

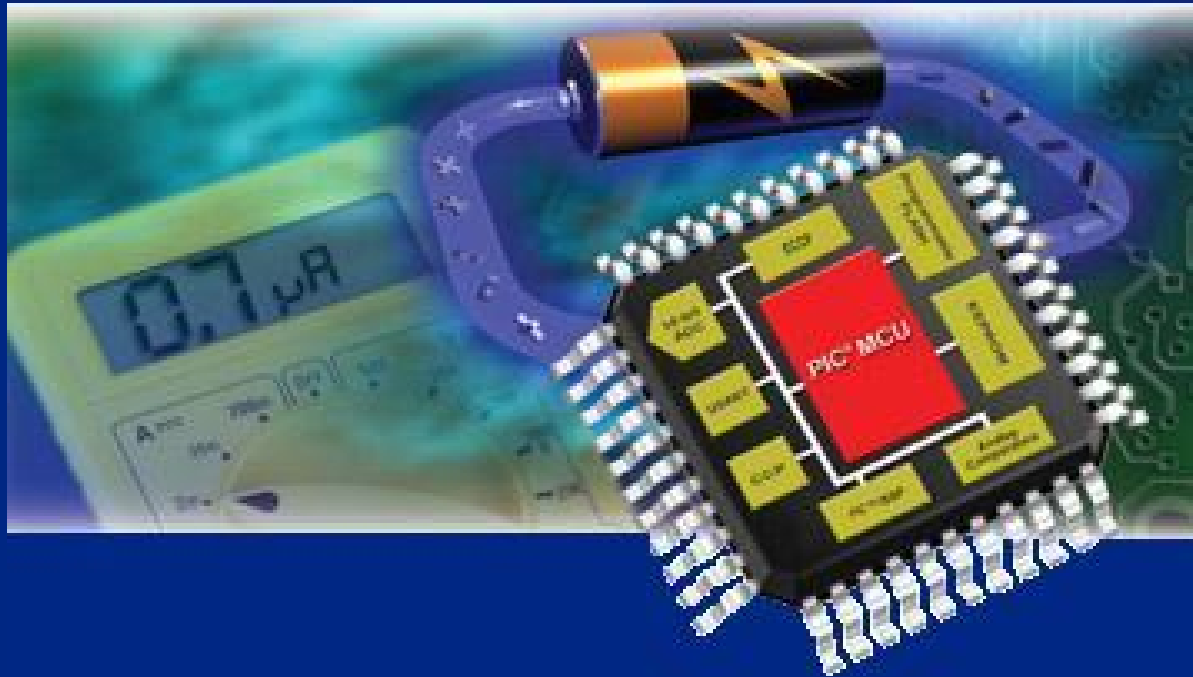


# 819 NNW

## What's New with the PIC18 Architecture Including nanoWatt Technology Features and the Extended Instruction Set



**MICROCHIP  
M A S T E R S**





# What's New on PIC18

## | **Class Goals:**

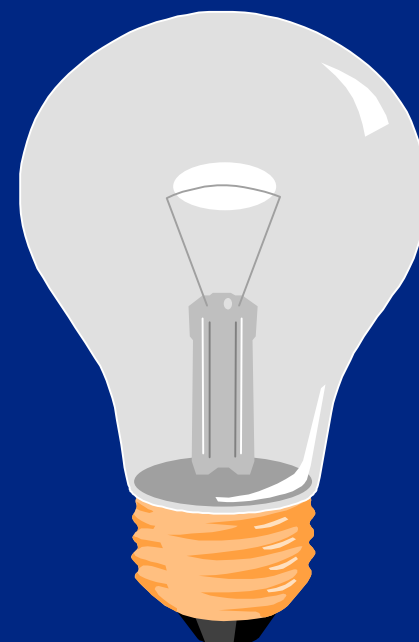
| *At the end this class, you will...*

- | be familiar with PIC18's revised & new modules
- | have an understanding of clock switching
- | know the "Latest and Greatest"
- | know how to use the extended instruction set
- | be able to apply nanoWatt technology to your application



# Agenda

- | ***nanoWatt Technology review***
- | PIC18 Features
- | Clock System
- | Power Managed Modes
- | PIC18 Devices
- | Extended Instruction Set
- | Summary





# What is nanoWatt Technology?

Microchip continues to make improvements over its existing portfolio:

**Redesigning Legacy  
Designing New Modules**



**Low Power & Greater  
Application flexibility**

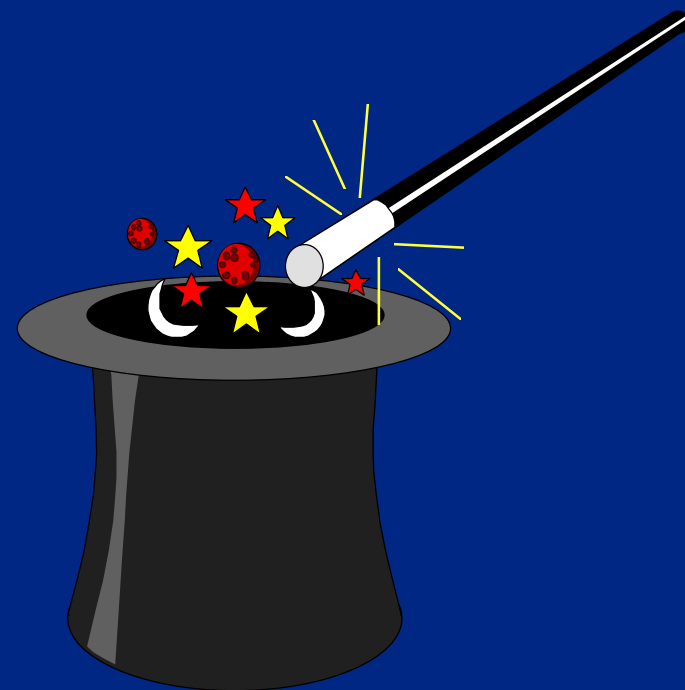
- Low Power Timer1
- Two-Speed Start-up
- Fail Safe Clock Monitor
- HLVD
- LCD
- USB
- Internal RC Oscillator

- EUSART
- Variable Boot Block
- Power Managed Modes
- LIN
- Motor Control
- BOR
- ECCP



# Agenda

- | nanoWatt Technology review
- | **PIC18 Features**
- | Clock System
- | Power Managed Modes
- | PIC18 Devices
- | Extended Instruction Set
- | Summary





# PIC18 Features: Timer1

## | Low Power Timer1 Oscillator:

- | 3ua vs 30ua
- | Oscillator amplitude is regulated
- | Constant current across VDD and Temperature
- | Robust operation
- | Enabled by T1OSCEN (T1CON<3>)
- | Commonly 32.768 kHz for a RTC time base
- | User code is responsible for determining when RTC is ready



# PIC18 Features: 2 - Speed Start-up

## | Two Speed Start-up:

- | Used with crystal based modes only
- | Immediate code execution
- | Automatically switches to Primary
- | Code may go back to SLEEP before Primary is ready
- | Used for:
  - | Start from RESET or wake from SLEEP mode
- | Primary clocks startup delays
  - | Crystal / Resonator oscillator start time
  - | OST delay (counts 1024 oscillator cycles)
  - | PLL delay (additional 2ms delay for HS/PLL)
  - | Power-up Timer (PWRT)



# PIC18 Features: FSCM

## Fail-Safe Clock Monitor:

- | Used to detect a loss of externally-based clocked sources

- | During start-up (Reset or Wake from Sleep)

- | INTRC/INTOSC provides system clocks until Primary clock is ready (similar to 2-speed start)

- | When the primary is ready, an automatic clock switch selects the primary clock

- | Dedicated interrupt: OSCFIF

- | 31 kHz INTRC clock source





# PIC18 Features: HLVD

## High/Low Voltage Detect

- Programmable Voltage trip point
- Programmable direction of change
- VDIRMAG** bit: indicates voltage direction

REGISTER 22-1: HLVDCON REGISTER (HIGH/LOW-VOLTAGE DETECT CONTROL)

R/W-0	U-0	R-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1
VDIRMAG		IRVST	HLVDEN	HLVDL3 <sup>(1)</sup>	HLVDL2 <sup>(1)</sup>	HLVDL1 <sup>(1)</sup>	HLVDL0 <sup>(1)</sup>
bit 7							bit 0

bit 7 **VDIRMAG:** Voltage Direction Magnitude Select bit  
1 = Event occurs when voltage equals or exceeds trip point (HLVDL3:HLVDL0)  
0 = Event occurs when voltage equals or falls below trip point (HLVDL3:HLVDL0)

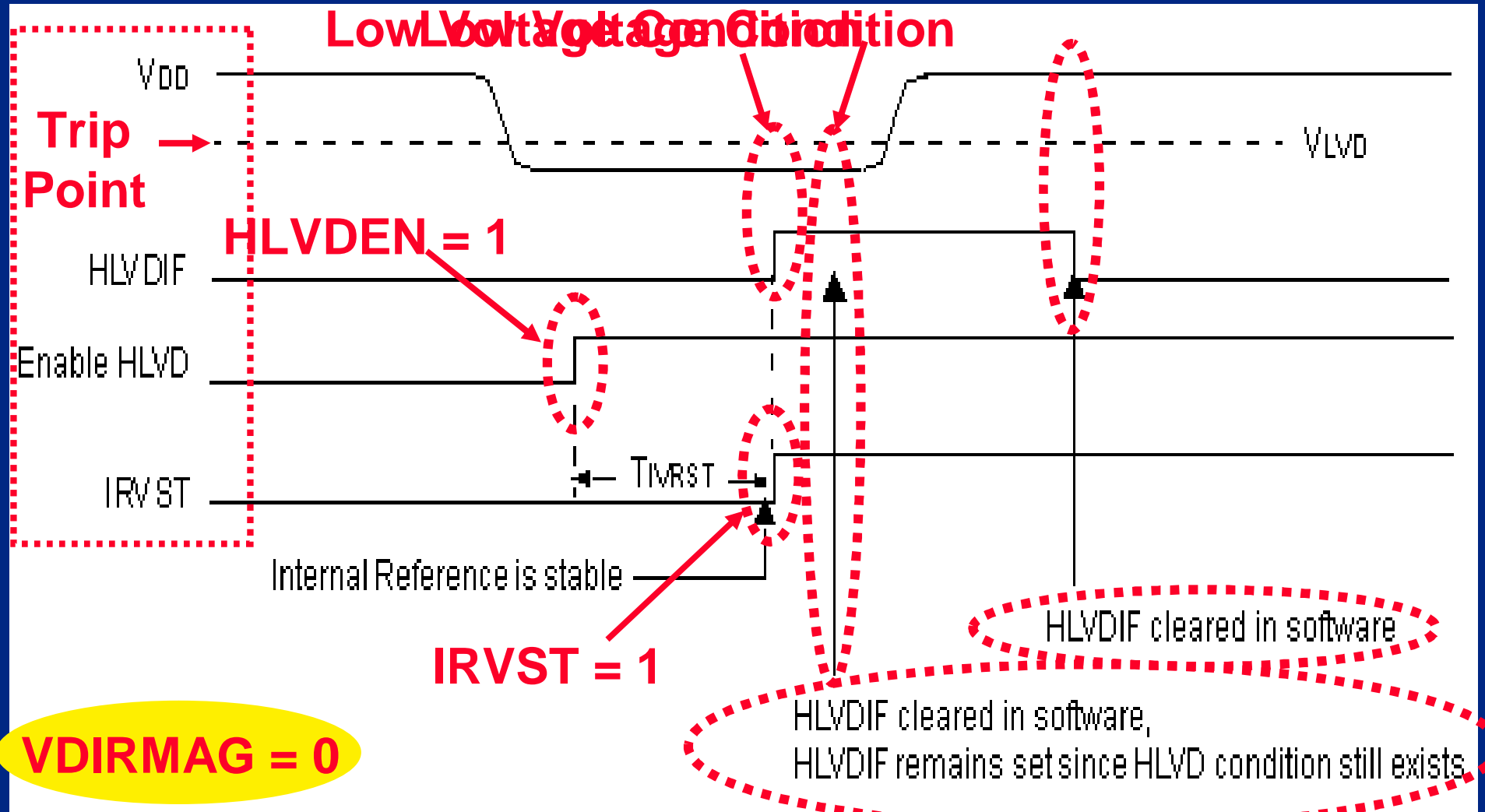
### **VDIRMAG:** Voltage Direction Magnitude Select Bit

