System Management Workshop

309SMW

Abstract

System Management architectures monitor system health characteristics such as temperatures, voltage levels, fan speed, and chassis intrusion. These applications typically utilize independent I^2C devices like real-time clock calendar chips, serial EEPROM, analog-to-digital converters, and thermal fan controllers to monitor and log physical health characteristics. This course teaches, in lectures and demonstration, how PIC microcontrollers can be used to add value to these applications in terms of flexibility and cost. The course will revolve around the PICDEM System Management board which uses the MSSP address mask feature to implement four I^2C devices: a real time clock, serial EEPROM, analog-to-digital converter, and thermal fan controller in one single microcontroller.

When you leave here today you will know…

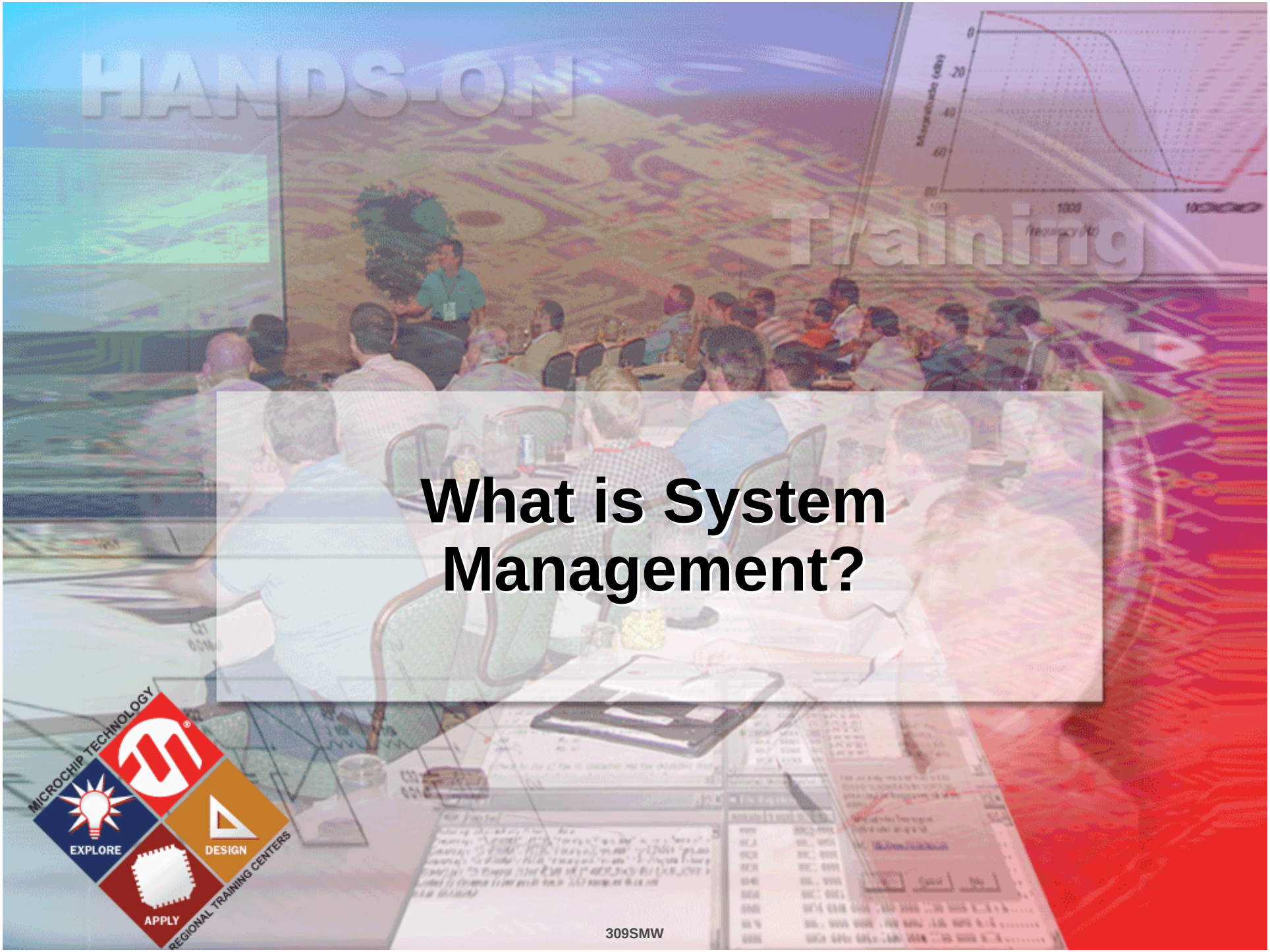
- How PIC® microcontrollers can be used to integrate system management functions
- How to use PIC microcontrollers to emulate a Serial EEPROM, Real-Time Clock/Calendar, Thermal Controller
- How to integrate multiple I²C™ devices in PIC microcontrollers

To get the most from this class

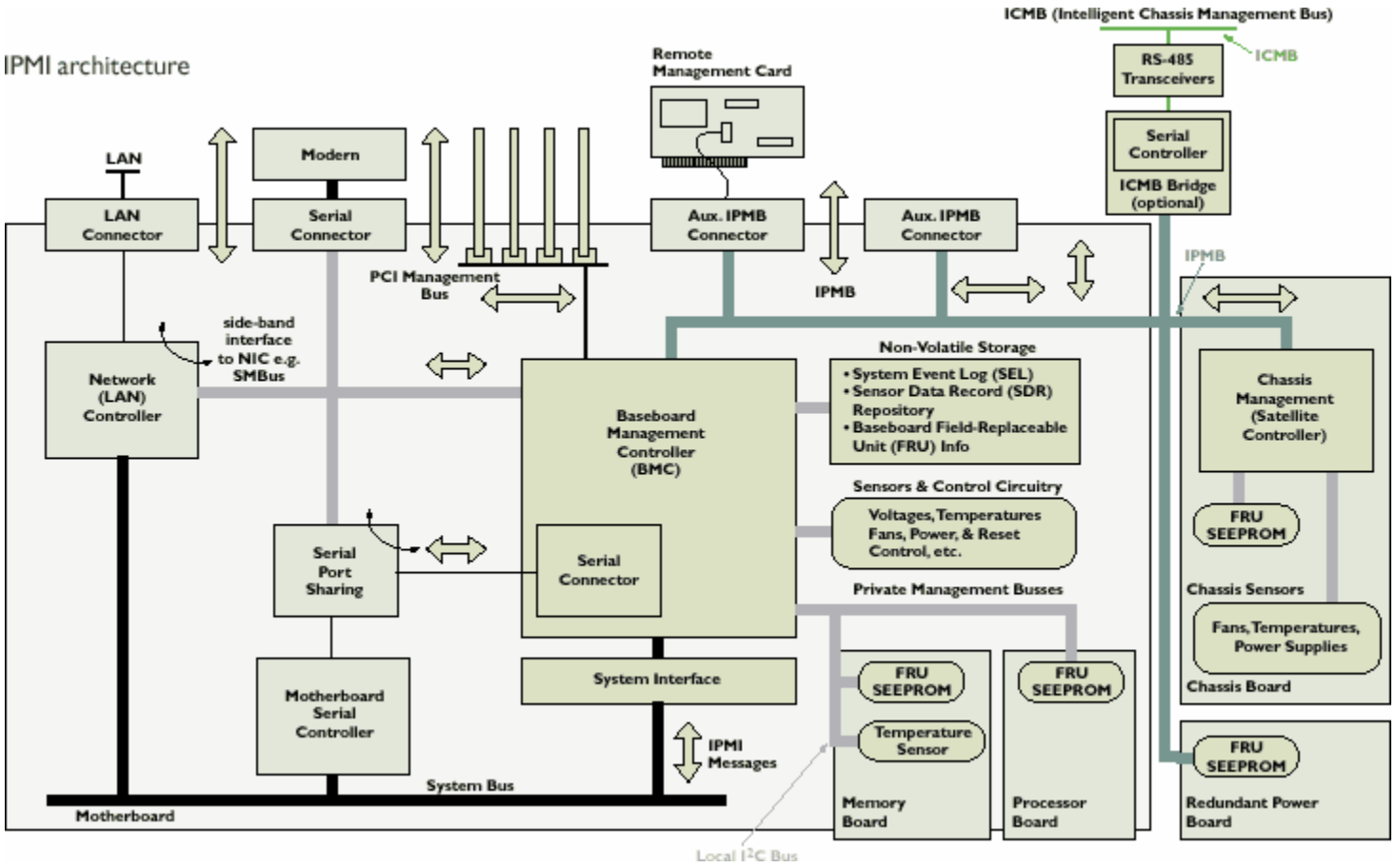
- You should be familiar with…
  - Mid-range PIC16 Architecture
  - Basics of Serial Communication
  - Common PIC® microcontroller peripherals
    - Timers
    - Analog-To-Digital
    - Comparators
    - Capture Compare PWM

Agenda

- What is System Management?
- Introduction to PICDEM™ System Management
- Introduction to PICkit™ Serial Analyzer
- Course Outline


HANDS-ON
Training
What is System Management?
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

Intelligent Platform Management Interface (IPMI)



Specification: http://www.intel.com/design/servers/ipmi/

System Management

- Requires many health monitoring functions:
  - Power Sequencing / Monitoring
  - Real-Time Clock (RTC)
  - Thermal Management
  - EEPROM

- Monitoring functions are implemented using several stand-alone devices

Where do PIC® microcontrollers fit in?

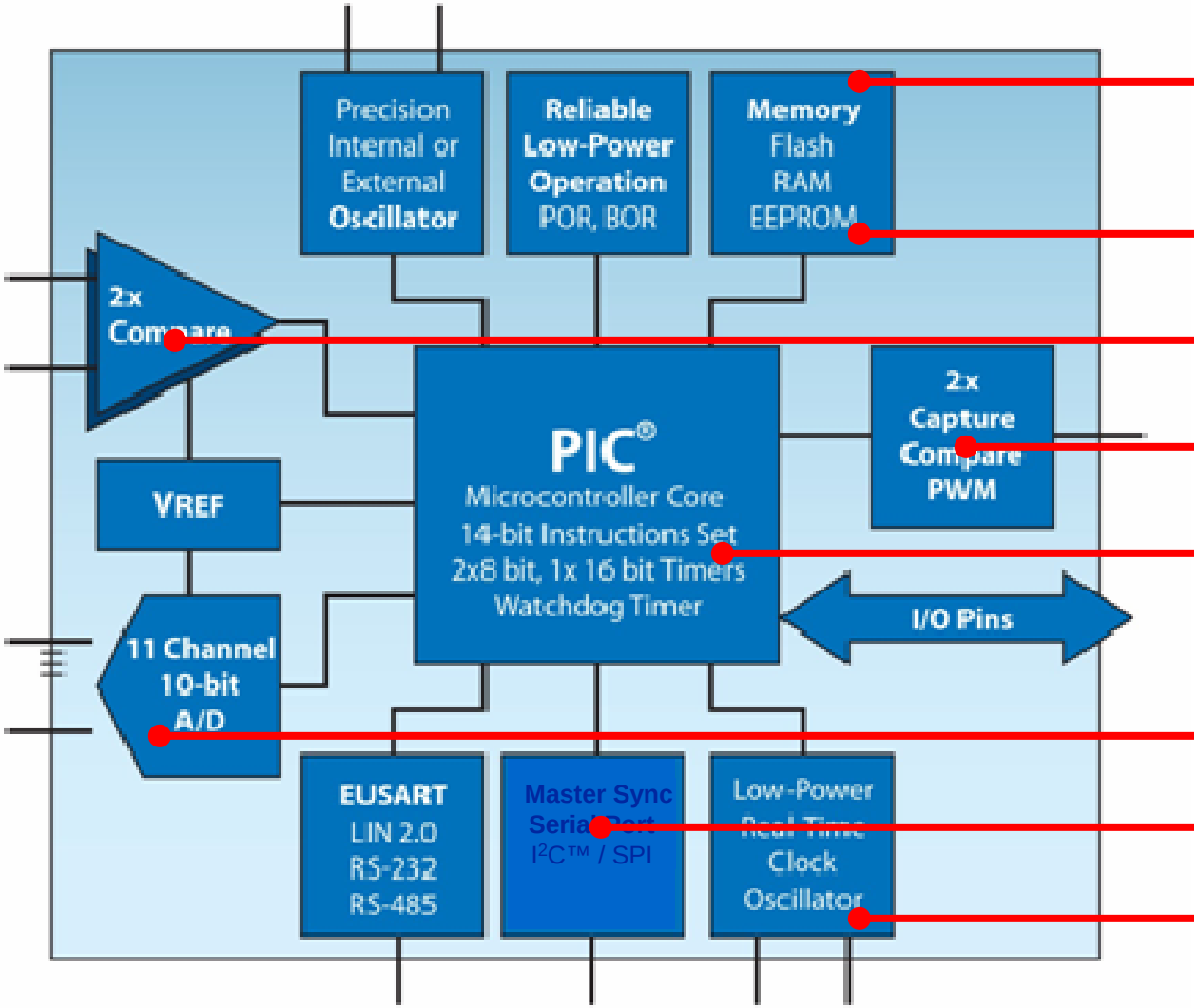
- Currently…
  - System designers use individual turnkey devices from numerous vendors
  - Stand-alone devices are not very flexible in meeting new specifications
  - Design cycles require more hardware redesign

Where do PIC® microcontrollers fit in?

- PIC microcontrollers can perform the following system management functions:
  - Power sequencing and start up control
  - Serial EEPROM
  - Real-Time Clock
  - Analog-to-Digital Converter
  - Thermal Management Fan Control

- Additional benefits: Integration, Programmability, Cost

PIC16F886
Typical Small Microcontroller



Reprogrammable Flash Memory 8K

Data EEPROM

Comparators

Control Peripherals

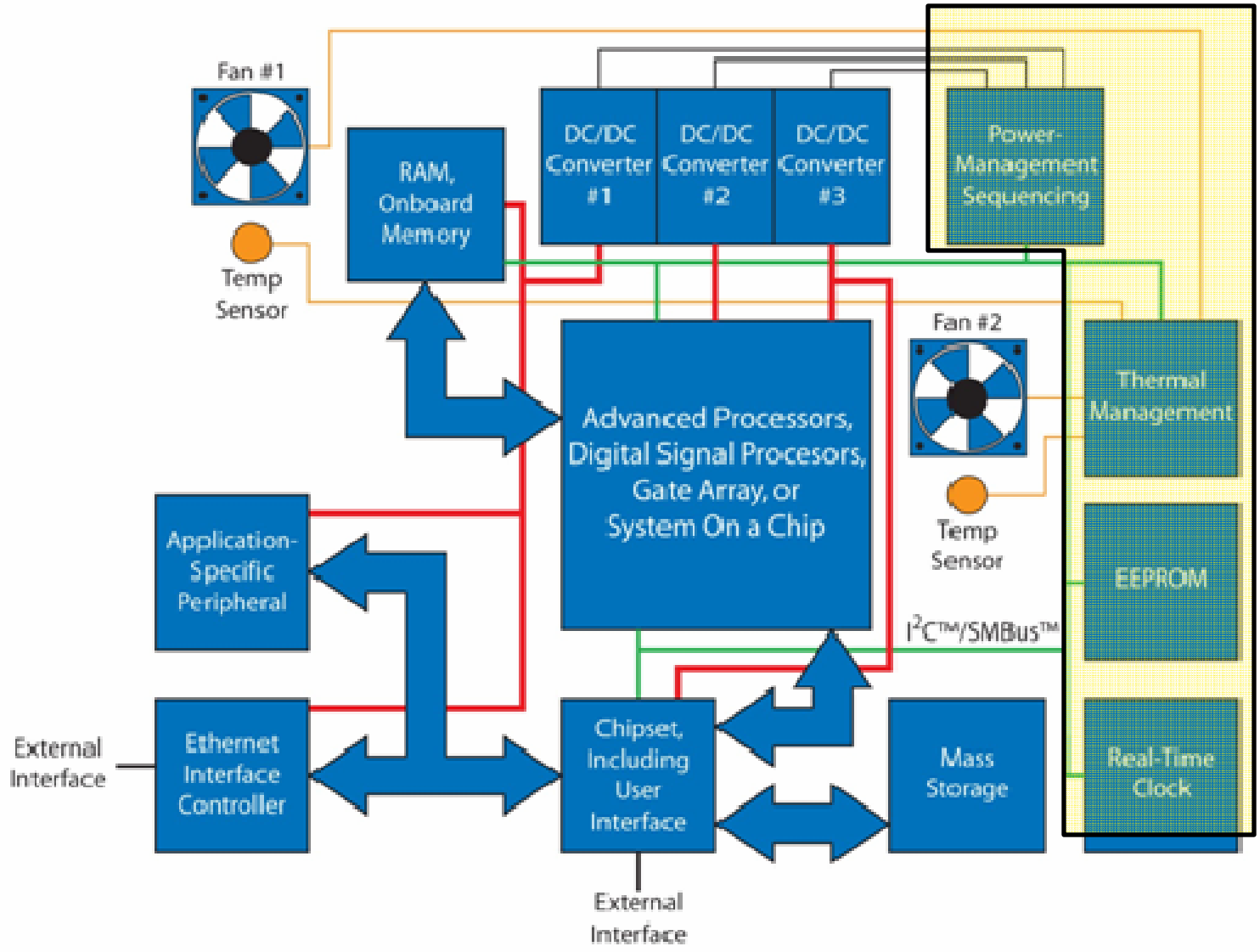
Programmable Processor

A/D Converter

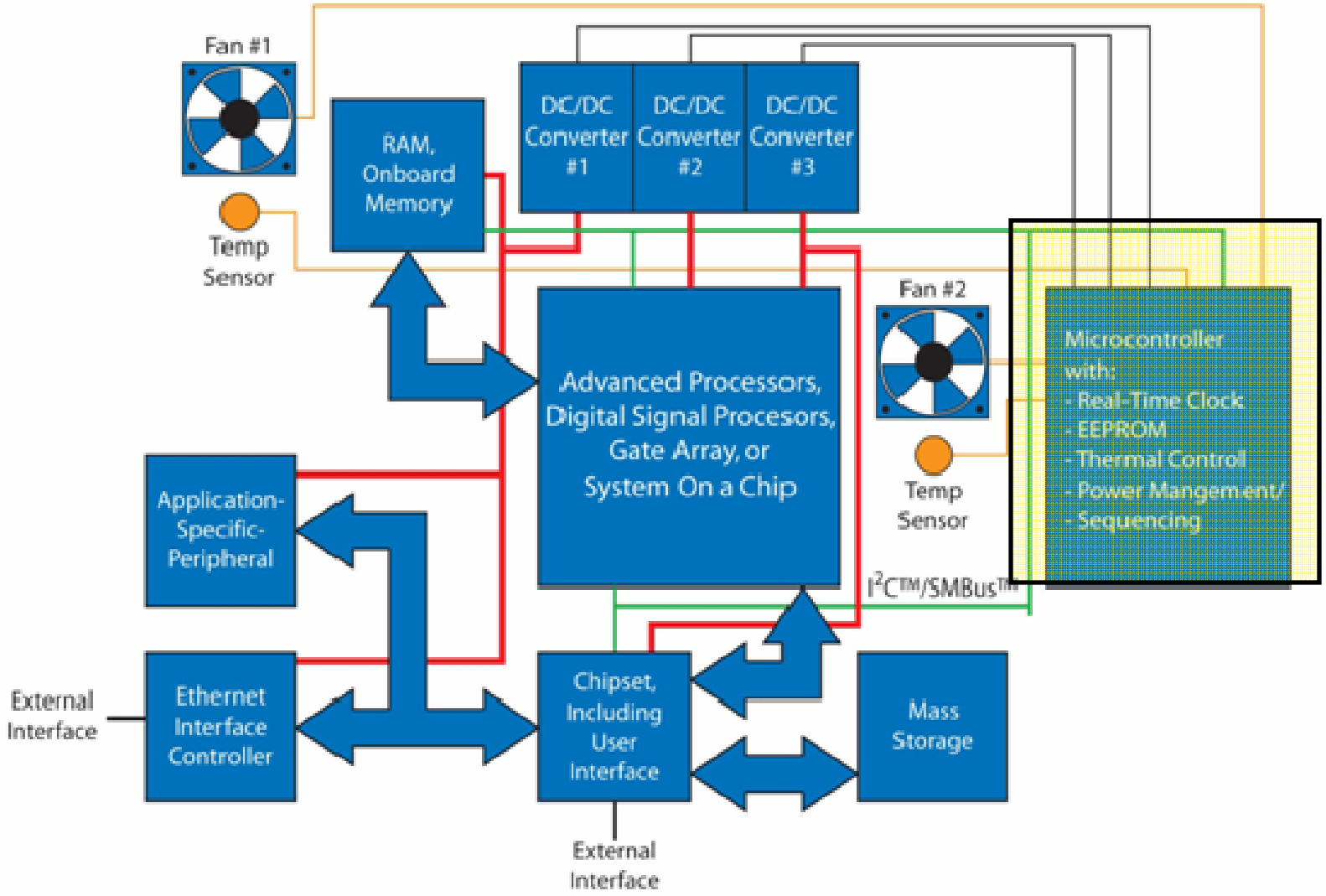
Serial Comm.

