204 ADV

16-bit

Advanced Peripherals

Class Objective

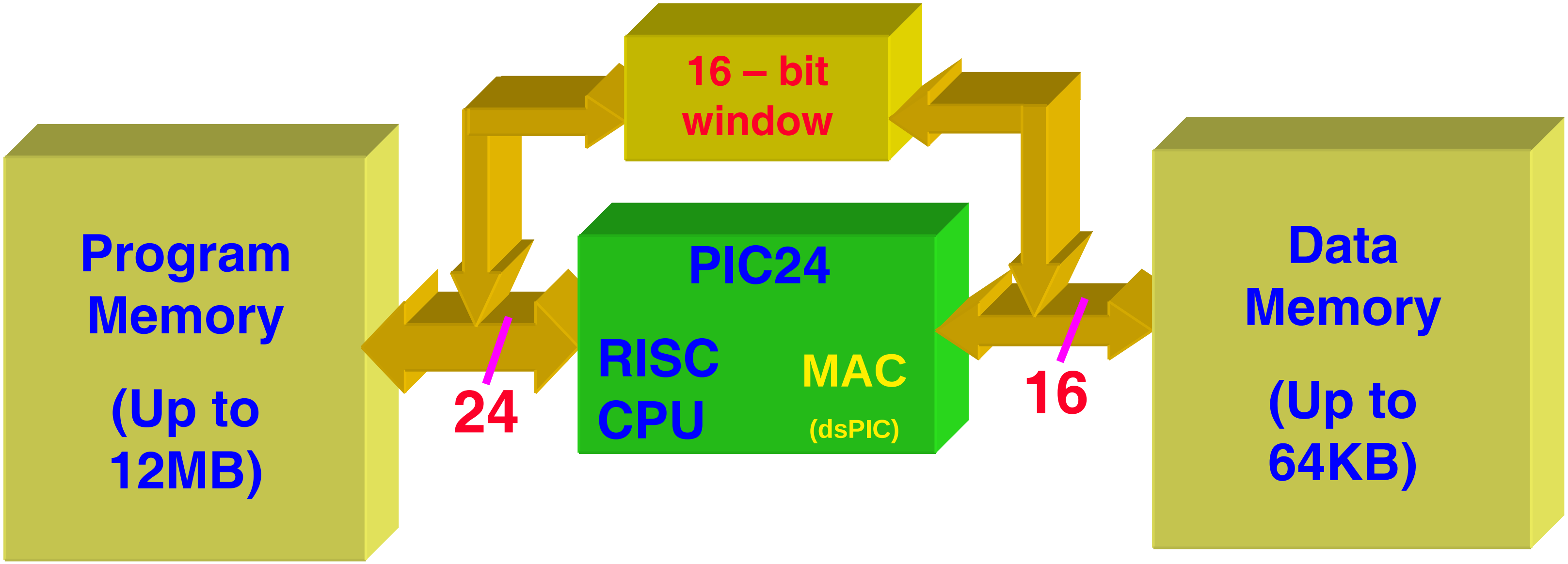
- **When you finish this class you will:**
  - Be familiar with using some of the advanced peripherals onboard Microchip’s16-bit devices
  - Be familiar with using MPLAB® IDE with the C30 compiler
  - Be familiar with using the Explorer16 Development Board

Agenda

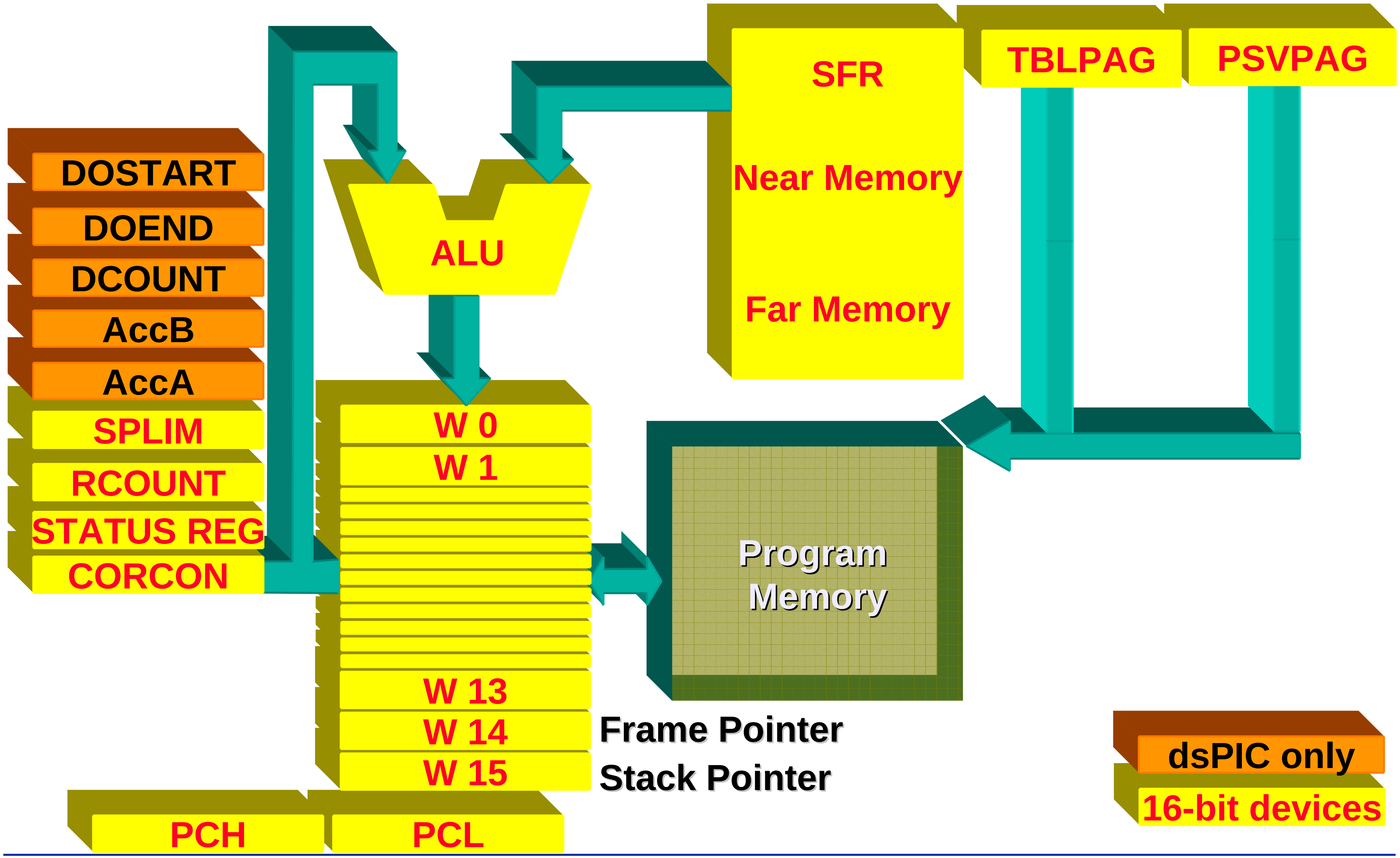
- **16-bit Devices**
  - 16-bit Refresher
    - **CPU architecture**
  - New Peripherals and Features
    - **PMP with hands-on**
    - **RTCC with hands-on**
    - **CRC Generator with hands-on**
    - **DMA with hands-on**

Harvard Architecture

- 16-bit microcontroller
- 24-bit Instruction width
- Data Transfer Mechanism between PM and DM

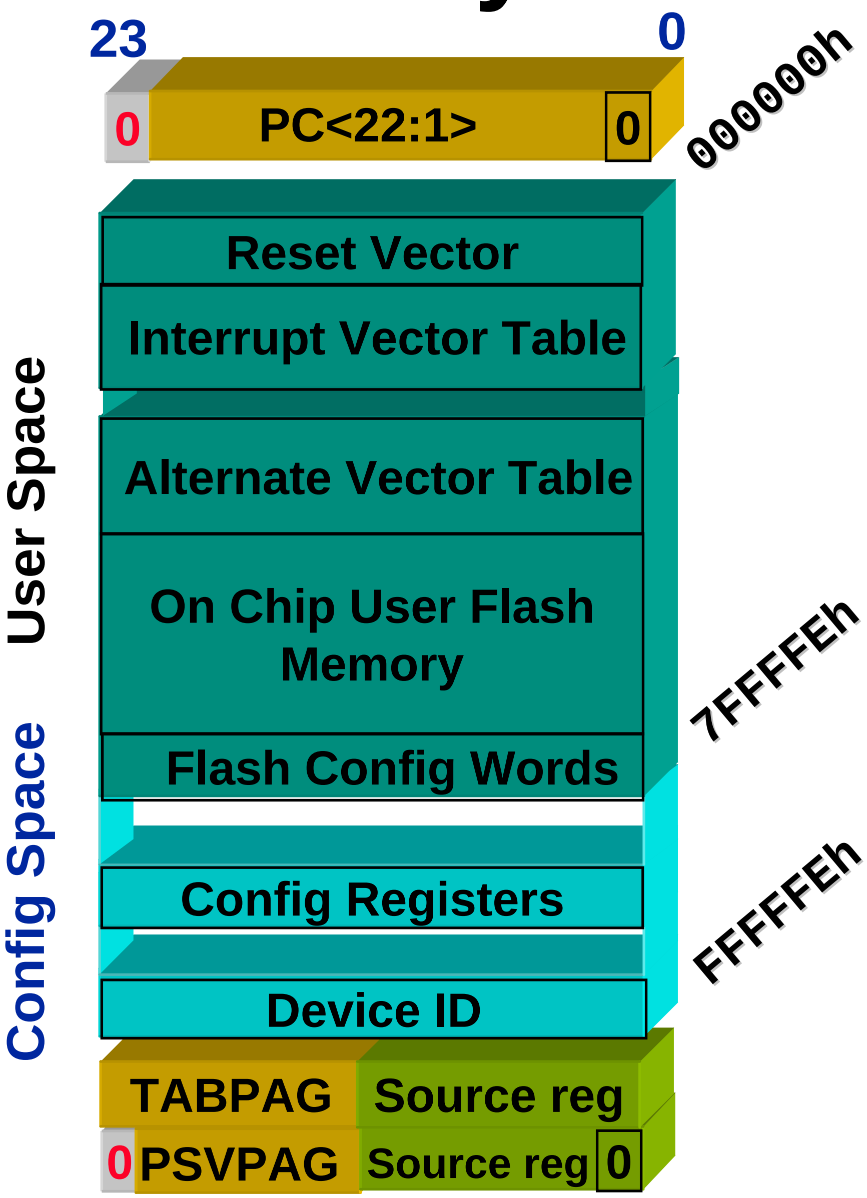


16-bit Architecture
Programmers model

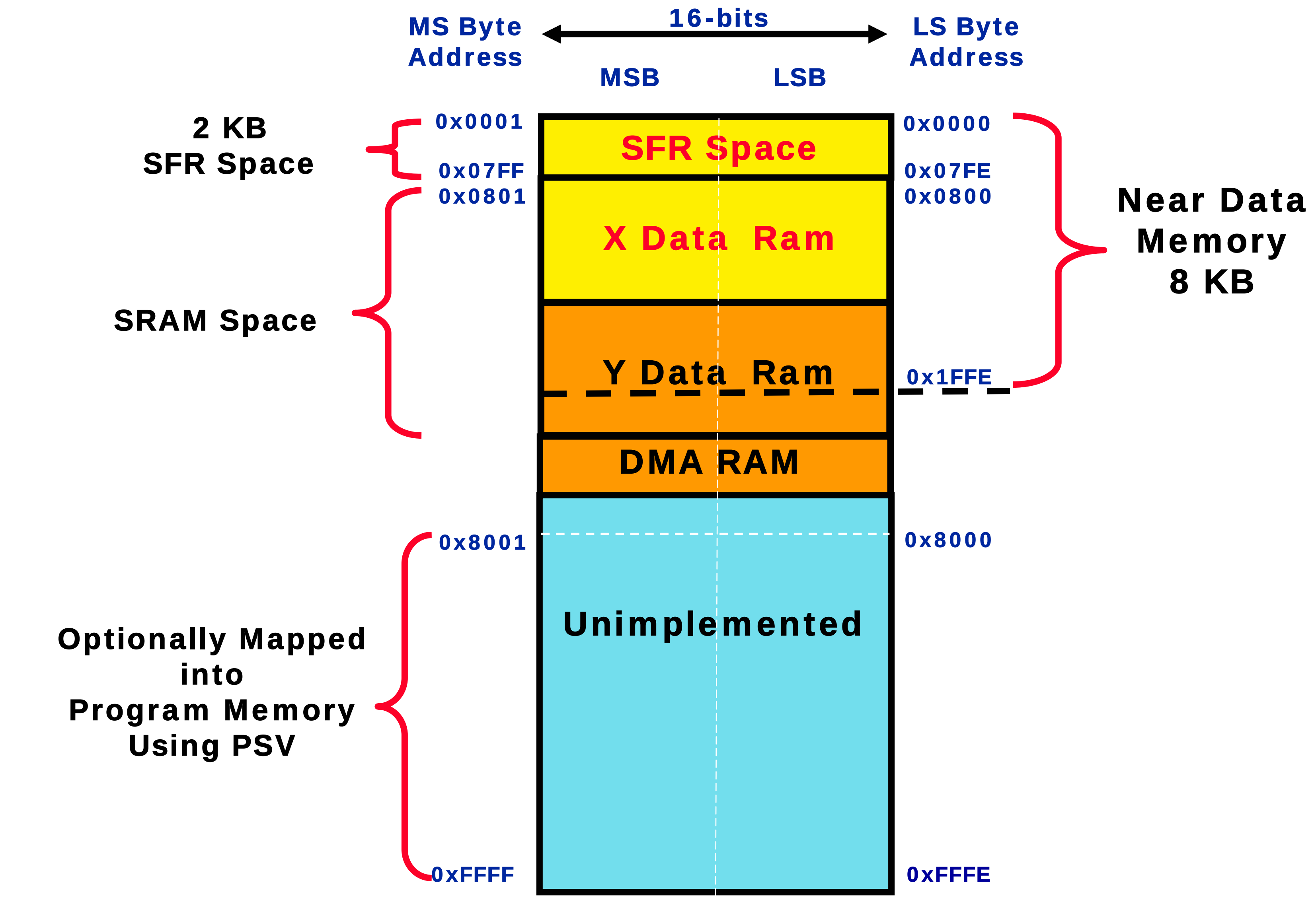


Program Memory

- Maximum 12MB
  - 4MB x 24-bit
  - 23-bit PC (PCH & PCL)
- PC increments in words (LSB always ‘0’)
- Reset Vector at 0
- Interrupt Vector Table from 4h to FEh
- User Code space from 200h to 7FFFFEh (what ever is implemented)



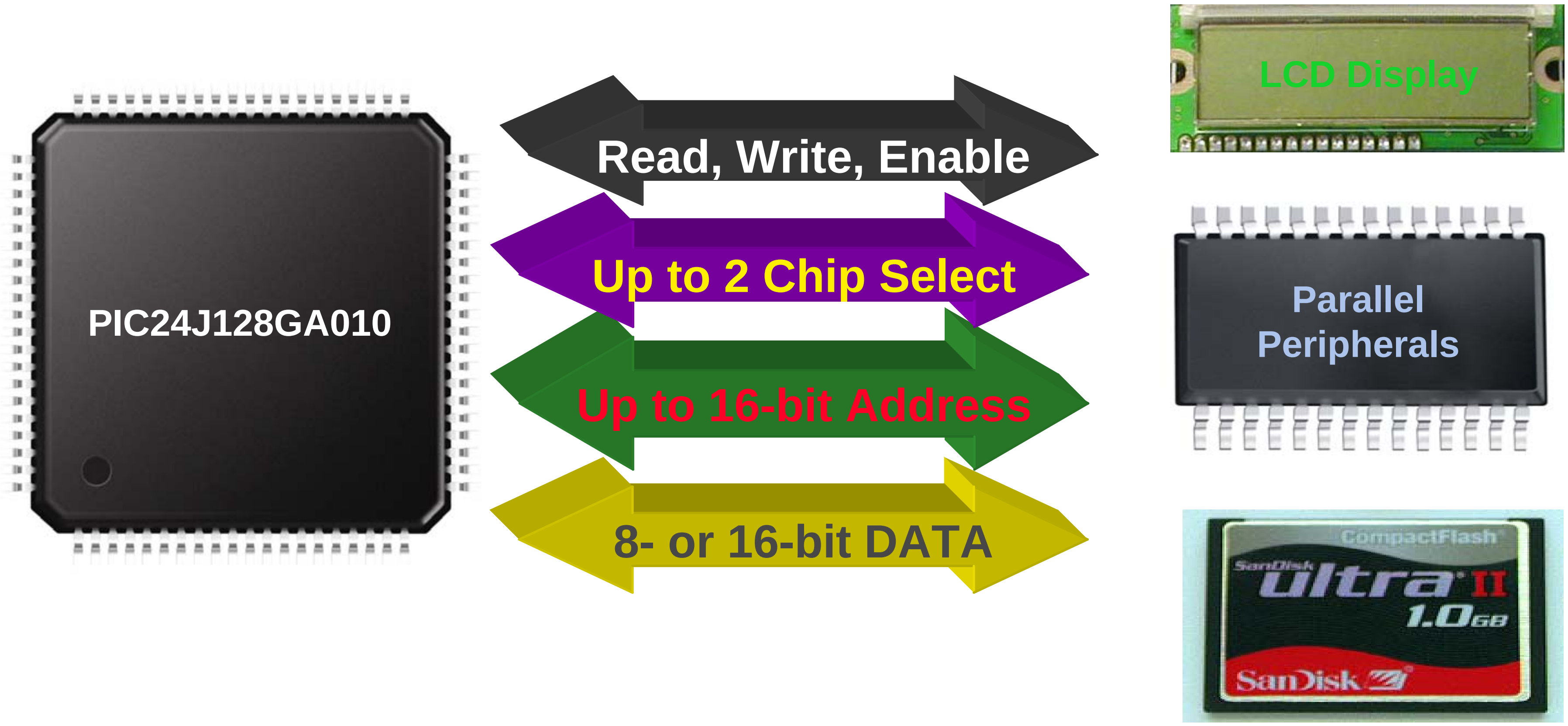
Data Memory Organization



Parallel Master Port

(PMP)

Parallel Master Port – PMP



PIC24J128GA010

Read, Write, Enable

Up to 2 Chip Select

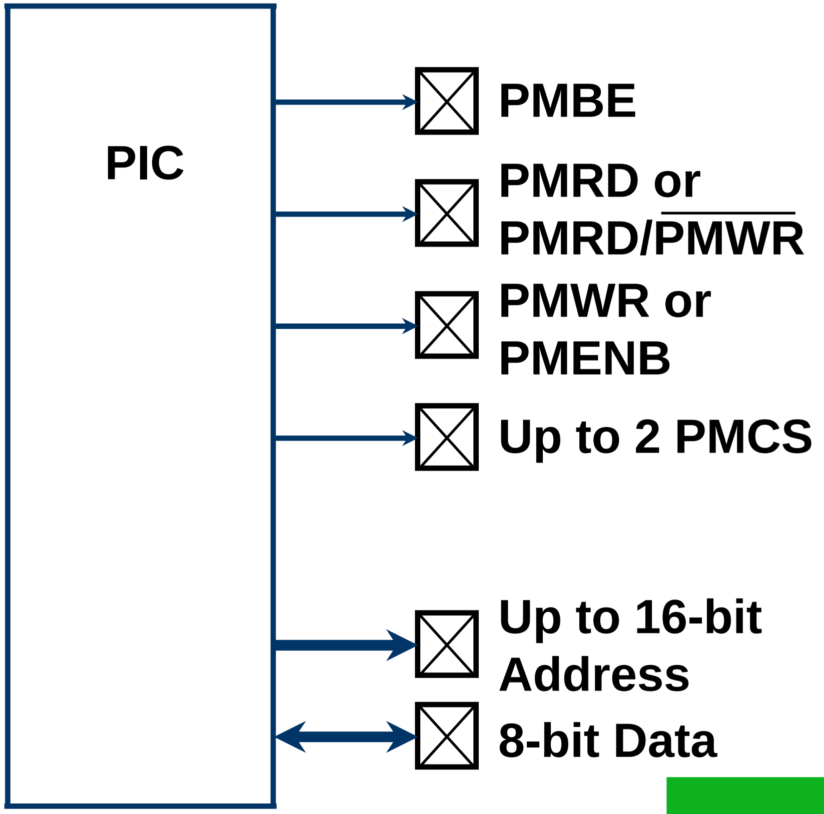
Up to 16-bit Address

8- or 16-bit DATA

LCD Display

Parallel Peripherals

PMP Configuration: Control Signals



PIC

PMBE

PMRD or PMRD/$\overline{\text{PMWR}}$

PMWR or PMENB

Up to 2 PMCS

Up to 16-bit Address

8-bit Data

The Signal can be individually Enabled or Disabled

PMCON: PTBEEN, PTWREN,
PMPEN: CS2, CS1

The Signal polarity can be individually selected

PMCON: ALP, CS2P, CS1P,
BEP, WRSP, RDSP

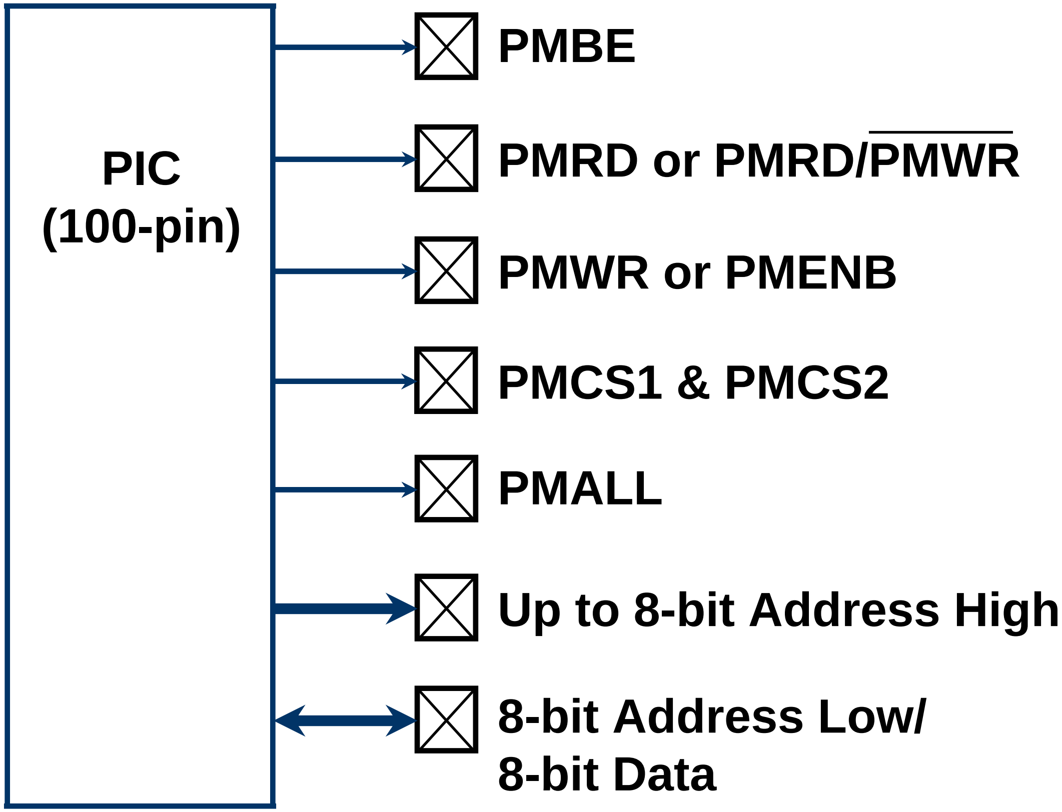
The Address pins can be individually Enabled or Disabled