1 = Event occurs when voltage equals or exceeds trip point (HLVDL3:HLVDL0)

0 = Event occurs when voltage equals or falls below trip point (HLVDL3:HLVDL0)

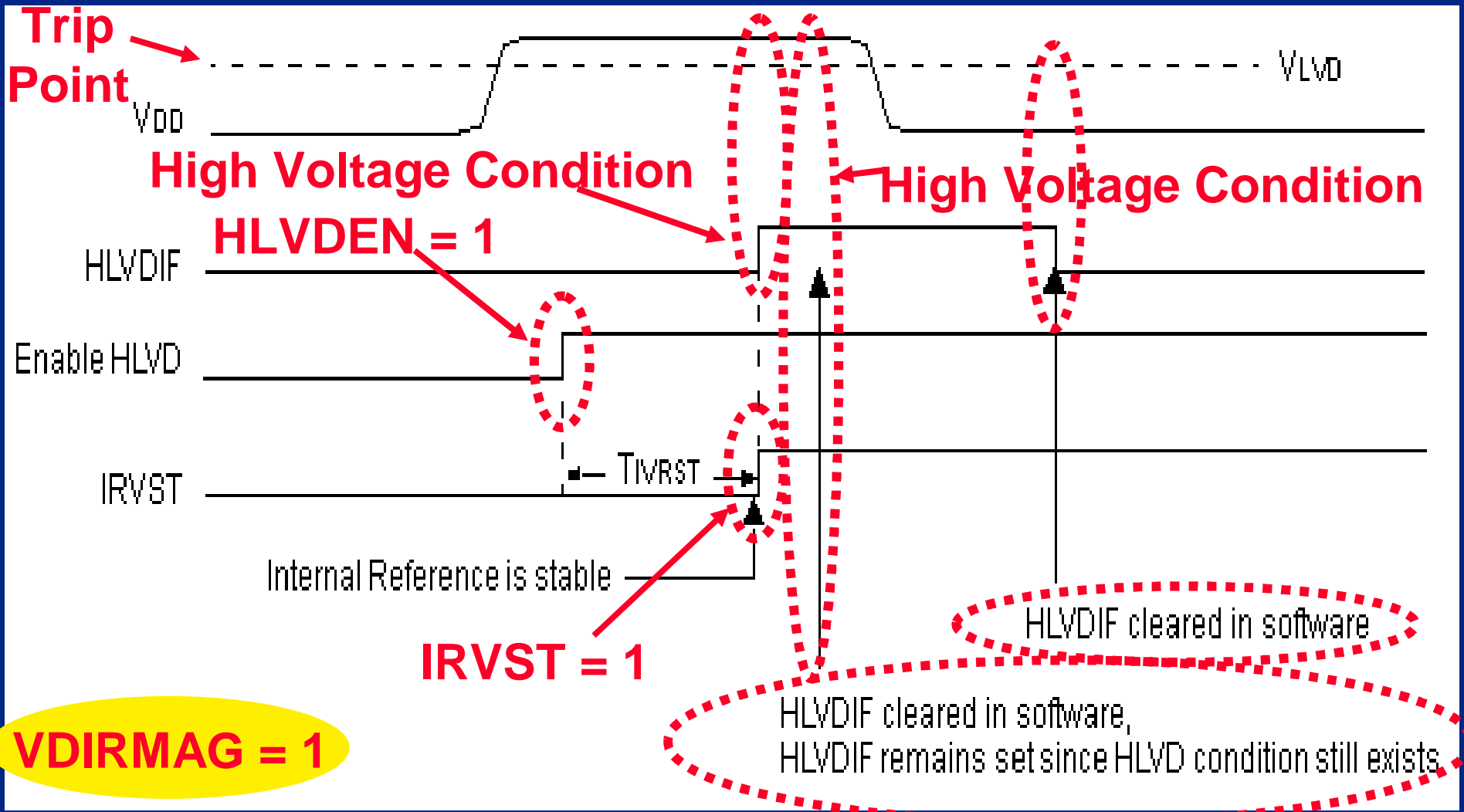


# PIC18 Features: HLVD





# PIC18 Features: HLVD



**VDIRMAG = 1**



# PIC18 Features: BOR

BOR Configuration		
BOREN1	BOREN0	SBOREN (RCON<6>)
0	0	Unavailable
0	1	Available
1	0	Unavailable
1	1	Unavailable

**BOREN1:0**

BOREN bit must be enabled by programming the configuration bits "OFF".

\* BOR enabled in software; operation controlled by SBOREN.

\* BOR enabled in hardware in RUN and IDLE modes, disabled during SLEEP mode.

BOR enabled in hardware; must be disabled by reprogramming the configuration bits "ON".

# PIC18 Features: LCD

## | Enhanced LCD Module

- | Direct Driving of LCD Panel
- | Can drive LCD in SLEEP Mode
- | Software select segments/pixel
- | Programmable LCD Module
  - | Three Clock sources
  - | Static, 1/2 or 1/3 bias configuration
  - | Up to 4 Commons: Static, 1/2, 1/3, or 1/4 multiplex



Device	Commons	Segments	Pixels	Program Memory
PIC18F6390	4	32	128	8K
PIC18F6490	4	32	128	16K
PIC18F8390	4	48	192	8K
PIC18F8490	4	48	192	16K



# PIC18 Features: LCD Applications

## Sensor

- | Utility Metering
  - | Electric, Water Gas

- | Thermostats

- | Pressure Sensors

- | Gas Detection

## Automotive

- | Dashboard Controller

- | Car Alarm

## Medical

- | Asthma Inhaler

- | Blood Sugar Monitor

- | Blood Pressure Meter

## Consumer

- | Microphone Display

- | DVD Player

- | Toy Speed Gun

- | Telescope Displays



# PIC18 Features: USB



## | Data RAM

| 2 Kbytes

| of which up to 1 Kbytes can be used for USB buffers

## | Data EEPROM

| 256 bytes

- | USB 2.0 Full-Speed
- | **Streaming Parallel Port**
- | **nanoWatt Technology**
- | **Enhanced Flash**
- | **I<sup>2</sup>C™/SPI™, EUSART**
- | **10-bit ADC**
- | **1xECCP + 1xCCP**
- | **48 MHz - 12 MIPS**
- | **Voltage Range: 2.0 - 5.5V**





# PIC18 Features Quiz

1. Two-Speed Start-up is used for XTAL based modes: True or False

True

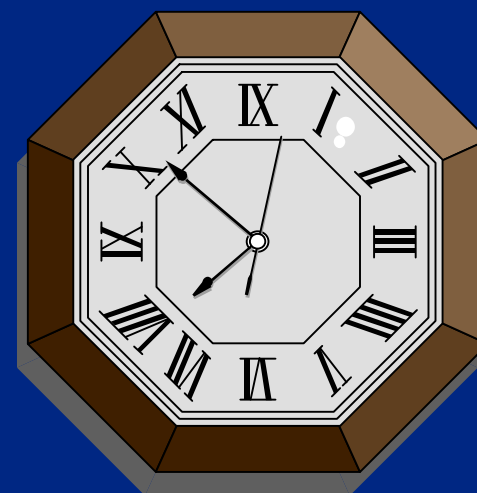
2. The BOR can be enabled by firmware: True or False

True



# Agenda

- | nanoWatt Technology review
- | PIC18 Features
- | ***Clock System***
- | Power Managed Modes
- | PIC18 Devices
- | Extended Instruction Set
- | Summary