Real-Time Clock Oscillator

Current System Management Block Diagram



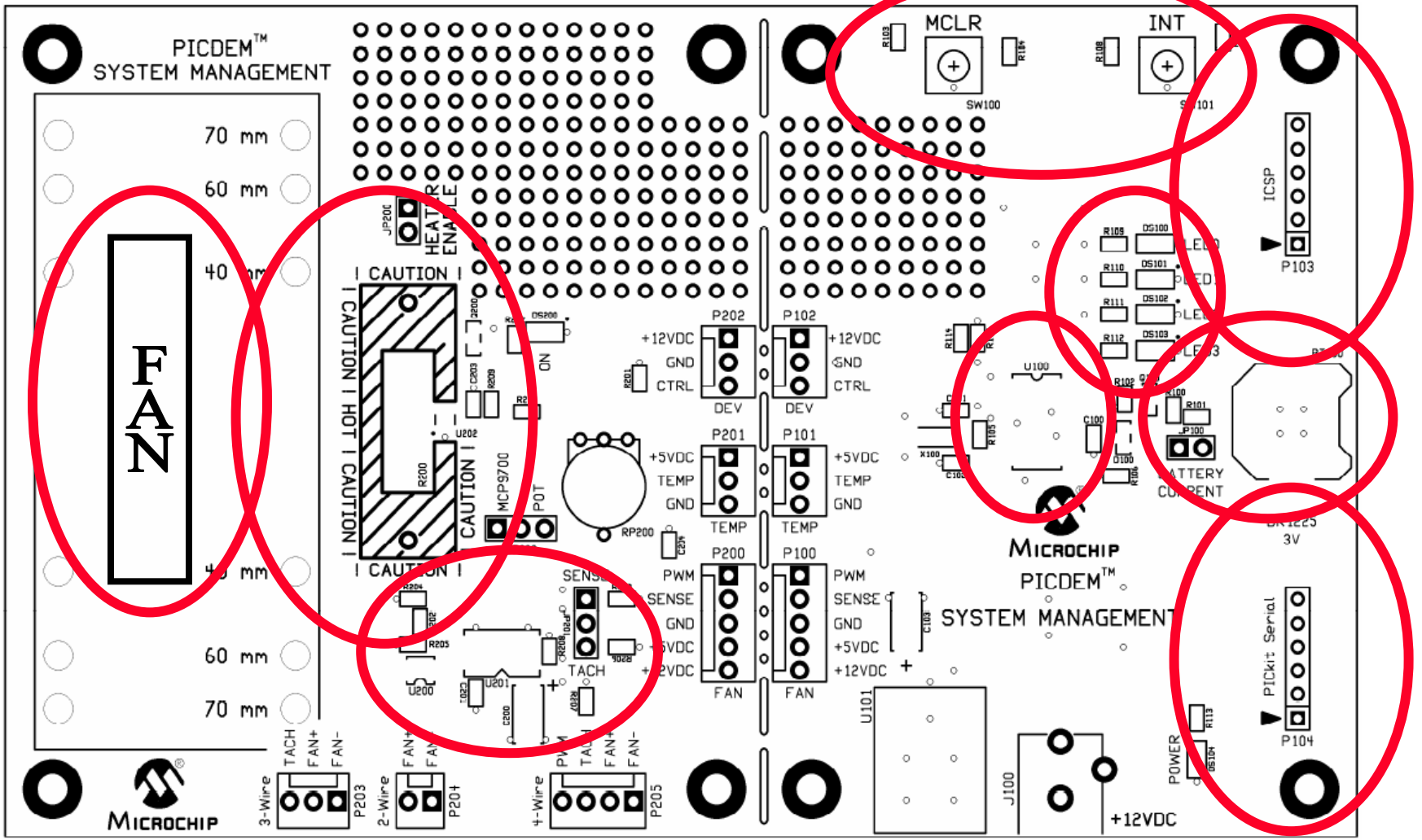
Revised System Management Block Diagram



PICDEM™ System Management

309SMW

PICDEM™ System Management



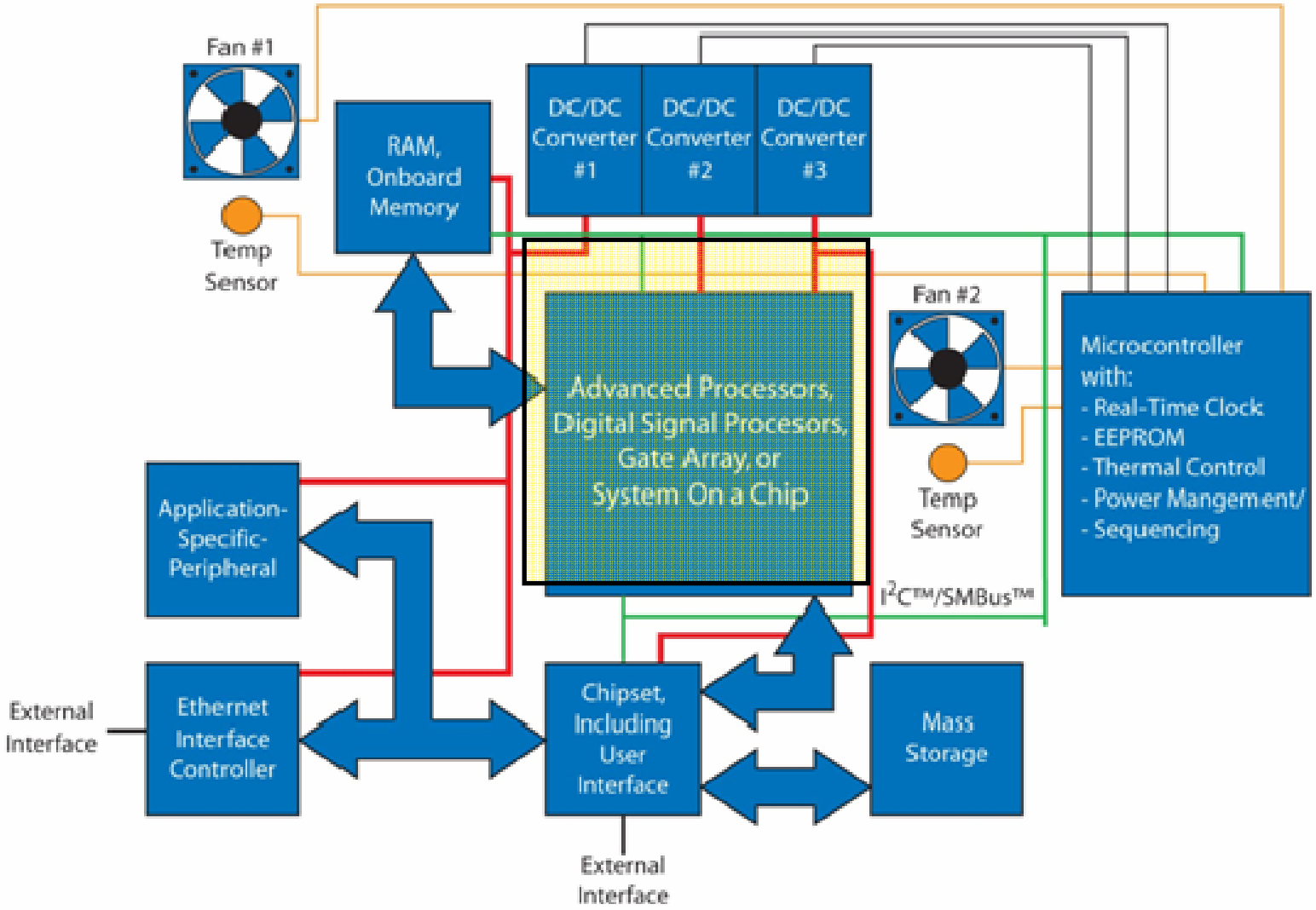
PIC16F886

Emulated I²C™ Device	PIC16F886 Peripherals
Serial EEPROM	Data EEPROM, Master Synchronous Serial Port (MSSP)
Real-Time Clock	Low-Power Timer 1 Oscillator, Timer 1, MSSP
Thermal Controller	Timer 1, Capture/Compare/PWM, Comparator, MSSP
Analog-to-Digital Converter	Analog-to-Digital Converter, MSSP

PICkit™ Serial Analyzer

309SMW

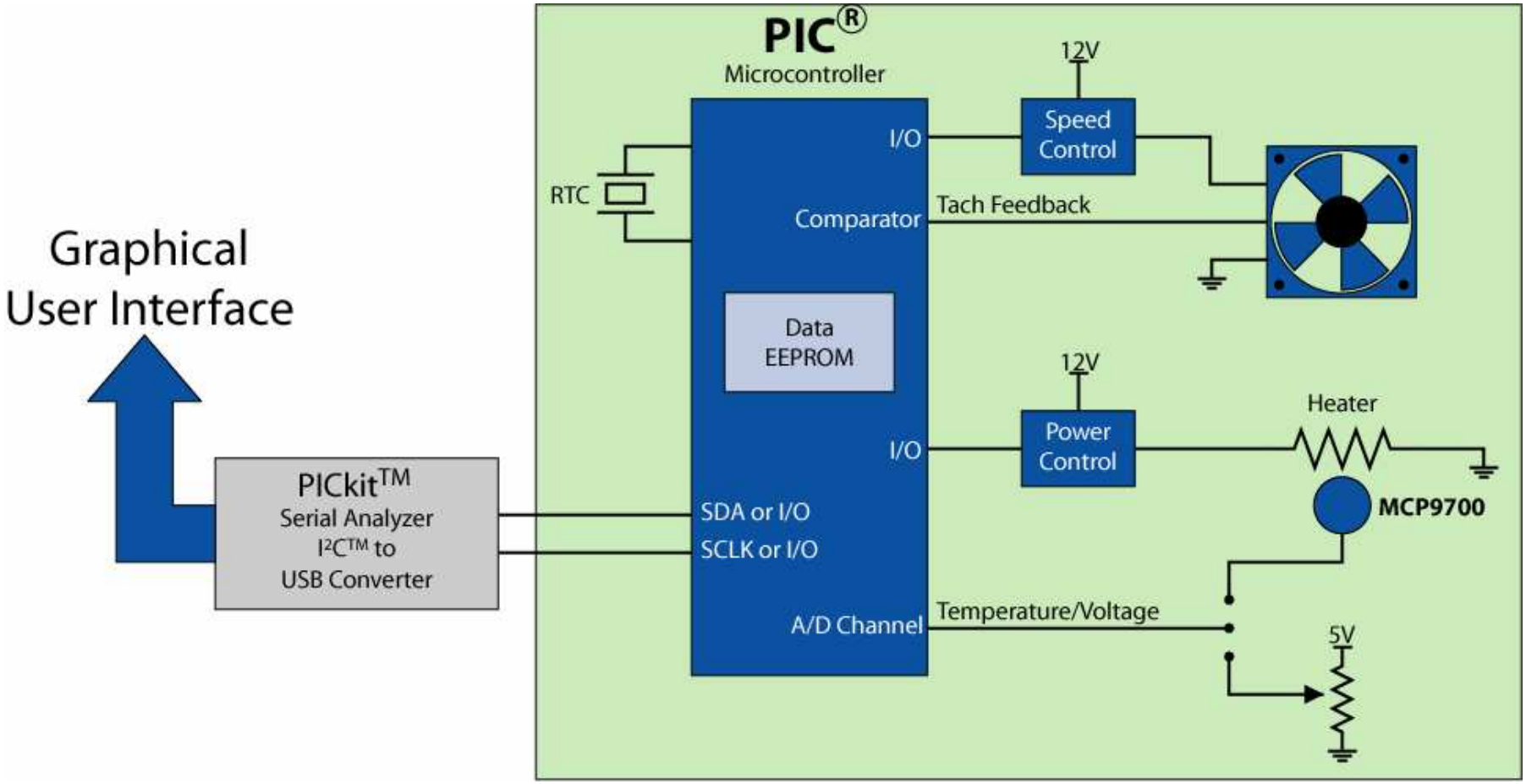
Recall: System Management Block Diagram



PICkit™ Serial Analyzer & PICDEM™ System Management

- PICkit Serial Analyzer acts as the I^2C™ bus Master
- Connects to PICDEM System Management Serial lines
- Monitors and controls the emulated I^2C devices on the PICDEM System Management Board

PICDEM™ System Management Application



Course Outline

- System Management Introduction
- I²C™ Serial Communications
- Serial EEPROM
- Real-Time Clock
- Thermal Control
- I²C Device Integration

Summary

- What is System Management?
- Introduction to PICDEM™ System Management
- Introduction to PICkit™ Serial Analyzer
- Course Outline

- Questions?

I²C™ Module

309SMW

Module Learning Objectives

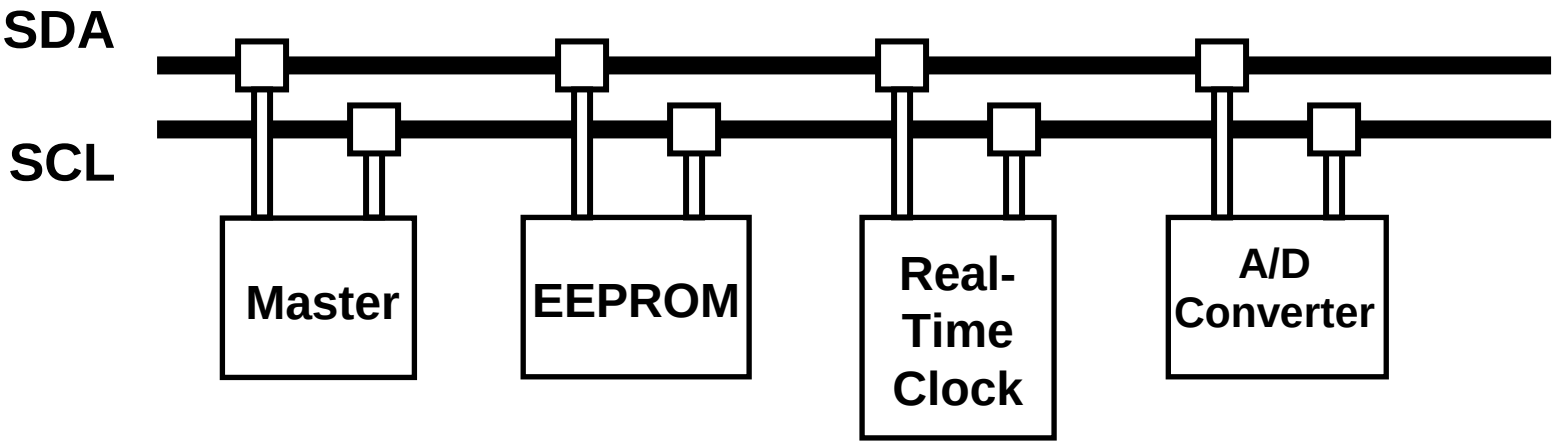
- Learning Objectives:
  - Explain I²C™ hardware architecture
  - Explain I²C communications elements
  - Explain I²C message formats

Module Agenda

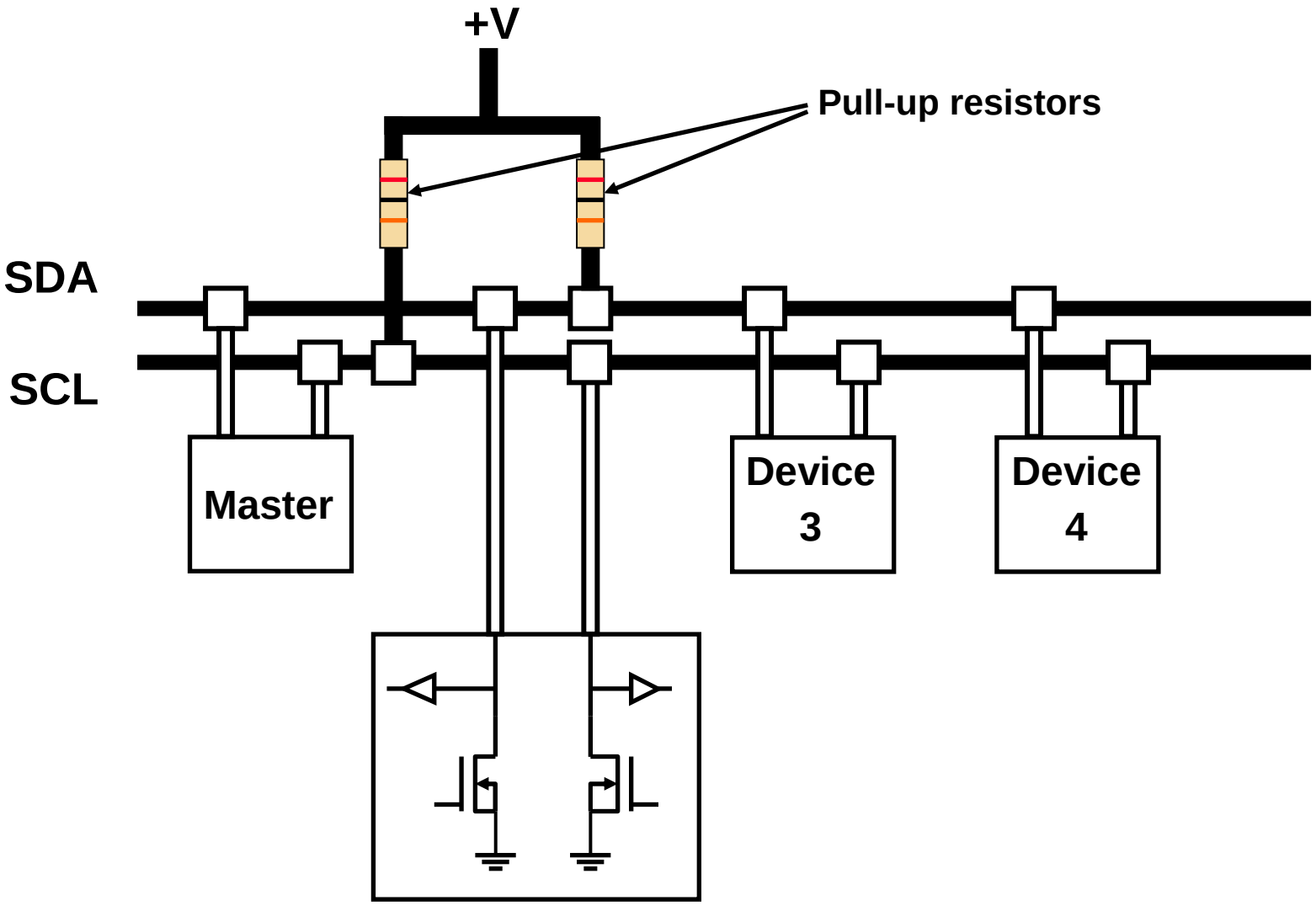
- I²C™
  - Introduction
  - Hardware Overview
  - Communication Elements
  - Message Formatting
- Summary

I²C™ Introduction

- NXP(Philips) Inter-Integrated Circuit (I²C)
  - Specification: www.standardics.nxp.com/literature/books/i2c/pdf/i2c.bus.specification.pdf
- Synchronous (not like RS232)
- Master-Slave Protocol
- Bidirectional
- Half Duplex Serial Interface

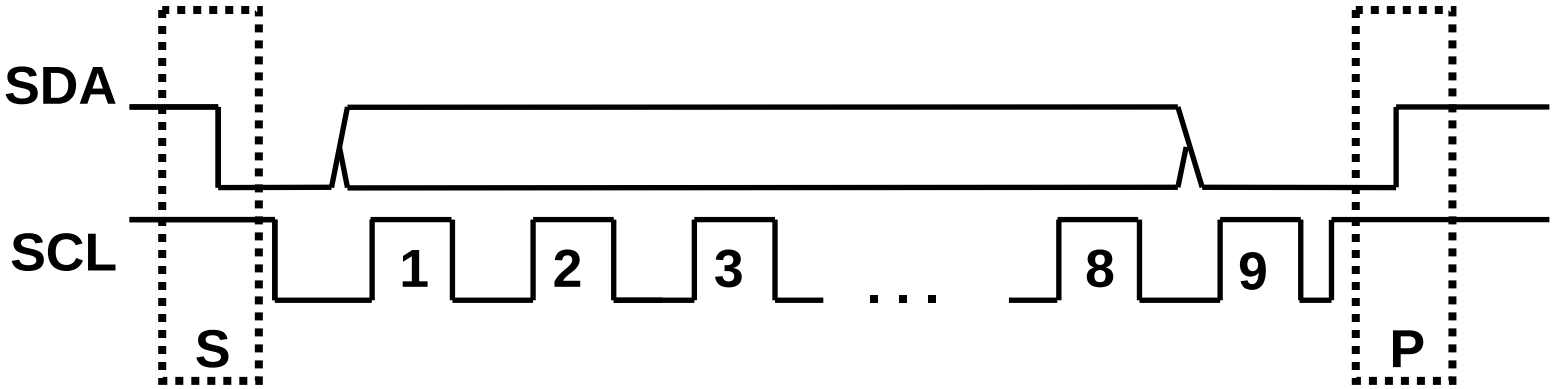


I²C™ Hardware Overview



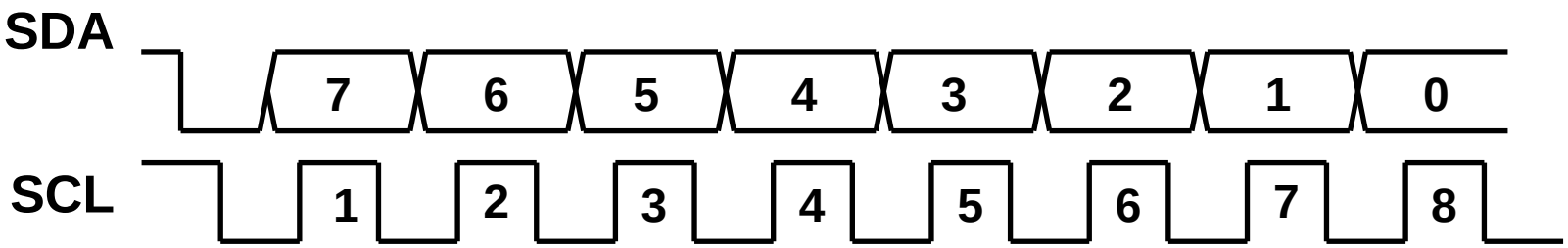
I²C™ Communication Elements

- **Start** and **Stop** Condition:
  - Generated by Master
  - After Start Condition: Bus is Busy
  - After Stop Condition: Bus is Free



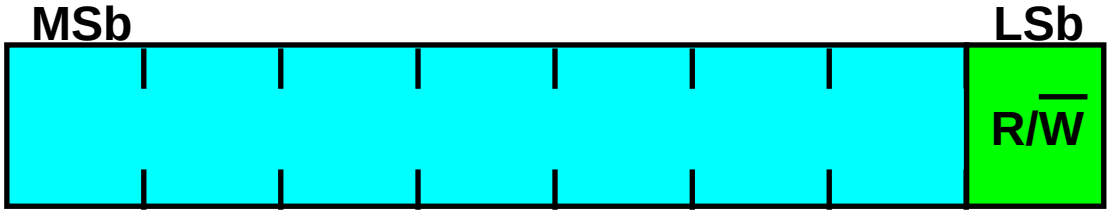
I²C™ Communication Elements

- **Data Transfer**
  - 8 bits of data are sent on the bus
  - Data is valid when SCL is high
  - Types of Data: Control Byte, Data Byte



I²C™ Communication Elements

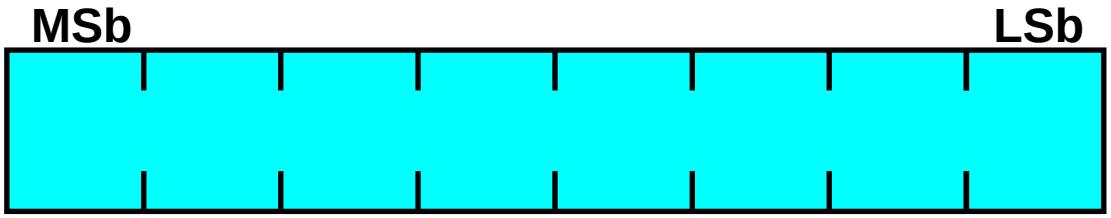
- Slave Address
  - Slave addresses are 7-Bit (10-bit also exists)
  - **Slave address** is the first byte sent out following the start condition
  - 7-bit Slave Address, Read Bit



