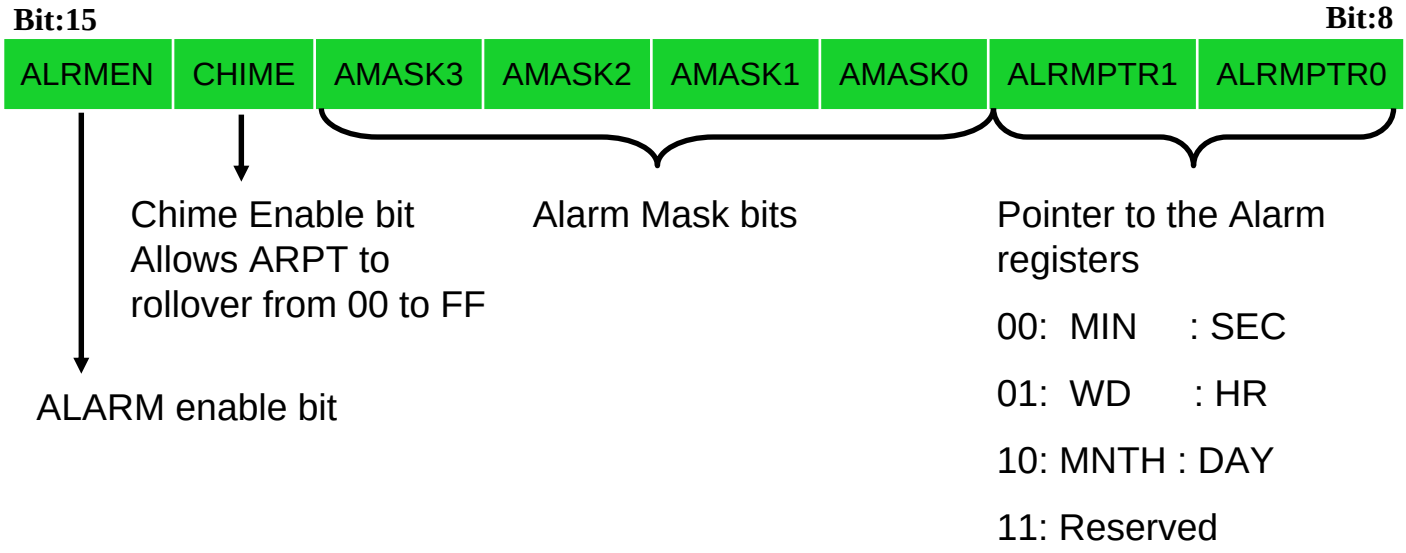


Crystal offset calibration bits (RTCC Drift calibration bits)