# Clock System

## | Clock Sources

### | Primary

- | Fixed Selection

- | LP, XT, HS, RC, EC, Int RC Osc

### | Secondary

- | Timer1 Oscillator - fixed frequency

- | Required for Real Time Clock time base

### | Internal RC Oscillator

- | INTOSC (8 MHz) source

- | 4, 2, 1 MHz, 500, 250, 125 and 31 kHz

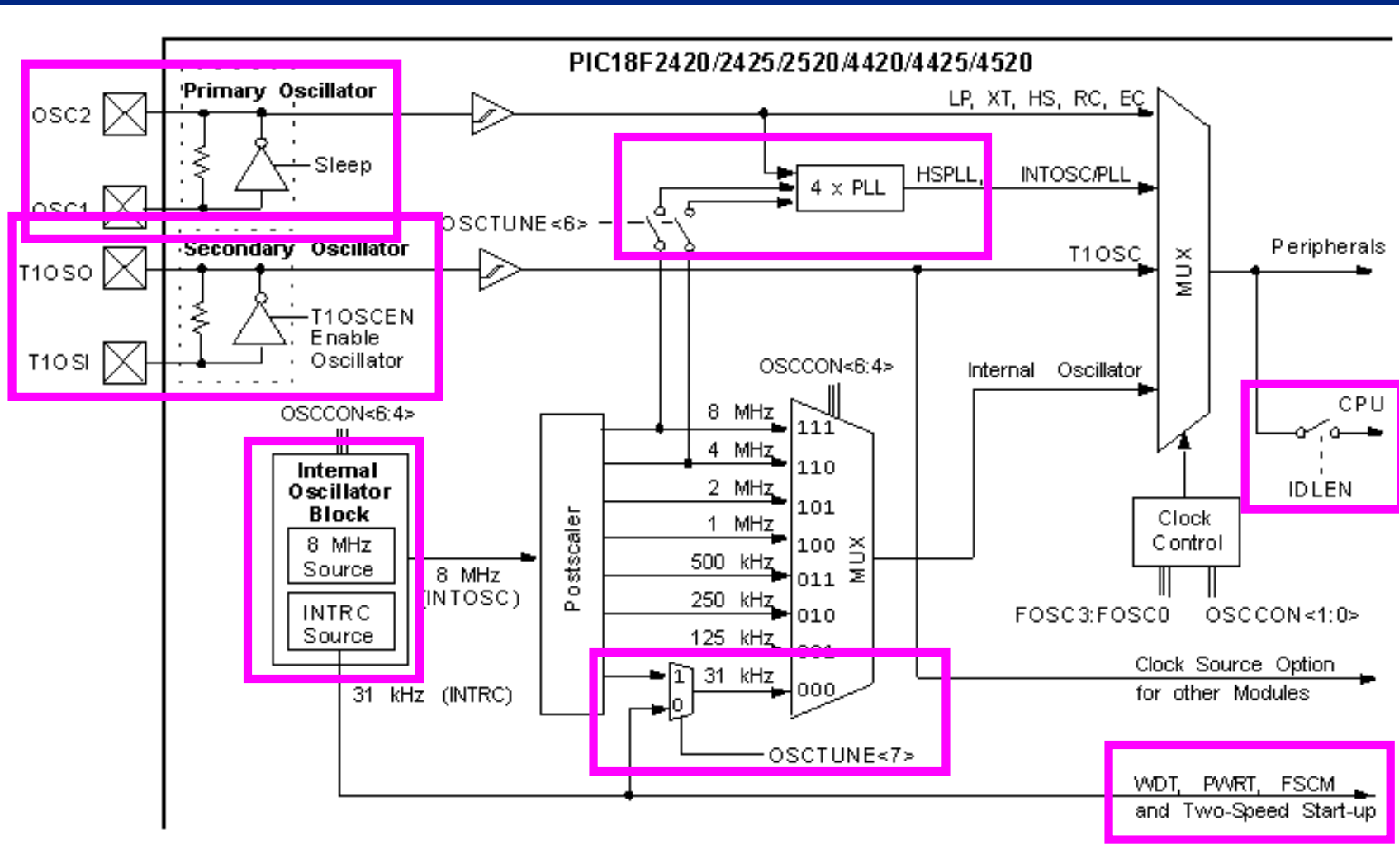
- | INTRC (31 kHz) source



# Clock System

- | New Internal RC Oscillator
  - | 2 separate RC sources
    - | 8 MHz (INTOSC)
    - | 31 kHz (INTRC)
  - | 2 - 31 kHz sources
  - | INTOSC 8 & 4 MHz can be routed through PLL
    - | 16 or 32 MHz
  - | Modifying IRCF<2:0> bits immediately selects a different INTOSC postscaler tap

# Clock System



# Clock System: OSCCON

2: **OSCCON REGISTER**

R/W-0	R/W-1	R/W-0	R/W-0	R <sup>(1)</sup>	R-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1

bit 7

bit 7 **IDLEN:** Idle Enable bit  
 1 = Device enters Idle mode on SLEEP instruction  
 0 = Device enters Sleep mode on SLEEP instruction

## IDLEN: Idle Enable Bit

1 = Device enters Idle mode on SLEEP instruction

0 = Device enters SLEEP mode on SLEEP instruction

## IRCF2:IRCF0: Internal Oscillator Frequency Select Bits

000 = 31kHz (from either INTOSC/256 or INTRC directly)

00 = Primary oscillator

- Note 1:** Reset state depends on state of the IESO configuration bit.  
**2:** Source selected by the INTSRC bit (OSCTUNE<7>), see text.  
**3:** Default output frequency of INTOSC on Reset.



# Clock System: OSCTUNE

RAW-0	RAW-0 <sup>(1)</sup>	U-0	RAW-0	RAW-0	RAW-0	RAW-0	RAW-0
INTSRC	PLLEN <sup>(1)</sup>	—	TUN4	TUN3	TUN2	TUN1	TUN0
bit 7							bit 0

**TUN4:TUN0:** Frequency Tuning bits

01111 = Maximum Frequency

00000 = Center Frequency

10000 = Minimum Frequency

**AN244**

**PLLEN:** Frequency Multiplier PLL for INTOSC Enable Bit

1 = PLL enabled for INTOSC (4MHz and 8 MHz only)

0 = PLL disabled

00001

00000 = Center frequency. Oscillator module is running at the calibrated frequency.

11111

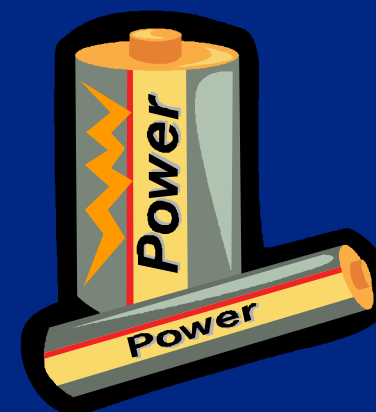
· ·

10000 = Minimum frequency



# Agenda

- | nanoWatt Technology review
- | PIC18 Features
- | Clock System
- | ***Power Managed Modes***
- | PIC18 Devices
- | Extended Instruction Set
- | Summary







# Power Managed Modes

## 3 Categories

- 4 **RUN** - 3 clock sources
- 4 **IDLE** - 3 clock sources
- 4 **SLEEP** - no clocks

**Total = 7 Modes**



# Power Managed Modes

## | PRI\_RUN Mode

- | Config Word defines Primary Clock Source
  - | FOSC3:FOSC0 (`_CONFIG1H<3:0>`) 10 modes
    - | Crystal Oscillator - LP, XT, HS, HSPLL
    - | External Clock - EC, ECIO
    - | External RC Oscillator - RC, RCIO
    - | Internal RC Oscillator - INTIO1, INTIO2

## | SEC\_RUN Mode

- | Clock switching mechanism in other PIC18 controllers
- | Timer1 source, Primary oscillator is disabled

## | RC\_RUN Mode

- | IRCF<2:0> selects clock speed
- | IOFS set after 1 us (typ.) delay if Freq  $\neq$  31 kHz



# Power Managed Modes

## RUN Modes

### Entry:

Configure the SCS<1:0> bits (OSCCON) for the desired clock source & execute a ~~SLEEP instruction~~

### Exit:

Configure the SCS<1:0> bits (OSCCON) for the desired clock source

Mode	Peripheral/CPU Clock	SCS bits
PRI_RUN	Primary Osc & Running	00
SEC_RUN	T1OSC & Running	01
RC_RUN	Int RC Osc Running	1x



# Power Managed Modes

## Switch from RC\_RUN to PRI\_RUN

```
;Application code here ←  
;Application code here ←
```

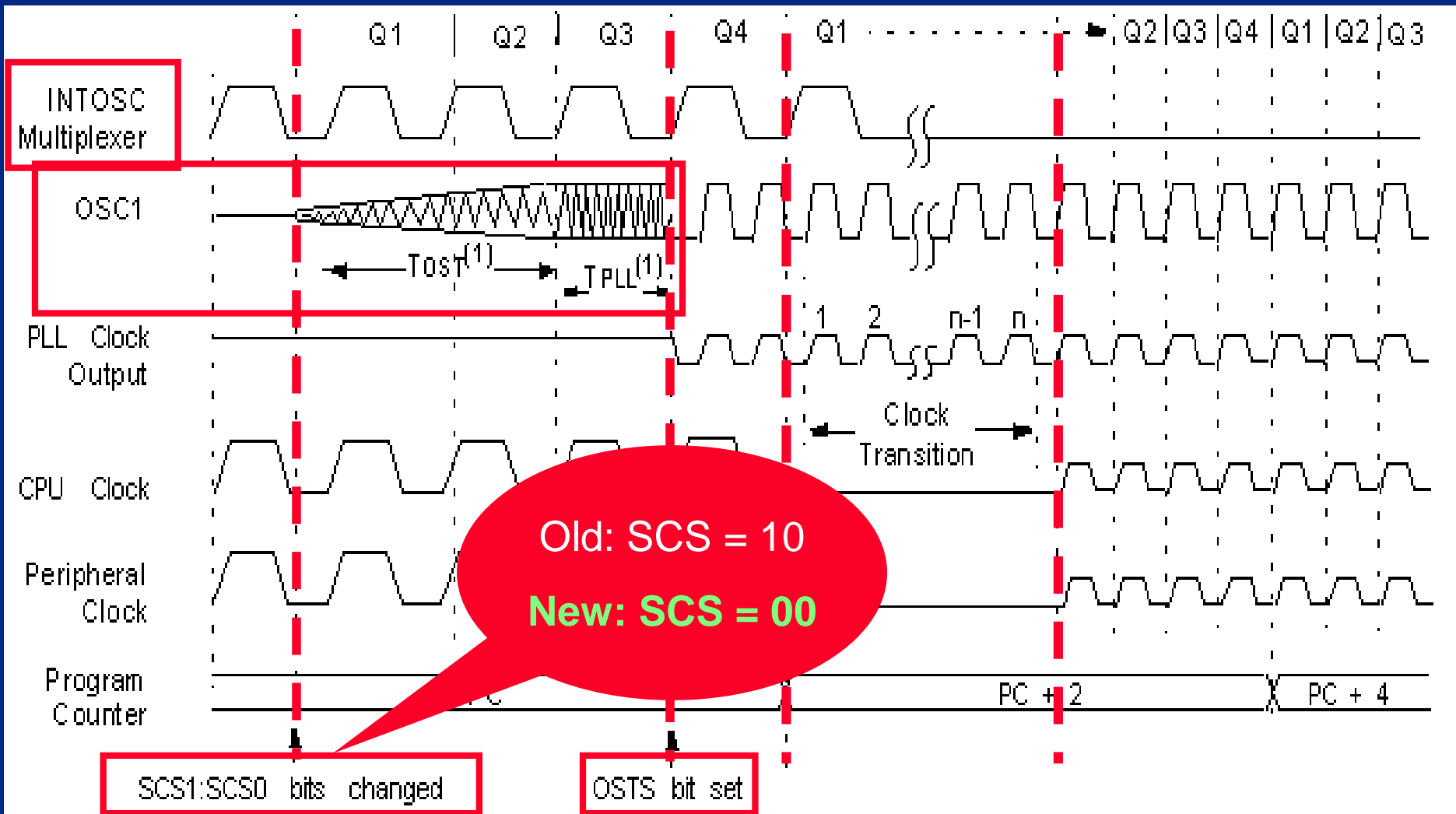
**“Switch from Internal RC Oscillator (SCS:10) to HS mode (SCS:00)”**

```
bcf OSCCON,SCS1 ;PRI_RUN mode is  
; selected
```