Slave Address

I²C™ Communication Elements

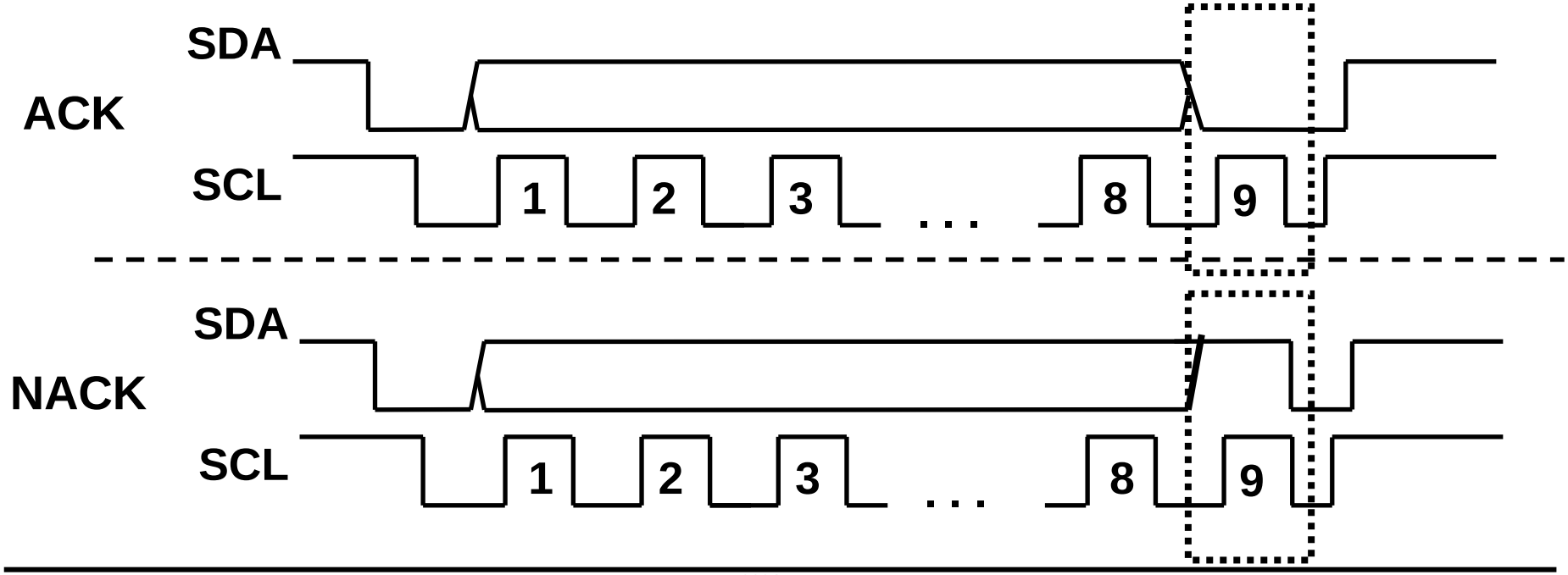
- Data Byte
  - Data bytes are 8 bits
  - Data to be read or written to I²C device
  - Word address to be written



Data Byte

I²C™ Communication Elements

- **Acknowledge/No Acknowledge**
  - Indicates success or failure of a data transmission or the continuation of a transfer
  - Generated by master or slave by holding the SDA low on the 9th clock pulse



I²C™ Message Formatting

- Master **Write** to Slave

From master to slave

From slave to master

A = **Acknowledge**

S = **Start** Condition

P = **Stop** Condition

I²C™ Message Formatting

- Master **Read** from Slave

From master to slave

From slave to master

A = **Acknowledge**

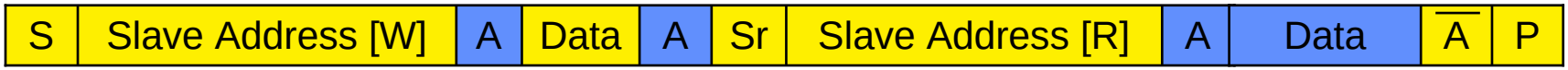
$\overline{A}$ = **No Acknolwedge**

S = **Start** Condition

P = **Stop** Condition

I²C™ Message Formatting

- Master **Combination Read** from Slave



From master to slave

From slave to master

A = **Acknowledge**

$\overline{A}$ = **No Acknowledge**

S = **Start**

Sr= **Repeated Start**

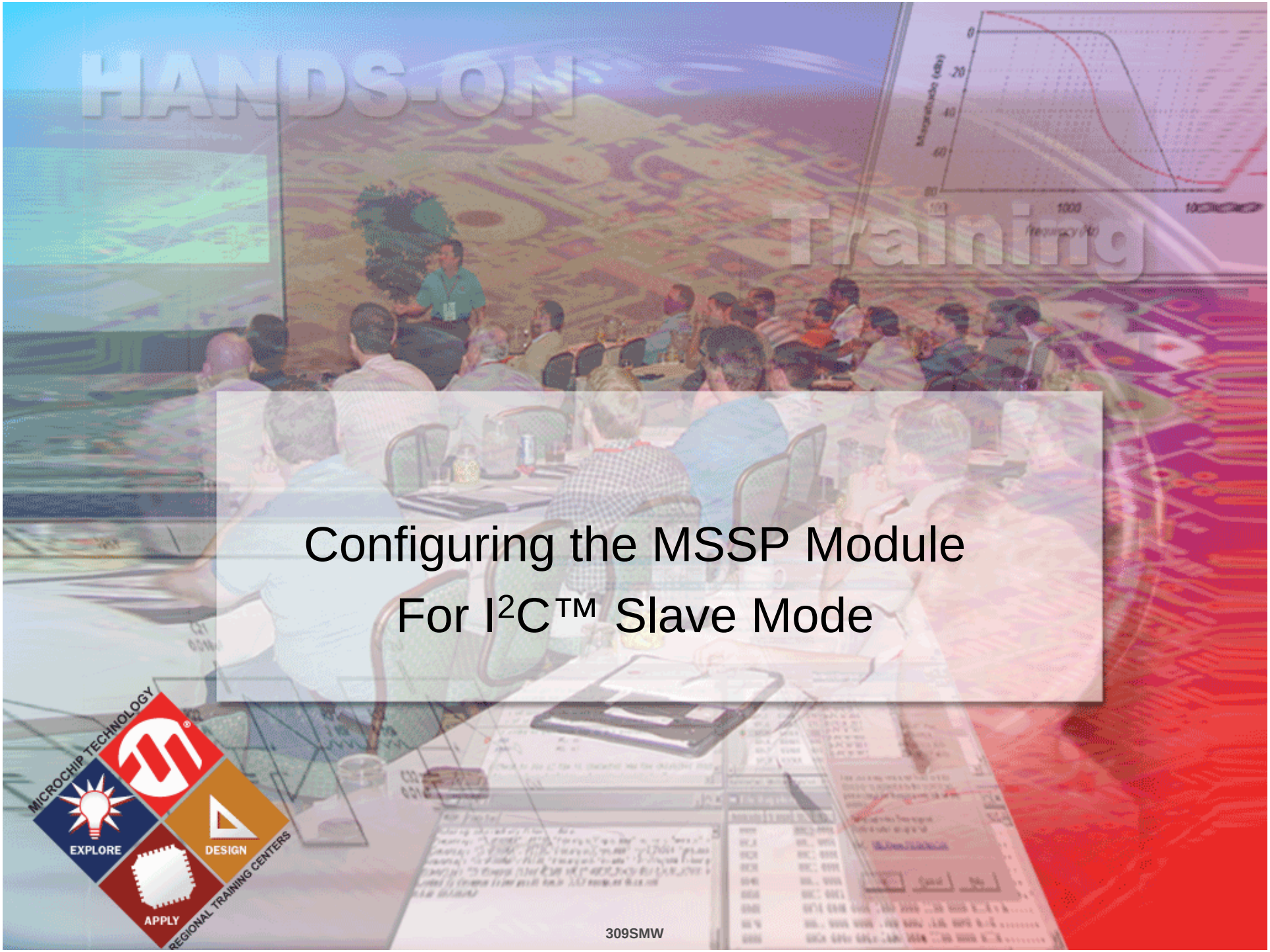
P = **Stop**

I²C™ Review

- I²C Introduction
- Hardware Overview
- Communication Elements
- Message Formatting

I²C™ Module

Questions?


HANDS-ON
Training
Configuring the MSSP Module
For I²C™ Slave Mode
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

Module Learning Objectives

- Learning Objectives:
  - Configure the Master Synchronous Serial Port module for I²C™ Slave Mode on PIC16F886
  - Explain the Master Synchronous Serial Port Address Mask Feature

- Prerequisites:
  - I²C Module

Module Agenda

In this module we will discuss the following:

- Master Synchronous Serial Port
  - I²C™ Slave Mode Configuration
  - MSSP Operation
  - Address Masking
- Summary

Master Synchronous Serial Port (MSSP) for Slave Mode I²C™

309SMW

MSSP Introduction

- MSSP I²C™ Mode Supports:
  - Master/Multi-Master Mode
  - Slave Mode
  - 7 or 10-bit Addressing
- Configurations:
  - I²C Master Mode
  - I²C Slave Mode
  - I²C Slave Mode with Start and Stop bit Interrupts enabled
  - I²C firmware controlled master, slave is idle

“SSP Configuration” Register

SSPCON Register

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

SSPEN: Synchronous Serial Port Enable Bit
1 = Configures SDA and SCL as serial port pins.

SSPM<3:0>: Synchronous Serial Port Mode Select Bits
1110 = I^2C™ Slave Mode, 7-Bit Address with Start/Stop bit interrupts enabled

WCOL = Write Collision Detect Bit
SSPOV = Receive Overflow Indicator
CKP = Clock Polarity Bit

“SSP Status” Register

SSPSTAT Register

SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF

D / $\overline{A}$ = Data / Not Address Bit
P = Stop Bit
S = Start Bit
R / $\overline{W}$ = Read / Not Write Bit
BF = Buffer Full Bit

SMP = Slew Rate Control for High Speed Enable

CKE = SPI Clock Edge Select
UA = Update Address Bit (for 10-bit address mode)

Configuring SSPMSK Register

SSPCON Register

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

SSPM<3:0>: Synchronous Serial Port Mode Select Bits
1001 = Load SSPMSK register at SSPADD SFR Address

SSPMSK Register

MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0

- Two Step Process
  1. **SSPCON, SSPM<3:0>** = 1001
  2. Accessed through **SSPADD** register

Configuring SSPADD Register

SSPCON Register

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

SSPM<3:0>: Synchronous Serial Port Mode Select Bits
1110 = I^2C™ Slave Mode, 7-Bit Address with Start/Stop bit interrupts enabled

SSPADD Register

ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0

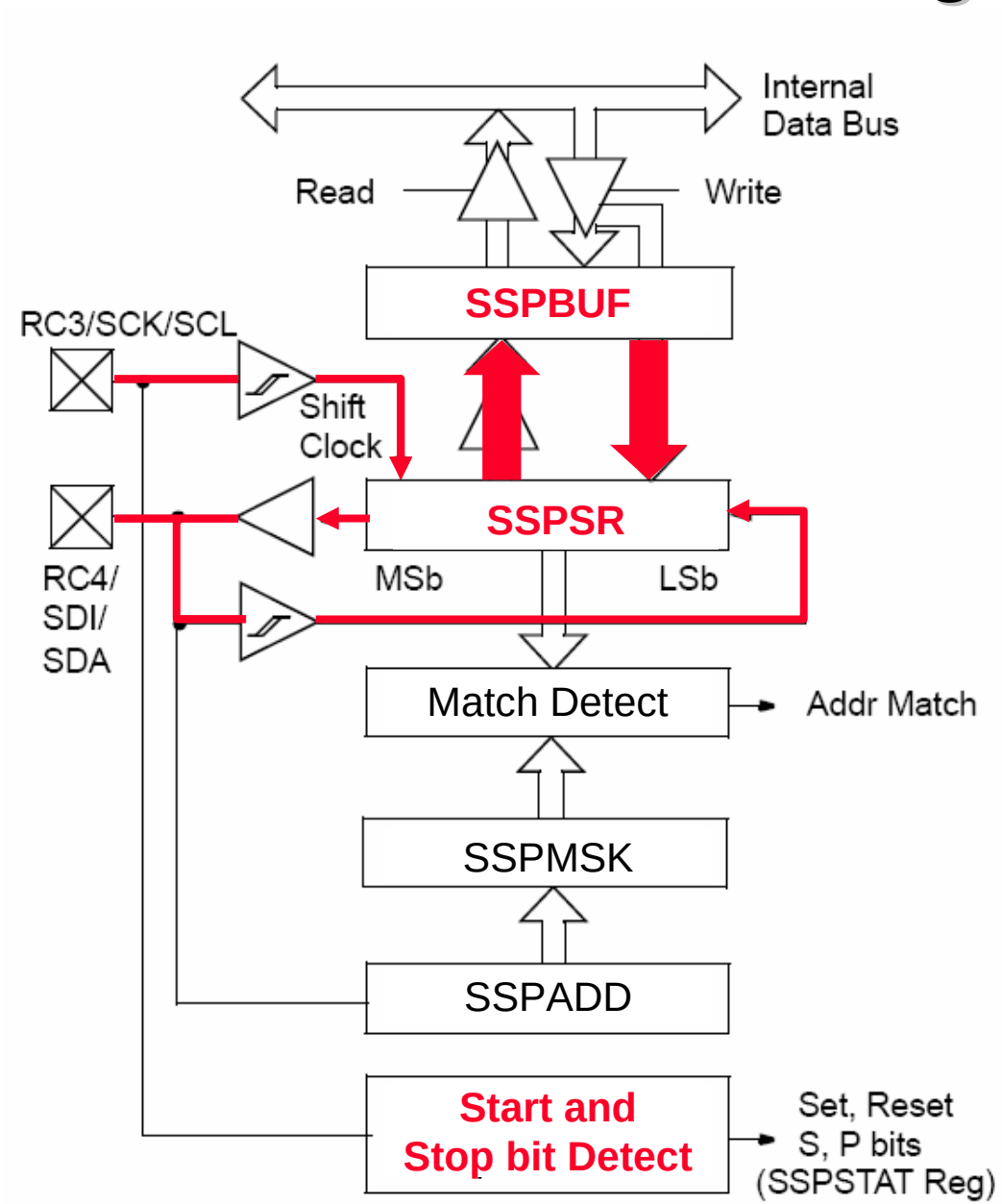
Reserve Addresses

B‘0000 000’ – General Call Address
B‘0000 001’ – CBUS address
B‘0000 010’ – Reserved for different bus format
B‘0000 011’ – Future Purposes
B‘0000 1XX’ – Hs-mode master code
B‘1111 1XX’ – Future Purposes
B‘1111 0XX’ – 10-bit Slave Address

MSSP I²C™ Block Diagram

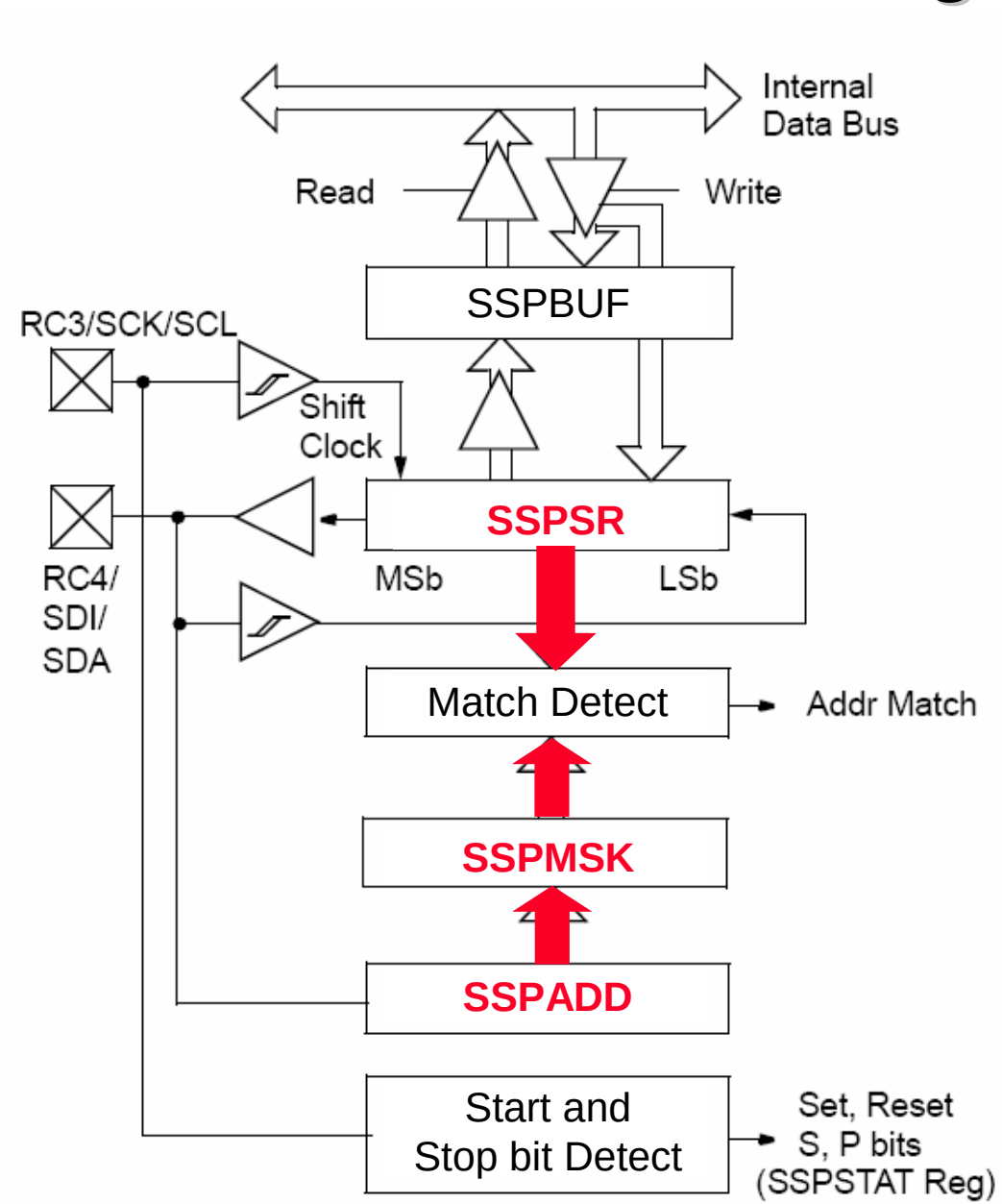
Master Write

Master Read



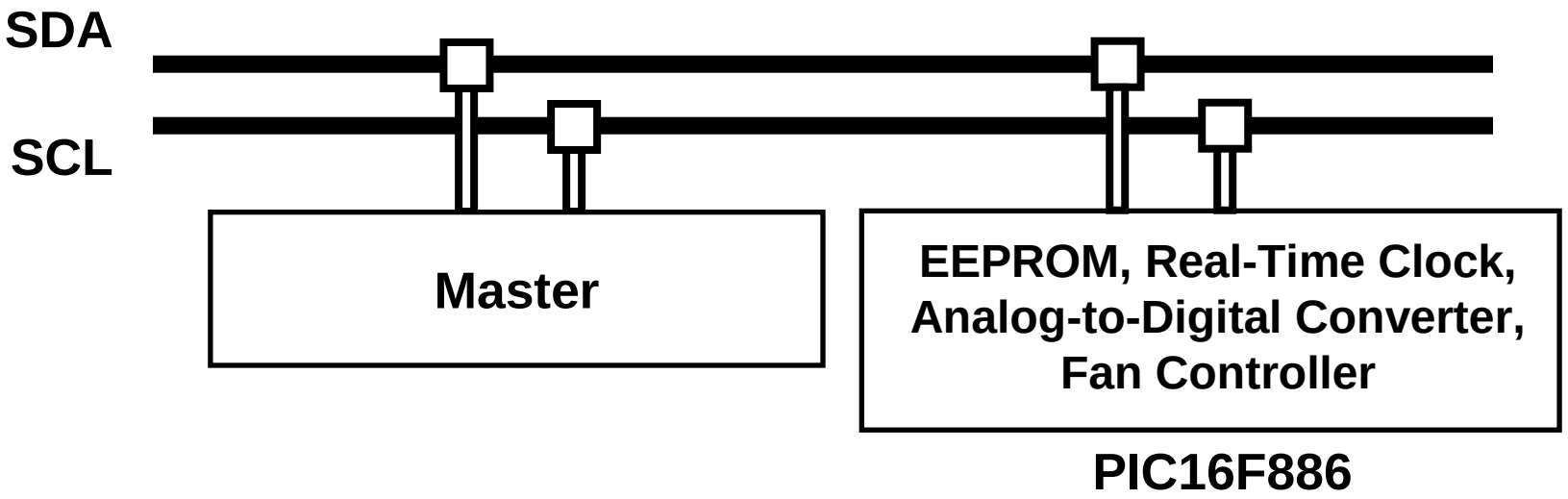
MSSP I²C™ Block Diagram

Address Match



Address Masking Feature

- **What is it?**
  - It allows a PIC® MCU to ACK more than one address.
- **What is it good for?**
  - Integrating multiple I²C™ devices in one PIC MCU



Address Masking Feature Example

- A “0” in the SSPMSK register ignores the corresponding address bit

- Example: SSPADD = 0b1010111R

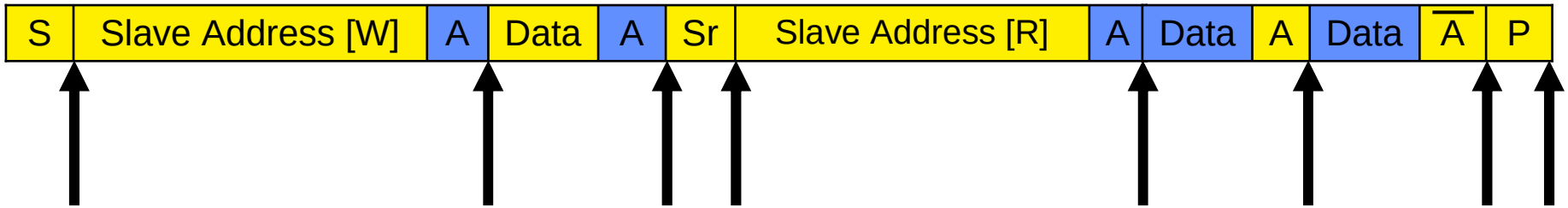
  SSPMSK = 0b1111100R

- Module will respond to addresses:
  - 0b1010111R, 0b1010110R, 0b1010101R, 0b1010100R

I²C™ Interrupt Events

PIR1 Register

--	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF



- Start and Stop Conditions
- Data Transmit and Receive

Interrupt Status and Enable

PIE1 Register

--	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

PIR1 Register

--	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

INTCON Register

GIE	PEIE	T0IE	INTE	RABIE	T0IF	INTF	RABIF

MSSP Review

- MSSP Introduction
- I²C™ Slave Mode Configuration
- Address Masking

I²C™ Module

Questions?