PMPEN: Reg

PMCON<CSF1:CSF0>	Chip Select Function
00	CS1, CS2, A15, A14
01	CS1, CS2, A15, A14
10	CS1, CS2, A15, A14

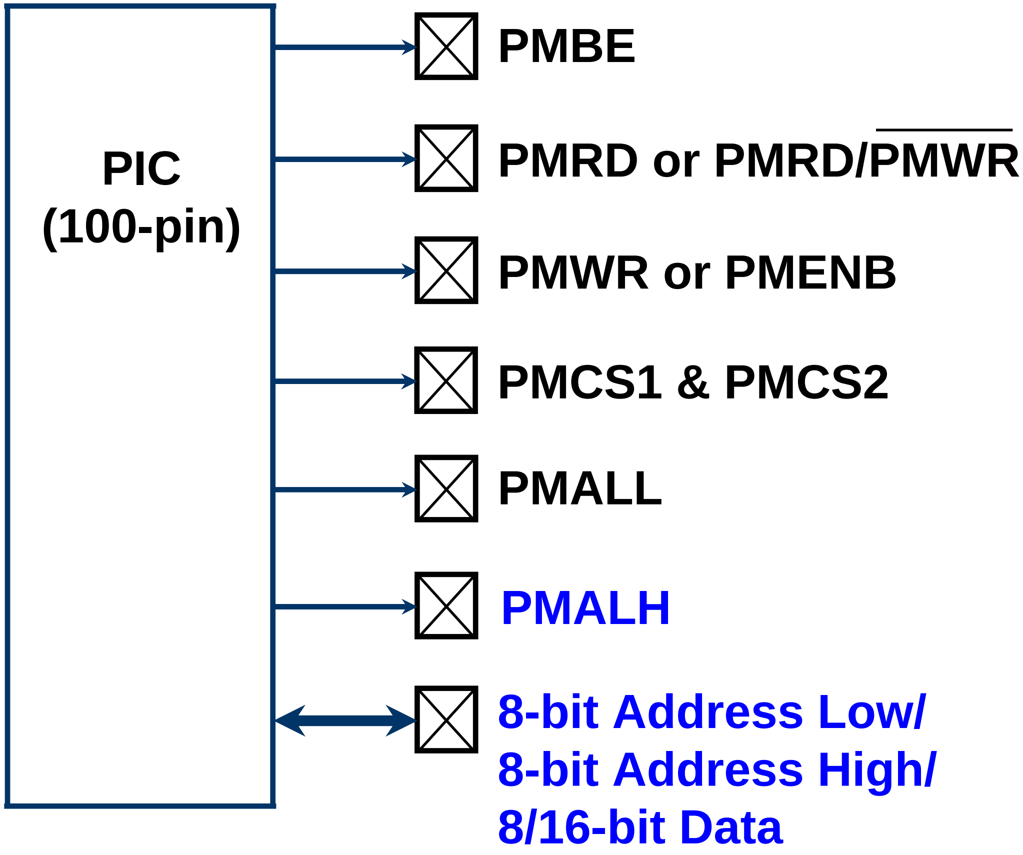
PMMODE<INCM1:INCM0>

PMP Configuration: Multiplexed Data Bus



PMCON<ADRMUX1:ADRMUX0> = 01

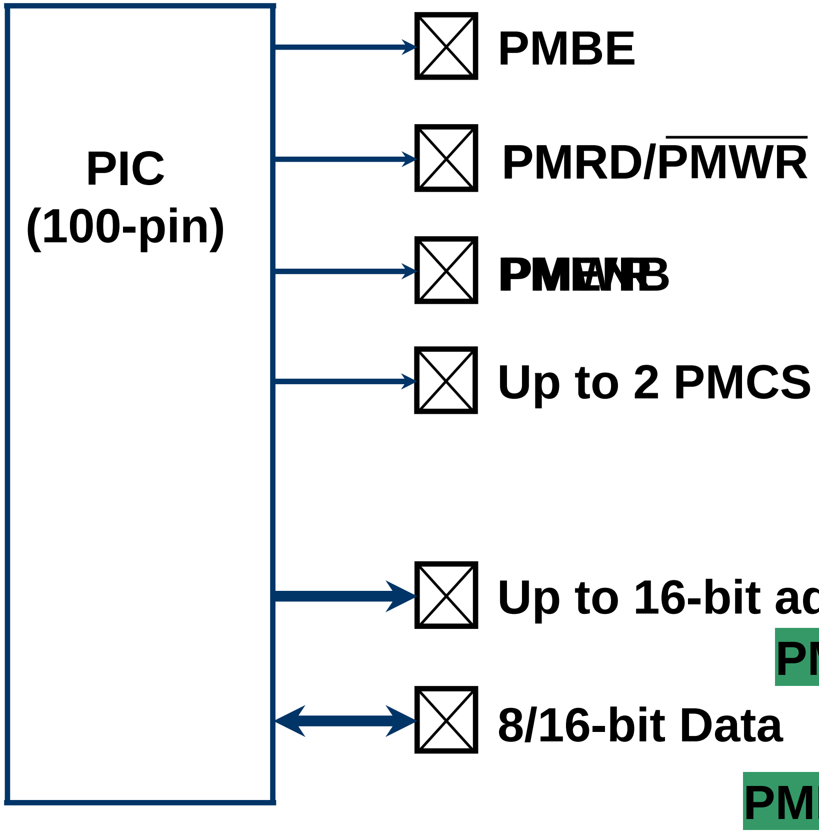
PMP Configuration: Multiplexed Data Bus

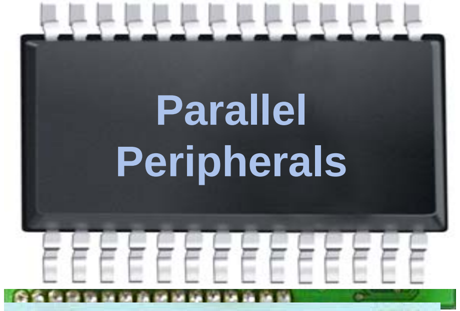


PMMODE<MODE16>

PMCON<ADRMUX1:ADRMUX0> = 10

PMP Configuration: Standard Peripherals





PMADDR

PMDIN2:PMDIN1

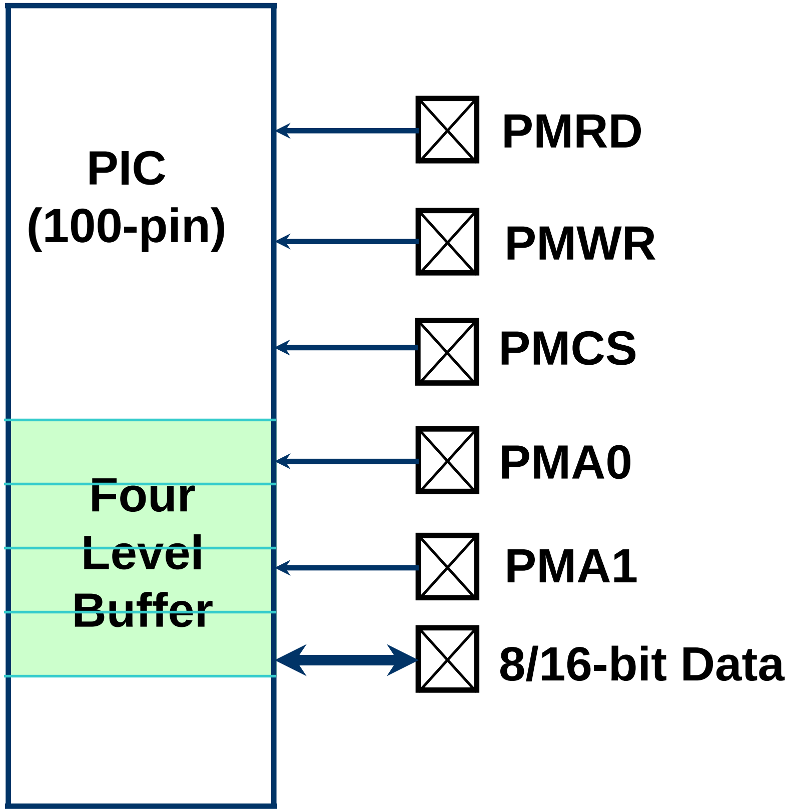
PMMODE<WAITB1:WAITB0>

PMMODE<MODE1:MODE0> = 10

PMMODE<WAITM3:WAITM0>

PMMODE<WAITE1:WAITE0>

PMP Configuration: Slave Mode



PMDIN2:PMDIN1

PMDOUT2:PMDOUT1

PMMODE<INCM1:INCM0> = 11

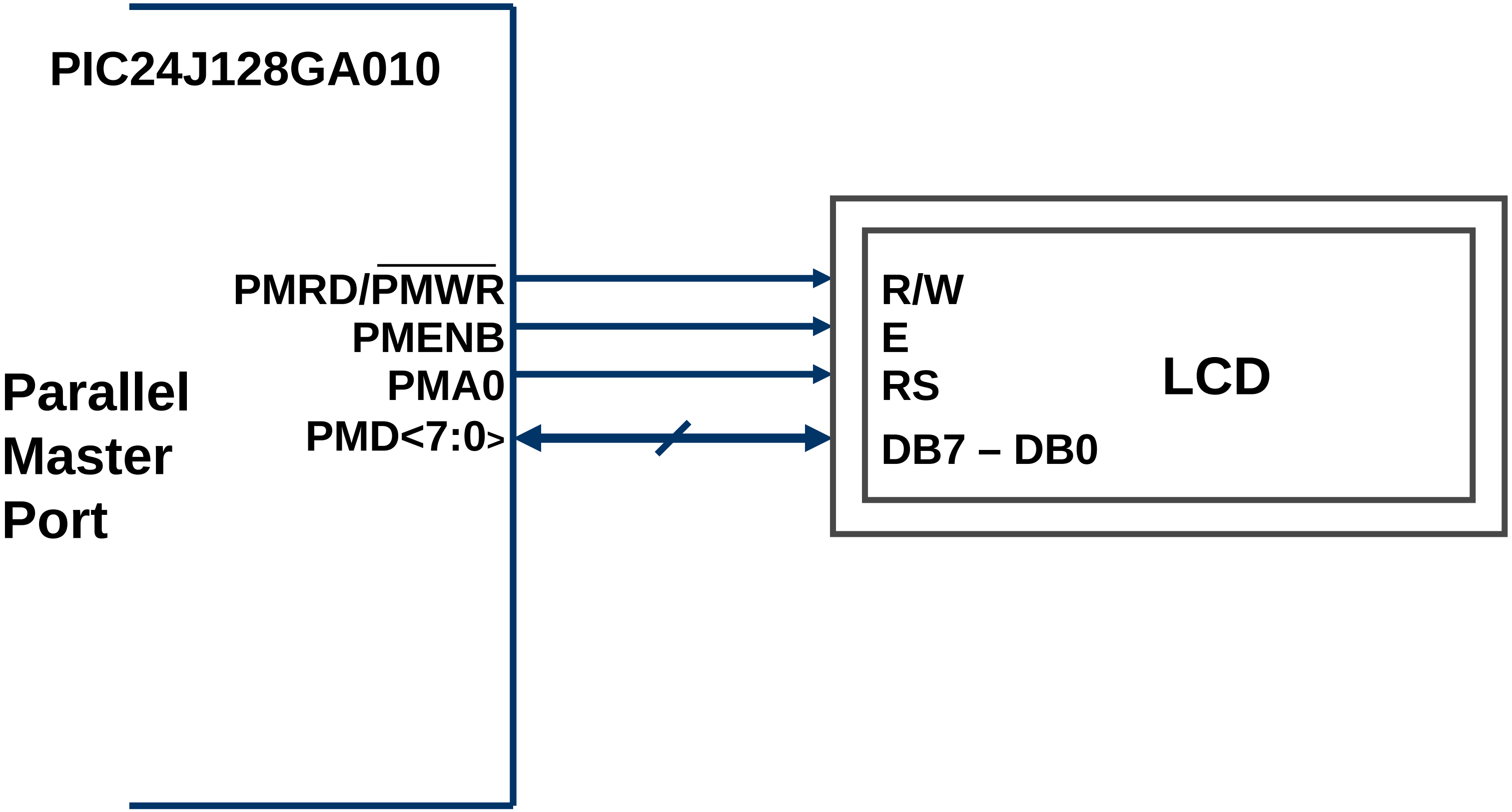
PMMODE<MODE1:MODE0> = 01

Let’s go Hands on

Lab 1

The Parallel Master Port

Lab 1 – LCD on the PMP

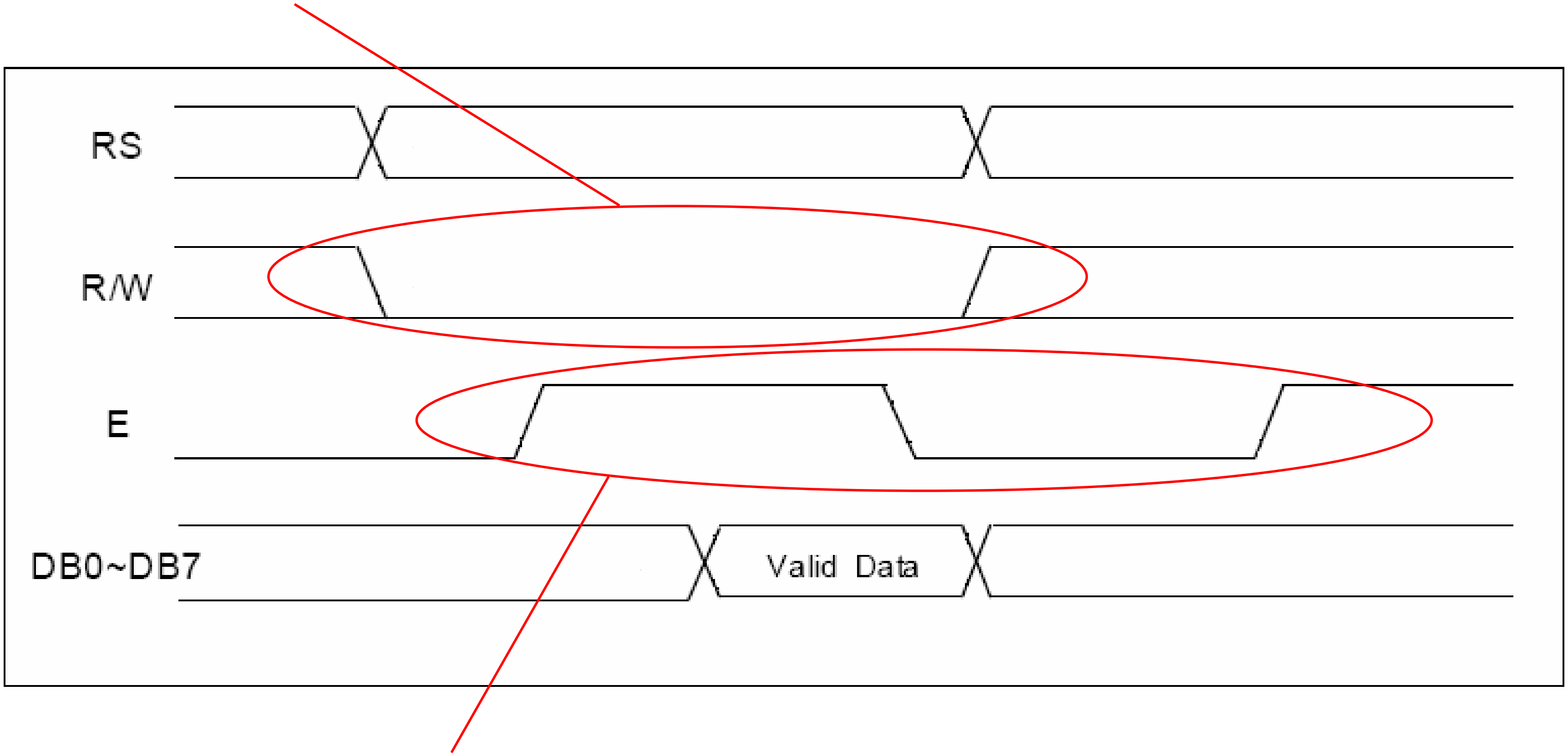


Lab 1 – LCD on the PMP

- **Setup PMP for an LCD, think about:**
  - Signals
  - Polarity
  - LCD interface
  - Addressing
  - Timing

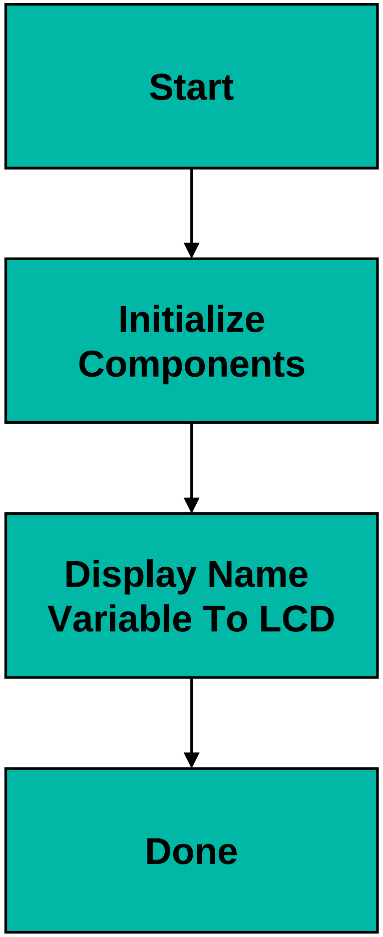
Lab 1 – LCD Write Timing

Write is active low.
But remember! PMP signal we are using is PMRD/$\overline{\text{PMWR}}$,
so the pin needs to be configured as active high.



Enable is active High.