**“Code pauses, CPU/Peripherals are not clocked”**  
**“Clock transition takes place”**  
**“Execution resumes on Primary clock”**

```
;Application code here ←  
;Application code here ←
```

# Clock Switch: RC\_RUN to PRI\_RUN





# Power Managed Modes

## | PRI\_IDLE Mode

- | Same clock source as PRI\_RUN
- | Fastest resumption of device operation
- | Wake event: CPU is clocked by Primary source

## | SEC\_IDLE Mode

- | Same clock as SEC\_RUN
- | Wake event: CPU is clocked by Timer1

## | RC\_IDLE Mode

- | Same clock as RC\_RUN
- | IRCF<2:0> selects clock speed
- | Wake event: CPU is clocked by Internal RC oscillator



# Power Managed Modes

## | IDLE Modes

### | Entry:

- | Enable IDLEN = 1 (OSCCON<7>)
- | Execute a SLEEP instruction

### | Exit:

- | Interrupt
- | WDT time-out
- | RESET

Mode	Peripheral Clock	CPU Clock	SCS bits
PRI_IDLE	Primary Osc	Halt	00
SEC_IDLE	T1OSC	Halt	01
RC_IDLE	Int RC Osc	Halt	1x



# Power Managed Modes

Switch from PRI\_RUN to PRI\_IDLE

```
;Application code here ←  
;Application code here ←
```

**“Switch from Primary RUN (SCS:00) to Primary IDLE (SCS:00)”**

```
bsf OSCCON, IDLEN ; select IDLE mode  
sleep             ; enters_PRI_IDLE mode
```

**“Execution pauses here”**  
**“CPU not clocked, peripherals clocked from primary clock source”**  
**“Waits for wake event”**





# Power Managed Modes

Peripheral Functions	SLEEP	IDLE
Flash Write	✓	✓
A/D	✓	✓
USART RX	✓	✓
MSSP RX	✓	✓
Timer 0/1/3 Counter	✓	✓
T1/3 Oscillator	✓	✓
Capture/Compare	✓	✓
Comparator	✓	✓
WDT,BOR,LVD,MCLR	✓	✓
PWM/Compare w/output		✓
Timer0/1/3 Timer mode		✓
Timer 2		✓
USART Sync TX,async		✓
MSSP master mode		✓
A/D (sys clock)		✓
USB		✓
CAN		✓
LCD		✓

## Why use Idle Mode?

- | Save Power
- | Peripheral Run/  
CPU off
- | Fast Wake-up



# Power Managed Modes

## | SLEEP Mode

- | Legacy SLEEP mode in all Microchip<sup>®</sup> controllers
- | CPU/Peripherals clock disabled
- | Only power managed mode where no system clock sources are running
- | Entry:
  - | IDLEN bit = 0 (POR default)
  - | Execute SLEEP instruction



# Power Managed Modes

## | SLEEP Mode (cont'd)

### | Exit:

- | No clock is selected, execution resumes when Primary becomes ready

Execution will resume with the clock source selected by the SCS bits

- | Immediately code execution if 2-Speed Startup or FSCM is enabled



# Power Managed Modes

## | Exit by Interrupt

- | CPU/Peripherals are clocked using selected clock until Primary becomes ready

**Execution will resume with the clock source selected by the SCS bits**

## | Exit by Reset

- | Primary clock is started
- | Execution pauses until primary becomes ready
- | Immediate code execution if 2-Speed Startup or FSCM is enabled
- | OSTS bit = 1 when Primary is providing clock



# Power Managed Modes

- | Exit by WDT Time-out
  - | TO bit is cleared (RCON <3>)
  - | Exit depend on CPU operating condition
    - | Executing code (all RUN modes)
      - | Device Reset
    - | No Code execution (SLEEP & IDLE modes)
      - | Exit from power managed mode



# Power Managed Modes

## Quiz

1. SEC\_RUN mode uses the Internal RC Osc:  
True or False

**False**

2. A SLEEP instruction must be executed when switching from PRI\_RUN to RC\_RUN: True or False

**False**

3. Which mode keeps the peripherals clocked but disables the CPU clock?

**Idle Mode**

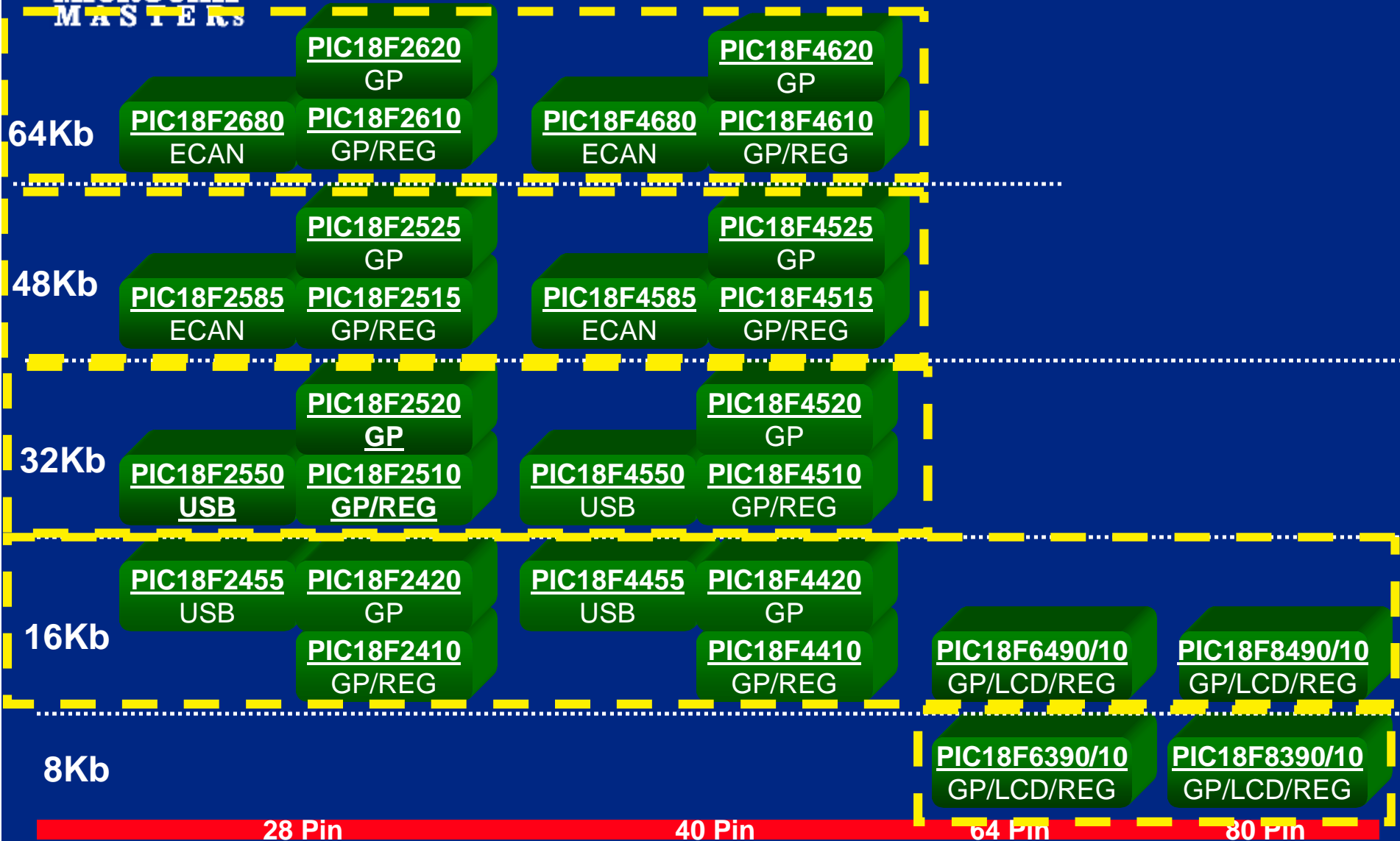


# Agenda

- | nanoWatt Technology review
- | PIC18 Features
- | Clock System
- | Power Managed Modes
- | **PIC18 Devices**
- | PIC18F Extended Architecture
- | Summary



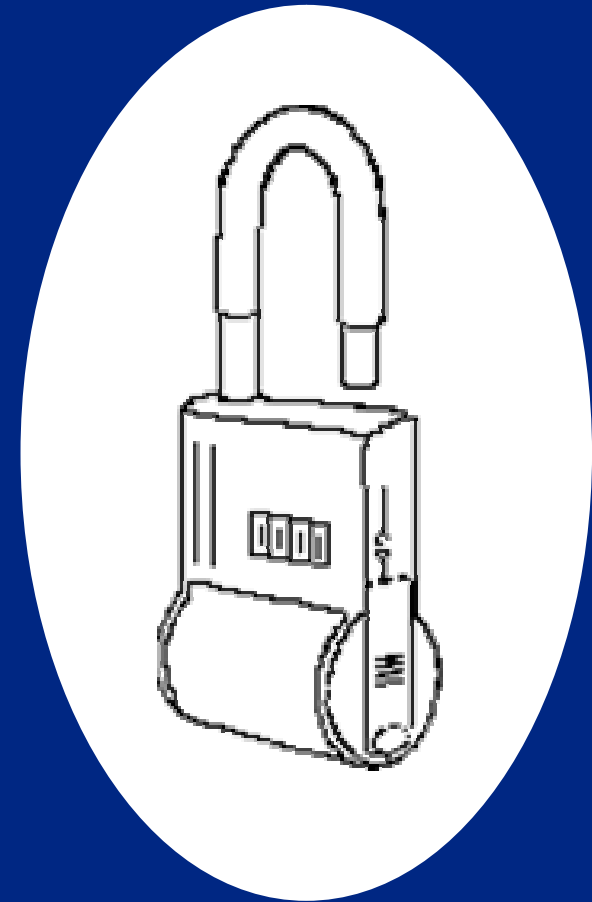
# PIC18F nanoWatt Technology 28-80 Pins





# PIC18F Application

- | “Lock Box”
  - | Purpose: Key Storage
  - | Residential/Commercial
- | Operation
  - | Waits on user input
  - | Decodes user id, logs name & time
  - | Opens locking mechanism
  - | Short duty cycle
  - | Battery powered
  - | Multiple users