Refer to PIC24FJ128GA010 Data Sheet, page 151

Lab 2 RTCC Registers

ALCFGRPT: RTCC Alarm Configuration register



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

Refer to PIC24FJ128GA010 Data Sheet, page 154

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

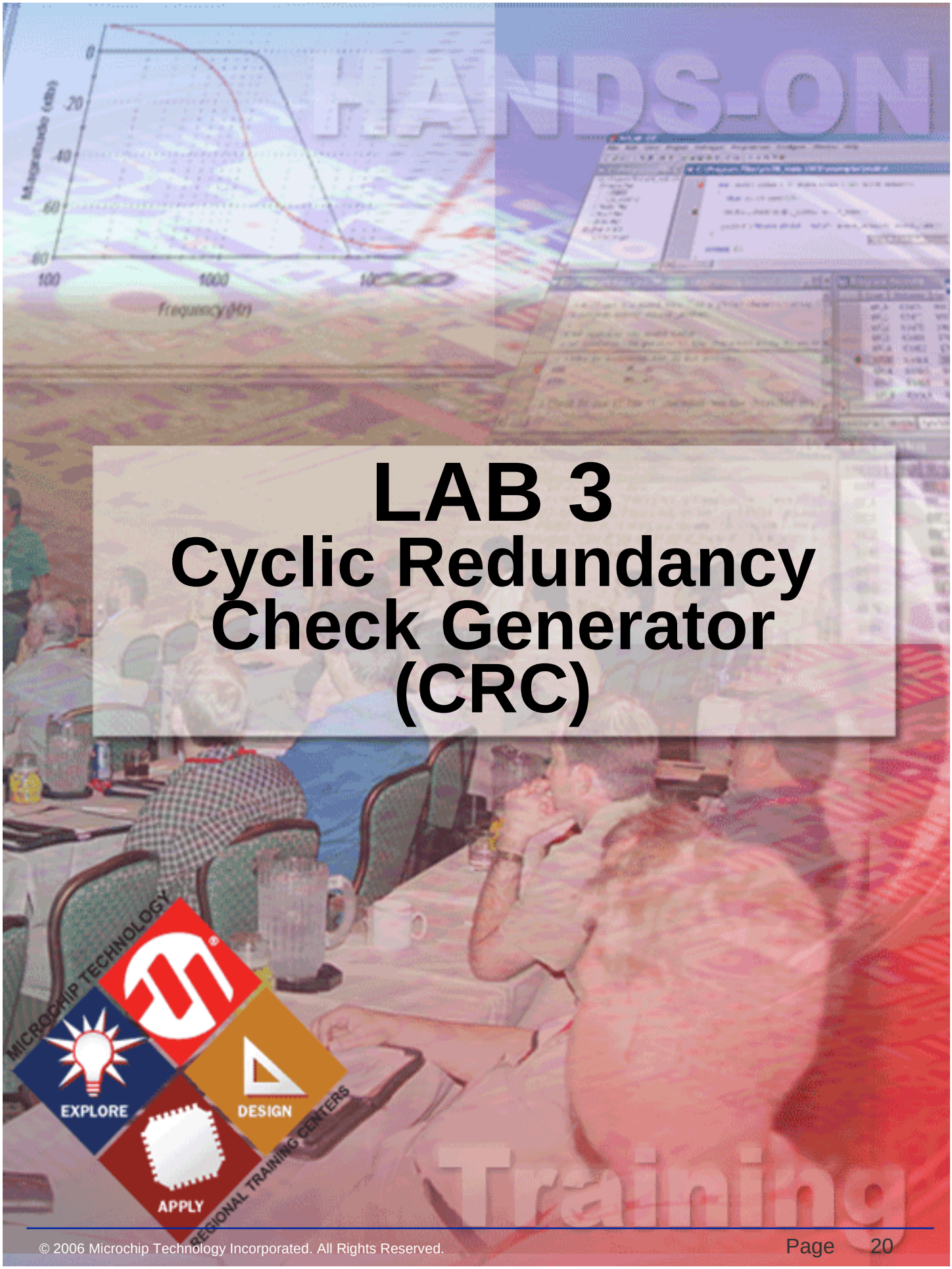
- **ALRMPTR<1:0> auto decrements when ALRMVAL<15:8> is read or written until it reaches '00'**

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

Refer to PIC24FJ128GA010 Data Sheet, page 155-158

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

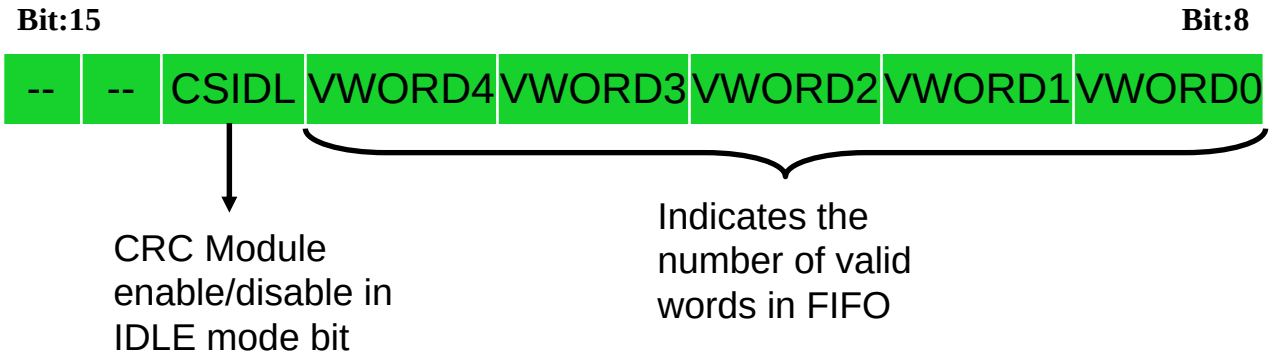
- Understand Configuration of CRC module
- Understand CRC operation
- Find the CRC Result of a data transmission