Lab 1 – LCD on the PMP



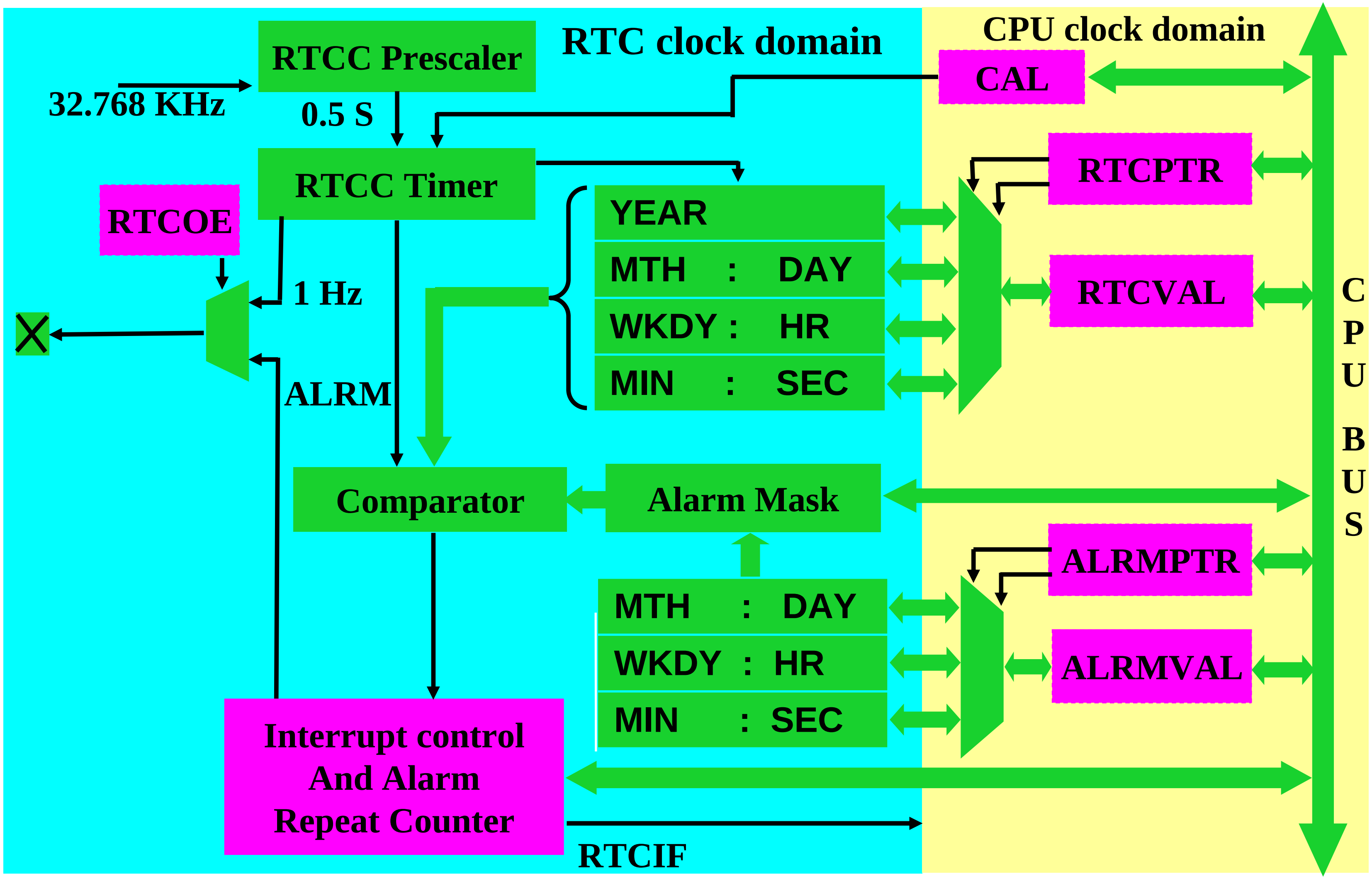
Lab 1 – LCD on the PMP

- **Goals**
  - Learn about the Parallel Master Port (PMP)
  - Refresh knowledge of the common LCD interface
  - Do some coding for PIC24
- **Lab**
  - Add setup code to initialize the PMP
  - Put your name on the display

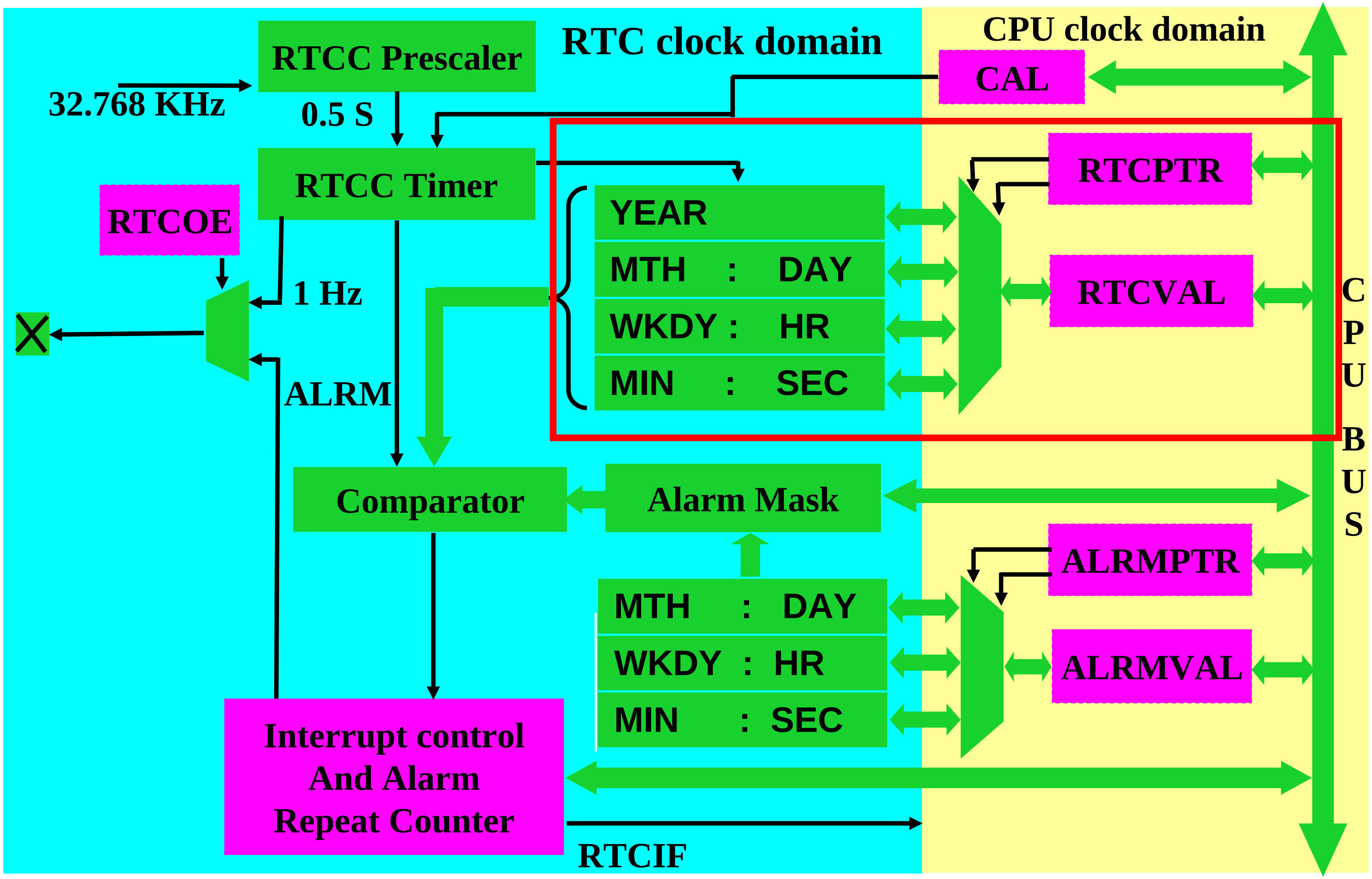
Real-time Clock & Calendar

(RTCC)

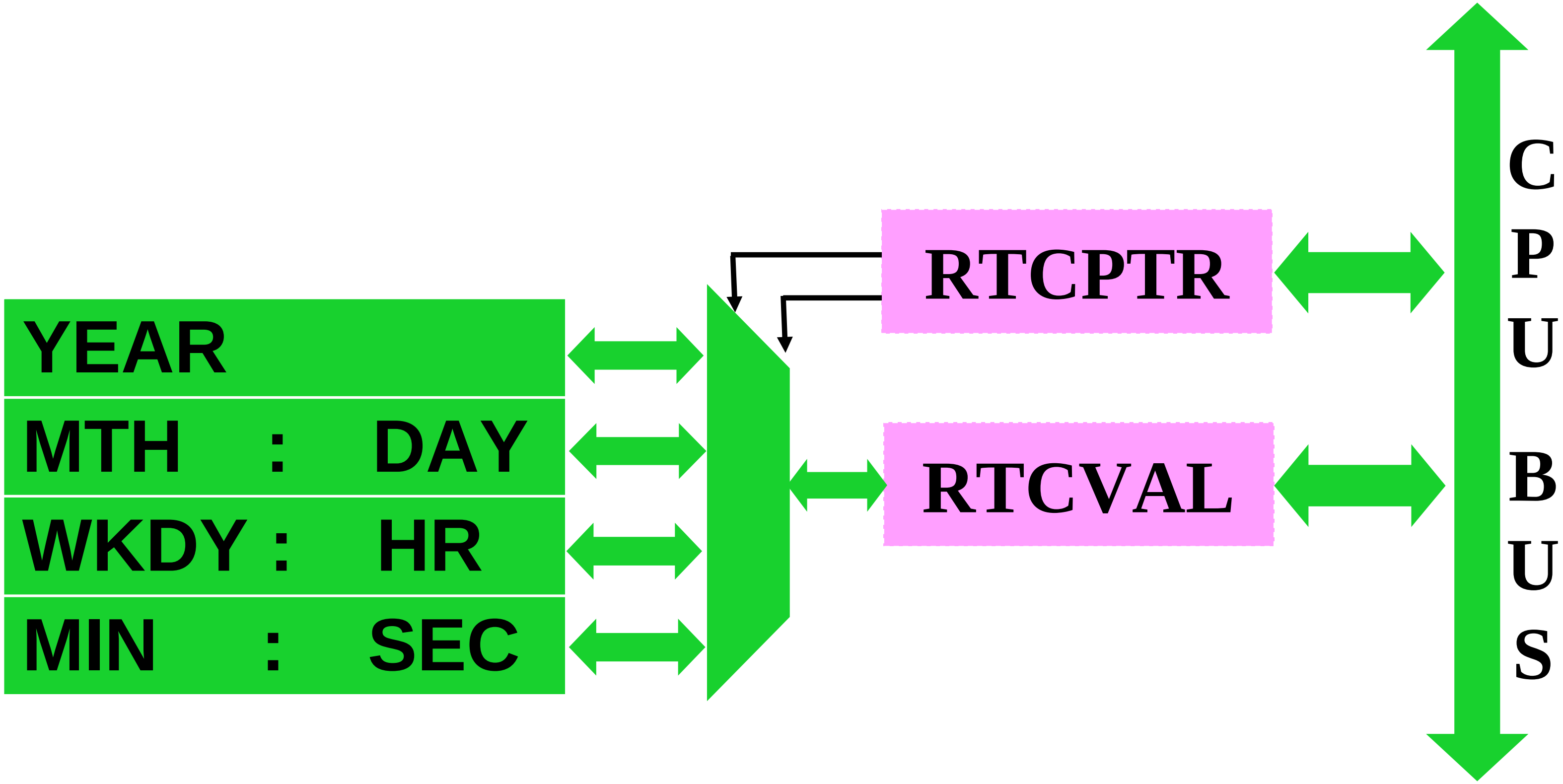
RTCC: Block Diagram



RTCC: Block Diagram



RTCC: Registers



RTCC: Registers

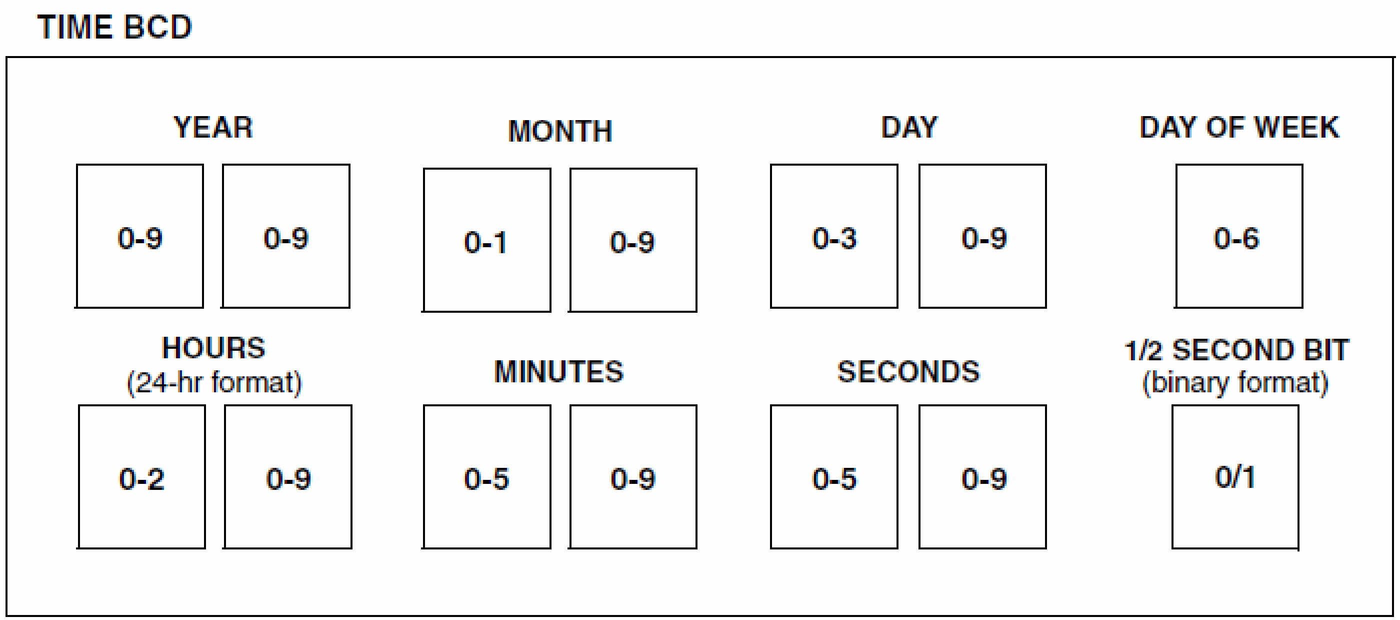
RTCVAL: RTCC Value Register

- Pointer bits, RTCPTR<1:0>, indicate which register is read from and written to RTCVAL
- 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

RTCC: Registers

- RTCVAL and ALRMVAL registers use BCD format
- In BCD, each nibble (4 bits) of a word encodes a number from 0-9.

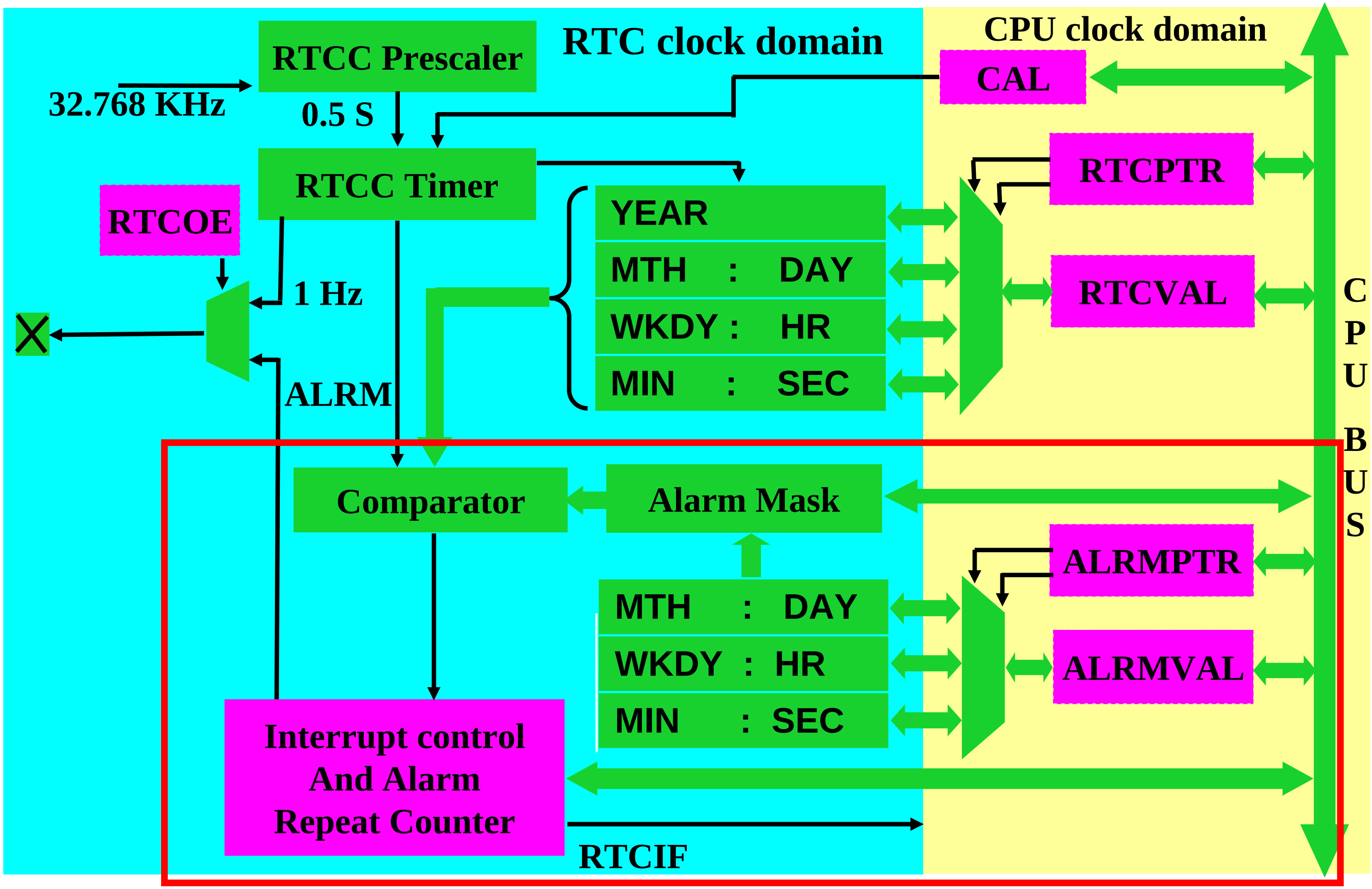


TIME BCD

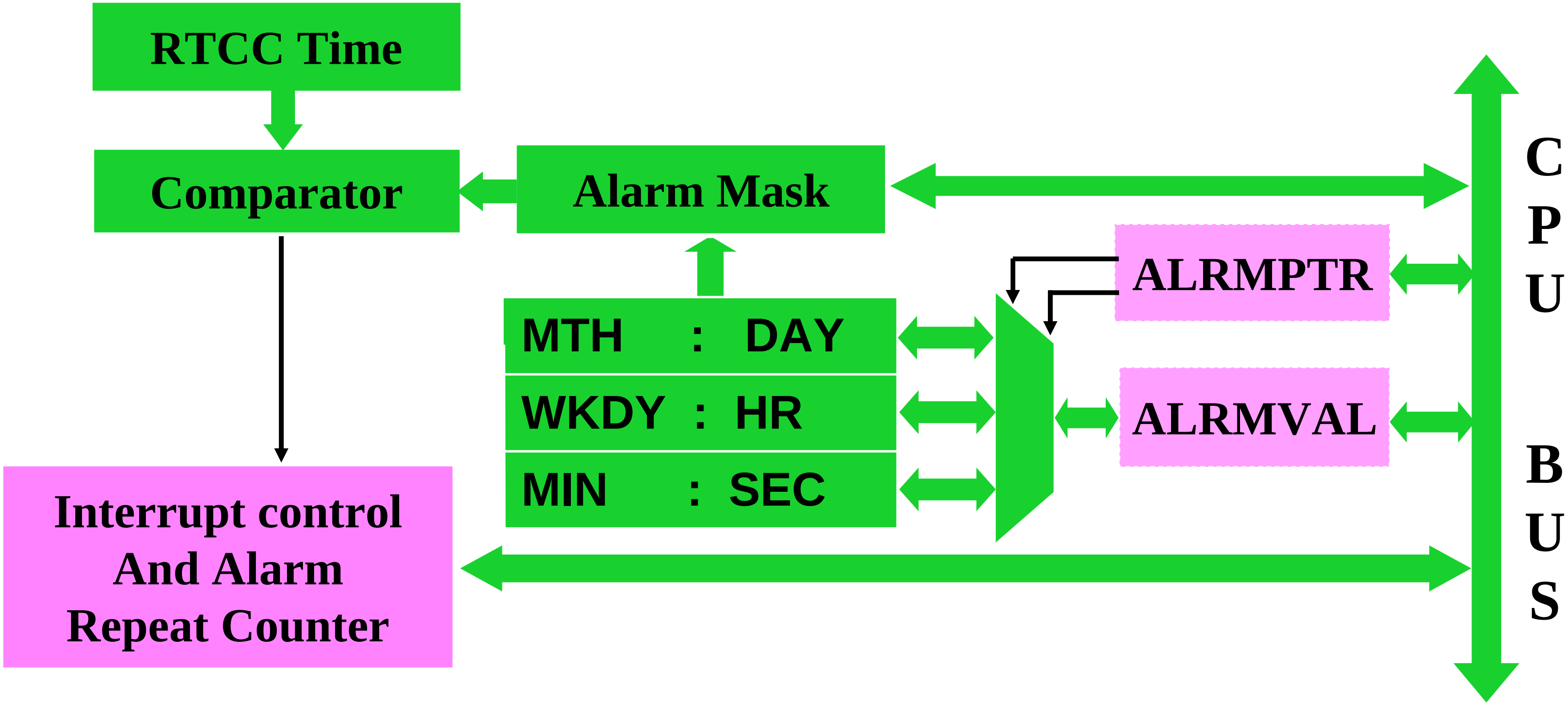
YEAR		MONTH		DAY		DAY OF WEEK
0-9	0-9	0-1	0-9	0-3	0-9	0-6

HOURS (24-hr format)		MINUTES		SECONDS		1/2 SECOND BIT (binary format)
0-2	0-9	0-5	0-9	0-5	0-9	0/1