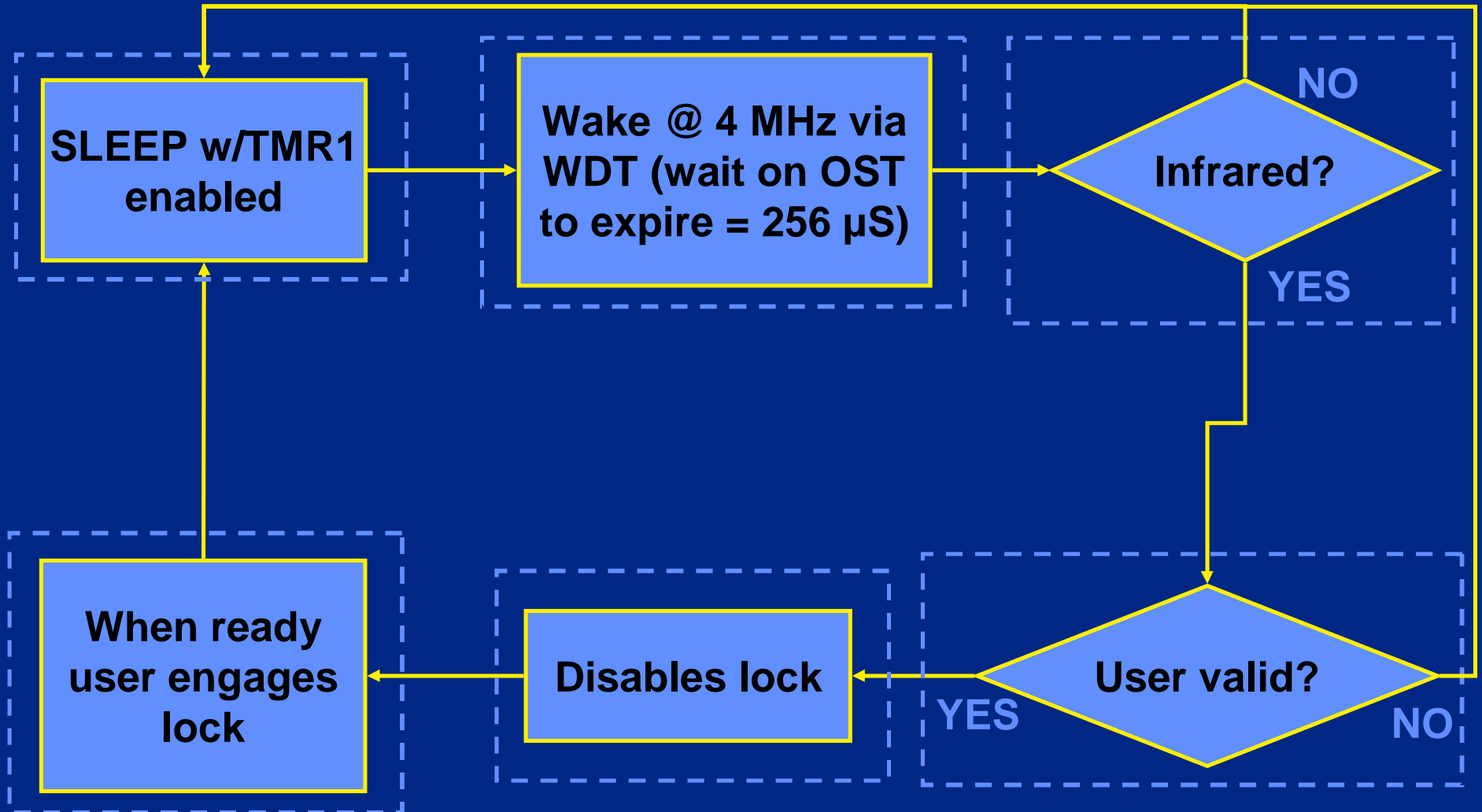


# PIC18F Application

- | Lock Box Firmware
  - | Operating modes
    - | SLEEP and wake on a regular period
    - | Check valid Infrared signal, not valid go back to SLEEP
    - | If valid, increase frequency & receive
    - | Valid user: lock disengages

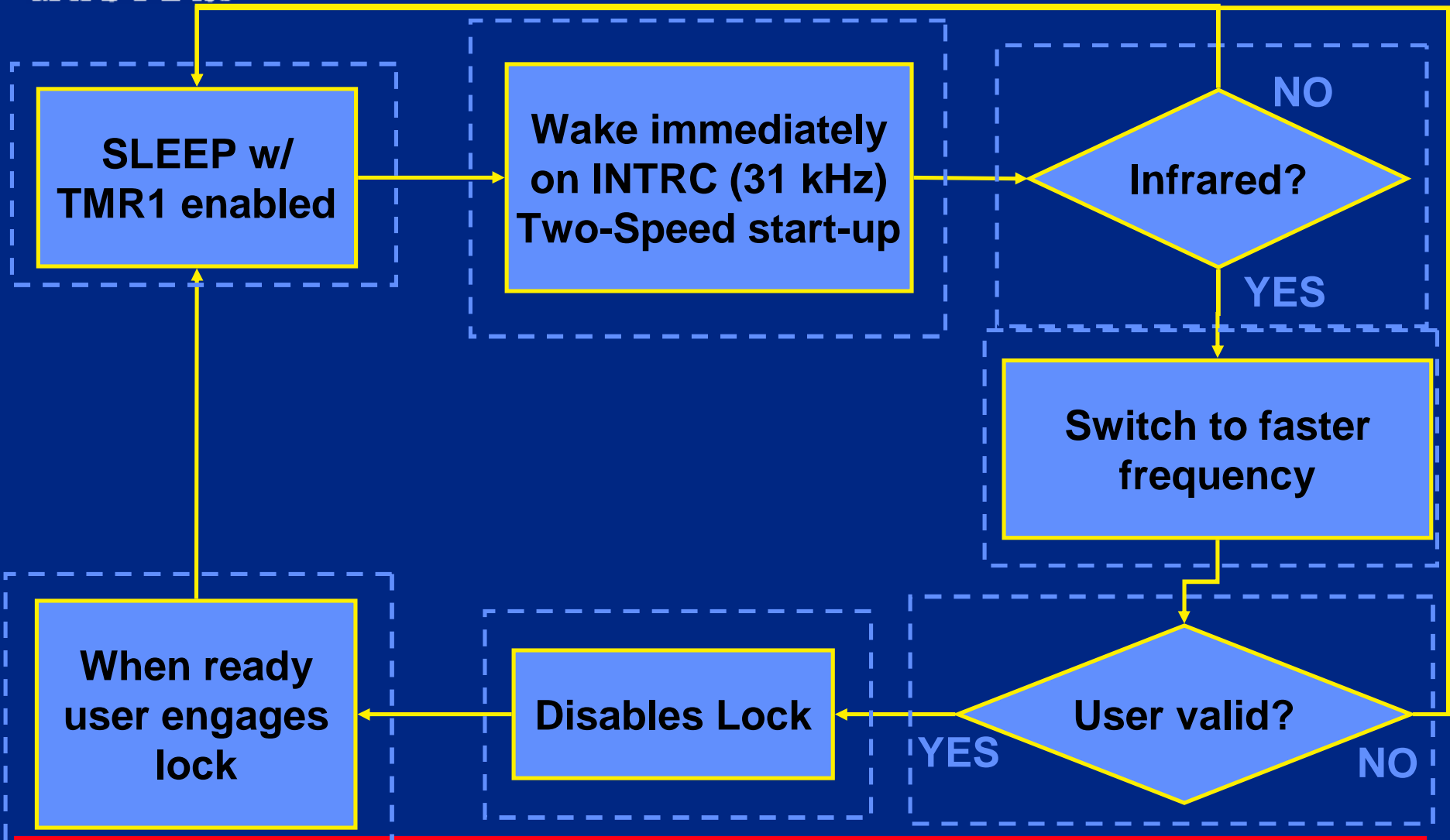


# Application: PIC18F452



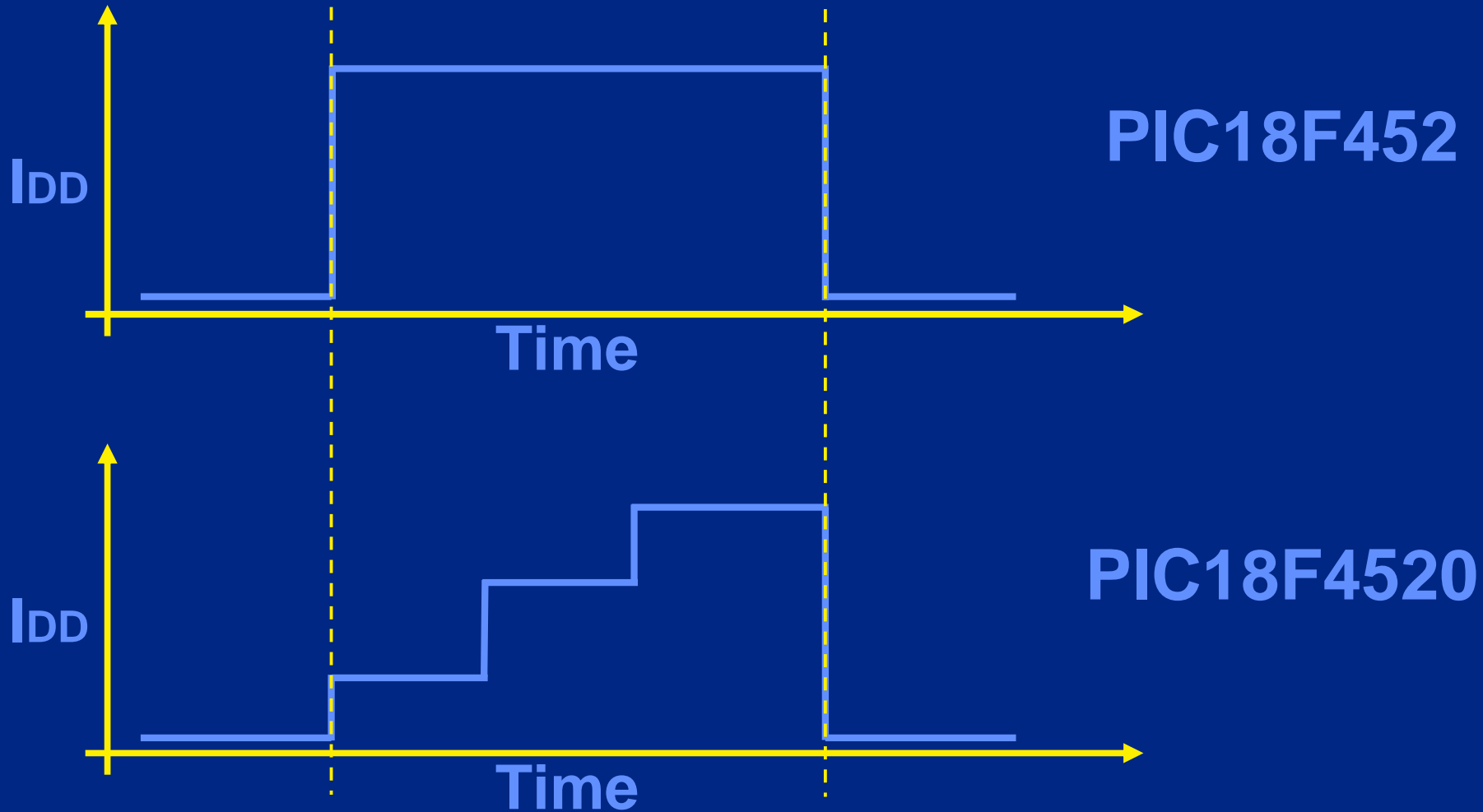


# Application: PIC18F4520





# PIC18F Application





# Agenda

- | nanoWatt Technology review
- | PIC18 Features
- | Clock System
- | Power Managed Modes
- | PIC18 Devices
- | ***PIC18F Extended Architecture***
- | Summary



# PIC18F Architecture Extensions

## Agenda

- | Background/Motivation
- | What is the extended architecture?
  - + New addressing mode
  - + New instructions
- | How to enable the extensions
- | Silicon support
- | Tools support
- | Impact of the extensions
- | Who should/shouldn't use the extensions?