I²C™ Slave Mode Firmware Module

309SMW

Module Learning Objectives

- Learning Objectives:
  - Explain the I²C™ Slave Mode Firmware & State Machine

- Prerequisites:
  - I²C Module
  - MSSP I²C Slave Mode Configuration Module

Module Agenda

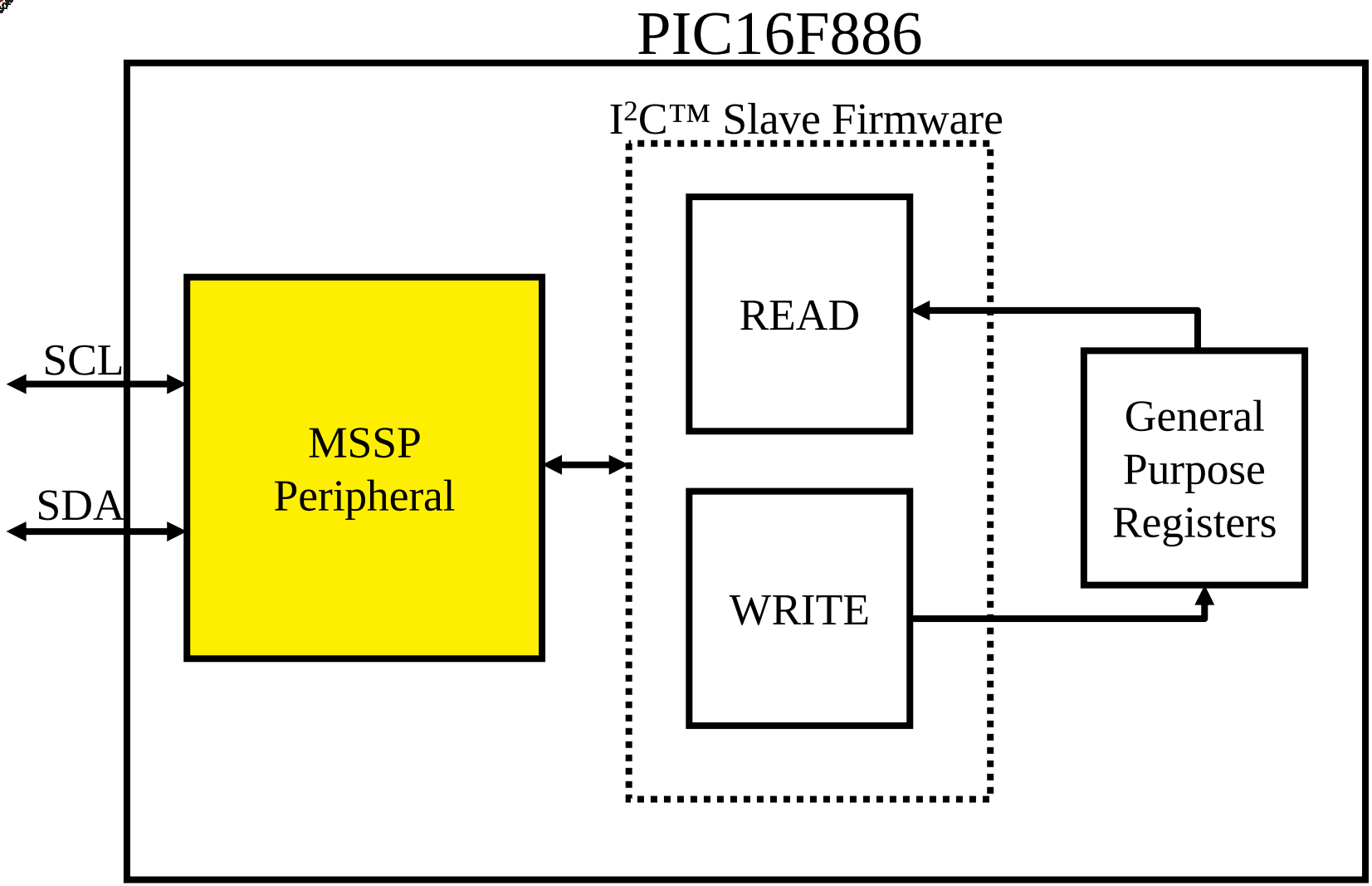
In the next hour we will discuss the following:

- I²C™ Slave Mode Firmware
- Lab 1: Introduction to the PICDEM™ System Management and PICkit™ Serial Analyzer
- Summary

I²C™ Slave Mode Firmware

309SMW

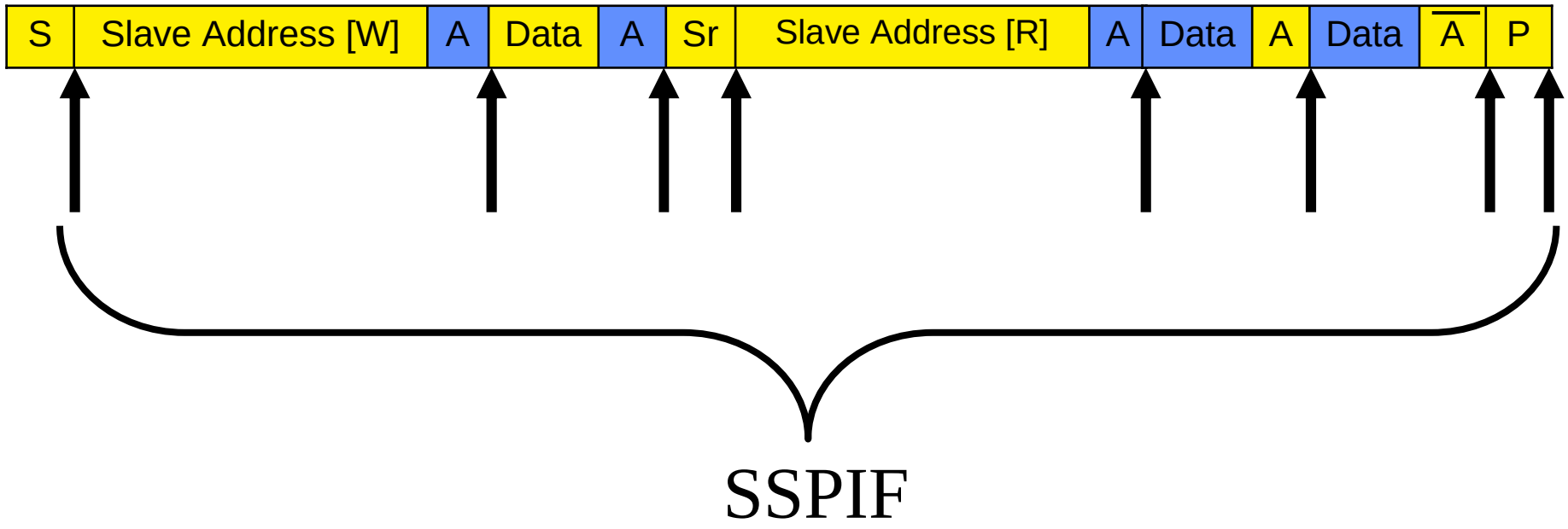
I²C™ Slave Mode Firmware Overview



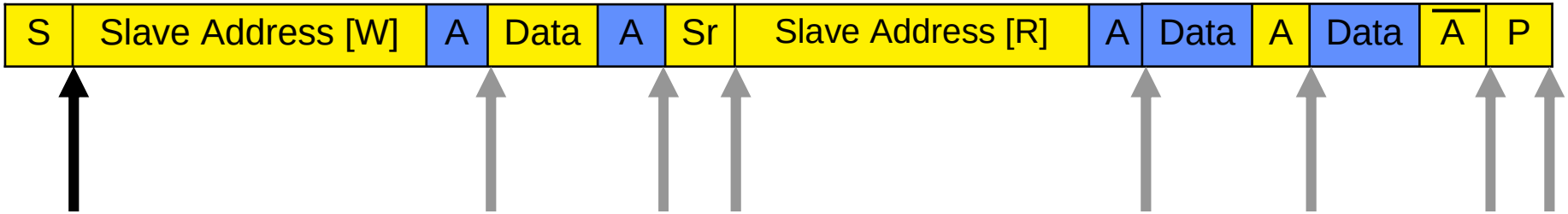
I²C™ Slave Mode Firmware Overview

- Sequential Process
- An event is a:
  1. Start or Restart Condition
  2. Master Write Address
  3. Master Read Address
  4. Master Write Data
  5. Master Read Data
  6. Stop Condition
- **SSPIF** bit indicates when an event has occurred
- Events will be identified by bits in the **SSPSTAT** register

I²C™ Slave Mode Firmware Overview



Start Condition

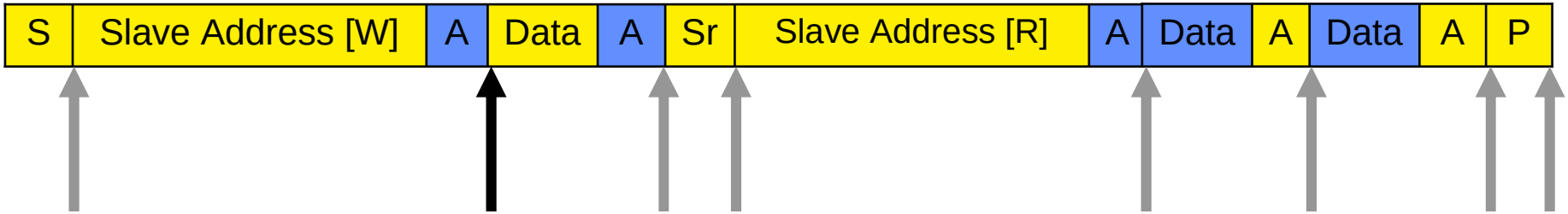


SSPSTAT Bits

- **S = 1**

Master Write Address

- Master has sent a **Write** request with a matching slave address

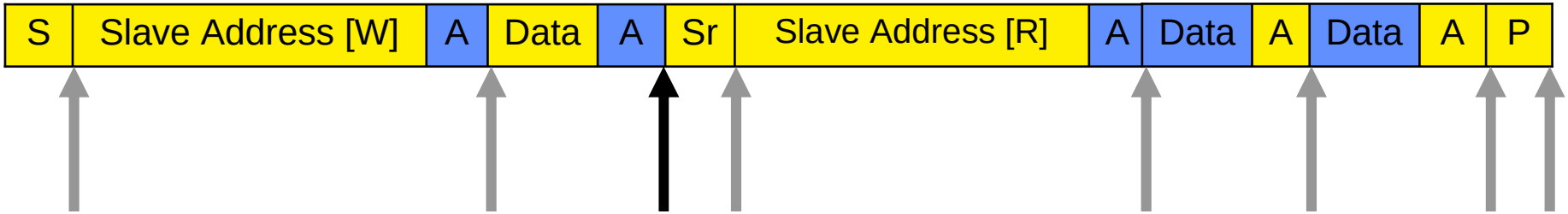


SSPSTAT Bits

- **D/$\overline{A}$ = 0**
- **R/$\overline{W}$ = 0**
- **BF = 1**

Master Write Data

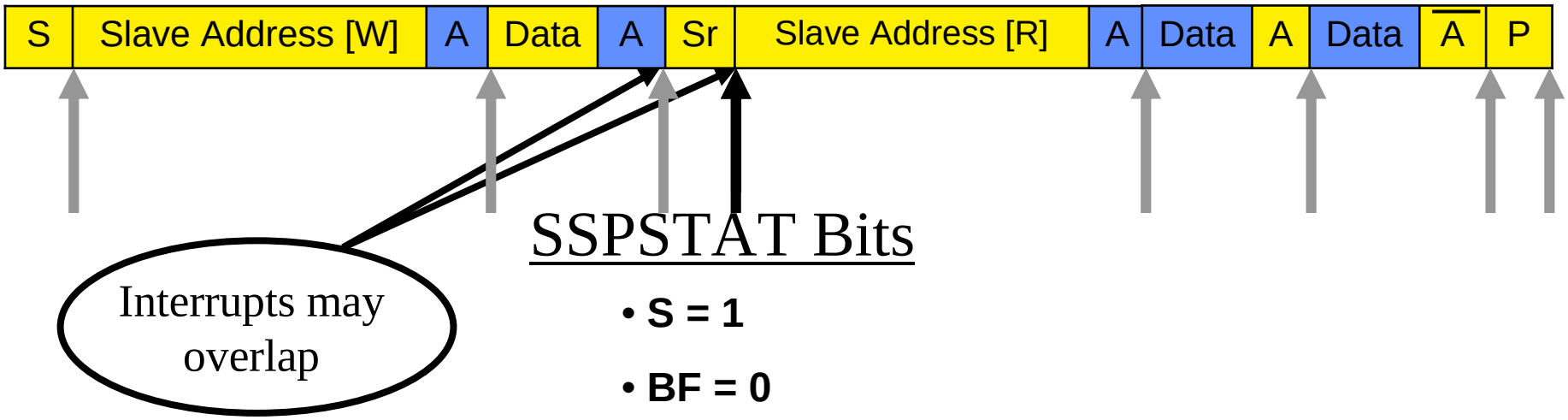
- Master has sent a **Write** request with a matching slave address



SSPSTAT Bits

- **D/$\overline{A}$ = 1**
- **R/$\overline{W}$ = 0**
- **BF = 1**

Restart Condition



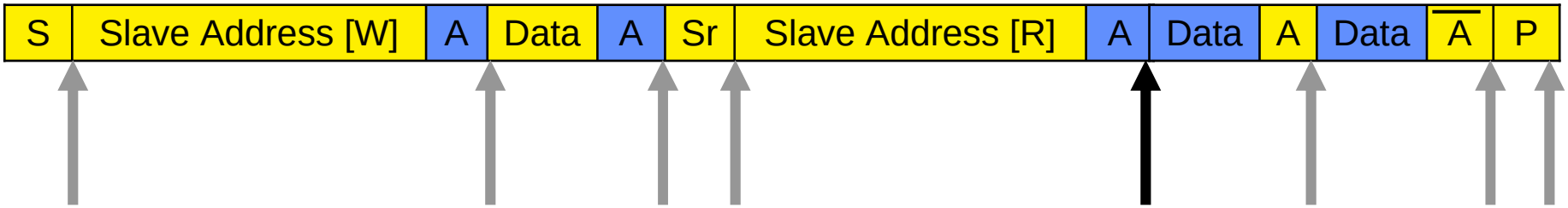
SSPSTAT Bits

- **S = 1**
- **BF = 0**
- **R/$\overline{W}$ = 0**

- **Check Buffer Full Flag & Read Flag**

Master Read Address

- Master has sent a **read** request with a matching slave address

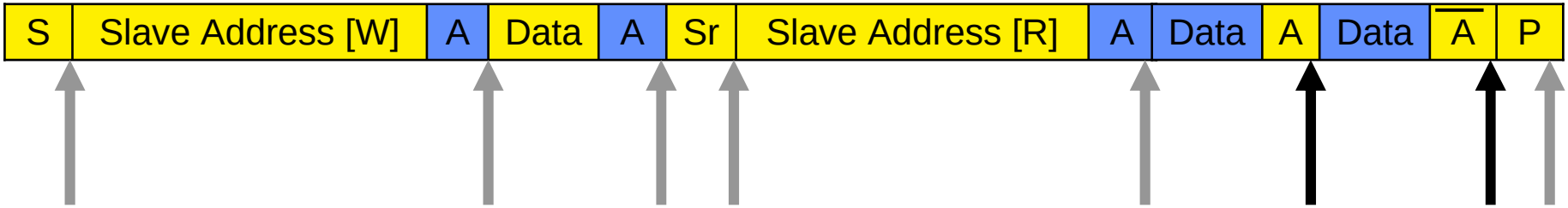


SSPSTAT Bits

- **D/$\overline{A}$ = 0**
- **R/$\overline{W}$ = 1**
- **BF = 1**

Master Read Data

- Master has read a **data** byte after a matching slave address and read request has been sent

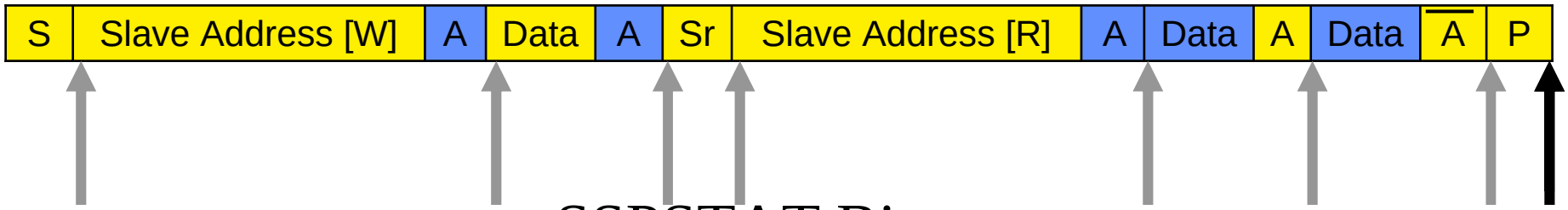


SSPSTAT Bits

- **D/$\overline{A}$ = 1**
- **R/$\overline{W}$ = 1**
- **BF = 0**

Stop Condition

- Indicates:
  - The end of communication

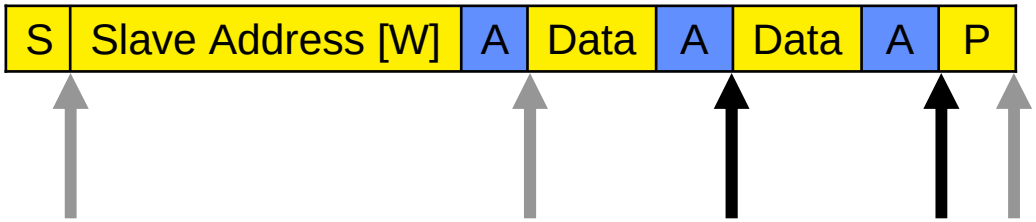


SSPSTAT Bits

- **P = 1**

Master Write Data

- Master has sent a **data** byte following a **write** request



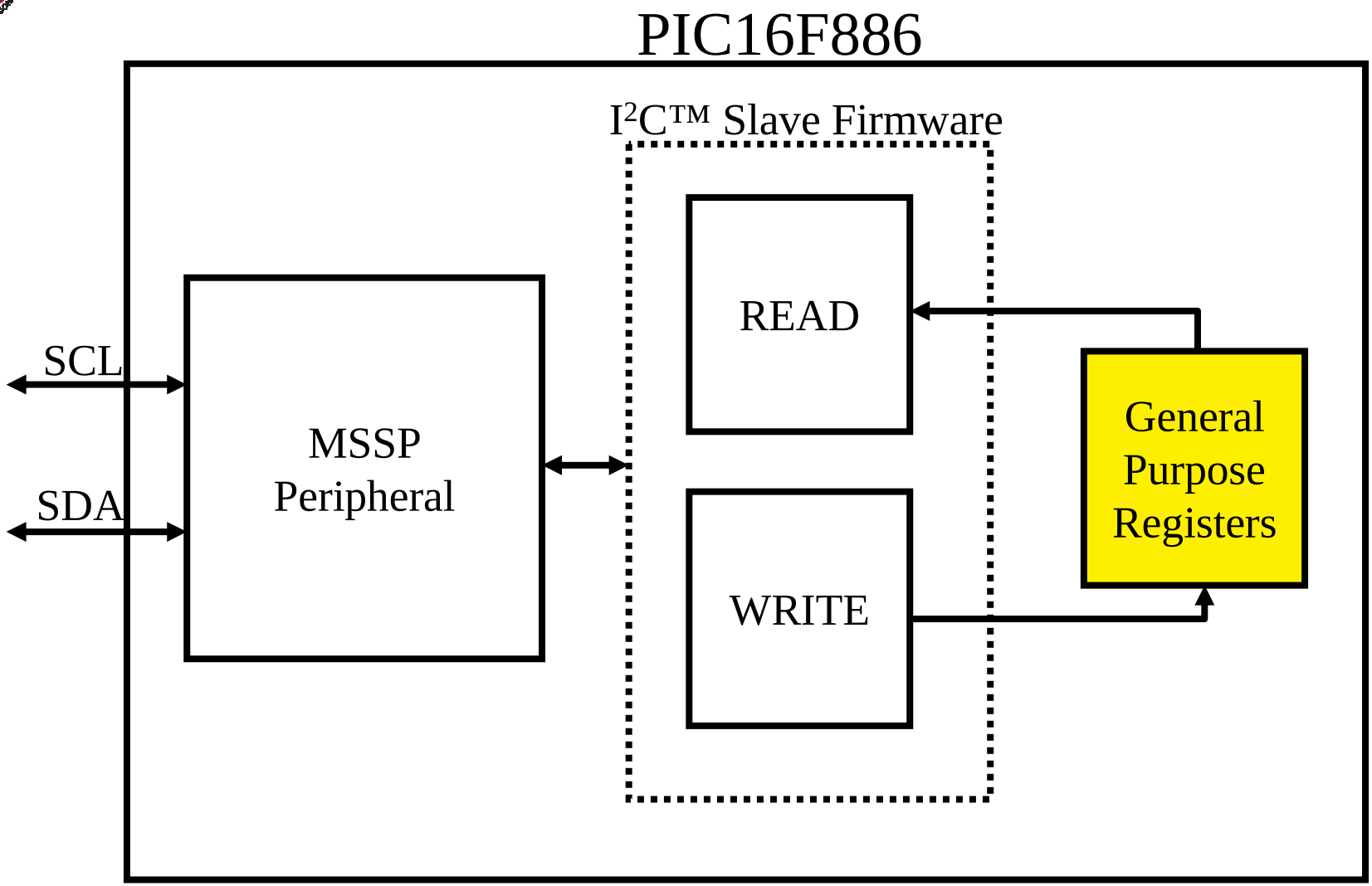
SSPSTAT Bits

- **D/$\overline{A}$ = 1**
- **R/$\overline{W}$ = 0**
- **BF = 1**

I²C™ Event Summary

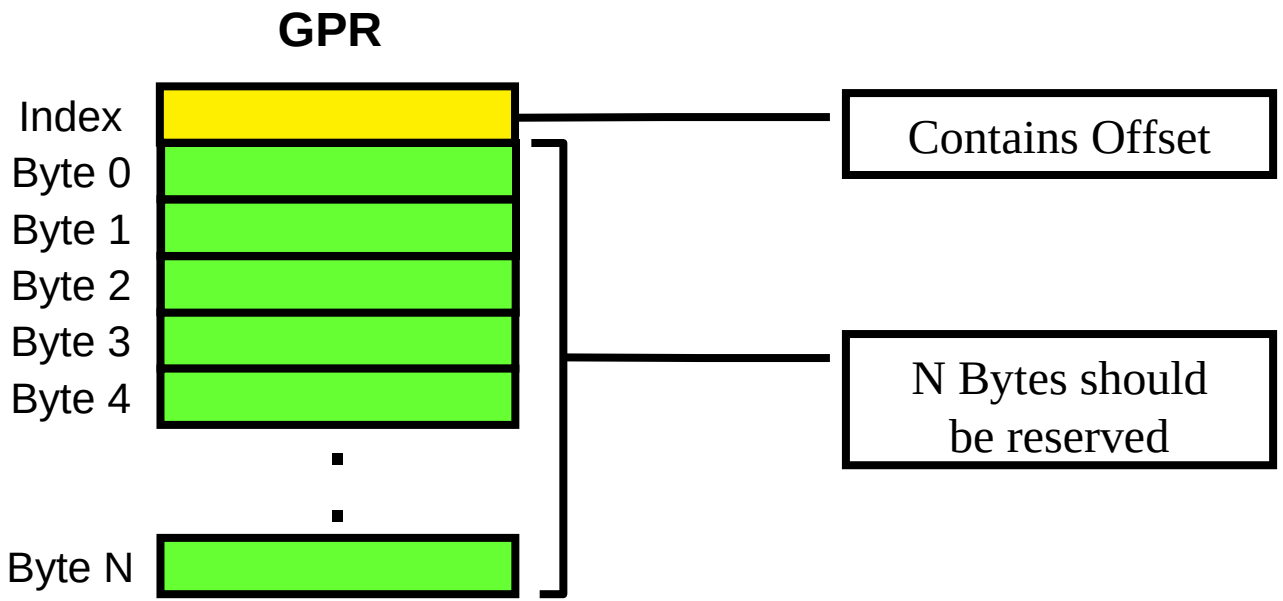
- We now know:
  - How to decode each event:
    - Write Events
    - Read Events
    - Start/Stop Conditions
- Next: Actions to perform at each event
  - Two types of message formats:
    - Read, Write