RTCC: Block Diagram

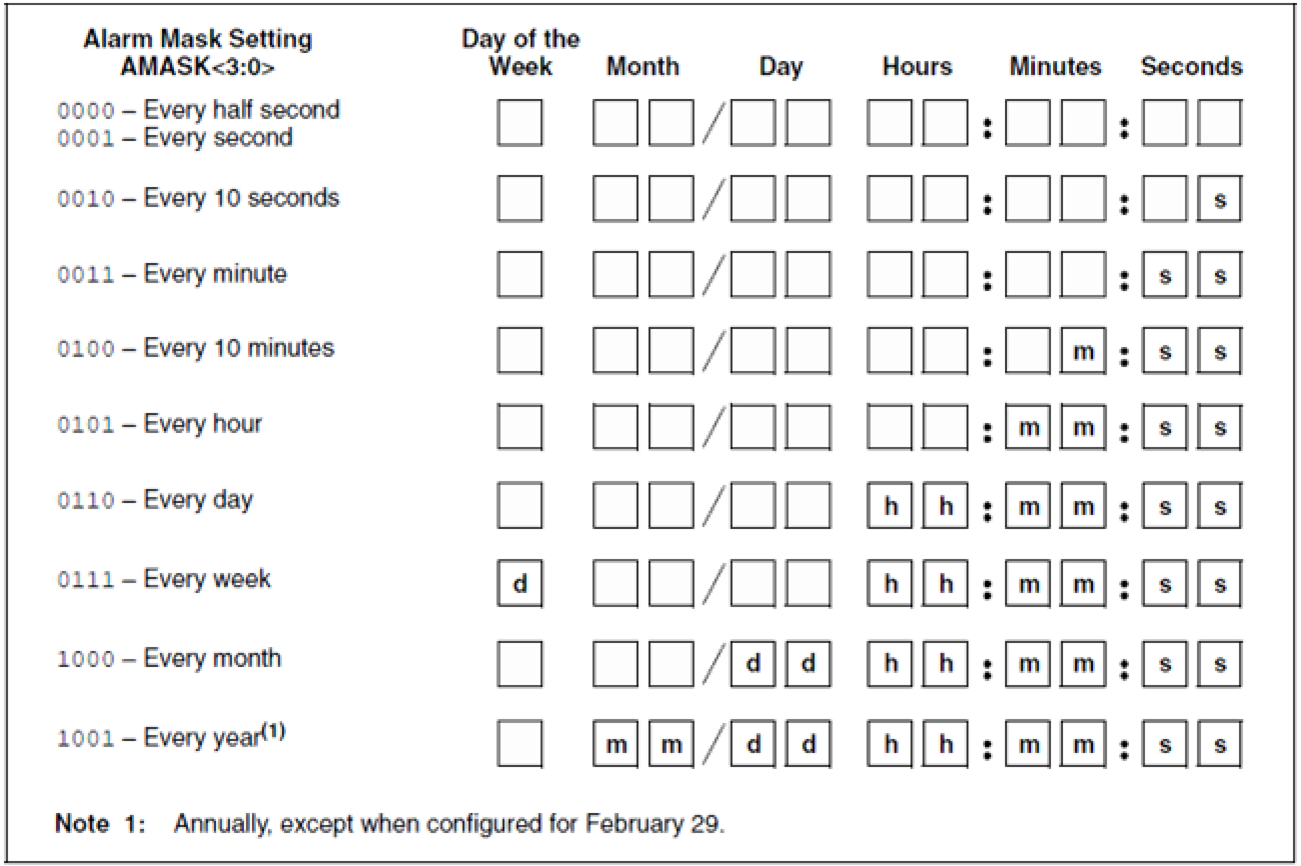


RTCC: Alarm


RTCC Time
Comparator
Alarm Mask
ALRMPTR
MTH : DAY
WKDY : HR
MIN : SEC
ALRMVAL
Interrupt control And Alarm Repeat Counter
CPU BUS

RTCC: Alarm

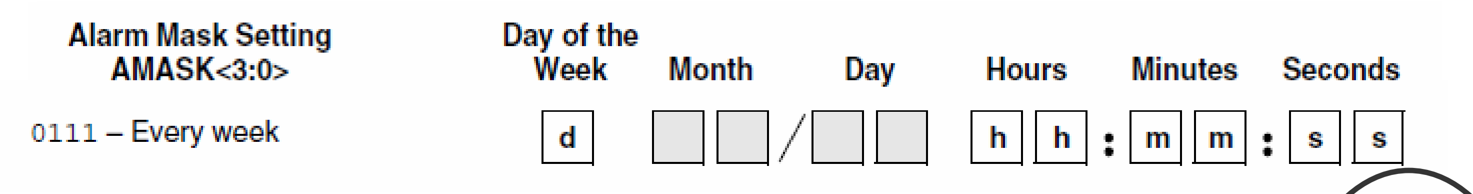
- **AMASK<3:0> controls which registers of ALRMVAL and RTCVAL are compared to generate an alarm**



Alarm Mask Setting AMASK<3:0>	Day of the Week	Month			Day		Hours			Minutes			Seconds	
0000 – Every half second 0001 – Every second	□	□	□	/	□	□	□	□	:	□	□	:	□	□
0010 – Every 10 seconds	□	□	□	/	□	□	□	□	:	□	□	:	□	s
0011 – Every minute	□	□	□	/	□	□	□	□	:	□	□	:	s	s
0100 – Every 10 minutes	□	□	□	/	□	□	□	□	:	□	m	:	s	s
0101 – Every hour	□	□	□	/	□	□	□	□	:	m	m	:	s	s
0110 – Every day	□	□	□	/	□	□	h	h	:	m	m	:	s	s
0111 – Every week	d	□	□	/	□	□	h	h	:	m	m	:	s	s
1000 – Every month	□	□	□	/	d	d	h	h	:	m	m	:	s	s
1001 – Every year[1]	□	m	m	/	d	d	h	h	:	m	m	:	s	s

Note 1: Annually, except when configured for February 29.

RTCC: Alarm



Alarm Mask Setting AMASK<3:0>	Day of the Week	Month		Day	Hours		Minutes		Seconds
0111 – Every week	d	□ □	/	□ □	h h	:	m m	:	s s

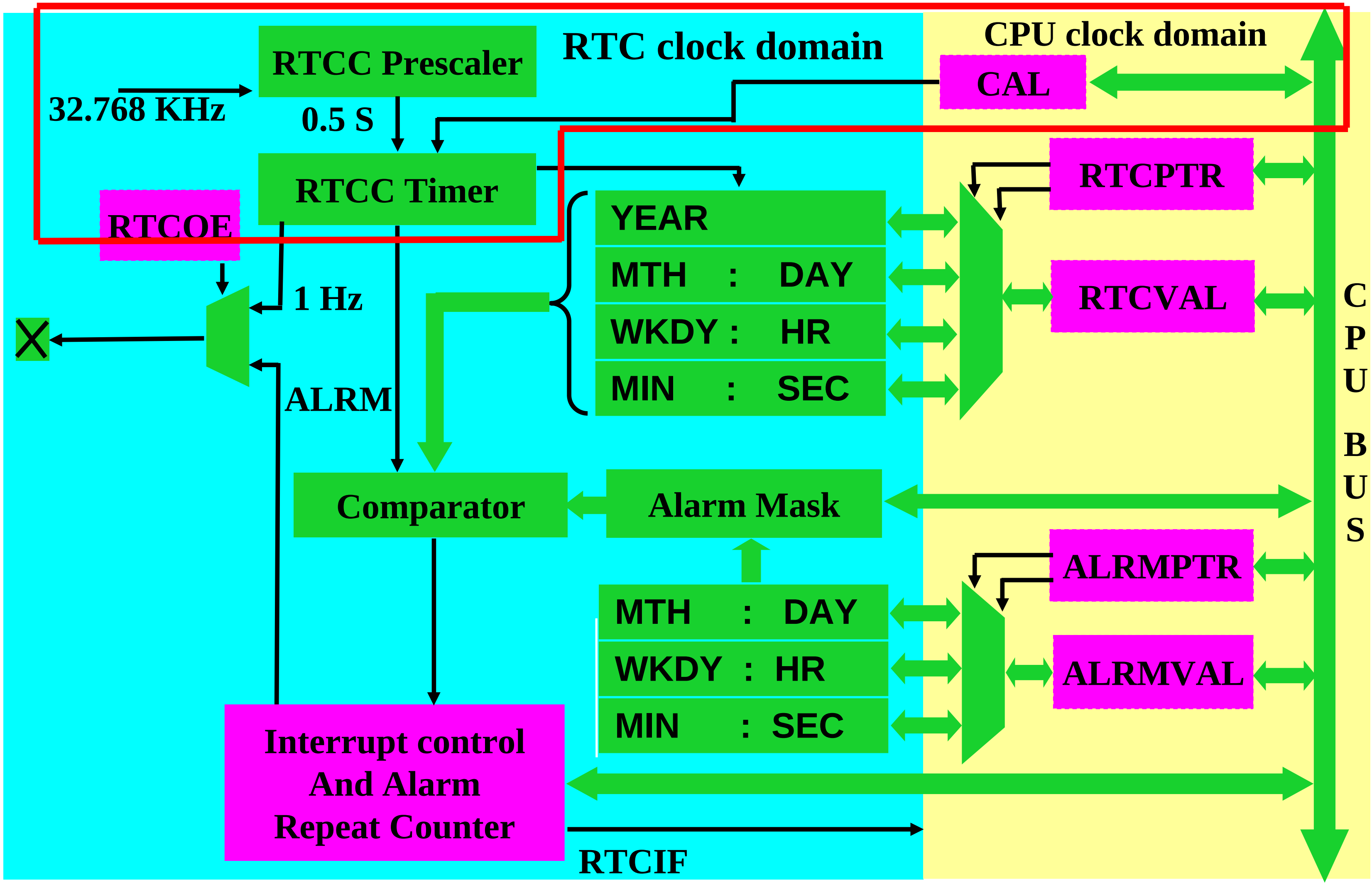
ALRMMNTH	ALRMDAY	ALRMWD	ALRMHR	ALRMMIN	ALRMSEC
12	24	Tuesday	4	45	53

YEAR	MONTH	DAY	WEEKDAY	HOURS	MINUTES	SECONDS
06	9	5	Tuesday	4	45	52

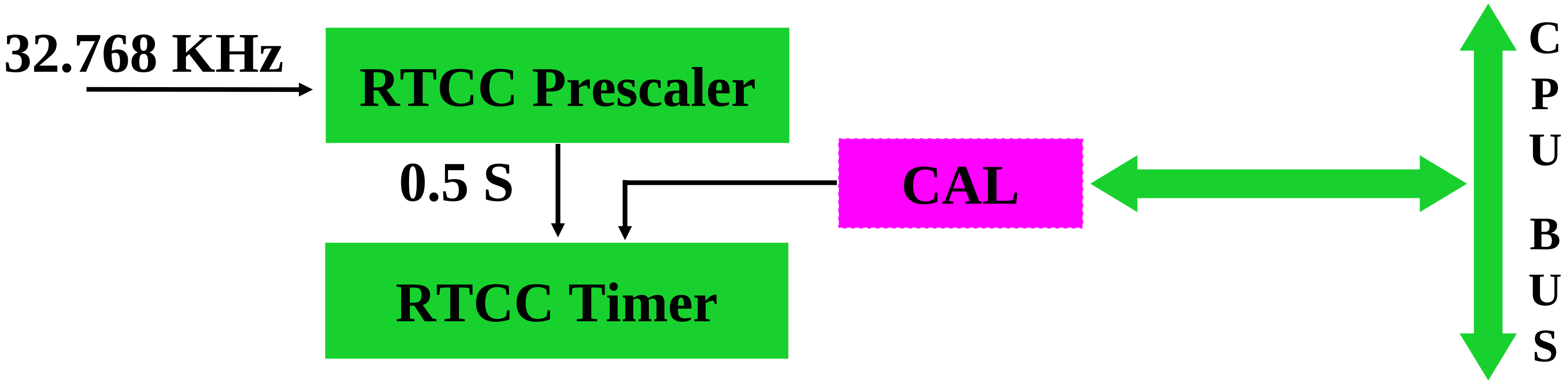
ALARM INTERRUPT

- **Chime allows ARPT to rollover from 00 to FF**
  - Alarms can be repeated indefinitely

RTCC: Block Diagram

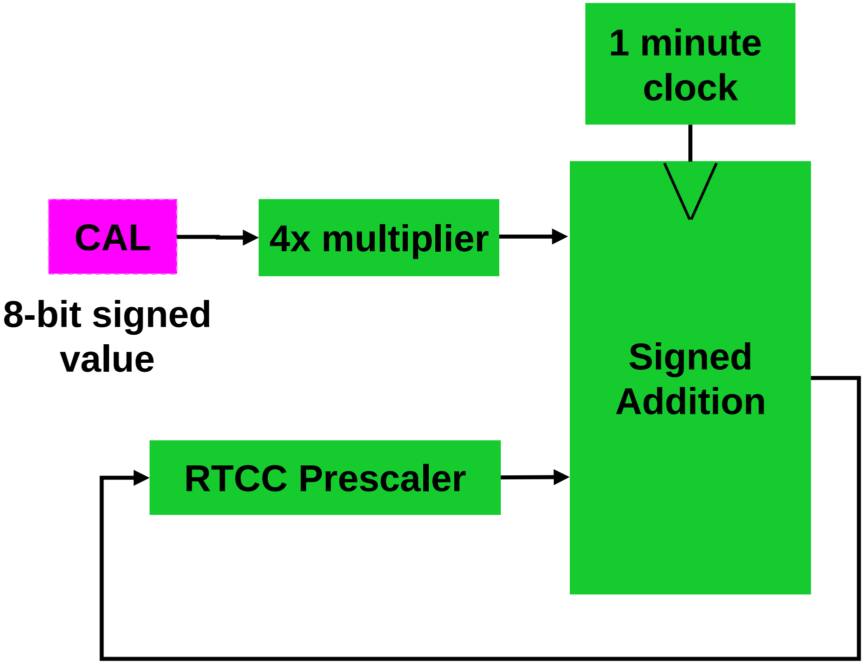


RTCC: Clock Calibration

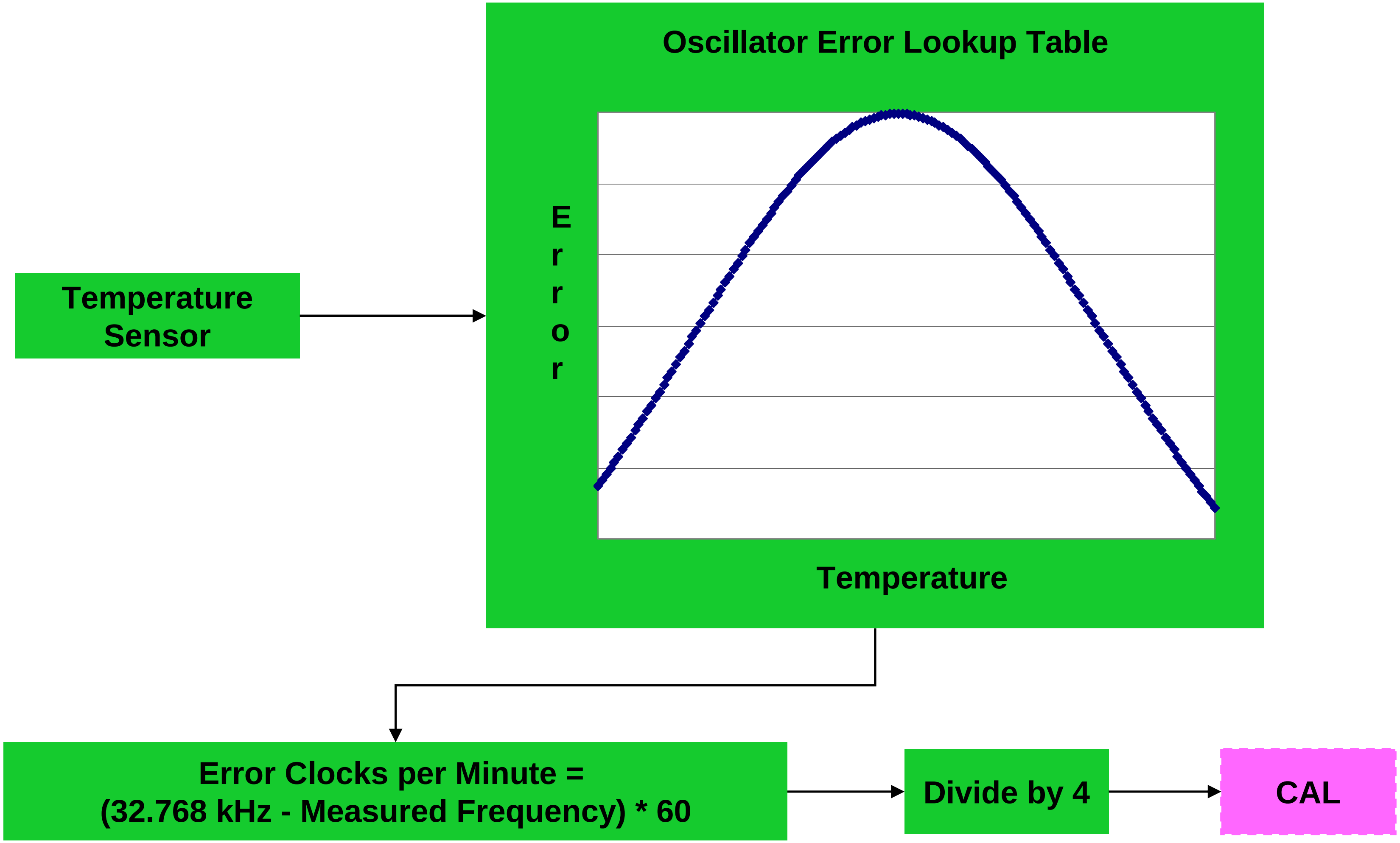


RTCC: Clock Calibration

- Calibrating the RTCC allows for accuracy with error less than 3 seconds per month



RTCC: Temperature Compensation



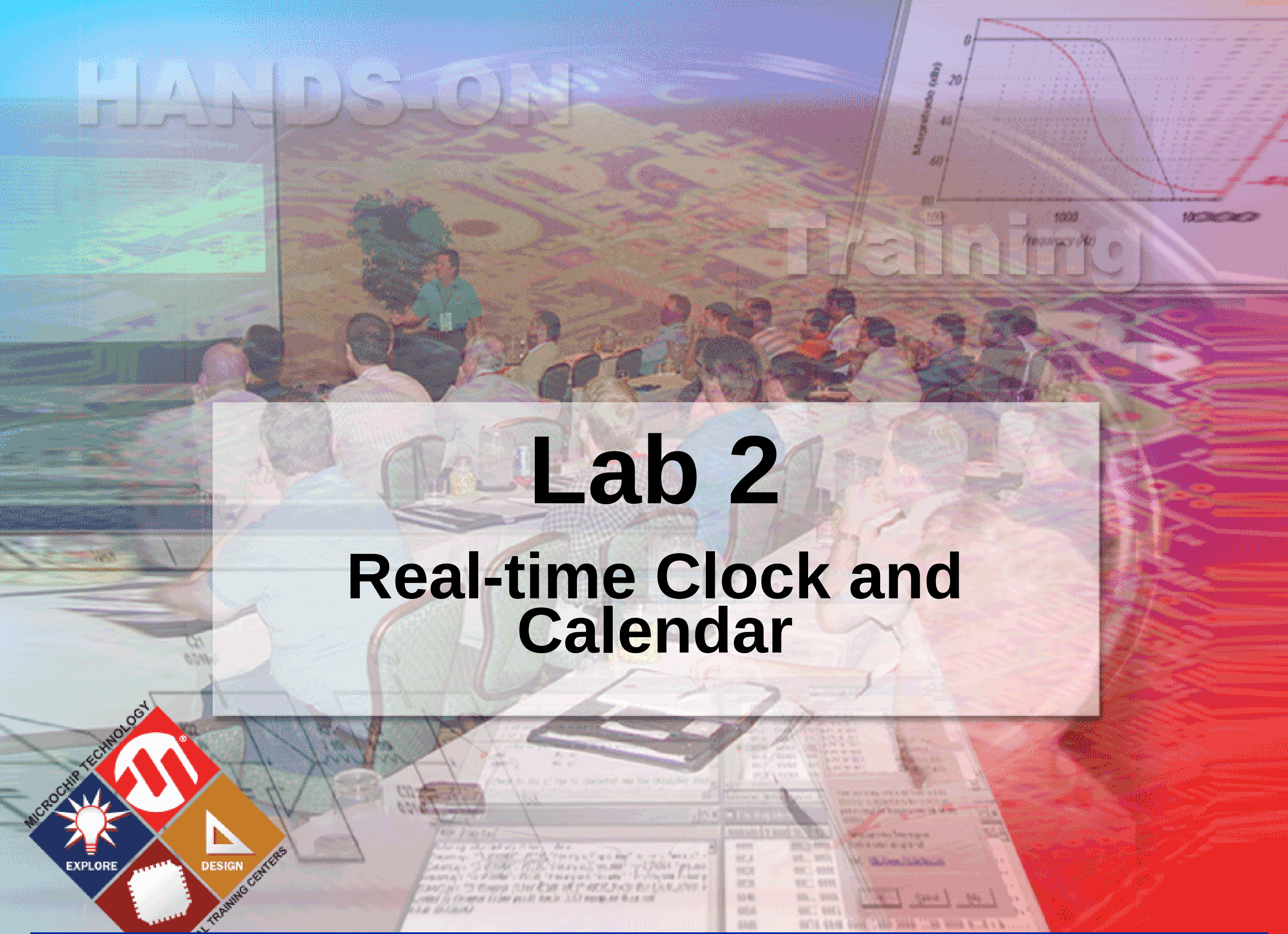
Temperature Sensor

Oscillator Error Lookup Table

Error Clocks per Minute =
(32.768 kHz - Measured Frequency) * 60

Divide by 4

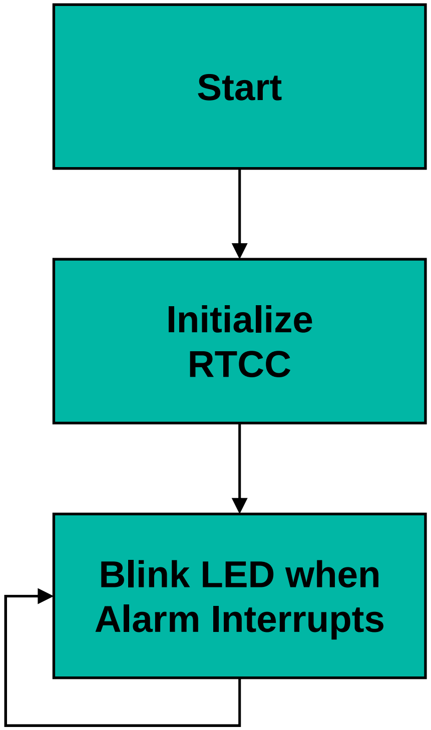
CAL



Lab 2

Real-time Clock and Calendar

Lab 2 – RTCC



Start

Initialize RTCC

Blink LED when Alarm Interrupts

Lab 2 – RTCC

- **Goals**
  - Learn about the Real-Time Clock and Calendar (RTCC)
  - Do some more coding for PIC24
- **Three parts to this lab**
  - Add code to unlock RTCC registers (rtcc.c)
  - Add code to set the time (rtcc.c)
  - Add code to set an alarm (rtcc.c)