# Background

The existing PIC18 architecture provides a number of features that support high-level languages:

- | byte-wide program memory addressing
- | indirection registers
- | pre/post increment/decrement
- | single-cycle multiply
- | PLUSW2 addressing





# Background

Some areas where we still saw room for improvement:

- | stack access (multiple use of WREG)
- | function pointers
- | cross-bank stack manipulations



# Our Goal

*To provide a memory-efficient model for  
high-level language support,  
including reentrant code*



# The Solution

## PIC18F Architectural Enhancements:

- | A new addressing mode (literal indexed)
- | Several new instructions

*These extensions greatly simplify the code that is needed for high-level language constructs*



# Why Do I Care?

- | Understanding compiler-generated code
- | Interfacing to C code in assembly language
- | Application in standalone assembly code



# PIC18F Architecture Extensions

## Agenda

- | Background / Motivation
- | What is the extended architecture?
  - + New addressing mode
  - + New instructions
- | How to enable the extensions
- | Silicon support
- | Tools support
- | Impact of the extensions
- | Who should / shouldn't use the extensions?



# Presentation of Extensions

For each extension, we will provide the following information:

- | Example of code on existing architecture
- | Description of new instructions/addressing
- | Comparison showing improvement



# Indexed Addressing

## Existing PIC18

Performing a simple assignment to a stack-based variable:

```
c = 5;
```


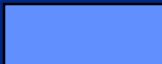
generates 10 bytes of instructions:

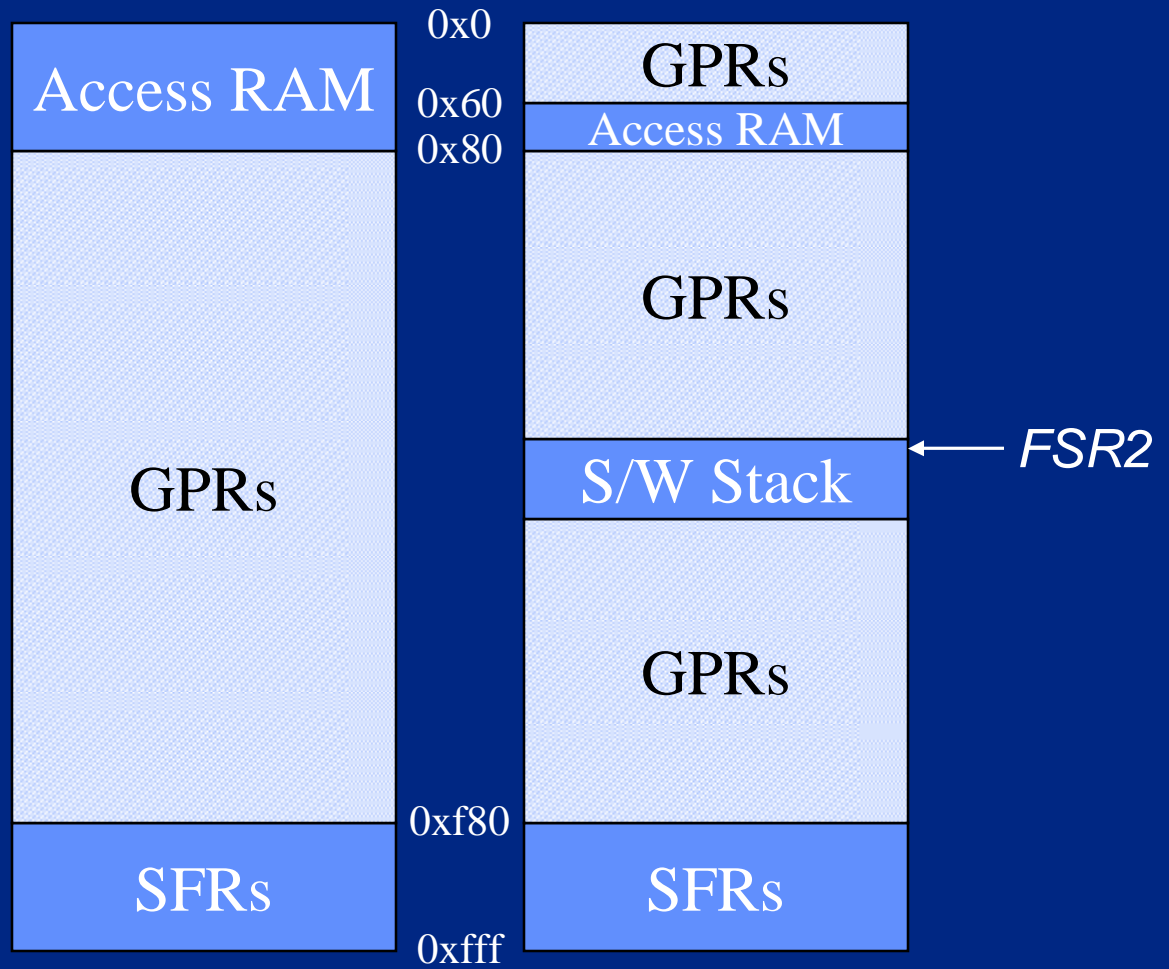
```
movlw      5  
movwf     PRODL, 0  
movlw     offset(c)  
movff     PRODL, PLUSW2
```



# Indexed Addressing New PIC18

**Indexed literal** addressing remaps the access RAM window to be a stack offset window

-  = banked addressing
-  = unbanked addressing



Traditional PIC18F

Extended PIC18F





# Indexed Addressing

## New PIC18

Assembler syntax:

*opcode* [*offset*],*f*

Example:

**addwf [12],1 ; add WREG to location (FSR2+12)**



# Indexed Addressing Improvement

Example of improvement on simple assignment:

**c = 5;**

```
movlw    5
movwf    PRODL, 0
movlw    offset(c)
movff    PRODL, PLUSW2
```

Traditional PIC18F  
*10 bytes*

```
movlw    5
movwf    [c], 0
```

Extended PIC18F  
*4 bytes*



# Stack Allocation

## Existing 18F Architecture

Stack allocation overhead (4 bytes) is required on most functions:

```
movlw    FrameSize
addwf    FSR1L,1,0
```

The penalty is even worse (12 bytes) if the stack is allowed to cross banks:

```
movlw    FrameSize
addwf    FSR1L,0,0
bnc      PC+6
setf     FSR1L,0
movf     POSTINC1,1,0
movwf    FSR1L,0
```



# Stack Allocation

## New PIC18

Assembler syntax:

**addfsr**      *literal, n*

**subfsr**      *literal, n*

Operation:

$FSR_n = FSR_n +/- \textit{literal}$

Example:

**addfsr**      **20,2**      ; add 20 to FSR2



# Stack Allocation Improvement

Example of improvement for function prologue:

## Single bank stack:

```
movlw    FrameSize
addwf    FSR1L,1,0
```

## Multi-bank stack:

```
movlw    FrameSize
addwf    FSR1L,0,0
bnc      PC+6
setf     FSR1L,0
movf     POSTINC1,1,0
movwf    FSR1L,0
```

```
addfsr  FrameSize,1
```

Extended PIC18F  
*2 bytes*

Traditional PIC18F  
*4-12 bytes*



# Stack Allocation w/ return

Assembler syntax:

**addulnk**     *literal*

**subulnk**     *literal*

Operation:

$FSR2 = FSR2 +/- \textit{literal}$

**return**

Example:

**subulnk**     **14**     ; subtract 14 from FSR2  
                                 ; and return



# Move Stack to Global Existing PIC18

Moving a local variable's contents into a global variable:

```
gc = lc;
```

Produces the following code:

```
movlw      offset(lc)  
movff     PLUSW2, gc
```



# Move Stack to Global

## New PIC18

Assembler syntax:

```
movsf      [src], dst
```

Operation:

```
dst = (FSR2 + src)
```

Example:

```
movsf [5], 0xac      ; move the contents of  
                      ; (FSR2 + 5) into memory  
                      ; location 0xac
```





# Move Stack to Global Improvement

Example of improvement:

`gc = lc;`

```
movlw    offset(lc)
movff    PLUSW2,gc
```

Traditional PIC18F  
*6 bytes, affects WREG*

```
movsf    [lc],gc
```

Extended PIC18F  
*4 bytes, WREG unchanged*



# Move Stack to Stack

## Existing PIC18

Moving a local to another local variable:

```
lc1 = lc2;
```

Produces the following code:

```
movlw    offset(lc2)  
movf     PLUSW2,0,0  
movwf    INDF1,0  
movlw    offset(lc1)  
movff    INDF1,PLUSW2
```



# Move Stack to Stack

## New PIC18

Assembler syntax:

```
movss      [src], [dst]
```

Operation:

$$(FSR2 + \mathit{dst}) = (FSR2 + \mathit{src})$$

Example:

```
movss [3], [5]    ; move the contents of  
                  ; (FSR2 + 5) into  
                  ; (FSR2 + 3)
```



# Move Stack to Stack Improvement

Example of improvement:

**lc1 = lc2;**

```
movlw    offset(lc2)
movf     PLUSW2,0,0
movwf    INDF1,0
movlw    offset(lc1)
movff    INDF1,PLUSW2
```

Traditional PIC18F  
*12 bytes*

```
movss   [lc2],[lc1]
```

Extended PIC18F  
*4 bytes*



# Function Pointer Invocation

## Existing PIC18

Invoking a function pointer:

```
fn( ) ;
```

Produces the following code:

```
bra          PC+12  
movff       fn+2,PCLATU  
movff       fn+1,PCLATH  
movlb       fn  
movf        fn,0,1  
movwf       PCL,0  
rcall       PC-10
```