I²C™ Slave Mode Firmware Overview



General Purpose Register

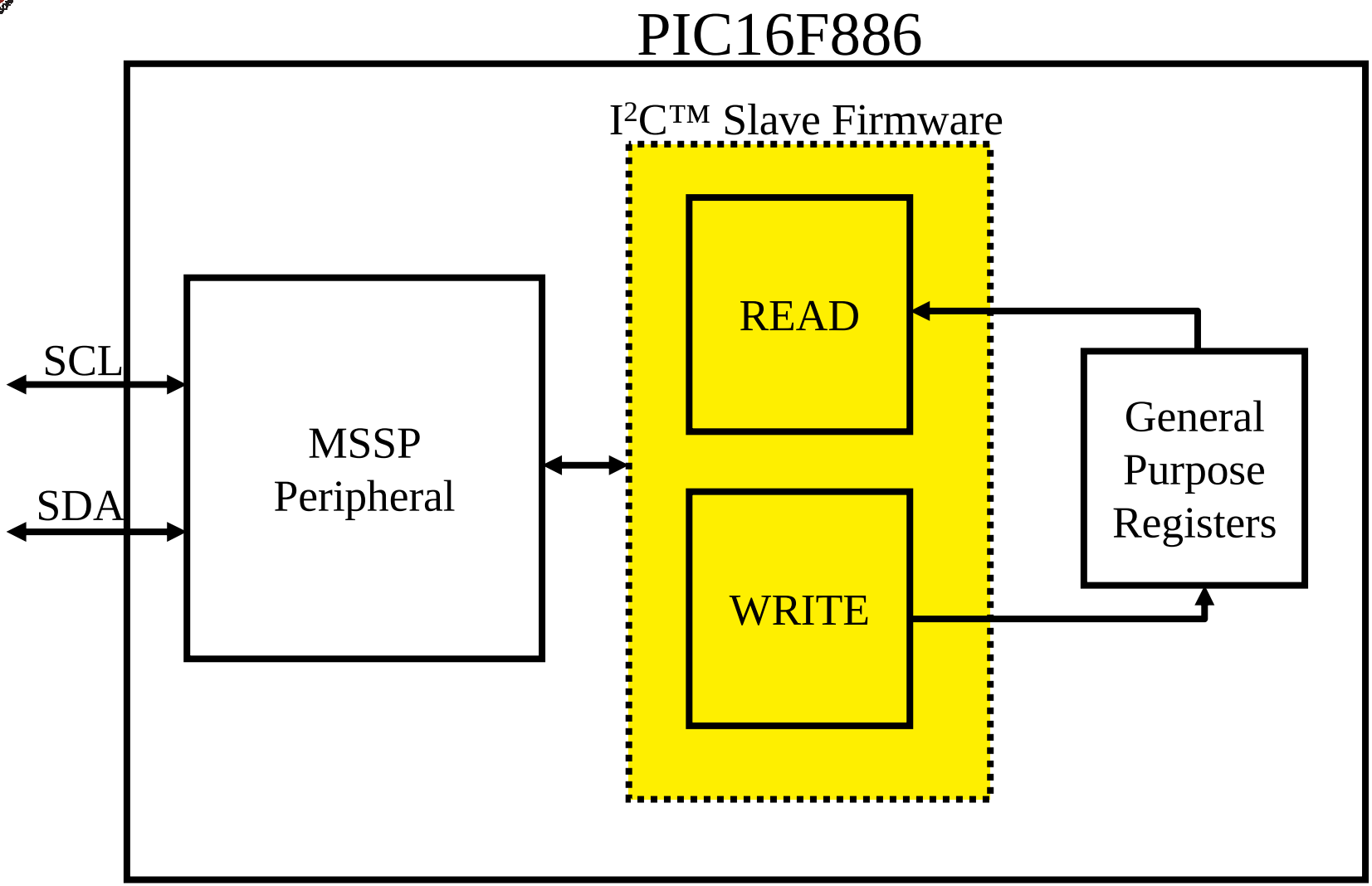
- Data must be sequential



Assembly Code Example

```
#define DEVICE_RAM_LENGTH .9  ; Device RAM Length

#define BYTE0                 (DEVICE_RAM + 0x01)
#define BYTE1                 (DEVICE_RAM + 0x02)
#define BYTE2                 (DEVICE_RAM + 0x03)
#define BYTE3                 (DEVICE_RAM + 0x04)
#define BYTE4                 (DEVICE_RAM + 0x05)
#define BYTE5                 (DEVICE_RAM + 0x06)
#define BYTE6                 (DEVICE_RAM + 0x07)
#define BYTE7                 (DEVICE_RAM + 0x08)
#define BYTE8                 (DEVICE_RAM + 0x09)

udata

DEVICE_RAM     res     DEVICE_RAM_LENGTH
```

I²C™ Slave Mode Firmware Overview

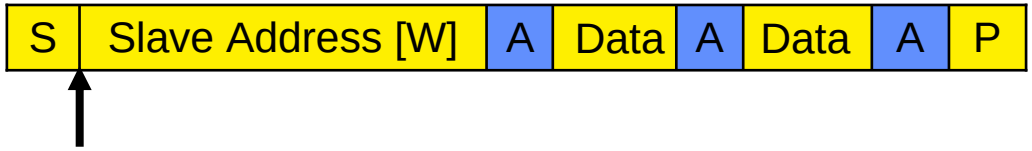


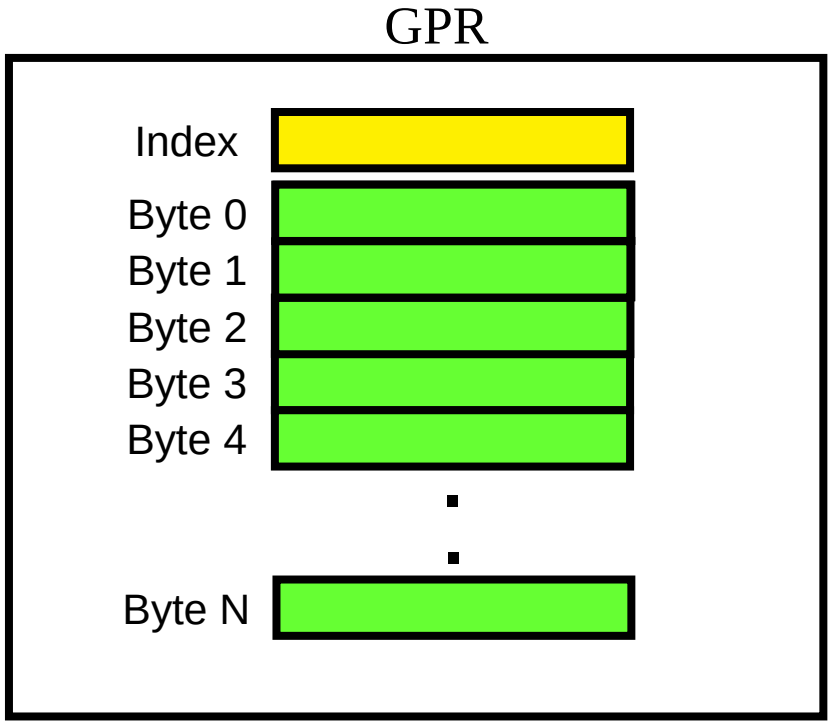
Flow Chart Overview

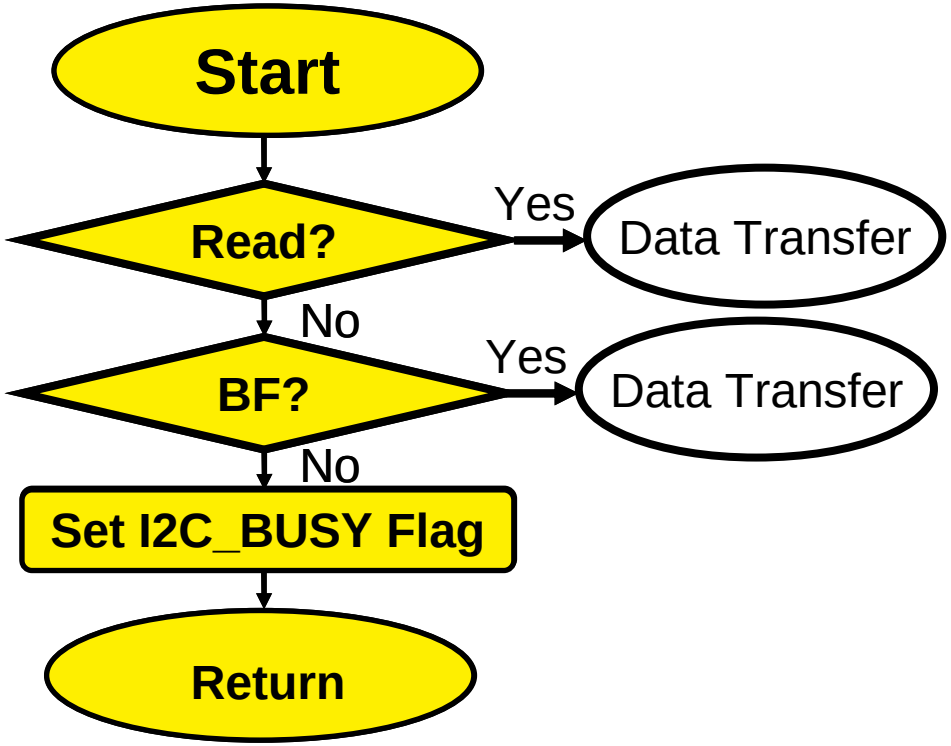
- Use the I²C™ Slave Mode Flow Chart Handout to follow along in the next slides

Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

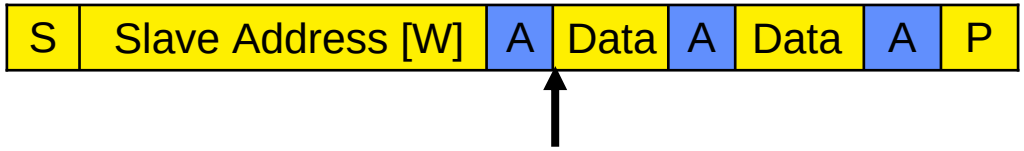


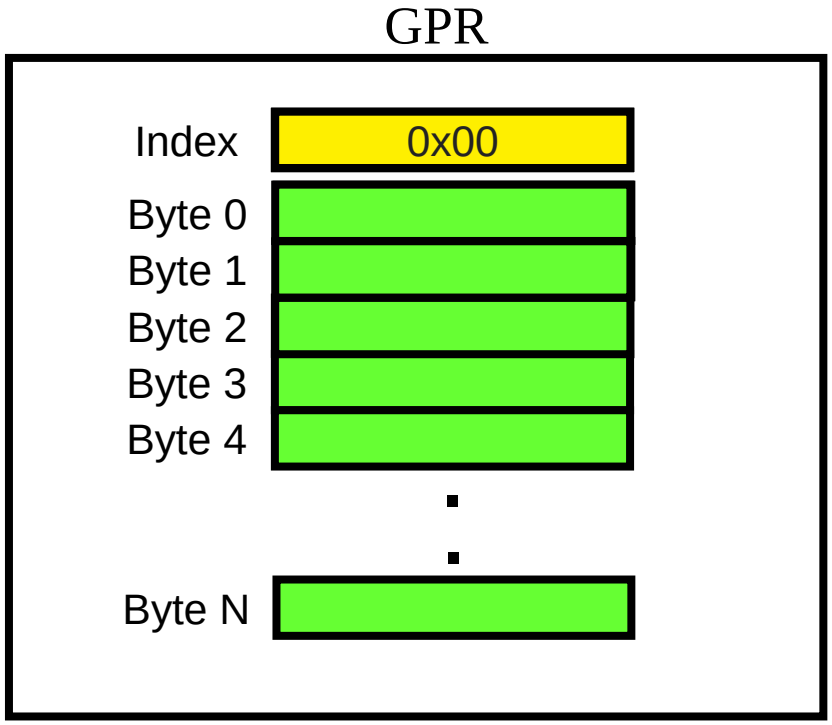


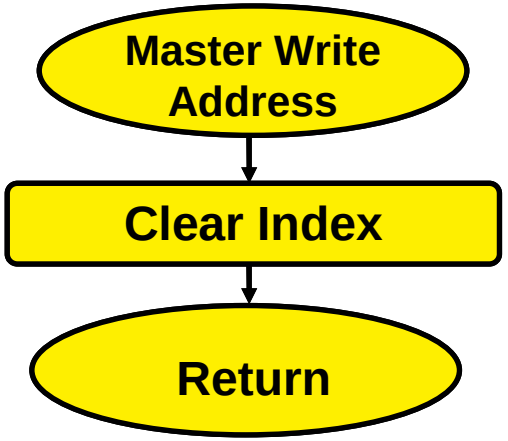


Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

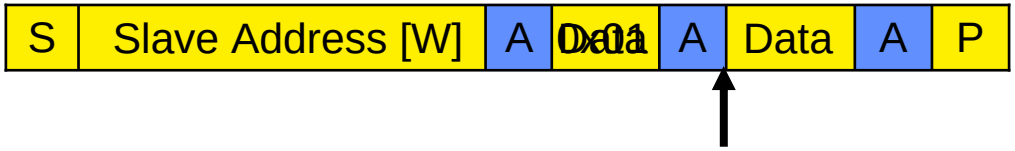


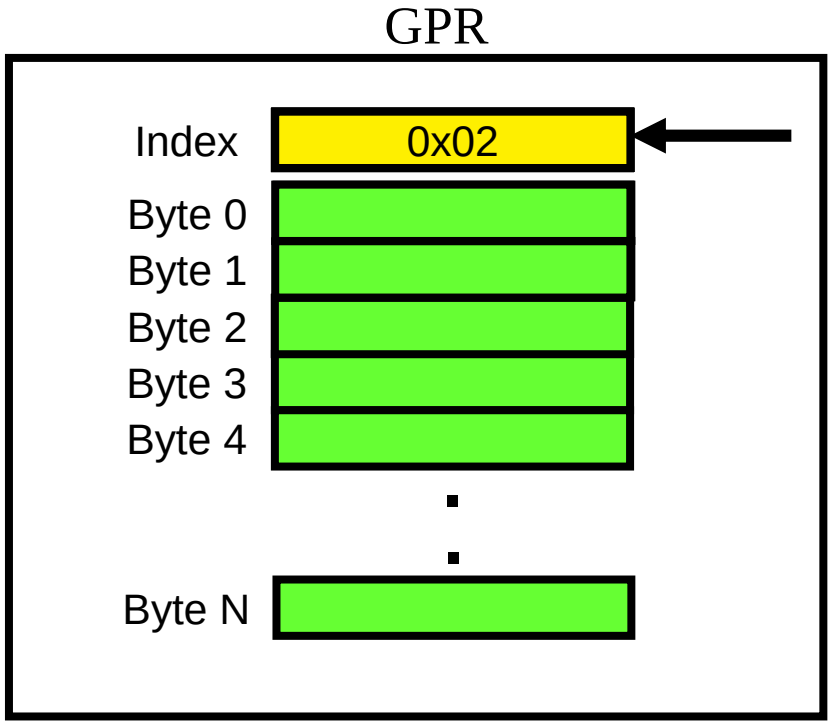


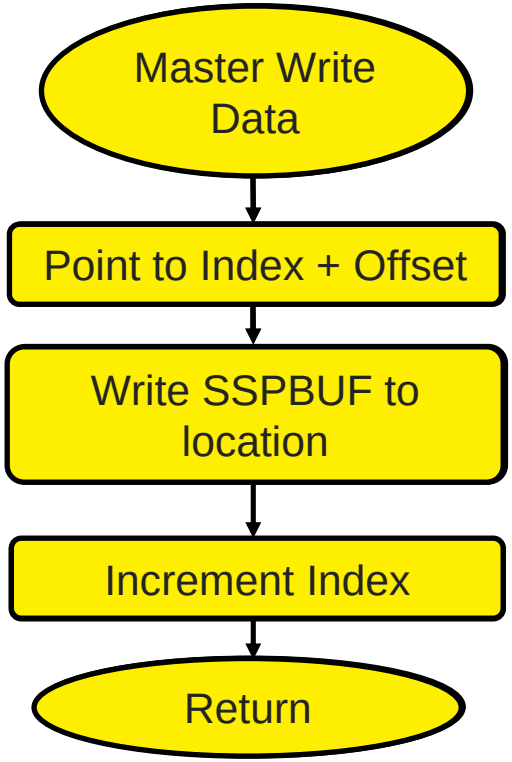


Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

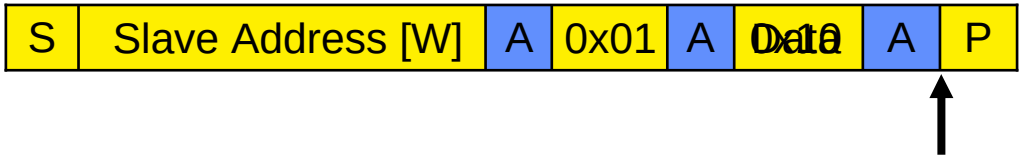


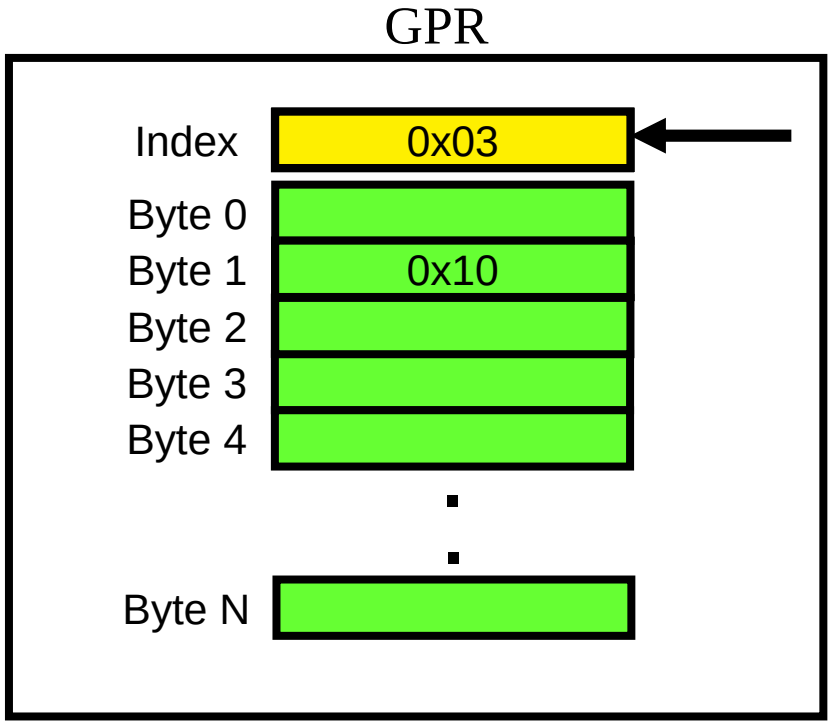


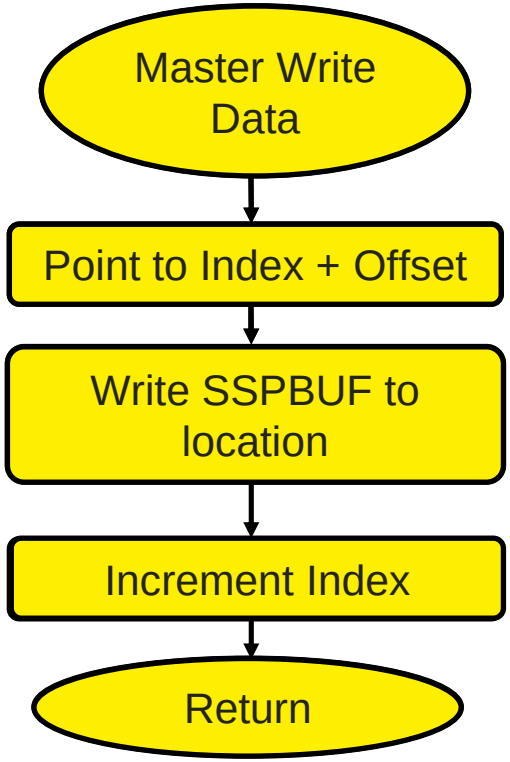


Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

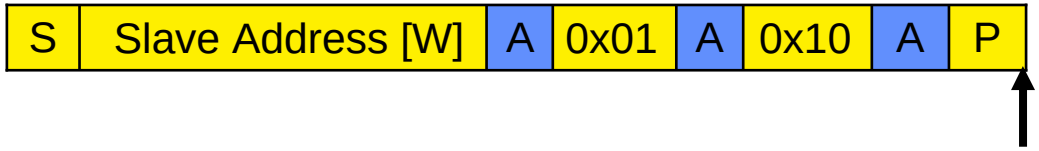


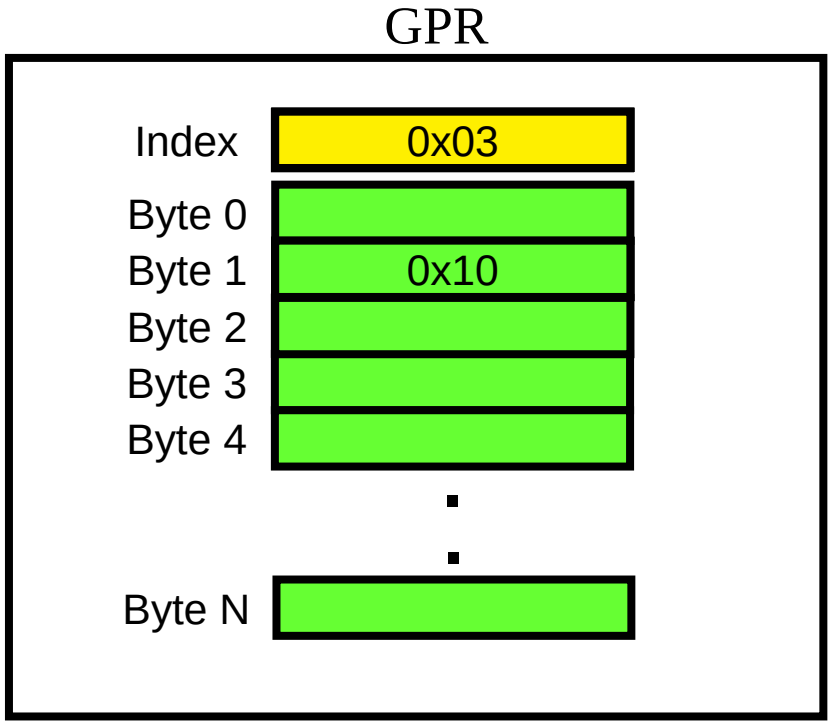


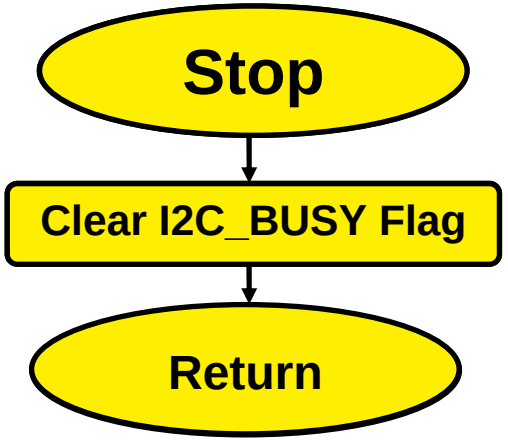


Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

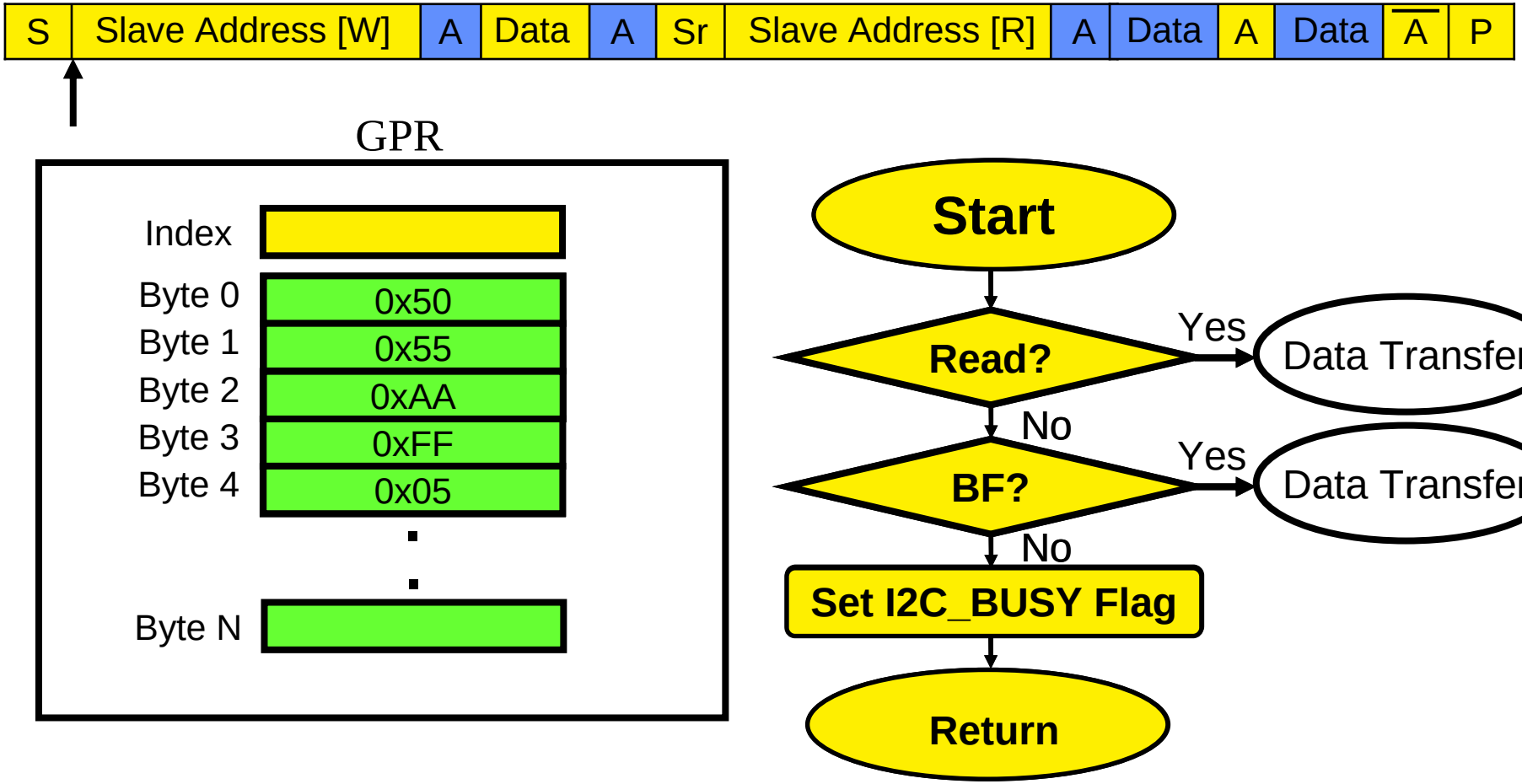






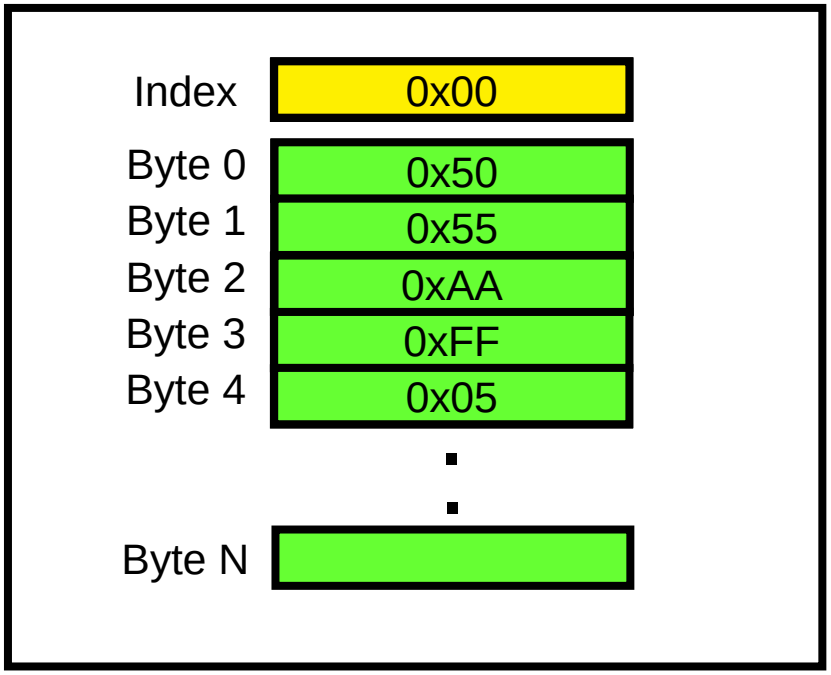
Handling I²C™ Read Messages

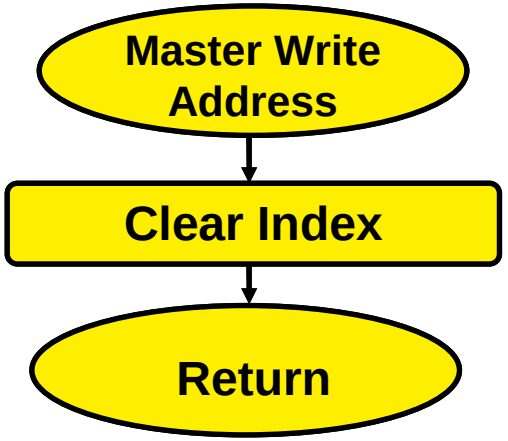
- Master writes address to be read then reads data



Handling I²C™ Read Messages

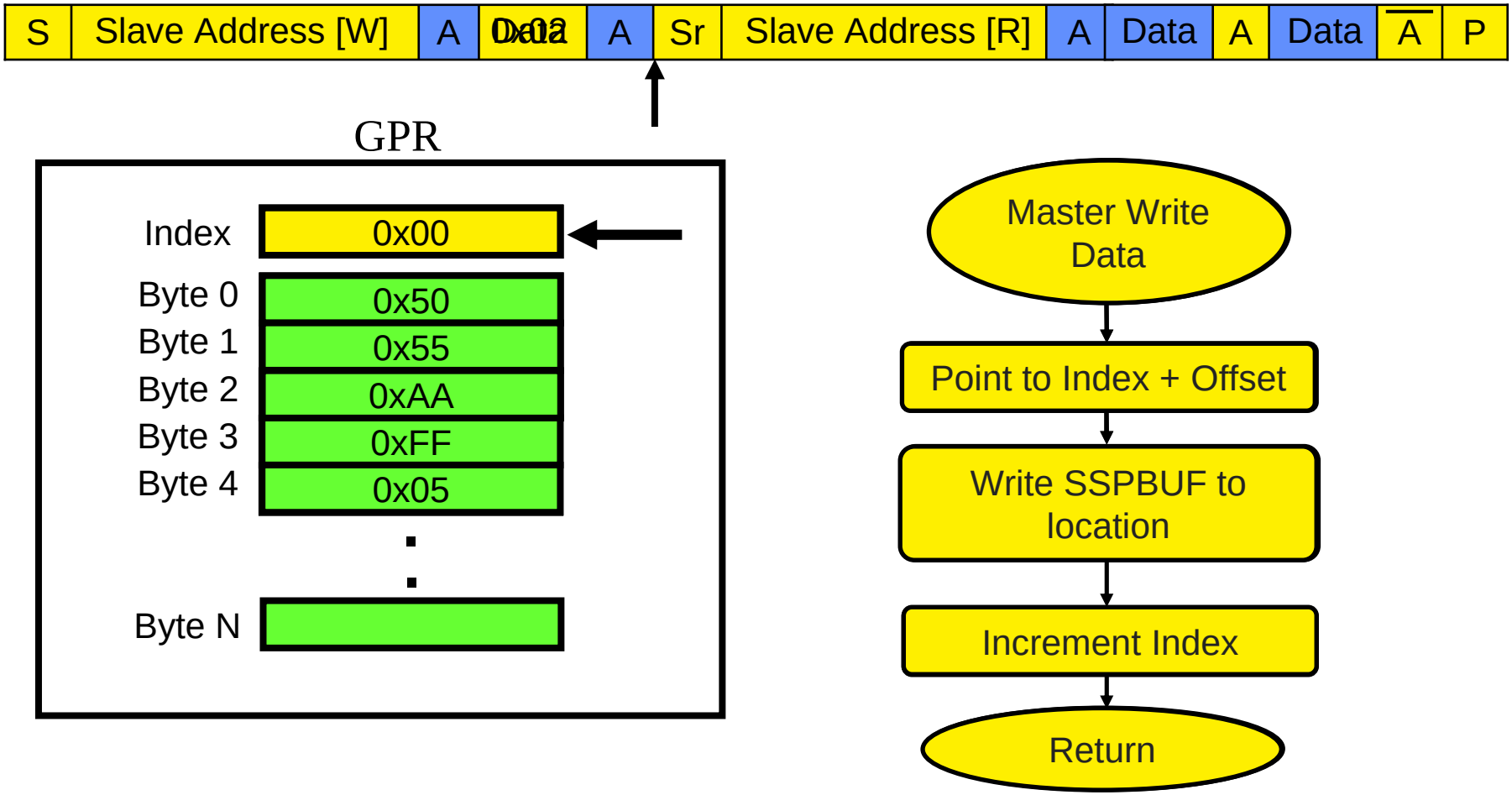
- Master writes address to be read then reads data





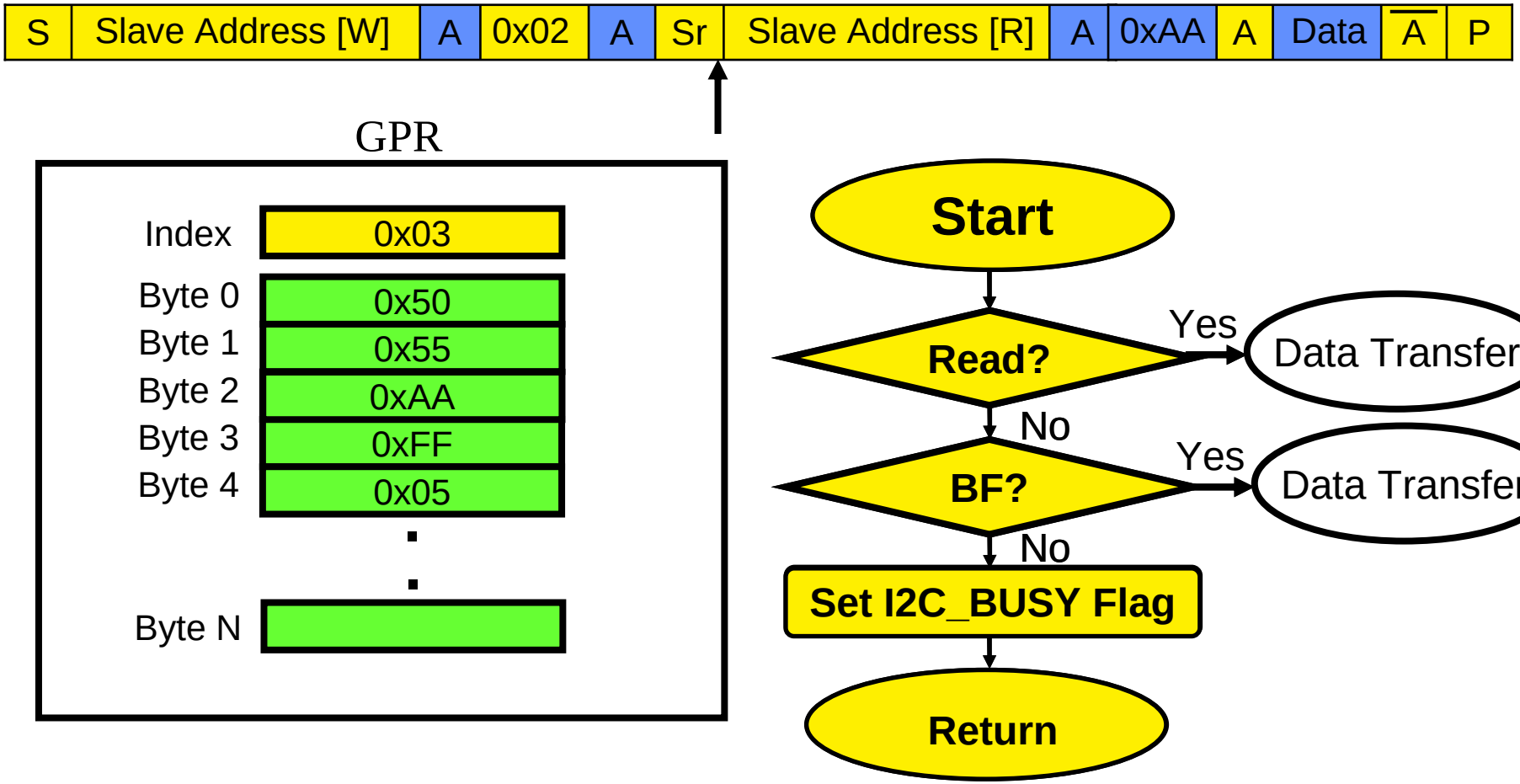
Handling I²C™ Read Messages

- Master writes address to be read then reads data



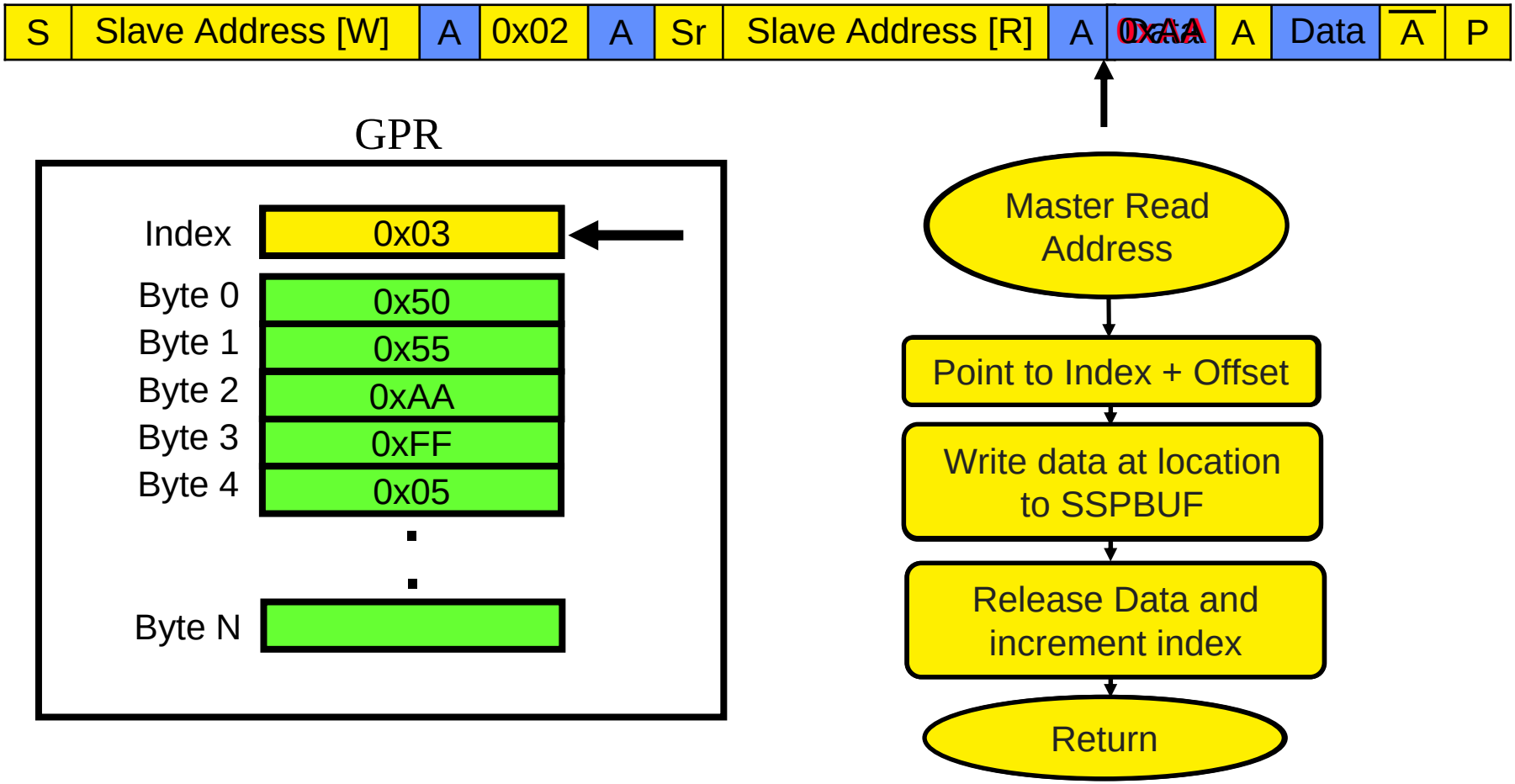
Handling I²C™ Read Messages

- Master writes address to be read then reads data



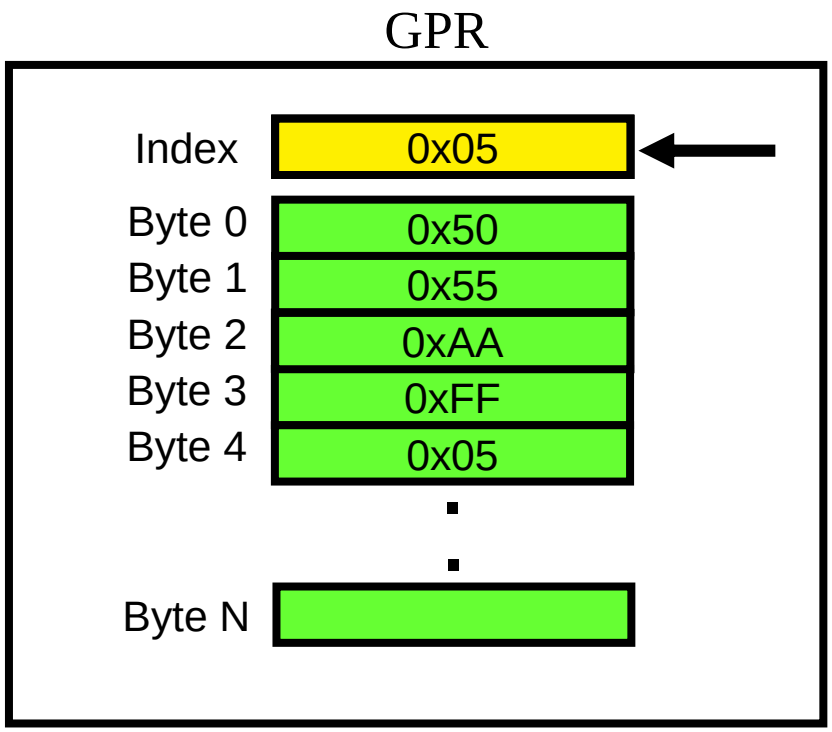
Handling I²C™ Read Messages

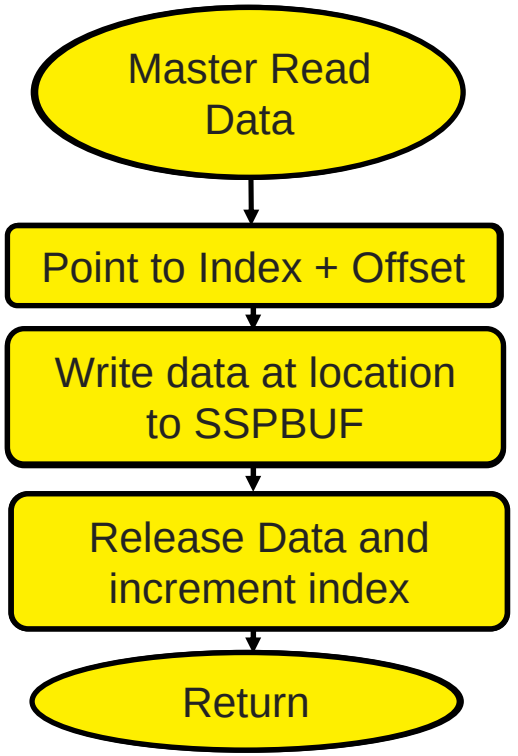
- Master writes address to be read then reads data



Handling I²C™ Read Messages

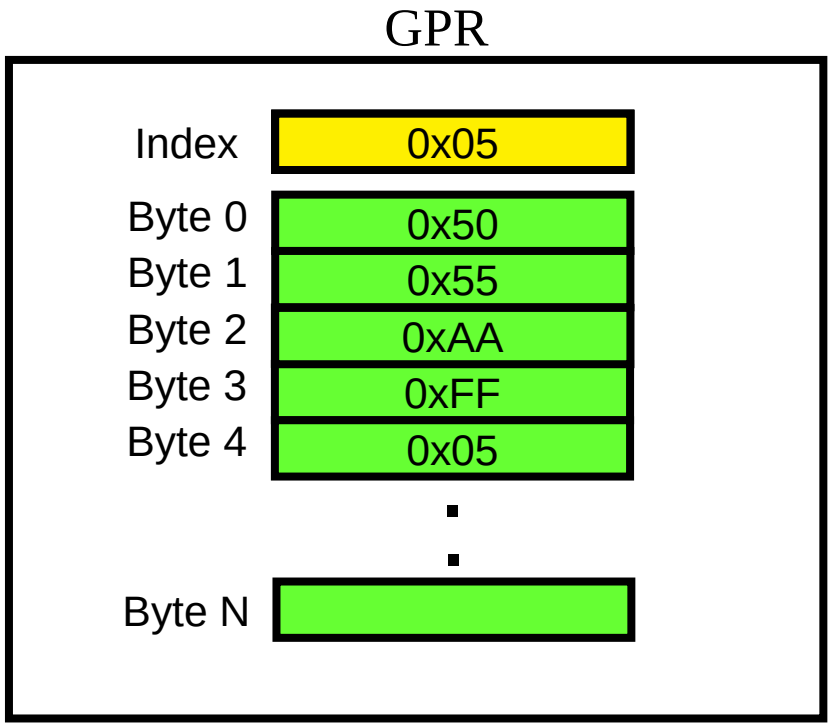
- Master writes address to be read then reads data