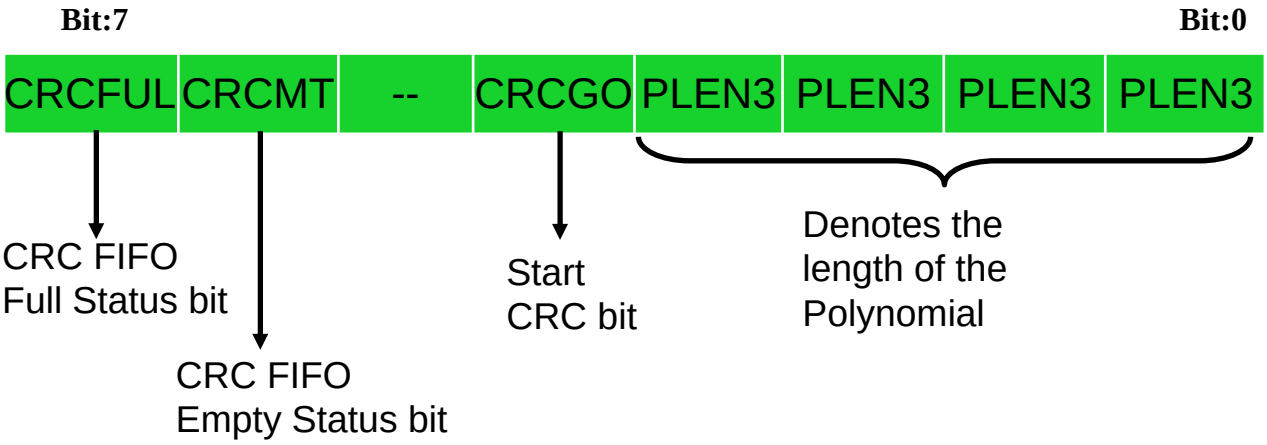
Lab 3 To Do

- **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, page 161

Lab 3 To Do

- Step 5 – Open Needed Files & Programs:
  - **Open HyperTerminal by, 1095_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**

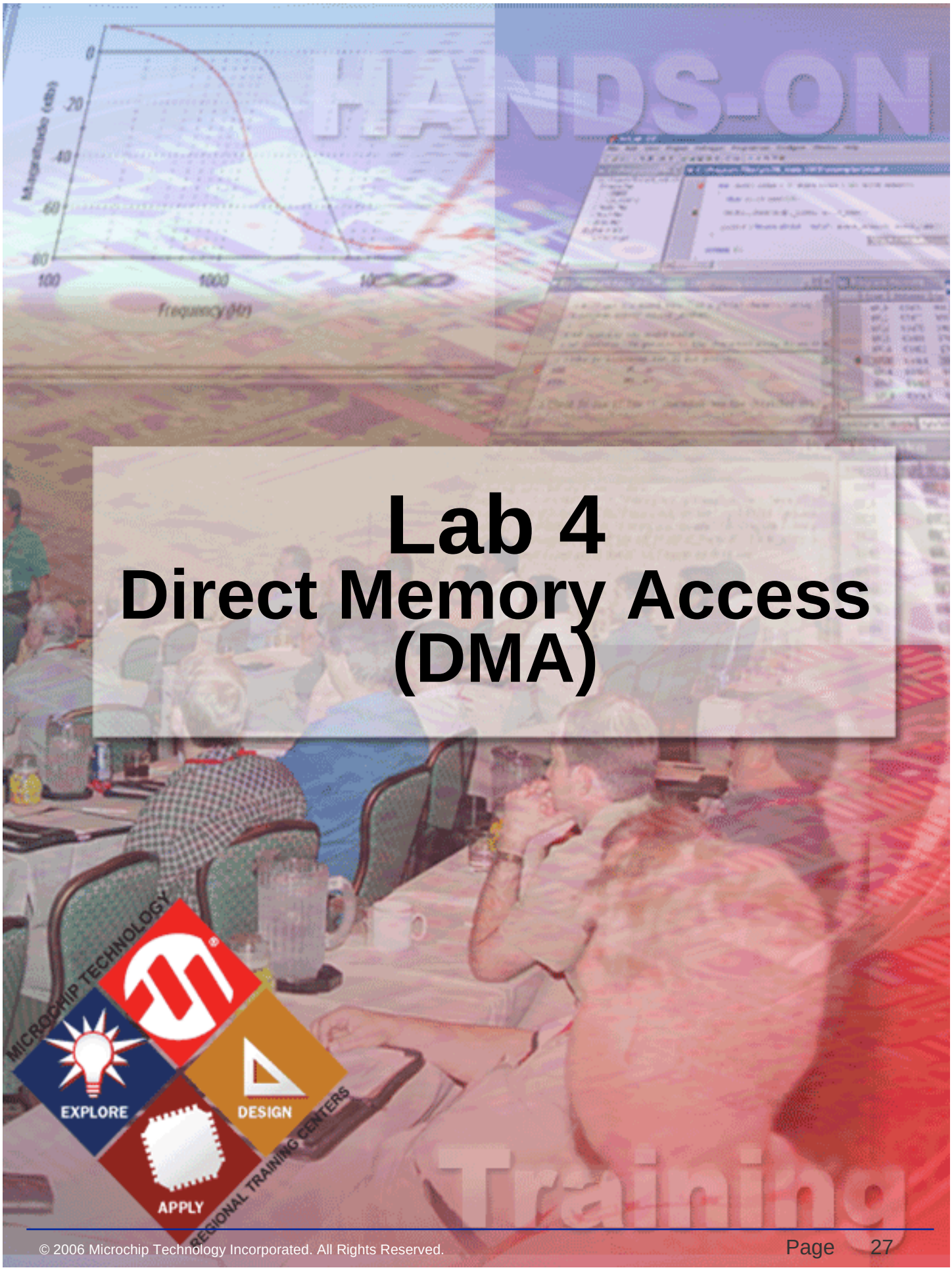
Lab 3 To Do

- 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!