# Function Pointer Invocation

## New PIC18

Assembler syntax:

**callw**

Operation:

**PC = PCLATU:PCLATH:WREG**



# Function Pointer Invocation Improvement

Example of improvement:

**fn();**

```
bra      PC+12
movff   fn+2,PCLATU
movff   fn+1,PCLATH
movlb   fn
movf    fn,0,1
movwf   PCL,0
rcall   PC-10
```

Traditional PIC18F  
*18 bytes, 2 branches*

```
movff   fn+2,PCLATU
movff   fn+1,PCLATH
movlb   fn
movf    fn,0,1
callw
```

Extended PIC18F  
*14 bytes, 0 branches*



# Passing a Literal Parameter Existing PIC18

Pushing a literal value onto a downward-growing stack:

```
fn( 0xaa );
```

Produces the following code:

```
movlw      0xaa  
movwf     POSTDEC2, 0
```





# Passing a Literal Parameter

## New PIC18

Assembler syntax:

```
pushl lit
```

Operation:

```
POSTDEC2 = lit
```

Example:

```
pushl 23 ; Store 23 to (FSR2) and  
; decrement FSR2
```



# Passing a Literal Parameter Improvement

Example of improvement:

`fn(0xaa);`

```
movlw    0xaa  
movwf    POSTDEC2, 0
```

Traditional PIC18F  
*4 bytes*

```
pushl    0xaa
```

Extended PIC18F  
*2 bytes*



# PIC18F Architecture Extensions

## Agenda

- | Background / Motivation
- | What is the extended architecture?
  - + New addressing mode
  - + New instructions
- | How to enable the extensions
- | Silicon support
- | Tools support
- | Impact of the extensions
- | Who should / shouldn't use the extensions?



# Enabling the extensions

*A single configuration bit selects between:*

## Traditional Mode

New instruction opcodes are interpreted as NOPs

Fast memory references in the range [0,0x5f] address general purpose access RAM

Default for Development Tools

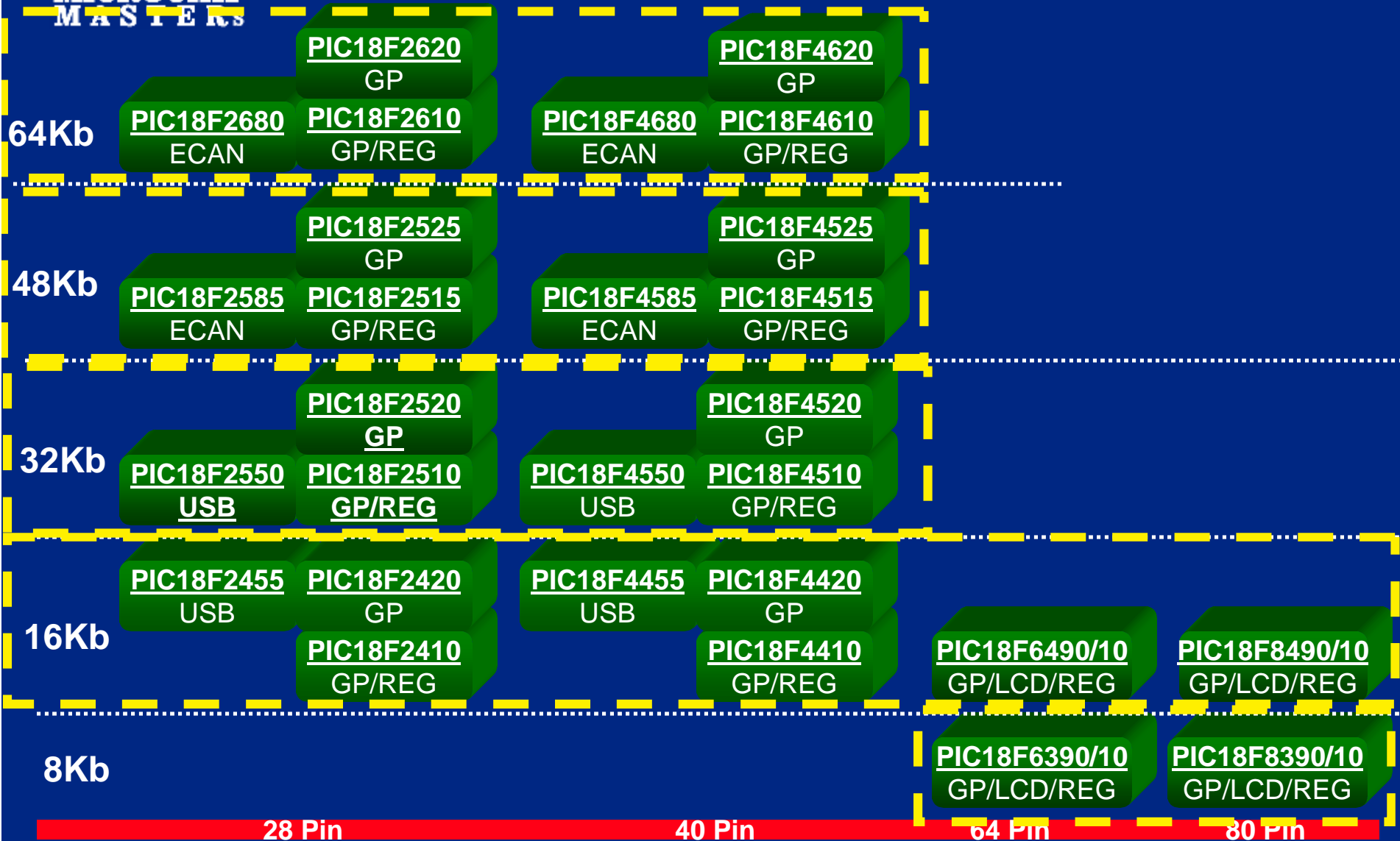
## Extended Mode

New instructions are available

Fast memory references in the range [0,0x5f] address FSR2 offsets



# PIC18F *Extended Instruction Set* Technology: 28-80 Pins





# Tools Support

Software tools are available NOW:

- | MPASM™ Assembler
- | MPLAB® IDE / SIM
- | MPLAB C18

Hardware tools are available NOW:

- | In-Circuit Debugger (ICD2)
- | Programmer (PRO MATE® 2)
- | Emulator (ICE 2000/4000)



# What Mode to Use

## *Use Extended Mode When...*

- Re-entrancy is required
- Starting a new design
- Code is mostly written in C
- Portability to non-extended cores is not needed
- RAM space is critical

## *Use Traditional Mode When...*

- The static model is O.K.
- Modifying an existing design
- Code is mostly written in ASM
- Code must also work on non-extended cores
- RAM space is not critical

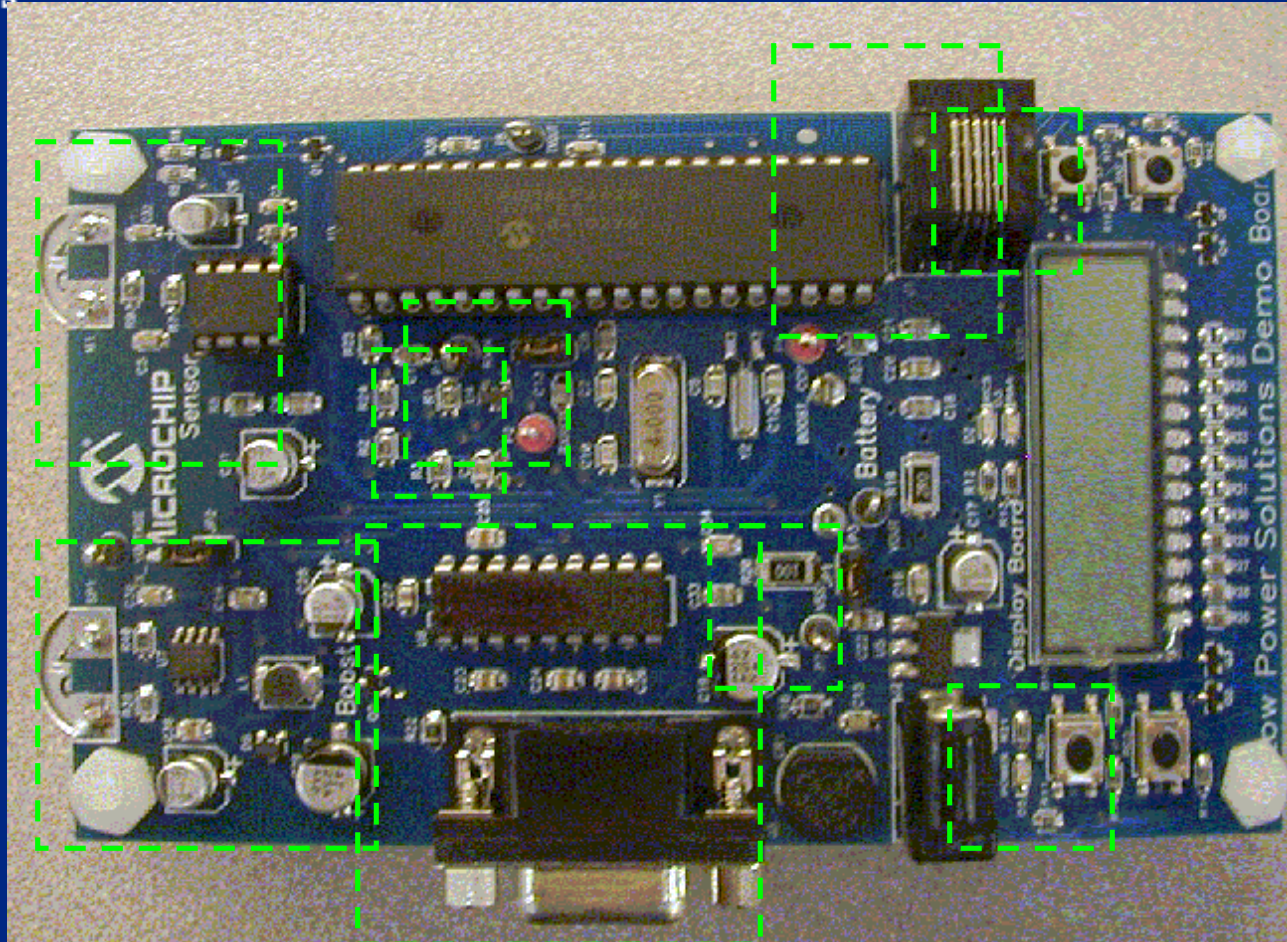


# Summary

- | The PIC18F architecture has been extended with:
  - + a new addressing mode (indexed literal)
  - + new instructions
- | These extensions provide efficient high-level language support
- | 100% backward-compatible behavior is available
- | Tools are available now
- | Applicability of the extensions will depend on the project