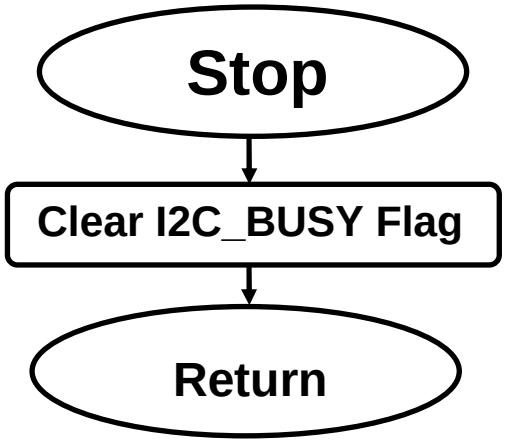




Handling I²C™ Read Messages

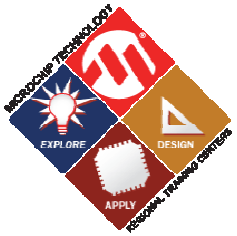
- Master writes address to be read then reads data



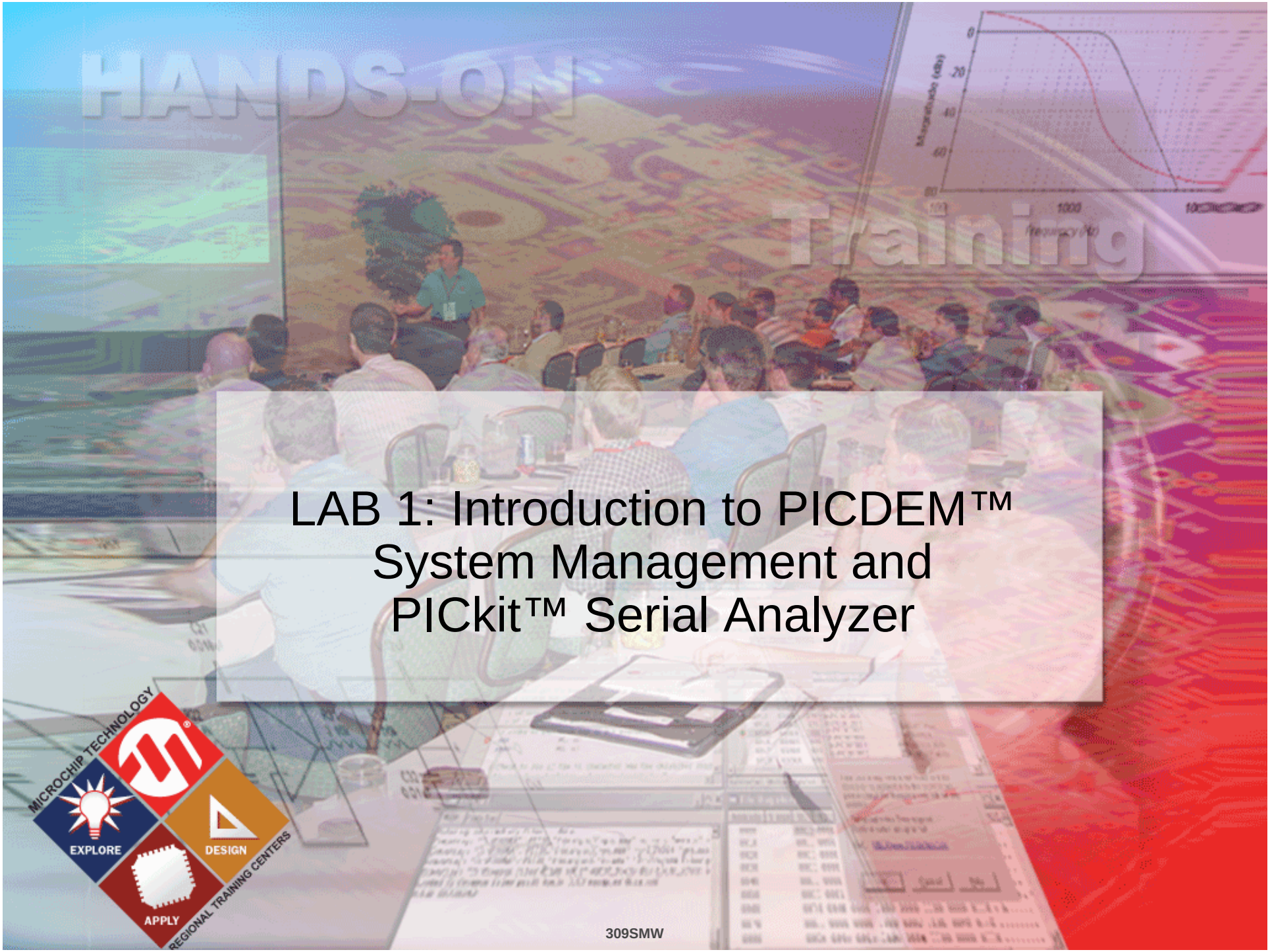


I²C™ Firmware Conclusion

- Performs operations to GPR
- Write Operation
- Read Operation
  - MUST specify the word address
- Gotchas:
  - SSPSTAT is updated during each interrupt event
  - Interrupt events may occur before an event is serviced



Questions?



LAB 1: Introduction to PICDEM™ System Management and PICkit™ Serial Analyzer

Serial EEPROM Module

309SMW

Module Learning Objectives

- Learning Objectives:
  - Explain how serial EEPROM devices function
  - Perform read and write operation to data EEPROM on a PIC16F886
  - Emulate an existing I^2C™ serial EEPROM device on a PIC16F886

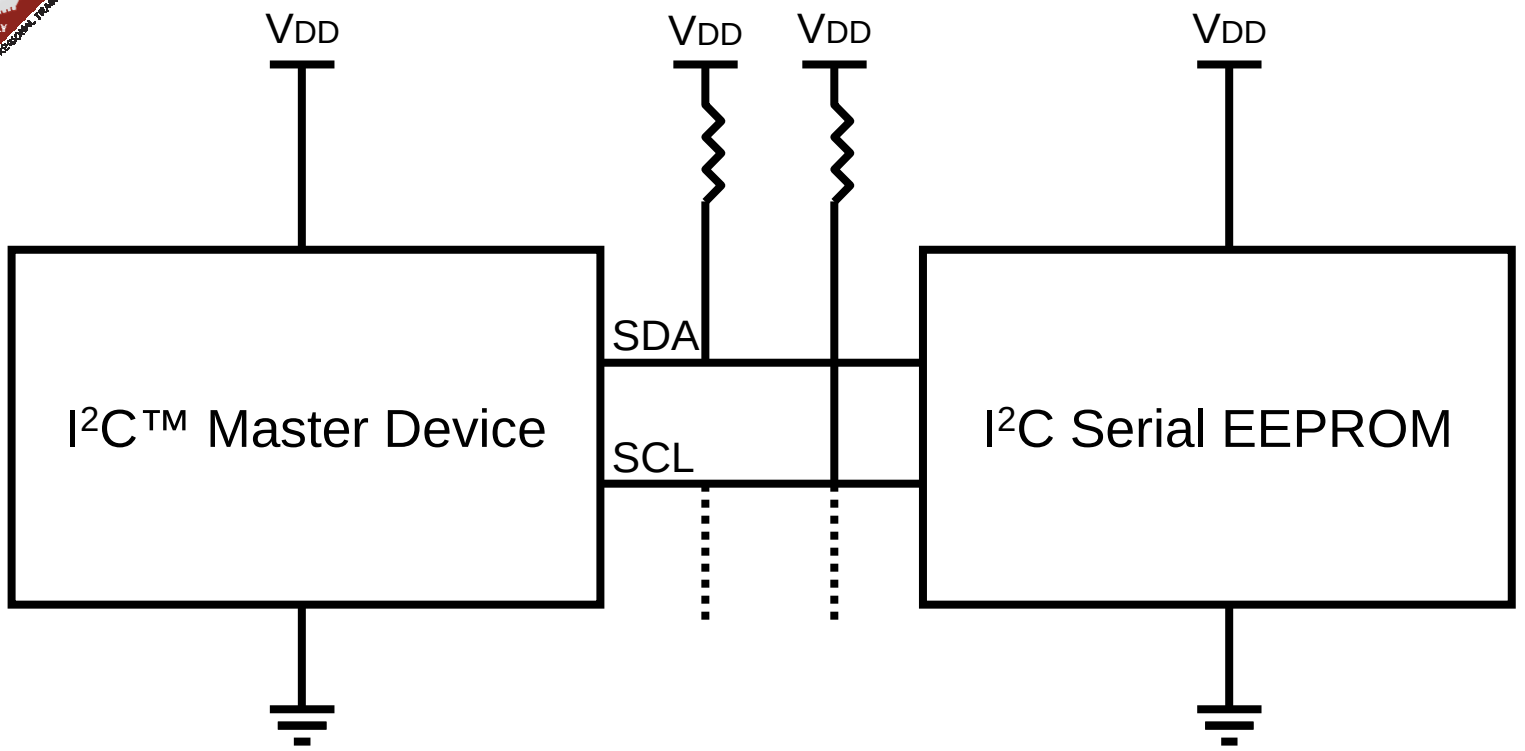
- Prerequisites
  - Mid-range Architecture
  - Synchronous Serial Port Peripheral
  - I^2C Slave Mode Firmware

Module Agenda

In the next hour we will discuss…

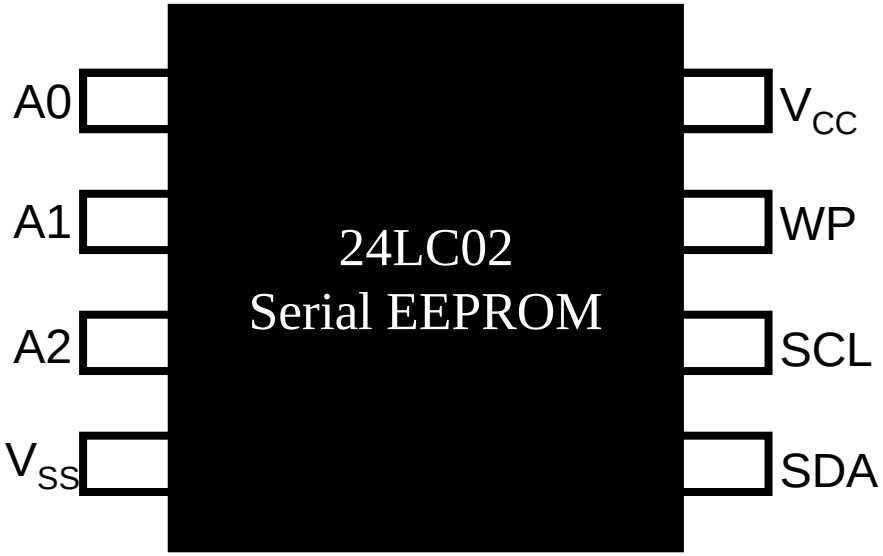
- Serial EEPROM Basics
- PIC® MCU Data EEPROM
- Emulating Serial EEPROM using PIC MCU Data EEPROM
- Lab Demonstration
- Summary

Introduction to Serial EEPROM



- Nonvolatile Data Storage
- Serial Communications: 2 Wire
- Applications: System Management, Data Logging

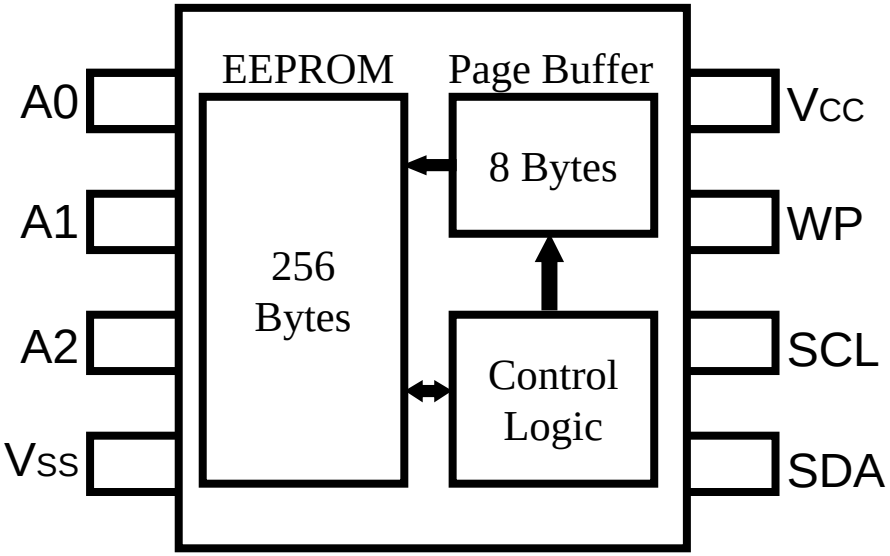
24LC02 Serial EEPROM



- I²C™ Serial Communication
- Hardware **slave address** allocation:

1	0	1	0	A2	A1	A0	R/W

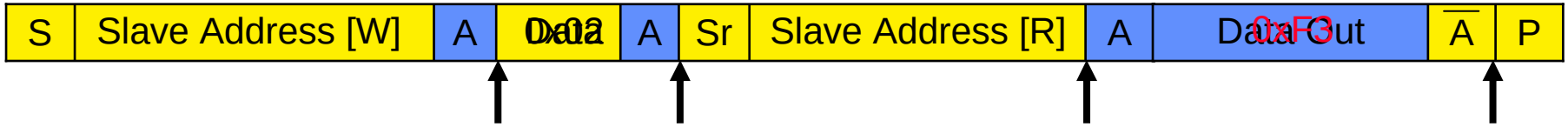
24LC02 Serial EEPROM

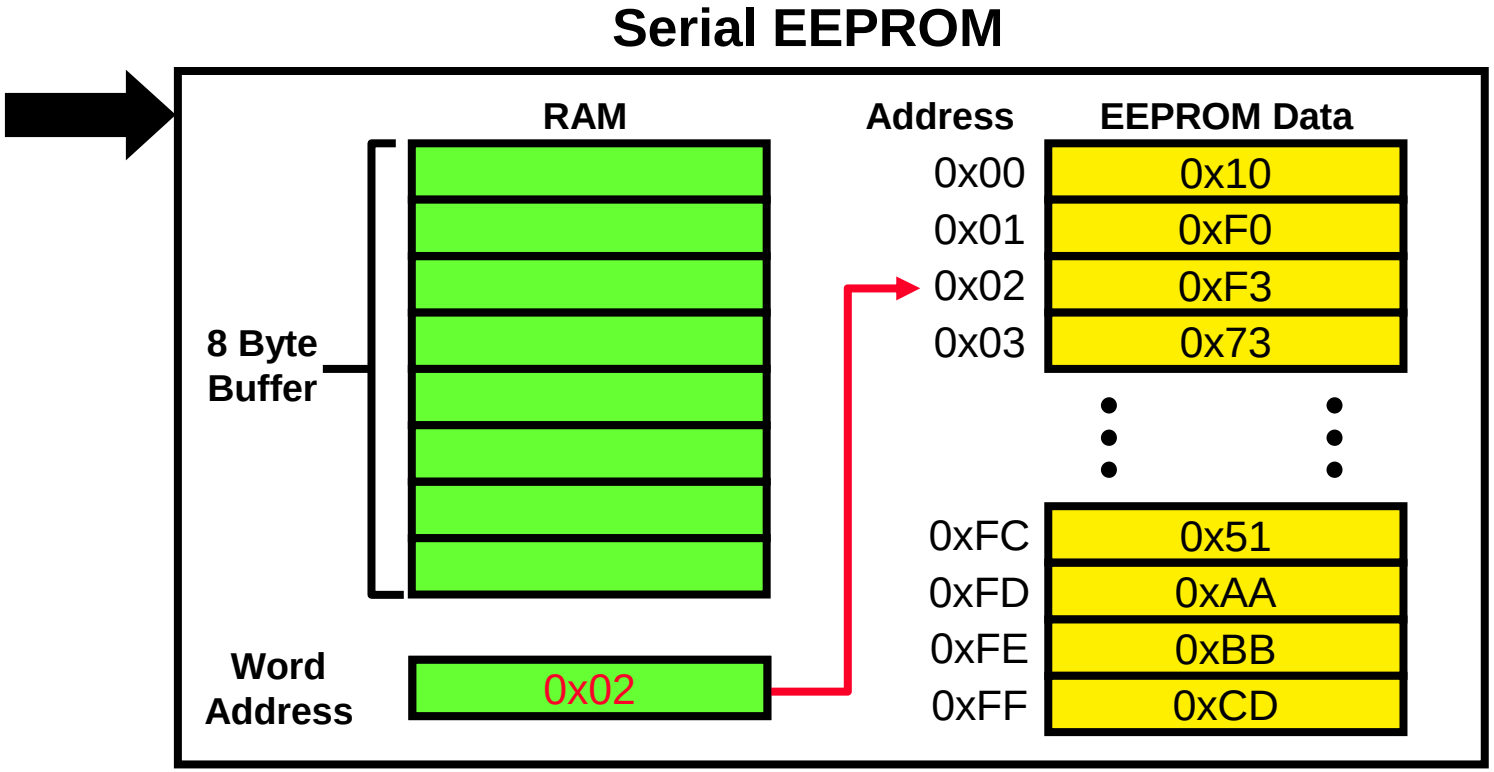


- 256 Byte Array Size
- 8-Byte Page Write Buffer
  - Used only for writes

Communicating to Serial EEPROM

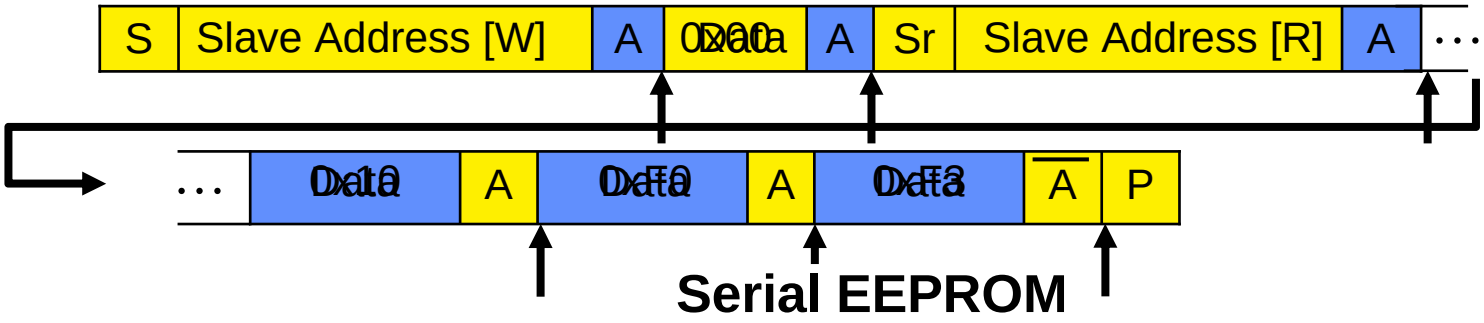
- Single Byte Read (I^2C™)

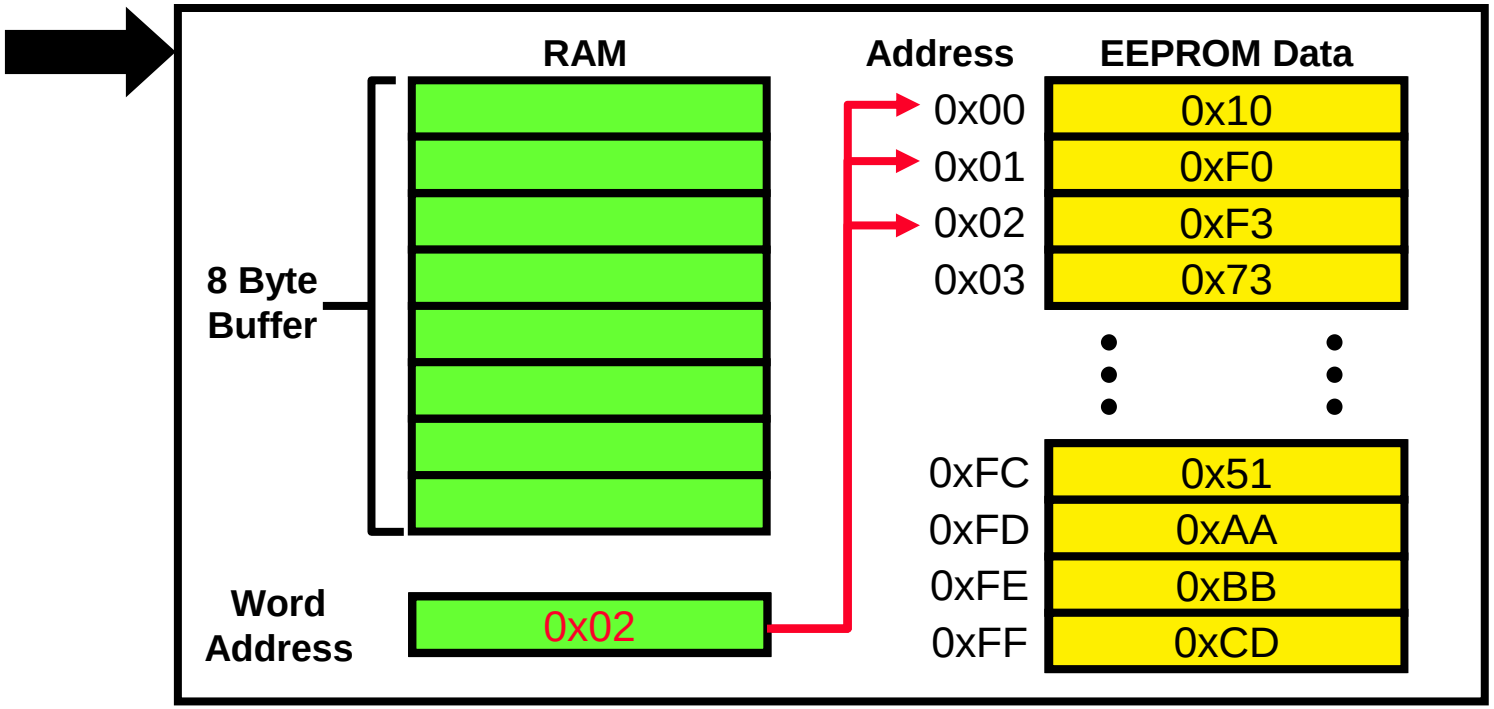




Communicating to Serial EEPROM

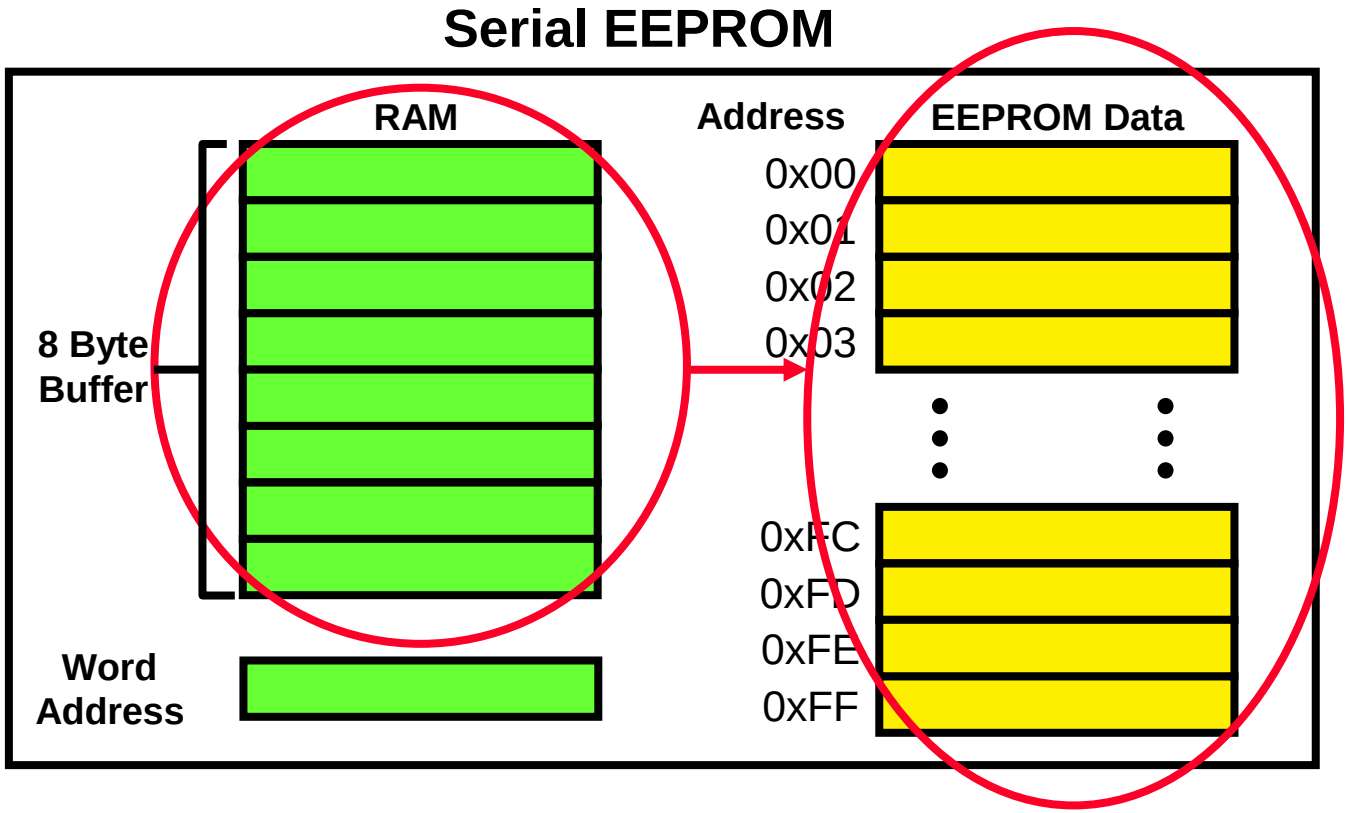
- Multiple Byte Read (I²C™)