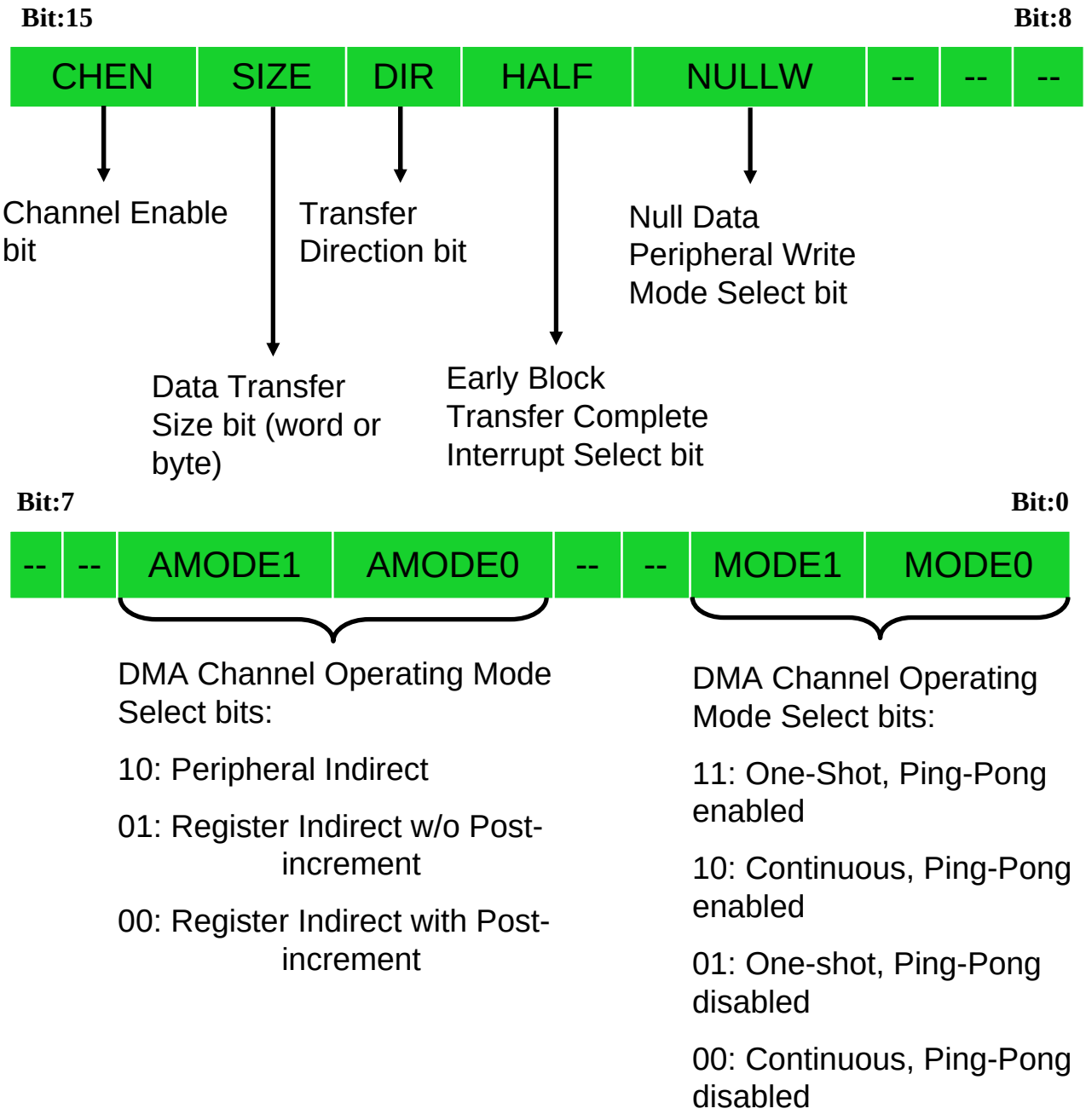


Lab 4 Goals

- 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 Resigers

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 (8-N-1, 9600)
  - Type characters into HyperTerminal

Lab 4 Expected Results

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