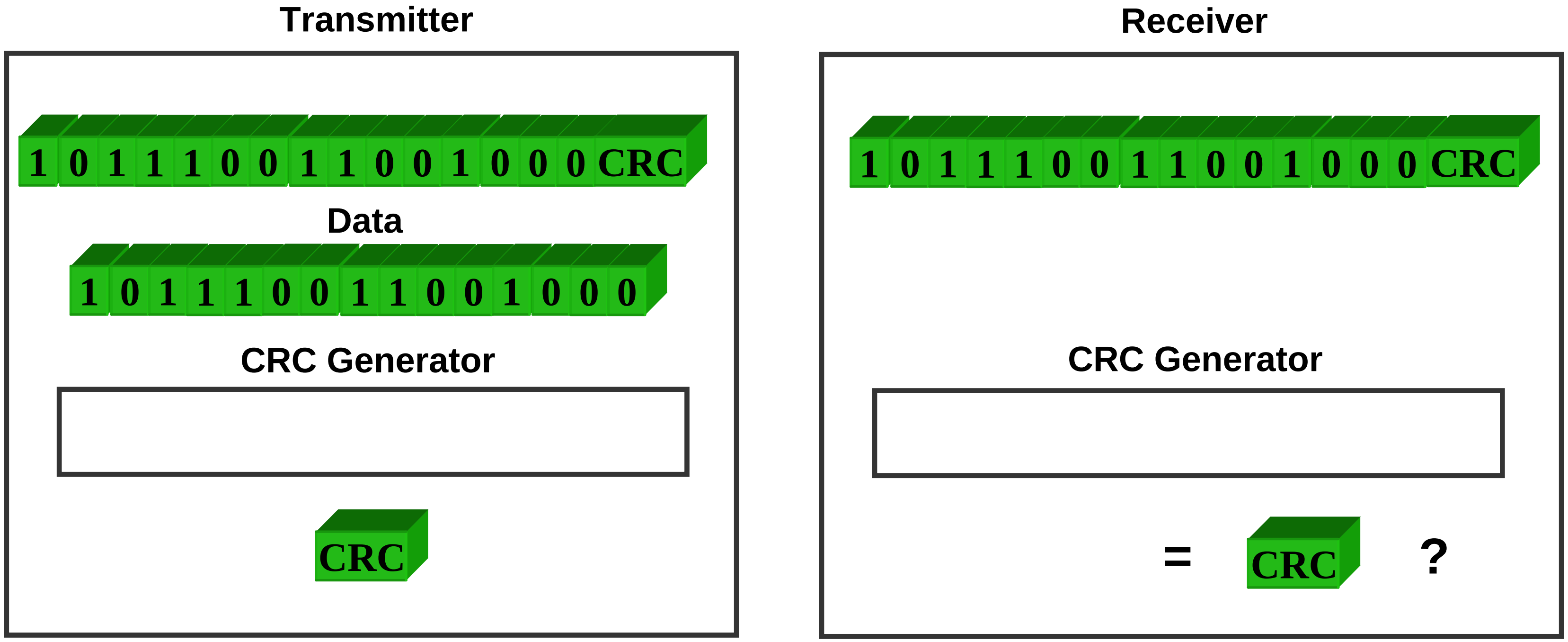
Programmable Cyclic Redundancy Check Generator (CRC)

What Is CRC?

- CRC - Cyclic Redundancy Check
- CRC provides a simple and powerful method for the detection of errors in memory and communications
- CRC is a technique to detect errors but not for correcting errors

CRC Checksum

- The CRC Checksum is appended to the end of the data
- The receiver runs the data through a CRC Generator and compares the result with the received CRC checksum



CRC Messages

- **The CRC method treats the data as a polynomial**
- **For example, if data is 11100101**
  - Data: 1 1 1 0 0 1 0 1
  - Poly: x7 x6 x5 x4 x3 x2 x 1
- **So the data polynomial will be**
  - x7+x6+x5+x2+1

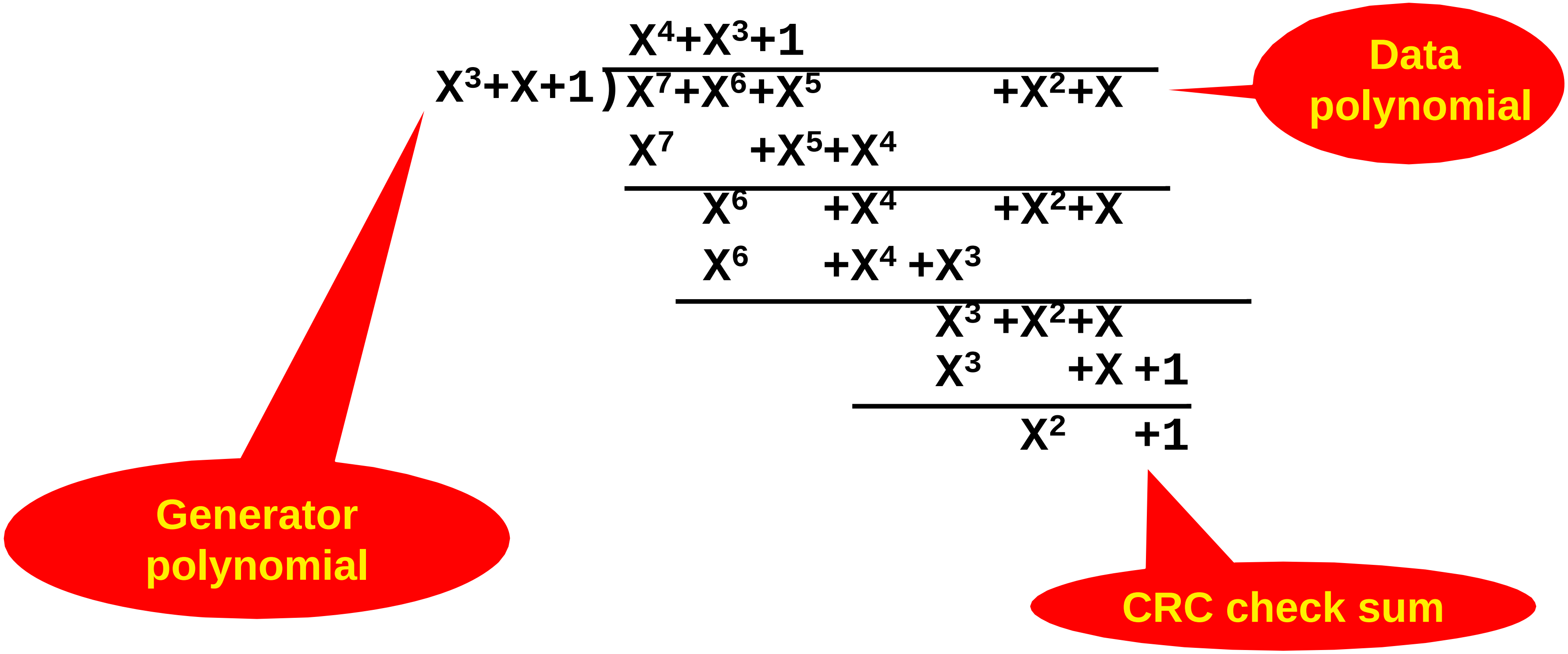
Configuring The CRC Generator Registers

- **Generator Polynomial Example**
  - Polynomial = $x^{16} + x^{15} + x^{3} + x^{2} + x + 1$
  - Polynomial Length (PLEN) is the highest order term minus one (i.e. 15)
  - CRCXOR Register is configured by setting a 1 for polynomial terms between [15:1] (i.e. 0x800E)

x^{15}	x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^{9}	x^{8}	x^{7}	x^{6}	x^{5}	x^{4}	x^{3}	x^{2}	x^{1}	N/A
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**

Programmable CRC Generator

- For example,
- Assume data polynomial to be $x^7+x^6+x^5+x^2+x$
- Assume a generator polynomial to be x^3+x+1



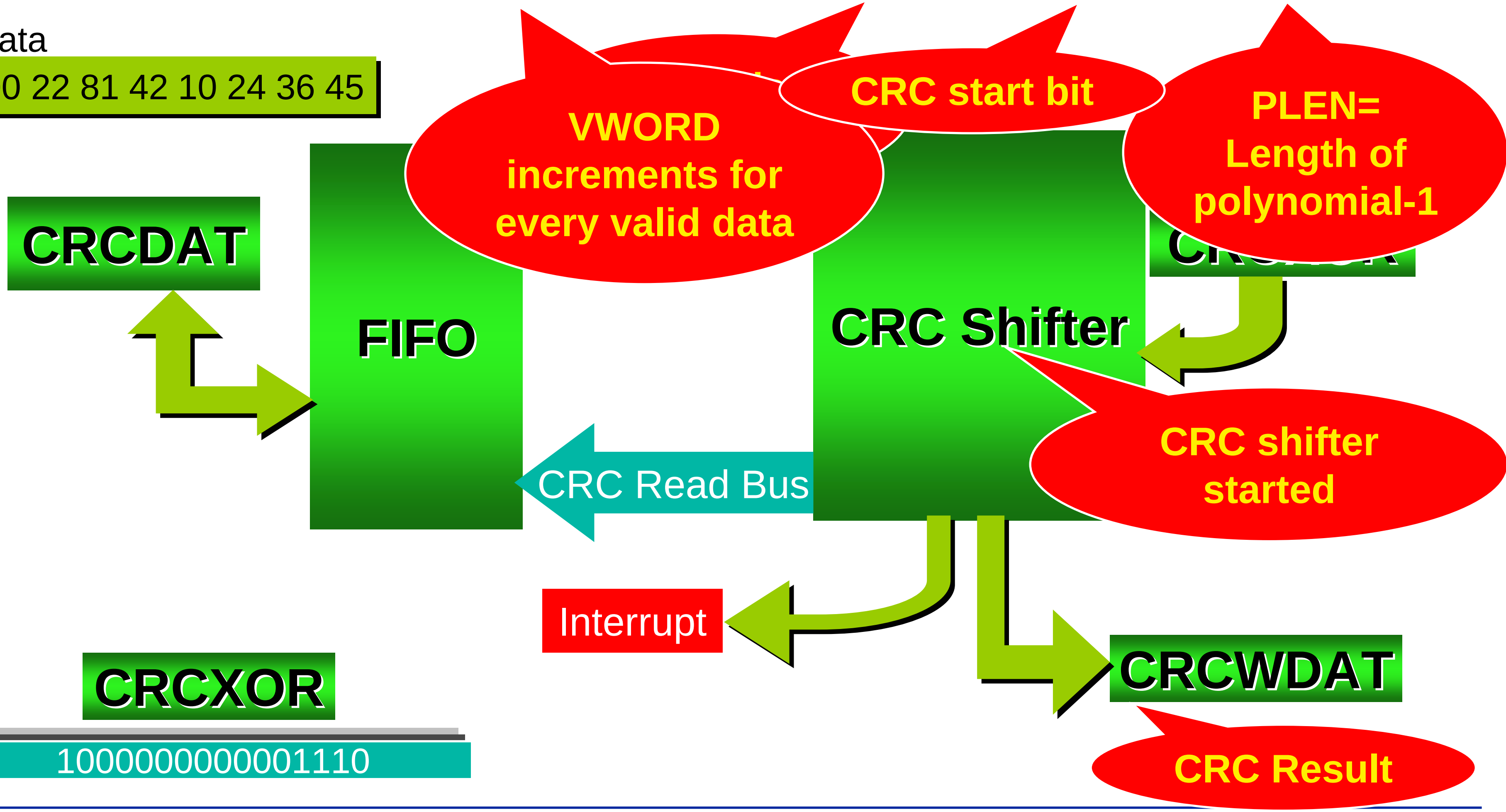
$$
\begin{array}{r|cccccccc}
 & X^4 & +X^3 & & & & & +1 & \\
\hline
X^3+X+1 & X^7 & +X^6 & +X^5 & & & & +X^2 & +X \\
 & X^7 & & +X^5 & +X^4 & & & & \\
\hline
 & & X^6 & & +X^4 & & & +X^2 & +X \\
 & & X^6 & & +X^4 & +X^3 & & & \\
\hline
 & & & & & X^3 & & +X^2 & +X \\
 & & & & & X^3 & & & +X & +1 \\
\hline
 & & & & & & & X^2 & & +1
\end{array}
$$

CRCCON

-	-	CSIDL	VWORD<12:8>	CRCFUL	CRCMPT	-	CRCGO	PLEN<3:0>
-	-	0	00000	0	0	-	0	0000

Data

00 22 81 42 10 24 36 45



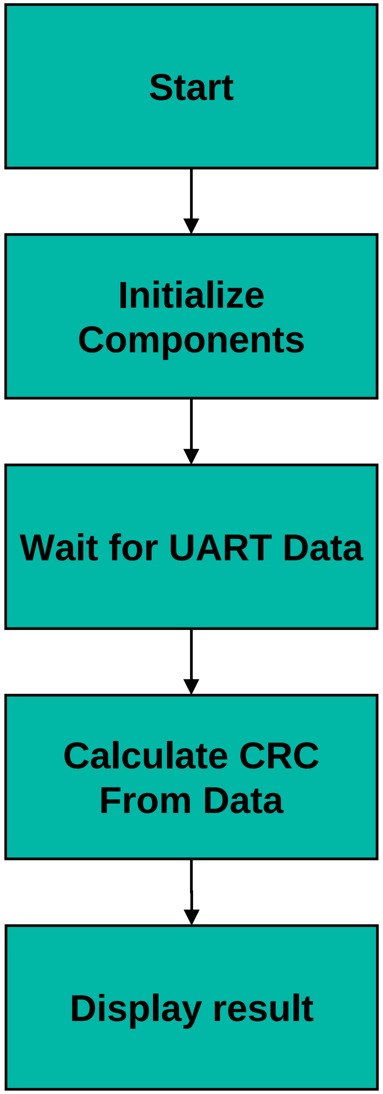
CRCXOR

1000000000001110

Lab 3

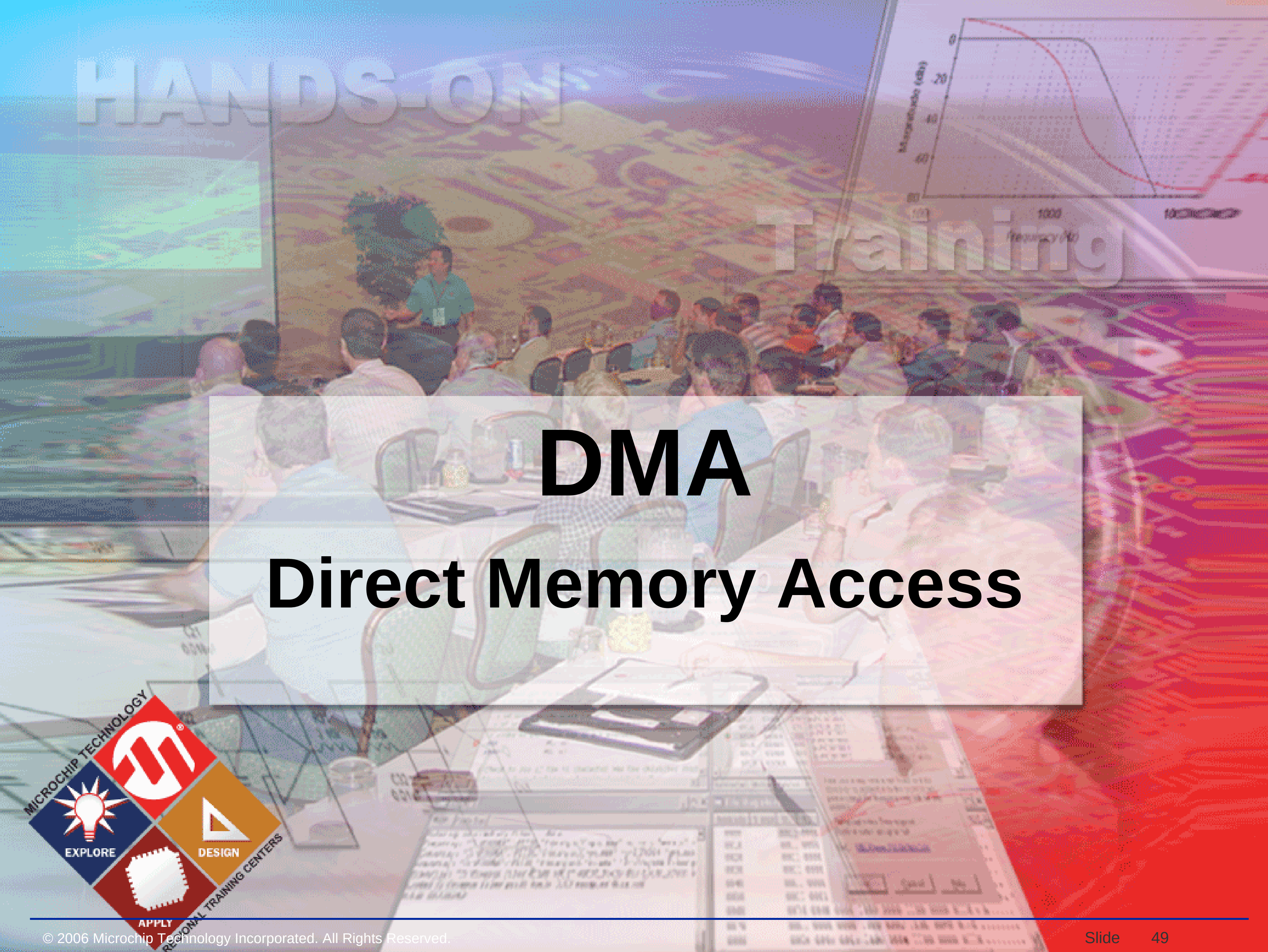
Programmable CRC Generator

Lab 3 CRC


Start
Initialize Components
Wait for UART Data
Calculate CRC From Data
Display result

Lab 3 – CRC

- **Goals**
  - Learn about the Programmable CRC Generator
  - Do some more coding for PIC24
- **Three parts to this lab**
  - Add code to configure the CRC generator for the polynomial $\mathbf{x^{16} + x^{15} + x^{2} + 1}$ (main.c)
  - Add code to start CRC generator (main.c)
  - Check results of CRC generation and verification on LCD Display