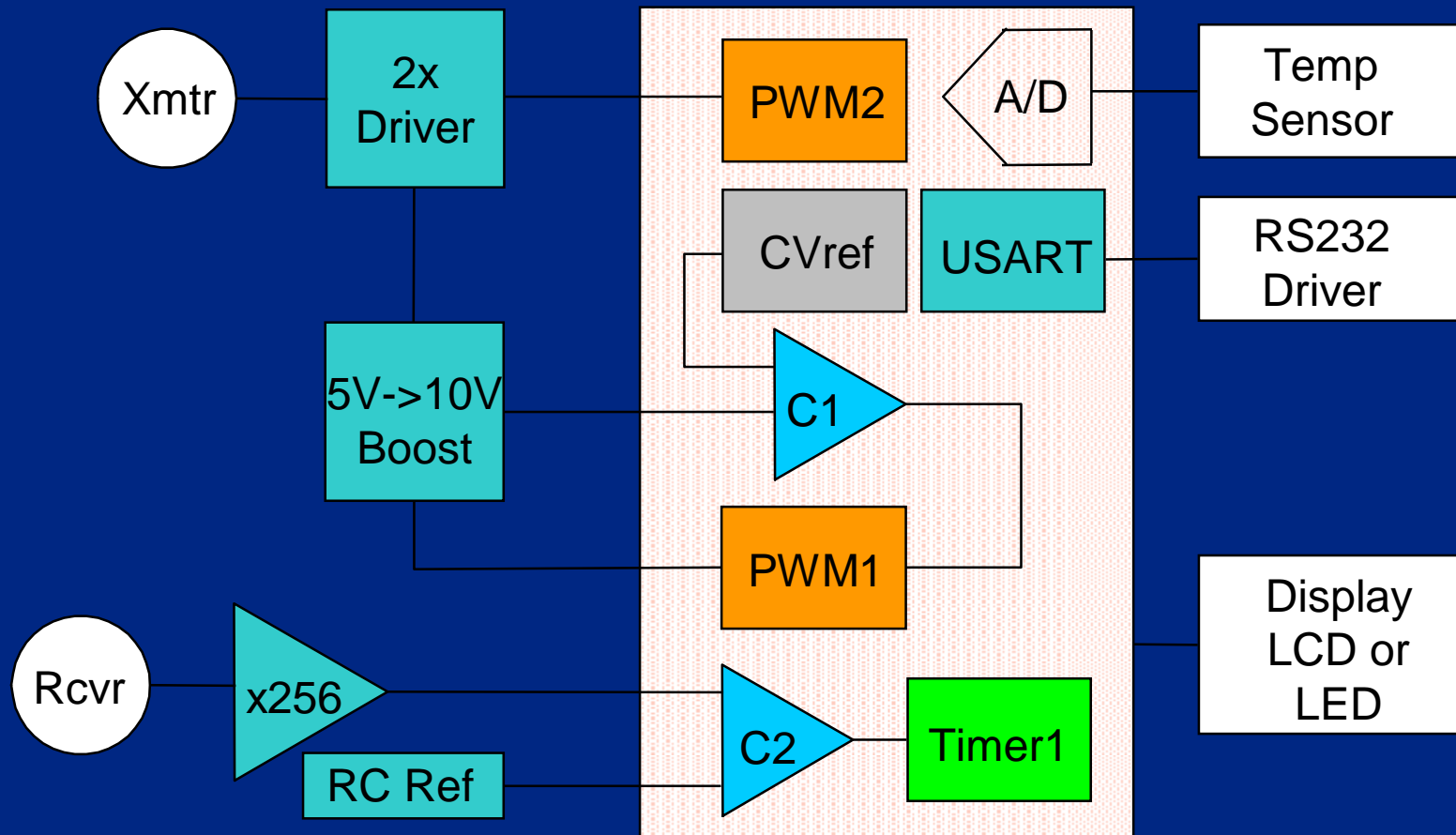
# Ultrasonic Range Finder



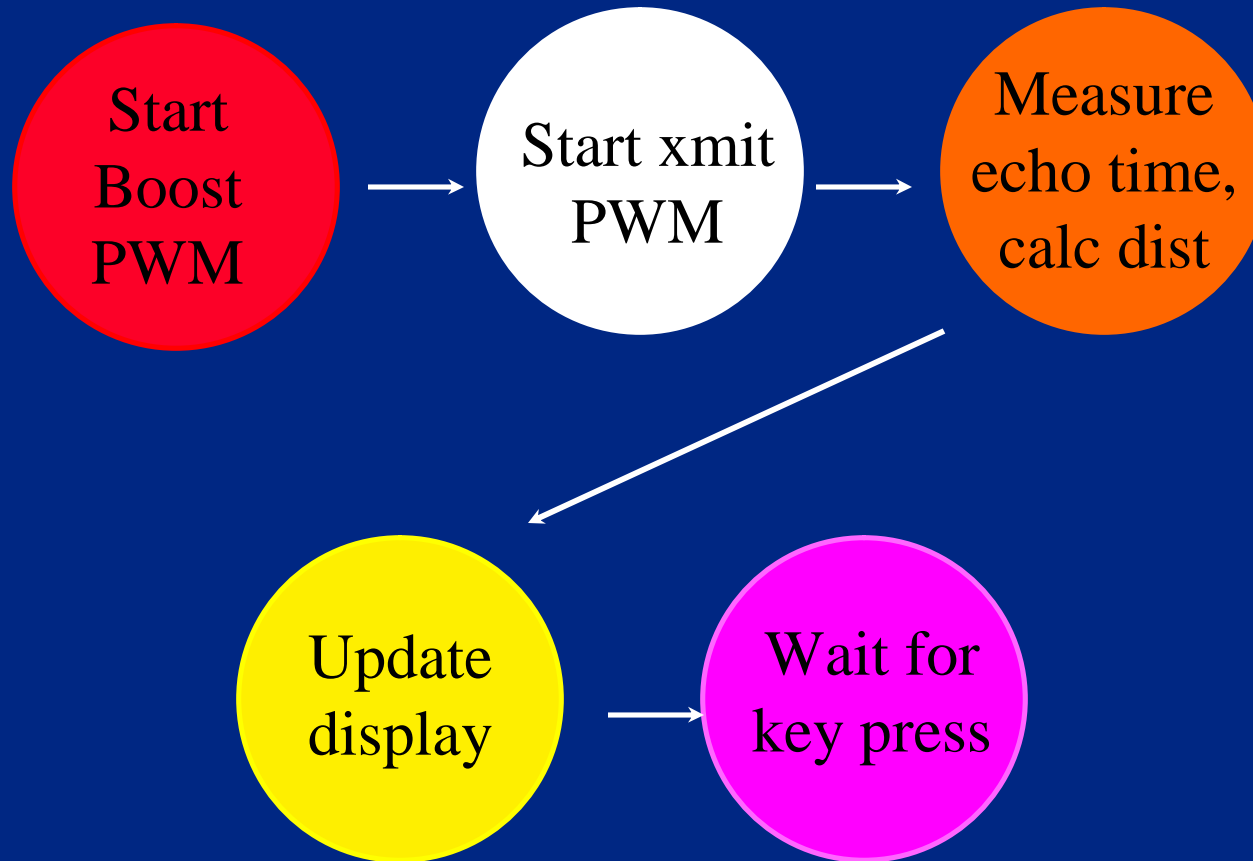


# Low Power Solutions Block Diagram

PIC18F4620



# State Diagram





# Lab: Putting It All Together

- | Purpose: To analyze an application and reduce its power consumption.
  - | Determine where power managed modes can be used
  - | Adjust clock frequency to minimize power consumption
  - | Maintain performance
  - | Measure system and PICmicro current consumption by connecting current meter to appropriate jumper.



## Lab: Tip 1

- | Disable unnecessary system components

LEDRA4\_OFF; //nW: Turn off LED

DISABLE\_TEMP; //nW: Turn off temp sensor

DISABLE\_USART; //nW: Turn off  
USART driver



## Lab: Tip 2

- | Select INTOSC as Primary Oscillator in Config Word
- | TRIS unused I/Os as outputs for INTOSC
- | Use INTOSC as primary oscillator and select 4MHz frequency.

```
int Fosc = 4; //Frequency in MHz
```

```
TRISAbits.TRISA6 = 0;  
TRISAbits.TRISA7 = 0;  
OSCCON = 0b01100000; //4MHz INTOSC  
while(!OSCCONbits.IOFS);
```



## Lab: Tip 3

- | Disable boost after transmit

```
DisableBoost();
```





**MICROCHIP  
M A S T E R S**

## Lab: Tip 4

Replace button while-loop with Sleep and PortB interrupt-on-change wake-up

```
// while(SW1_NOT_PRESSED);  
  
TRISBbits.TRISB7 = 0;  
TRISBbits.TRISB6 = 0;  
Disable_Ints();  
WREG = PORTB;  
INTCONbits.RBIF = 0;  
INTCONbits.RBIE = 1;  
Sleep();  
INTCONbits.RBIE = 0;
```





## Lab: Tip 5

- | Replace 100ms for-loop delay with TMR2 rollover at  $F_{osc}=512\text{Khz}$ .
  - | 100ms settling time is required between samples.
  - |  $F_{osc} \geq 512\text{KHz}$  required to minimize LCD T3 interrupt latency.

```
// for(SampleDelay=0;SampleDelay<190*Fosc;  
    SampleDelay++);
```



MICROCHIP

## Lab: Tip 5 Continued

```
OSCCON_sh = OSCCON;           //Store previous value
OSCCON = 0b00110000;         //512KHz INTRC, Sleep
while(!OSCCONbits.IOFS);

TMR2 = 0;                      //~100ms rollover time
PR2 = 0xFF;

T2CON = 0b00100101;          //1:5 post, 1:4 pre, on
PIR1bits.TMR2IF = 0;
PIE1bits.TMR2IE = 1;
while(T2CONbits.TMR2ON)
    Sleep();
PIE1bits.TMR2IE = 0;
OSCCON = OSCCON_sh;          //Restore value
while(!OSCCONbits.IOFS);     //Wait for switch
```



# Class Summary

Timer1, BOR, HLVD,  
LCD, USB

Extended  
Instruction set

PIC18  
Devices

Your Application

Clock  
System

Power Managed  
Modes

nanoWatt  
Technology



# Appendix

## Demo Boards

- | PICDEM™ 4
- | PICDEM MC
- | PICDEM Full Speed USB

## Application Notes

- | Internal RC Oscillator Calibration AN244
- | USB Power Management AN<sub>TBD</sub>
- | RS-232 to USB Migration AN<sub>TBD</sub>
- | USB Mass Storage Class AN<sub>TBD</sub>
- | J1939 C Library AN930
- | Brushless DC Motor Control AN899