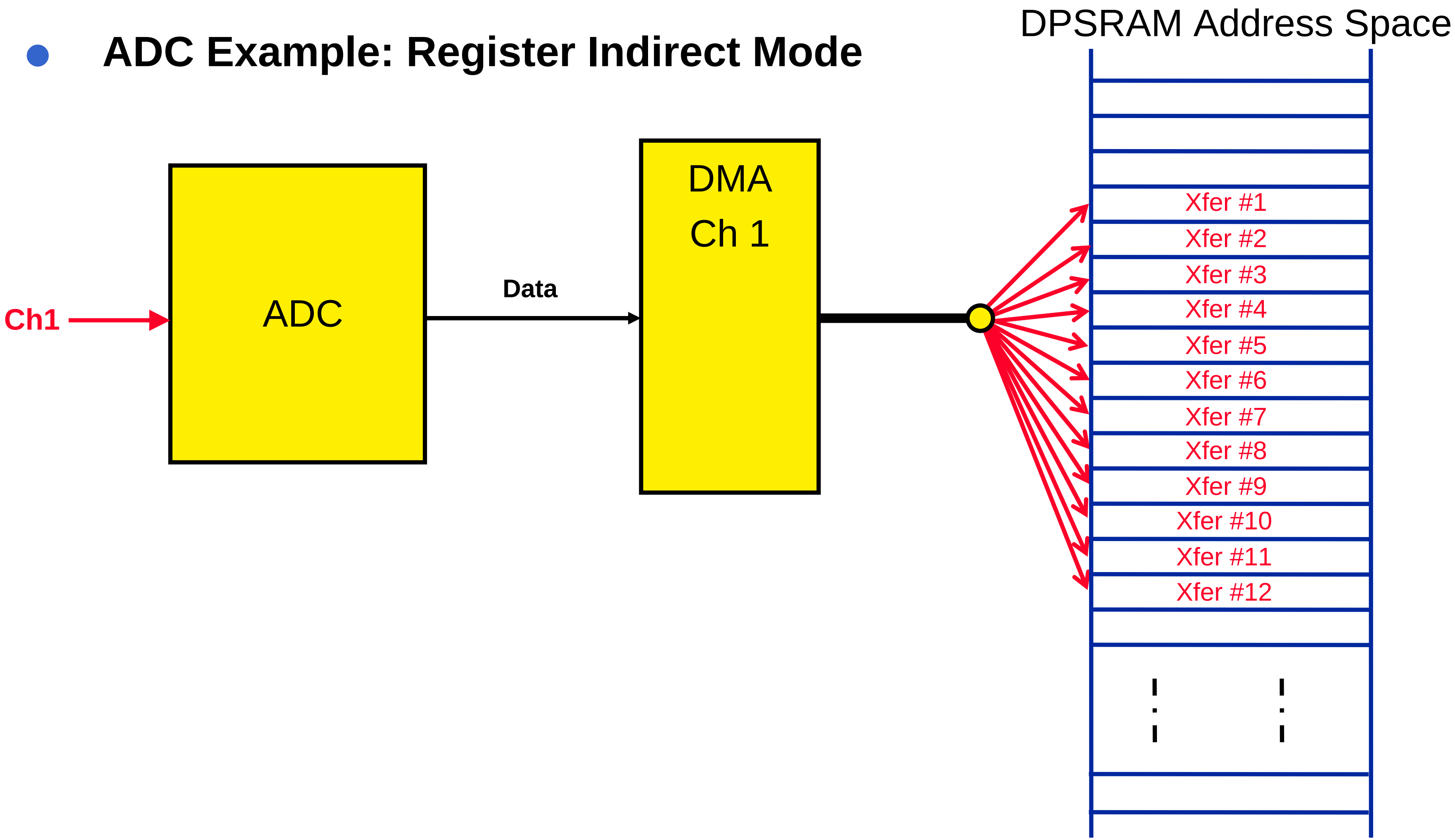


DMA

Direct Memory Access

DMA Example: ADC

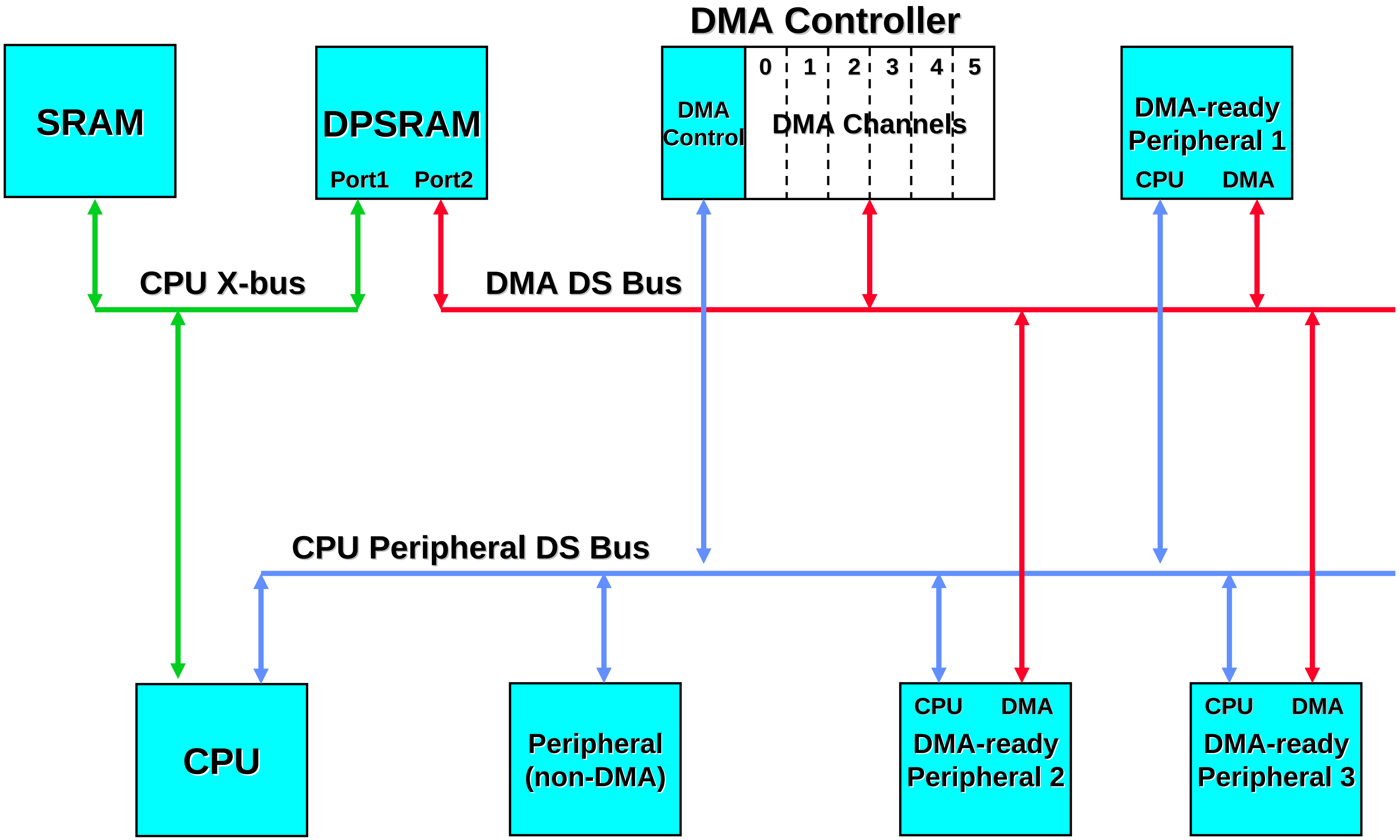
- **ADC Example: Register Indirect Mode**



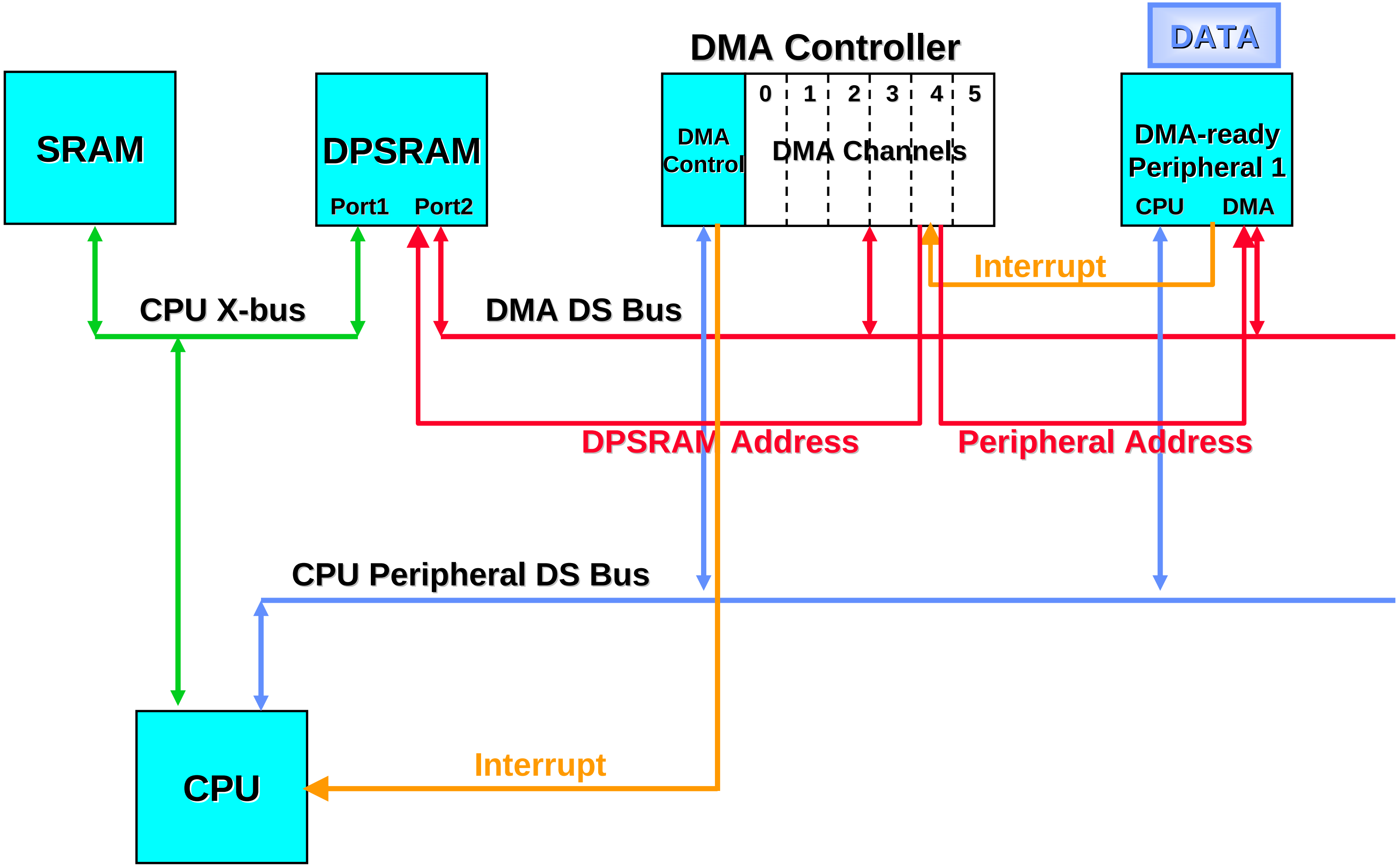
DMA Features

- **8 DMA channels**
- **Register indirect with post increment addressing mode**
- **Peripheral indirect addressing mode**
  - peripheral generates destination address
- **CPU interrupt after half or full block transfer complete**
- **Byte or word transfers**
- **Fixed priority channel arbitration**
- **Manual or Automatic transfers**
- **One-shot or Auto-Repeat transfers**
- **‘Ping-pong’ mode**
  - automatic switch between two buffers
- **DMA request for each channel can be selected from any supported interrupt sources**
- **Debug support features**

DMA Controller Block Diagram



DMA Controller Operation



DMA Support

- **Supported 16-bit Families**
  - PIC24H
  - dsPIC33F
- **Supported 16-bit Peripherals**
  - ECAN Module
  - Data Converter Interface (DCI)
  - 10-bit/12-bit A/D Converter
  - Serial Peripheral Interface (SPI)
  - UART
  - Input Capture
  - Output Compare
- **DMA Request Support Only**
  - Timers
  - External Interrupts

Enabling DMA Operation

1. **Associate DMA channel with peripheral**
2. **Configure DMA capable peripheral**
3. **Initialize DPSRAM data start addresses**
4. **Initialize DMA transfer count**
5. **Select appropriate addressing and operating modes**

Step 1: Associate DMA and Peripheral

- Associate peripheral IRQ with DMA via DMAxREQ
- Provide peripheral read/write address via DMAxPAD

Example: Associate DMA Channel 0 and 1 with UART2 Transmitter and Receiver respectively

```
DMA0REQbits.IRQSEL = 0x1F;
DMA0PAD = (volatile unsigned int) &U2TXREG;

DMA0REQbits.IRQSEL = 0x1E;
DMA0PAD = (volatile unsigned int) &U2RXREG;
```

Step 2: Configure DMA-Ready Peripheral

- **Configure peripherals to generate interrupt for every transfer (if applicable)**

Example: Configure UART2 to generate DMA request after each Tx and Rx character

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

- **Enable Error interrupts (if applicable)**

Example: Enable and process UART2 error interrupts

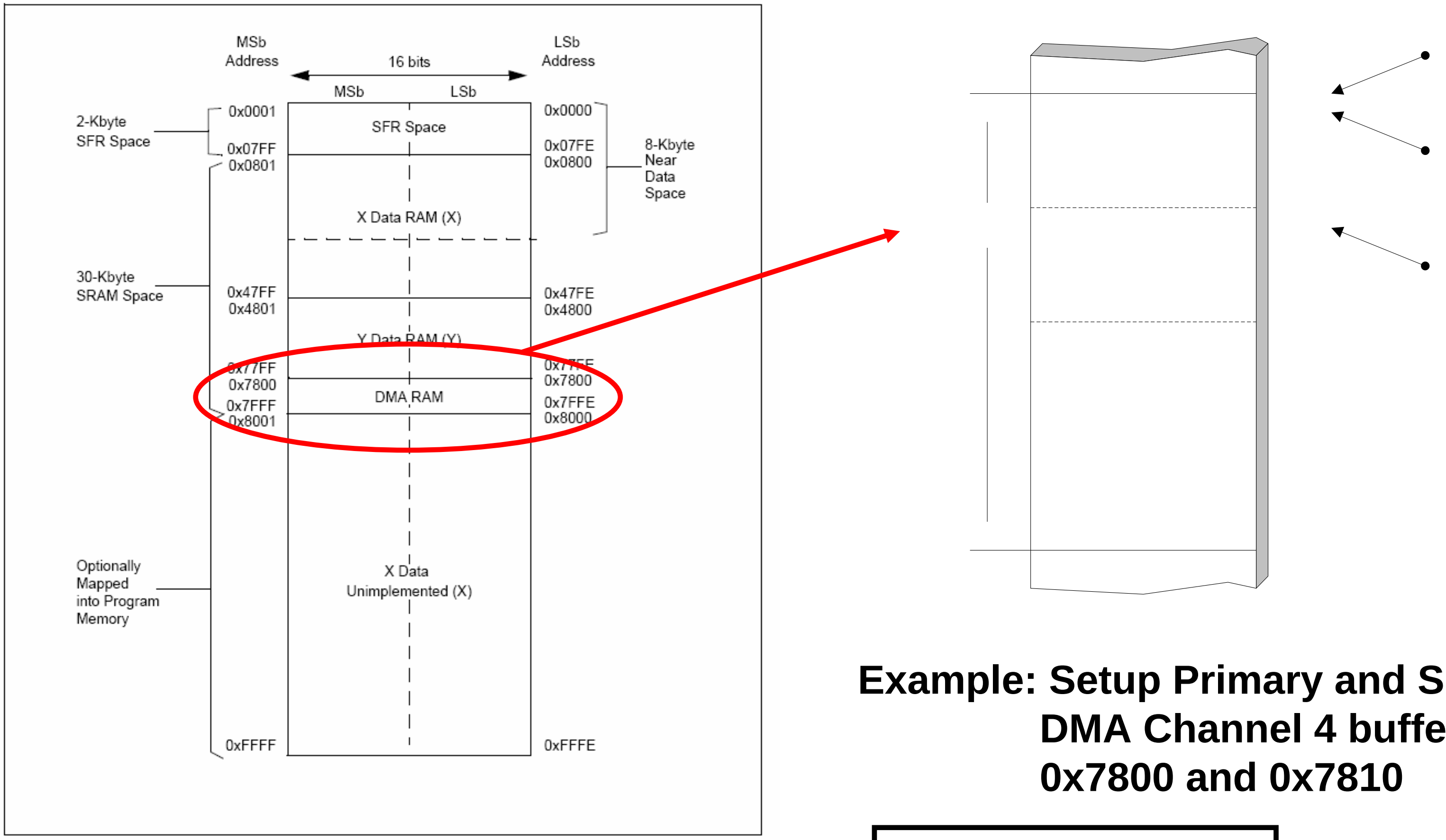
```
IEC4bits.U2EIE = 0;               // Enable UART2 Error Interrupt

void __attribute__((__interrupt__)) _U2ErrInterrupt(void)
{
      /* Process UART 2 Error Condition here */

      IFS4bits.U2EIF = 0; // Clear the UART2 Error Interrupt Flag
}
```

Step 3: Initialize DPSRAM data start addresses

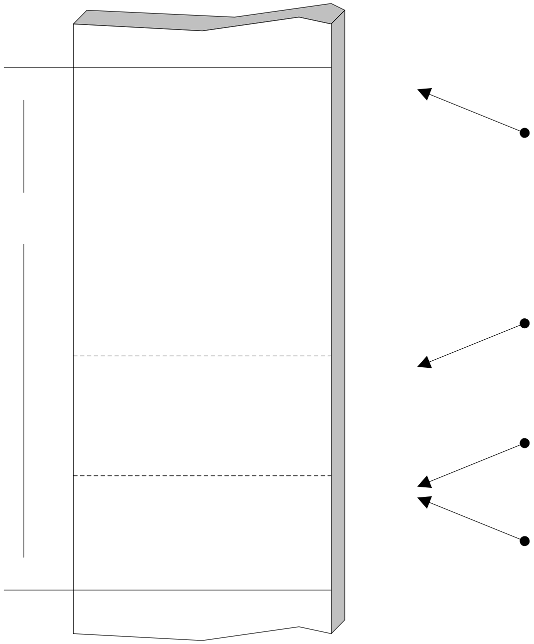
FIGURE 3-5: DATA MEMORY MAP FOR dsPIC33F DEVICES WITH 30 KBs RAM



Example: Setup Primary and Secondary DMA Channel 4 buffers at 0x7800 and 0x7810

```
DMA4STA = 0x0000;
DMA4STB = 0x0010;
```

Step 3: MPLAB Support for DMA



- Use __attribute__(space(dma)) and __builtin_dmaoffset()

Example: Allocate two buffers 8 words each in DMA memory for DMA Channel 1; Associate DMA Channel 0 with one of the buffers as well

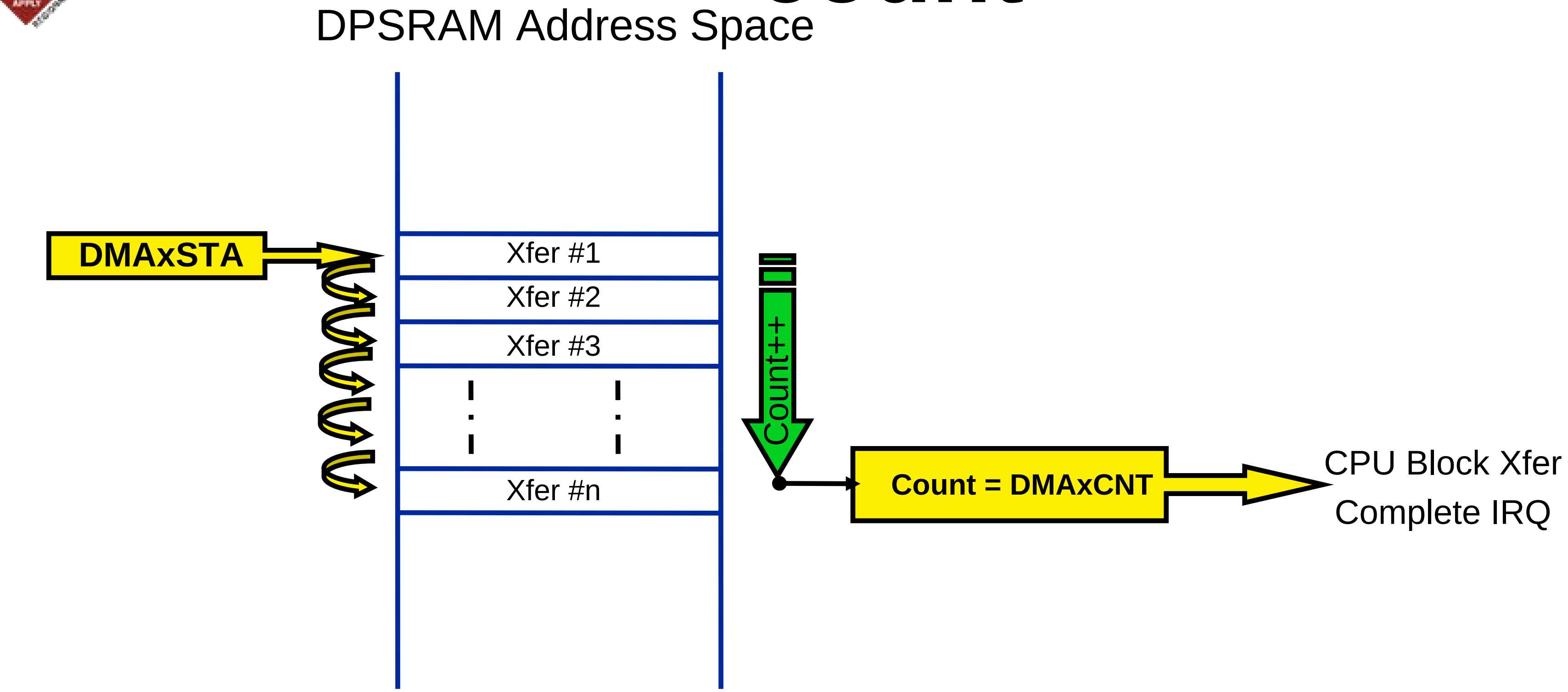
```
unsigned int BufferA[8] __attribute__(space(dma));
unsigned int BufferB[8] __attribute__(space(dma));

DMA1STA = __builtin_dmaoffset(BufferA);
DMA1STB = __builtin_dmaoffset(BufferB);

DMA0STA = __builtin_dmaoffset(BufferA);
```

DMA RAM

Step 4: Initialize DMA transfer count



Example: Setup DMA Channel 0 and 1 to handle 8 DMA requests

```
DMA0CNT = 7;   // 8 DMA Requests
DMA1CNT = 7;   // 8 DMA Requests
```

Step 5: Select appropriate DMA addressing and operating modes

- Word or byte size data transfers
- Peripheral to DPSRAM, or DPSRAM to peripheral transfers
- Post-increment or static DPSRAM addressing
- One-shot or continuous block transfers
- Interrupt the CPU when the transfer is half or fully complete
- Auto switch between two start addresses offsets (DMAxSTA or DMAxSTB) after each transfer complete (‘ping-pong’ mode)
- Peripheral indirect addressing
- Null data write mode
- Manual Transfer mode

DMA Modes: Size and Direction

- Word or byte size data transfers
- Peripheral to DPSRAM, or DPSRAM to peripheral transfers

Example: Setup DMA Channel 0 and Channel 1 to transfer words to and from peripheral respectively

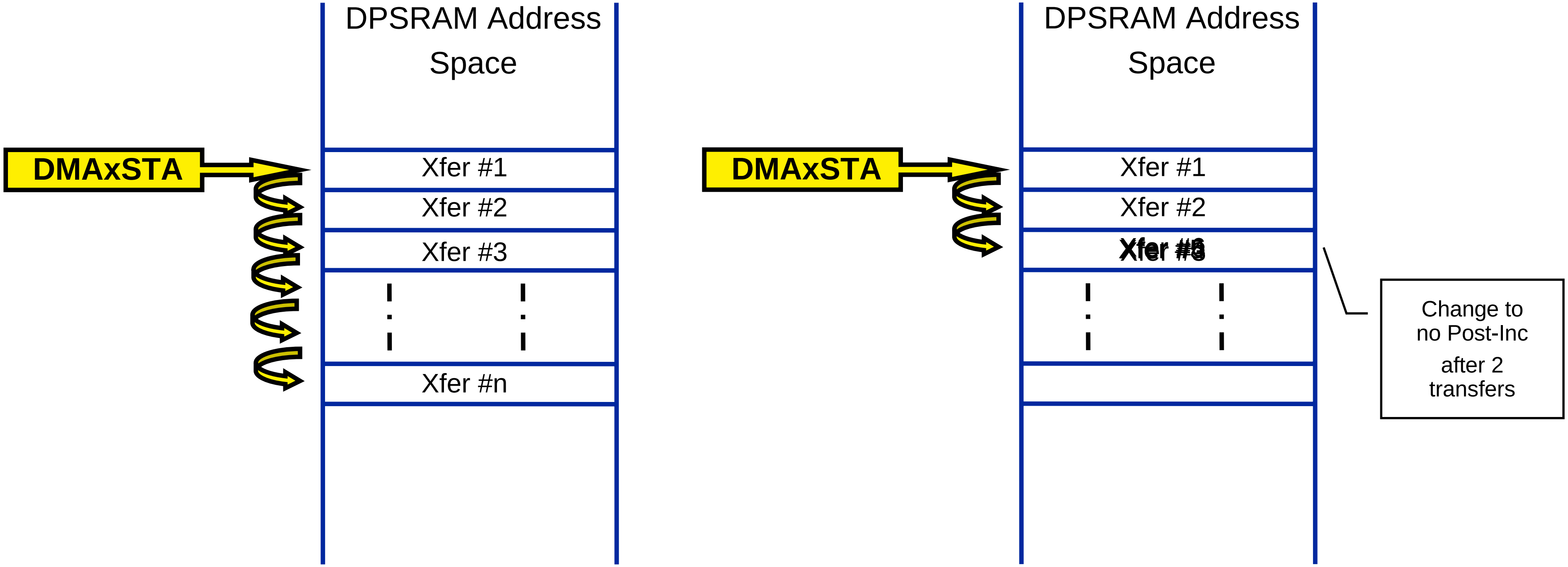
```
DMA0CONbits.SIZE = 0;   // Word transfers
DMA0CONbits.DIR  = 1;   // RAM-to-Peripheral direction

DMA1CONbits.SIZE = 0;   // Word transfers
DMA1CONbits.DIR  = 0;   // Peripheral-to-RAM direction
```

DMA Modes: Register Indirect Addressing

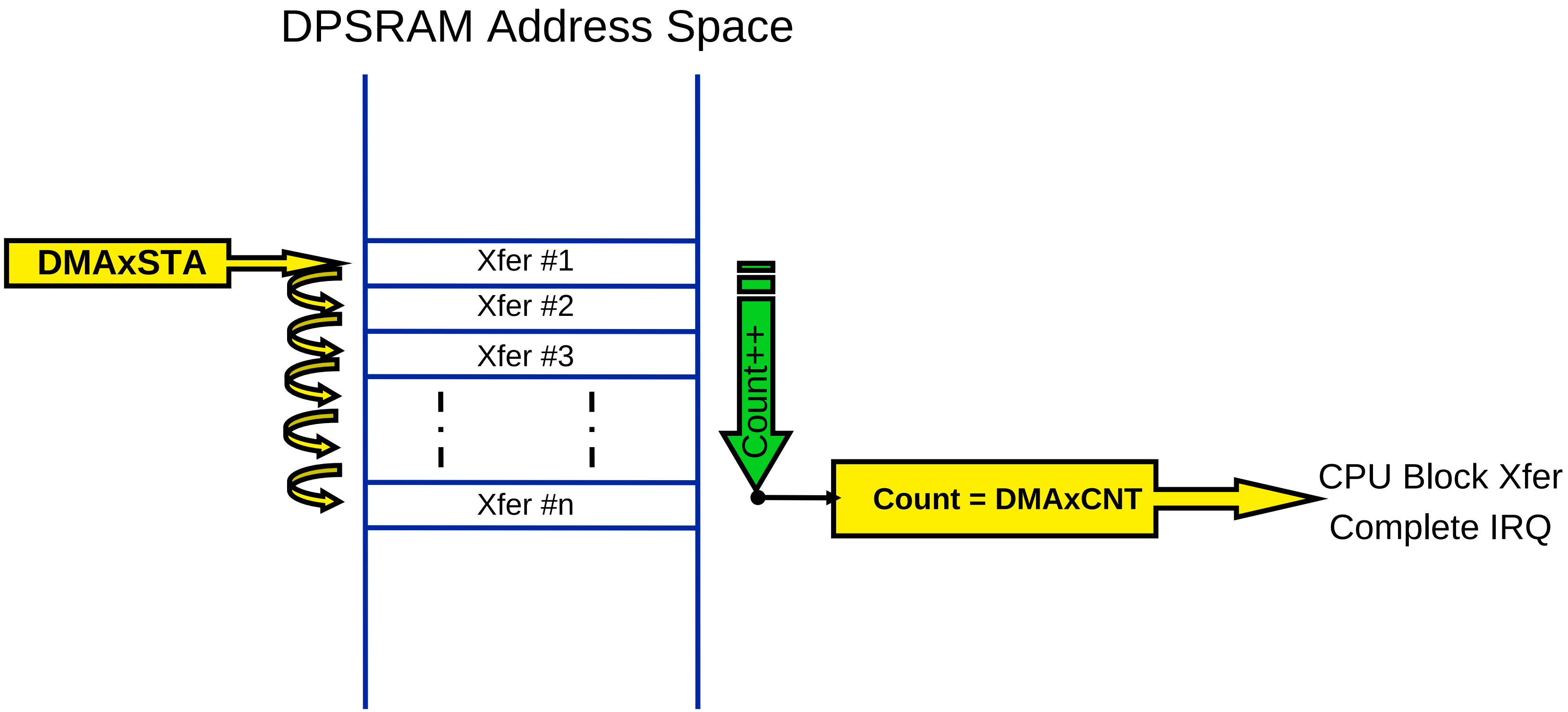
Post-Increment

Without Post-Increment



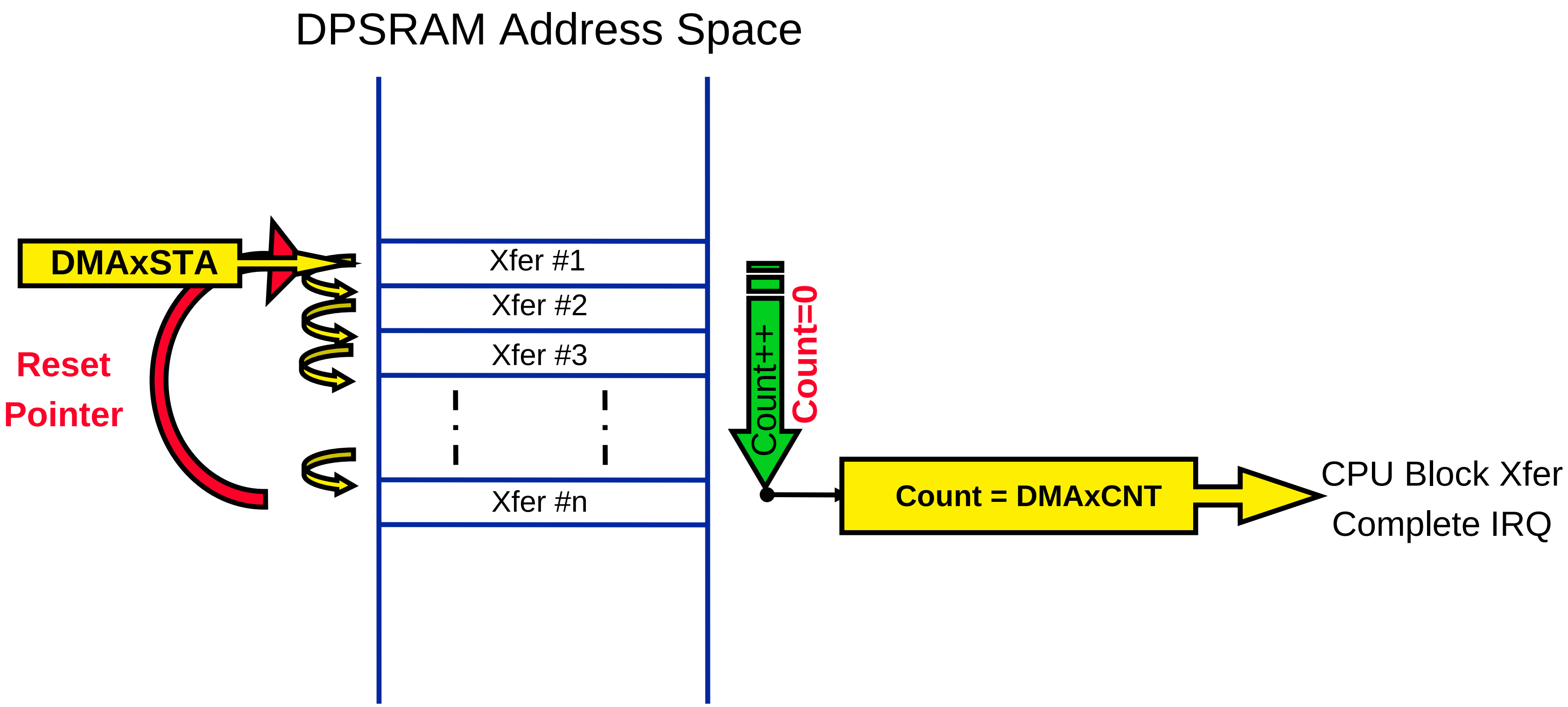
Change to no Post-Inc after 2 transfers

DMA Modes: One-Shot



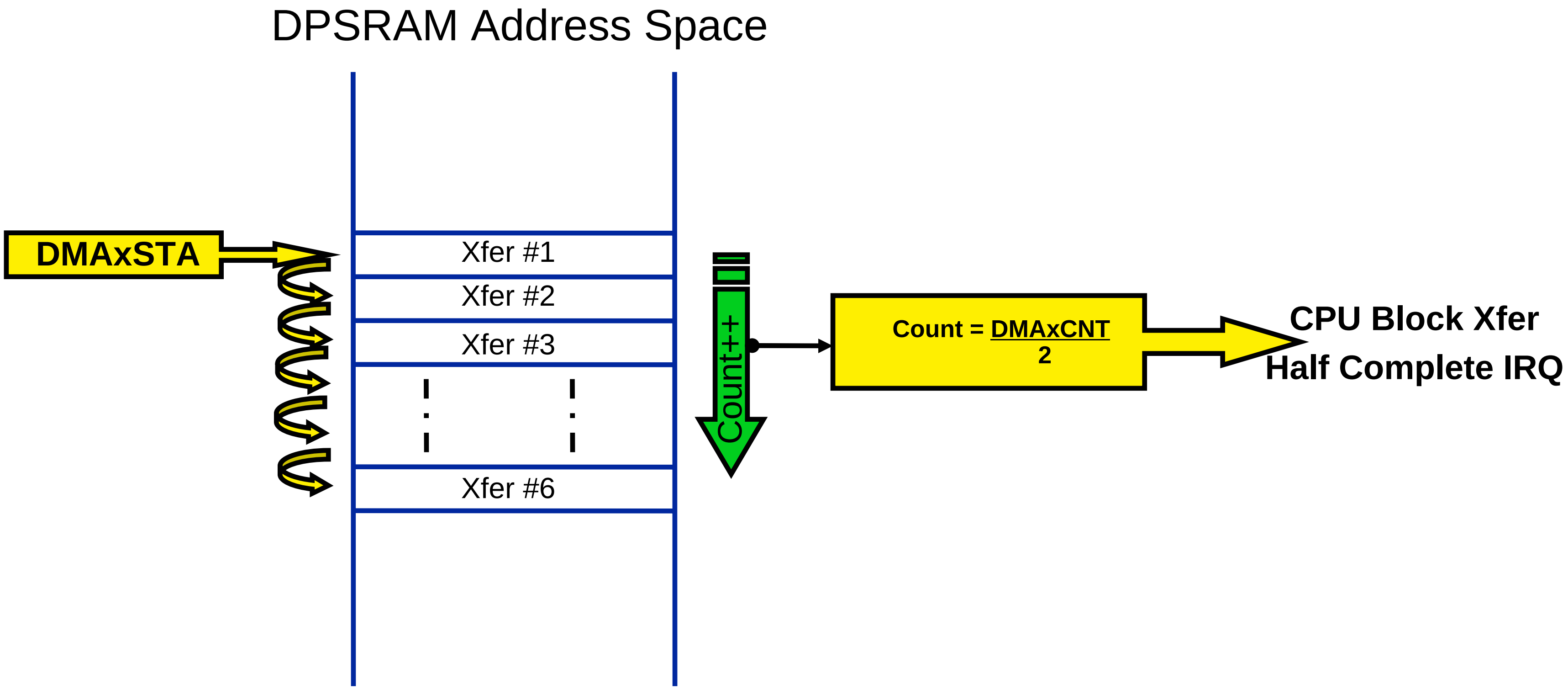
– Move 1 block of data then disable channel

DMA Modes: Continuous



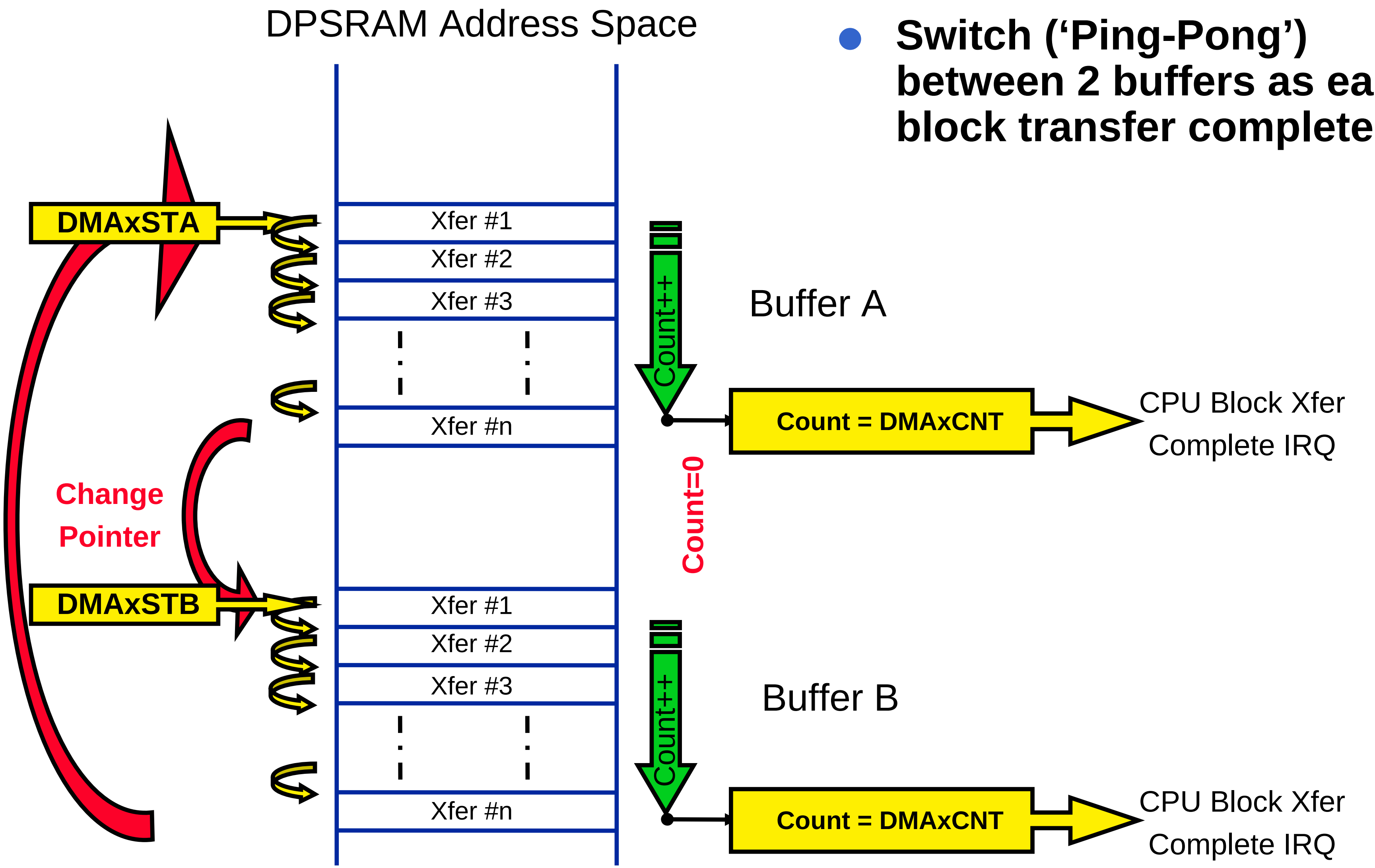
– Move a block of data then automatically configure channel ready to repeat transfer

DMA Modes: Half or Full Transfer



- Move ½ block of data then issue interrupt
- Continue moving second ½ block of data

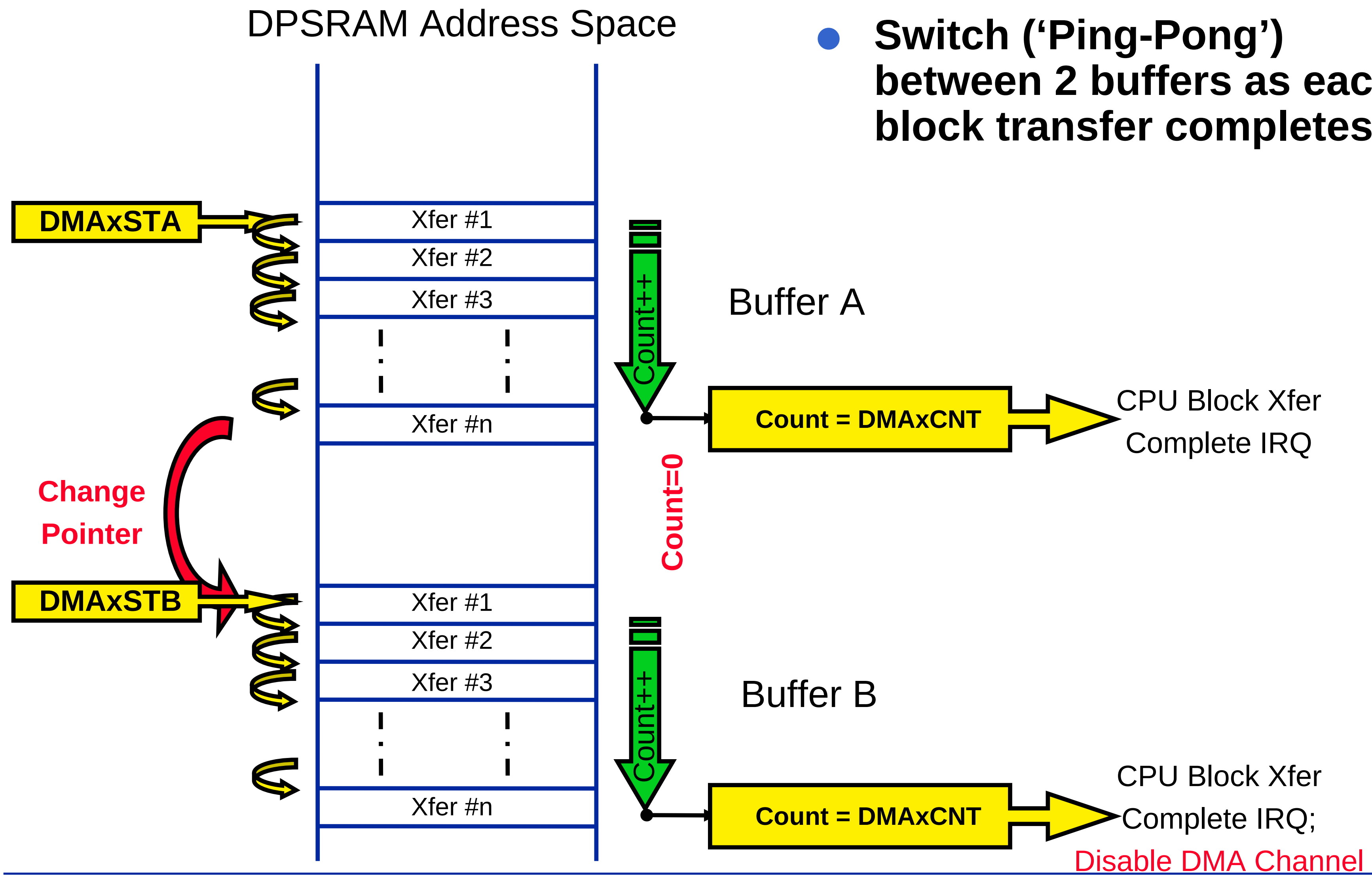
DMA Modes: ‘Ping-Pong’ and Continuous



- **Switch (‘Ping-Pong’) between 2 buffers as each block transfer completes**

DMA Modes: 'Ping-Pong' and One-Shot

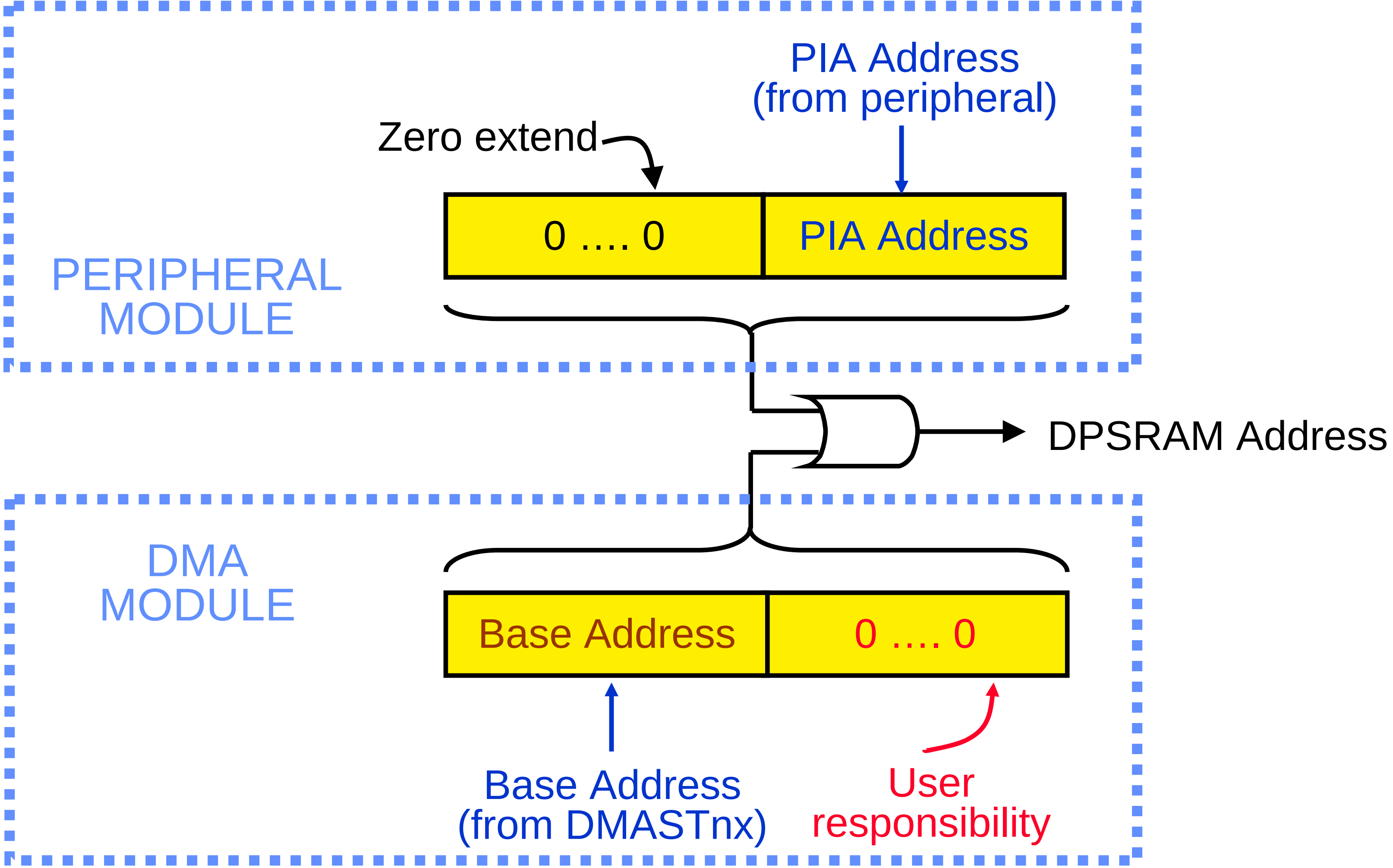
- **Switch ('Ping-Pong') between 2 buffers as each block transfer completes**



DMA Modes: Peripheral Indirect

- **Least significant bits of address supplied by DMA request peripheral**
- **Allows 'scatter/gather' addressing schemes to be tailored to the needs of each peripheral**

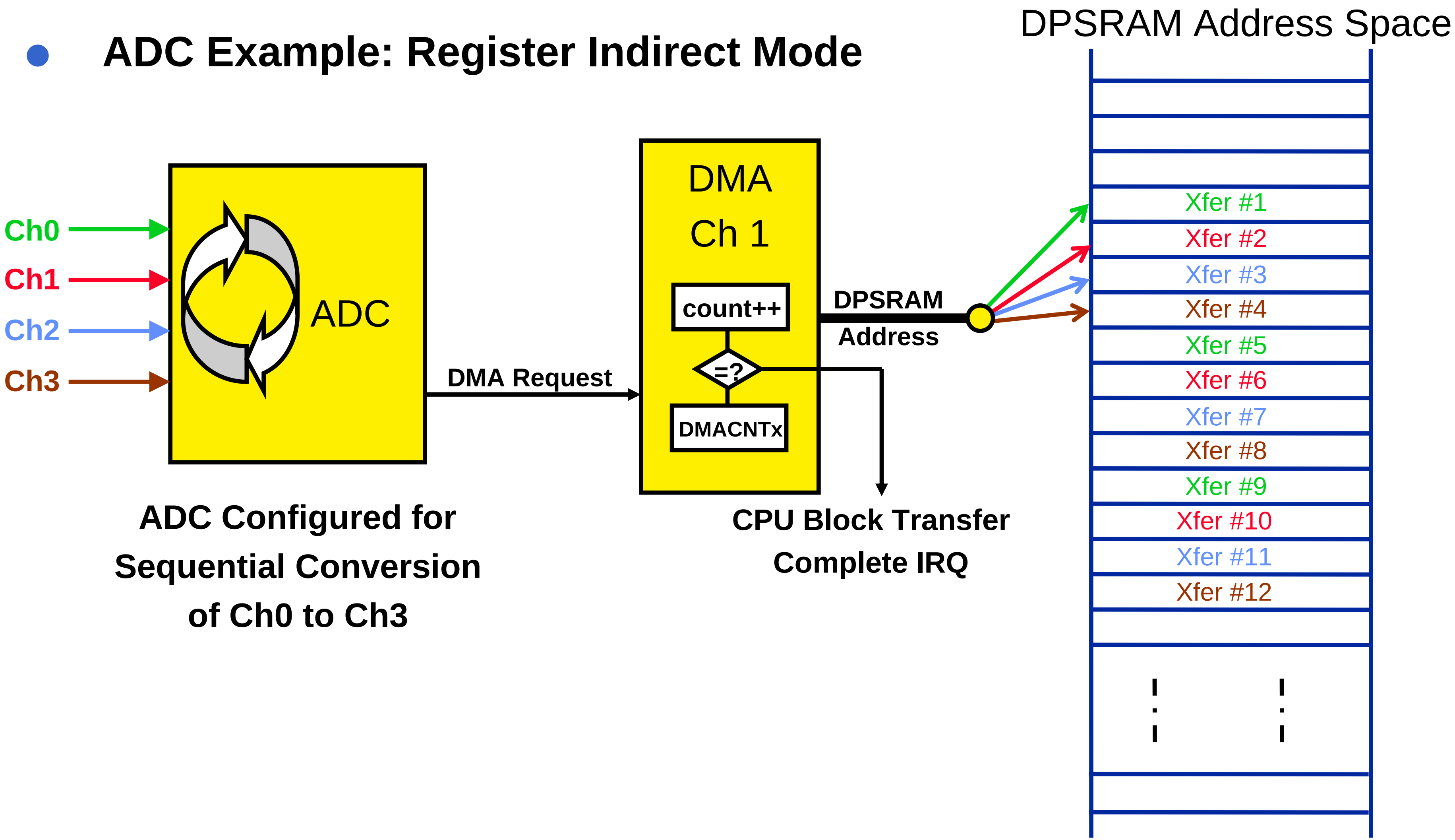
DMA Modes: Peripheral Indirect



- PIA Mode is also compatible with Auto-Repeat and ‘Ping-Pong’ modes

DMA Modes: Peripheral Indirect

- **ADC Example: Register Indirect Mode**

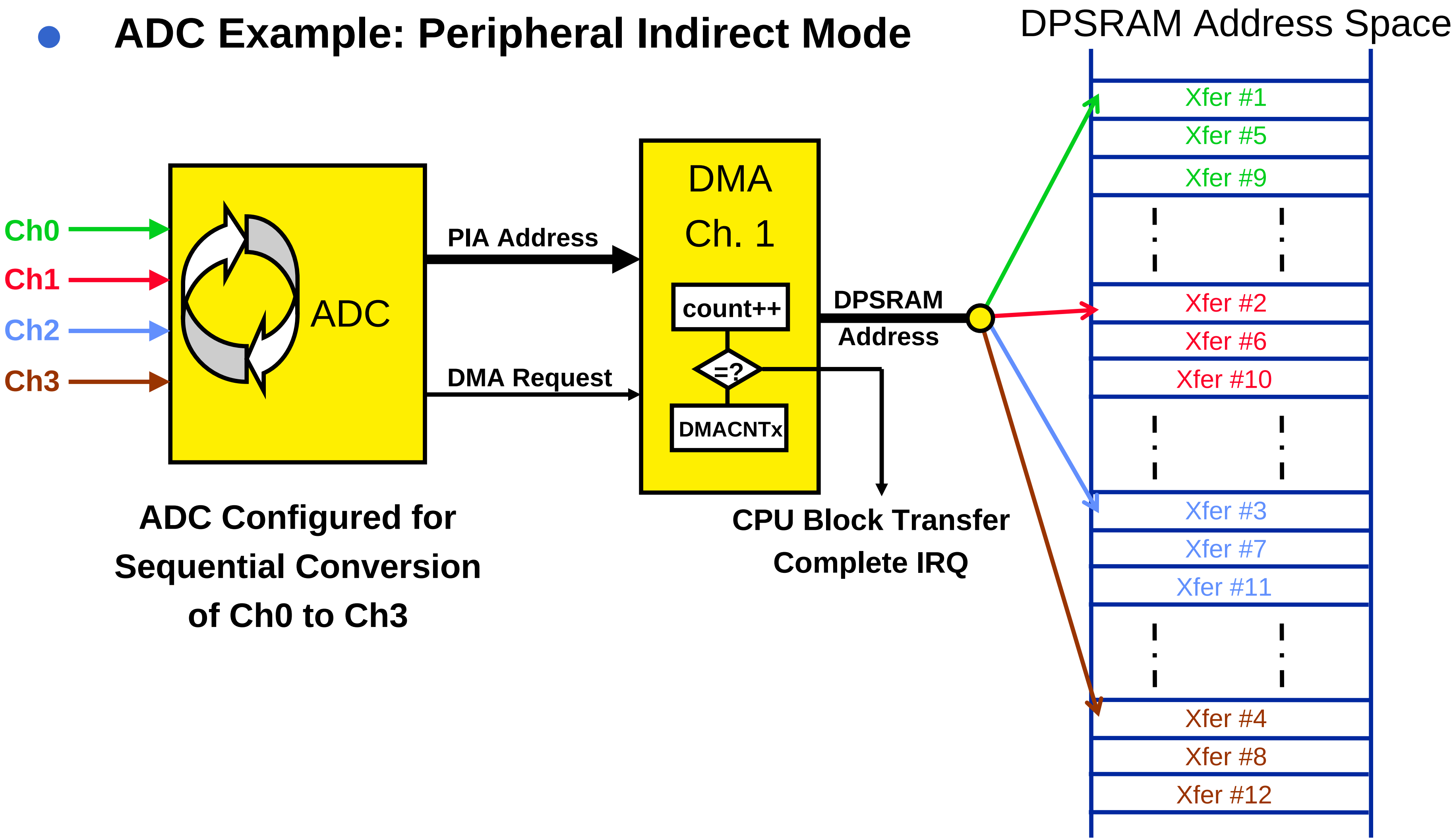


ADC Configured for Sequential Conversion of Ch0 to Ch3

CPU Block Transfer Complete IRQ

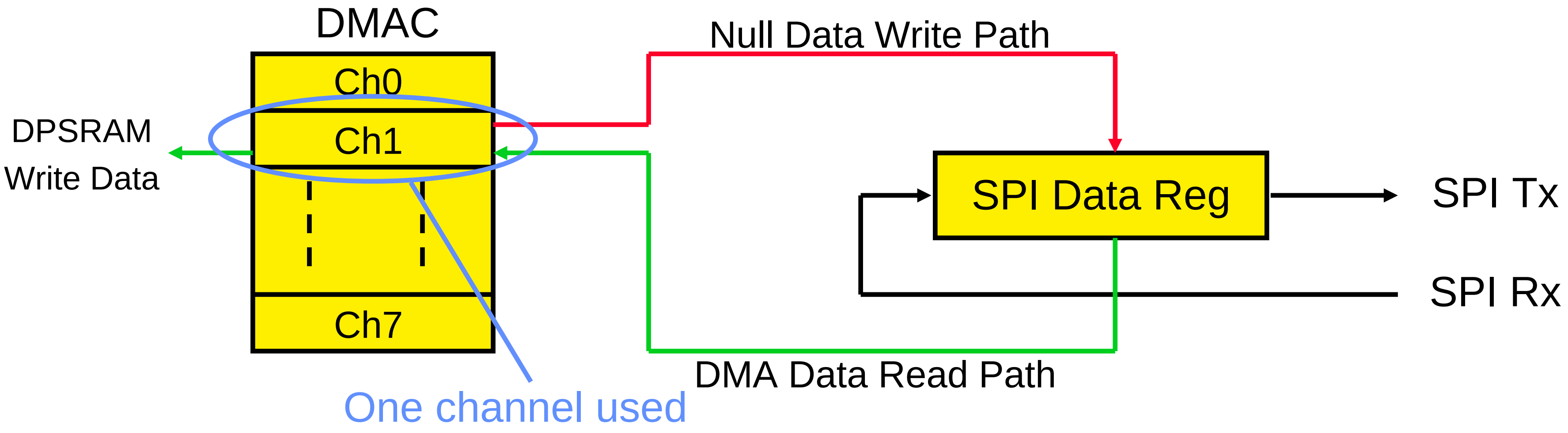
DMA Modes: Peripheral Indirect

- **ADC Example: Peripheral Indirect Mode**



DMA Modes: Null Data Write

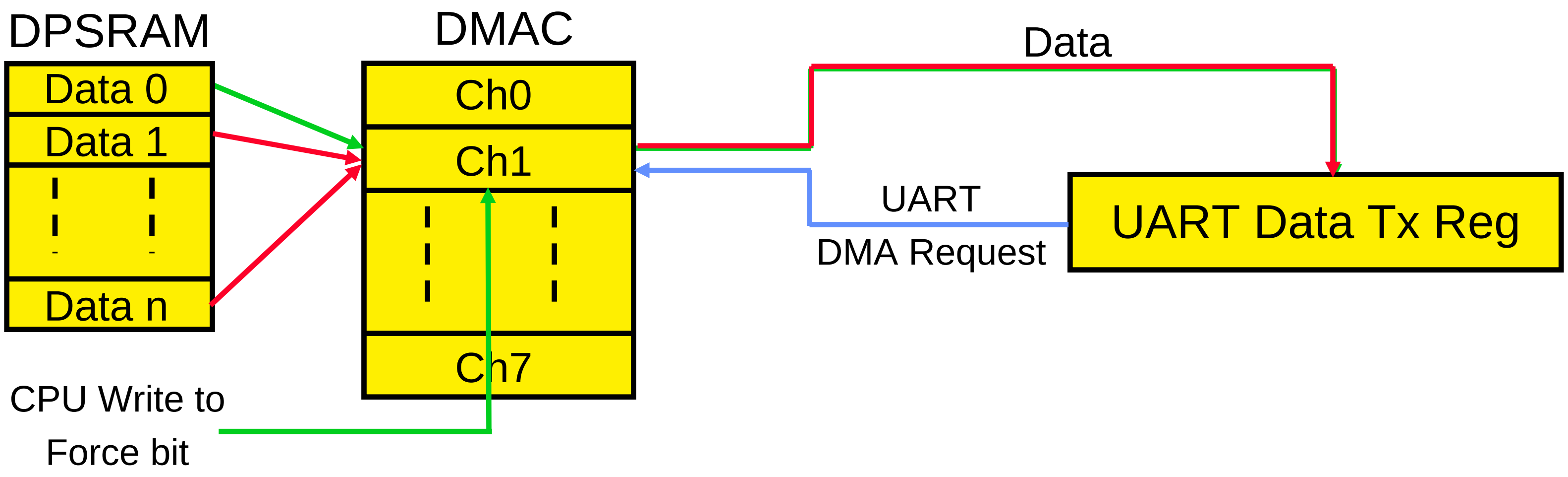
- **SPI has a single peripheral read/write data address**
  - Requires that data be transmitted (written) in order for external data to be received (read)
  - If only data reception required, “null” (zero) write necessary



- **“Null write” mode simplifies DMAC operation with SPI**
  - Automatically writes null (zero) data to peripheral data register after each DMA read
  - Avoids wasting another channel for null write

DMA Modes: Manual Transfer

- **Provides a means to start a DMA transfer using software**
  - Setting the FORCE bit in the selected DMA channel mimics a DMA request
- **Useful for sending the first element from a block of data to a serial peripheral (e.g. UART)**
  - Starts the sequence of DMA request to load data into a peripheral
  - When peripheral data buffer is empty (data sent), peripheral will issue a DMA request for the next data element



DMA Modes: Examples

Example: Configure DMA Channel 0 for: One-Shot,
Post-Increment,
RAM-to-Peripheral,
Single Buffer

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

Example: Configure DMA Channel 1 for: Continuous,
Post-Increment,
Peripheral-to-RAM
Ping-Pong

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

DMA Interrupts

- **Transfer Complete Interrupt**
- **Write Collision Interrupt (DMA Trap)**
  - Should never happen but if they do, handled robustly
  - CPU will win; DMAC write ignored
  - Cause DMA Fault trap and set channel write collision flag

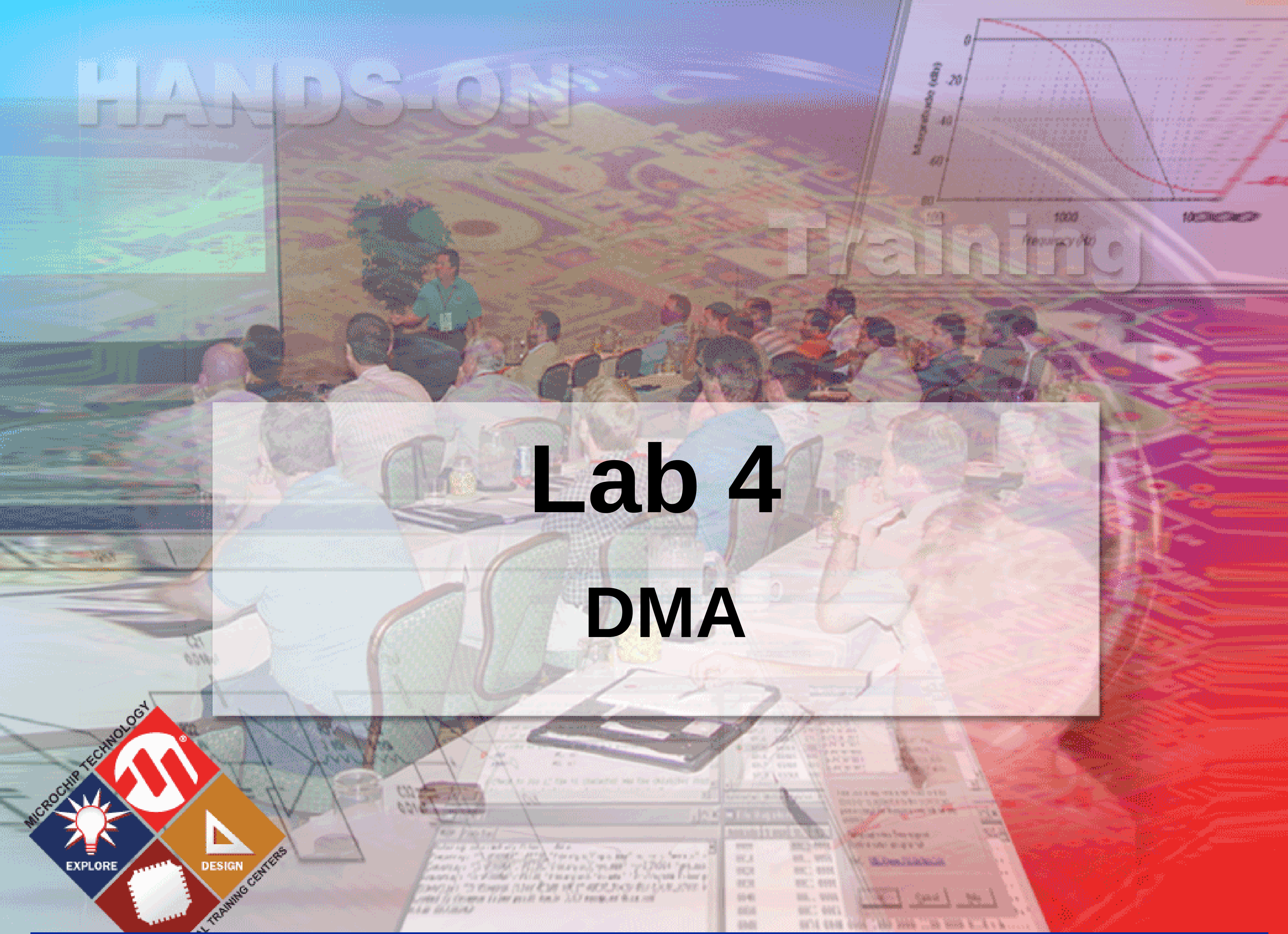
Example: Enable and process DMA Channel 0 and 1 interrupts

```
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

void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
      /* Process DMA Channel 0 interrupt here */
      IFS0bits.DMA0IF = 0;        // Clear the DMA0 Interrupt Flag
}
void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
      /* Process DMA Channel 1 interrupt here */
      IFS0bits.DMA1IF = 0;        // Clear the DMA1 Interrupt Flag
}
```

DMA Controller
Debug Support

- **2 DMA debug assist registers included**
  - DSADR<15:0> : Captures the DPSRAM address of the most recent DMA transfer
  - DMACS1 :
    - **LSTCH<2:0> : Captures the ID of the most recently active DMA channel**
    - **PPSTx : ‘Ping-Pong’ mode status bits, one per channel.**
      Indicates which buffer is active (A or B)



Lab 4

DMA

Lab 4 – Using DMA

- **Goals**
  - Learn DMA module
- **Lab**
  - 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 – PIM Swap

- **Lab 4 uses the dsPIC33FJ256GP710**
- **To swap the processor:**
  - Disconnect Explorer 16 power and ICD2 connections
  - Remove PIC24FJ128GA010 PIM
  - Place dsPIC33FJ256GP710 PIM on board, making sure to properly align the notched corner
  - Reconnect Power and ICD2

Summary

- We learned how to use some of the new peripherals onboard the 16-bit devices
- We used the Microchip Development Tools Suite for developing with the 16-bit PICs
- We became familiar with some of the PIC24 and dsPIC33 documentation

References

- PIC24 & dsPIC33 Datasheet
- Explorer 16 User’s Guide
- MPLAB® IDE
- C30 Compiler
- ICD2 In Circuit Debugger

All Done!

Thank you all for attending

Please remember the evaluation sheets

Trademarks