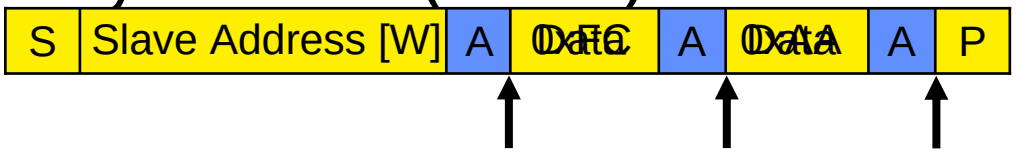
Page Write Buffer

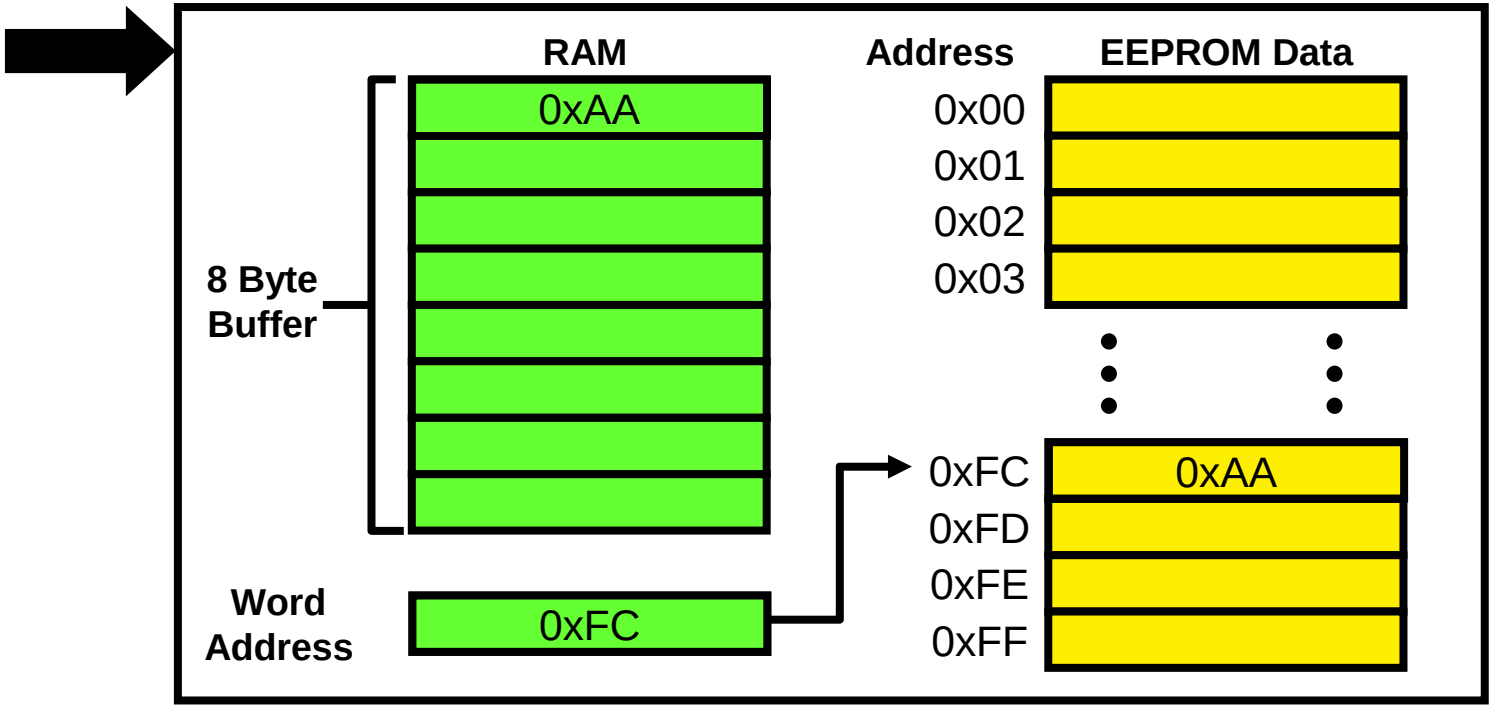
- Write time ~ 5 ms
- Buffer incoming data
- Internally the write occurs in two steps:



Communicating to Serial EEPROM

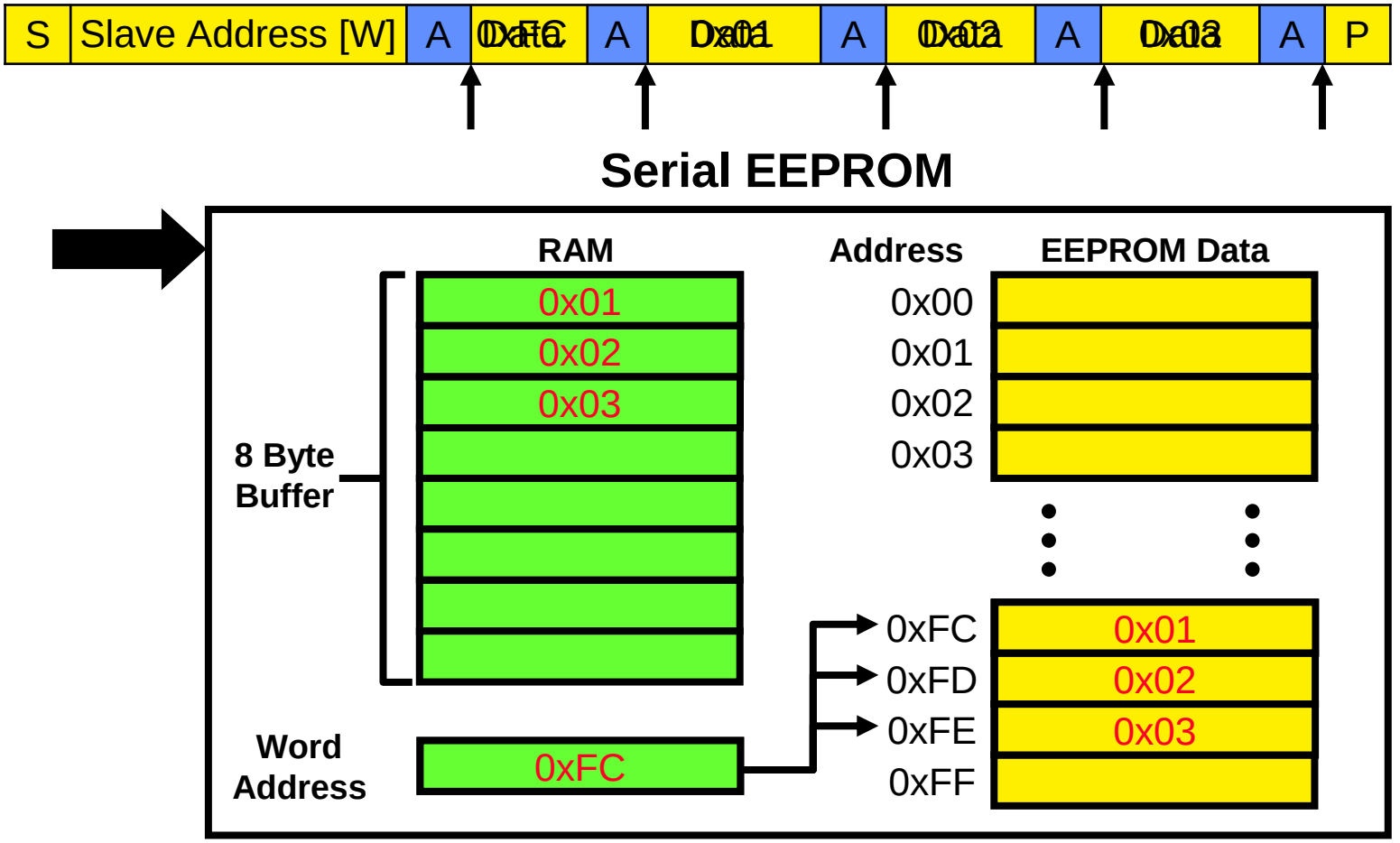
- Single Byte Write (I^2C™)





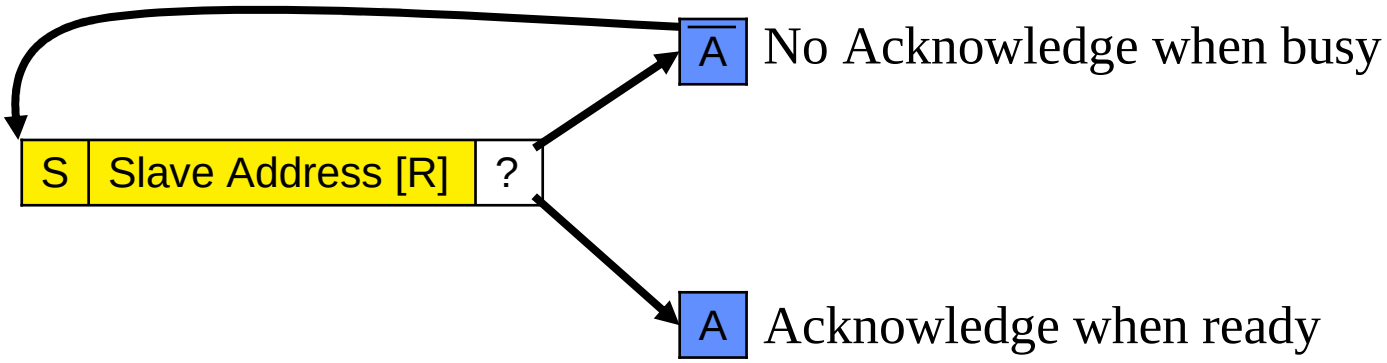
Communication to Serial EEPROM

- Multiple Byte Write (I^2C™)



Acknowledge Polling

- Serial EEPROM does not acknowledge during internal write cycle
- Acknowledge polling helps determine the instant EEPROM is ready for a write/read

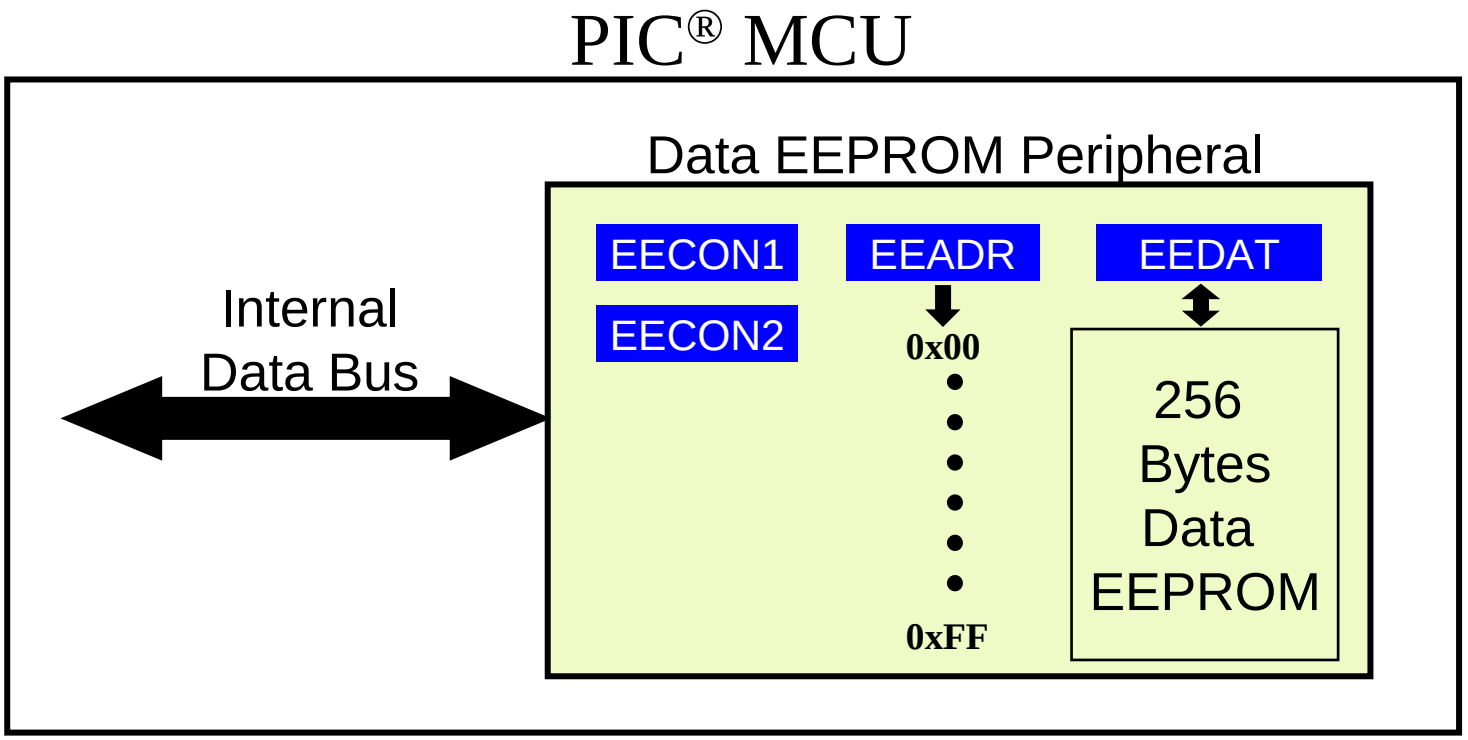


Serial EEPROM Review

- Nonvolatile memory
- 256 Bytes
- Communication via I²C™
- Single or Multi-Byte Read
- Single or Multi-Byte Write
  - 8-Byte Page Buffer
- Acknowledge Polling


HANDS-ON
Training
PIC16F886
Data EEPROM
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

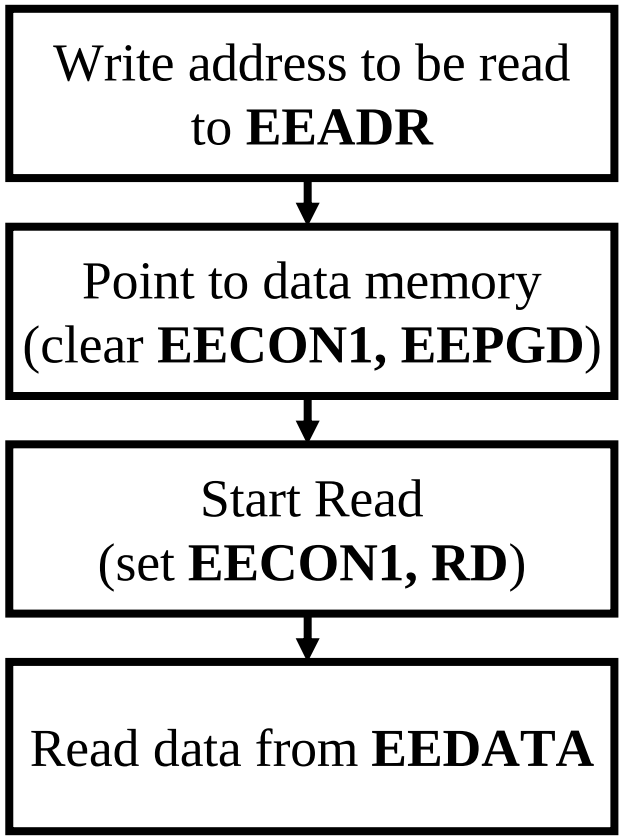
PIC16F886 Data EEPROM Peripheral



- Addressed through Special Function Registers (SFR)
  - EECON1: EEPROM Control Register
  - EECON2: Not a Physical Register, Used for Writes
  - EEADR: EEPROM data address to write or read
  - EEDAT: EEPROM data to write or read

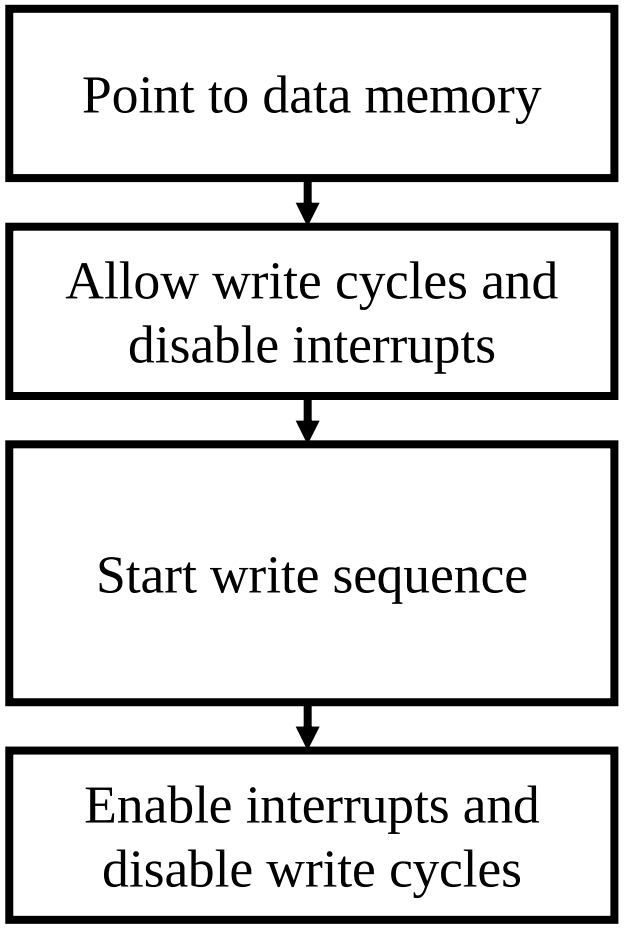
Reading from data EEPROM

```
movlw       EEDATA_ADDRESS
banksel     EEADR
movwf       EEADR

banksel     EECON1
bcf         EECON1, EEPGD

bsf         EECON1, RD


banksel     EEDATA
movf        EEDATA, W
```

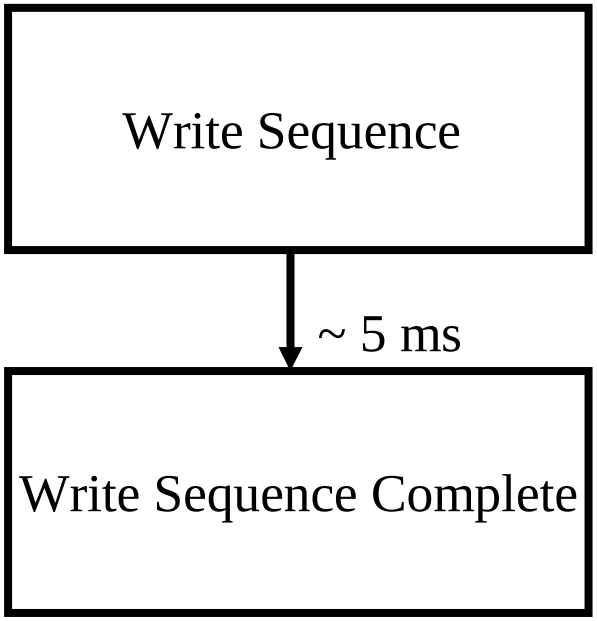


Writing to data EEPROM

```
banksel     EECON1
bcf         EECON1, EEPGD

bsf         EECON1, WREN
bcf         INTCON, GIE

movlw       0x55
movwf       EECON2
movlw       0xAA
movwf       EECON2
bsf         EECON1, WR

bsf         INTCON, GIE
bcf         EECON1, WREN
```



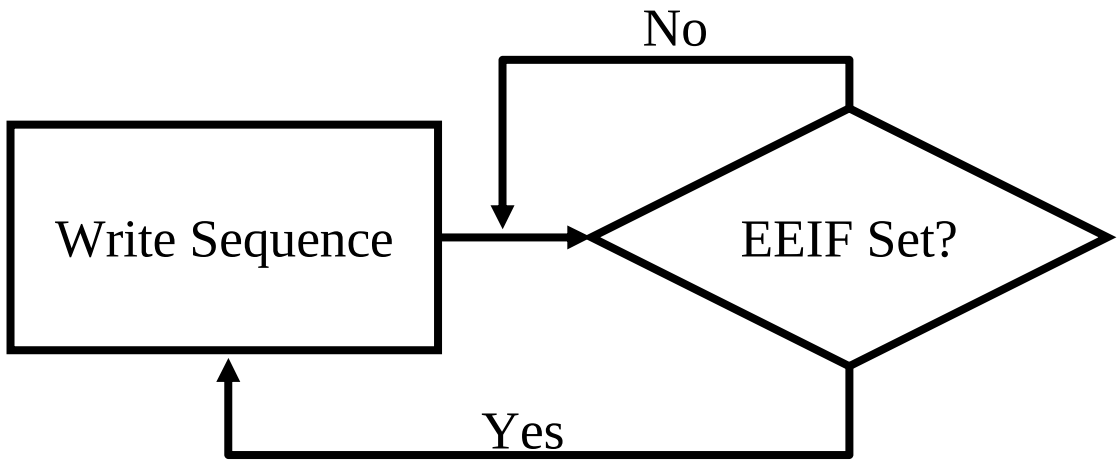
Writing to data EEPROM



- Write time is fixed
- Challenge: Optimizing multiple byte writes
  - Polling
  - Interrupts

Optimizing Multiple Byte Writes

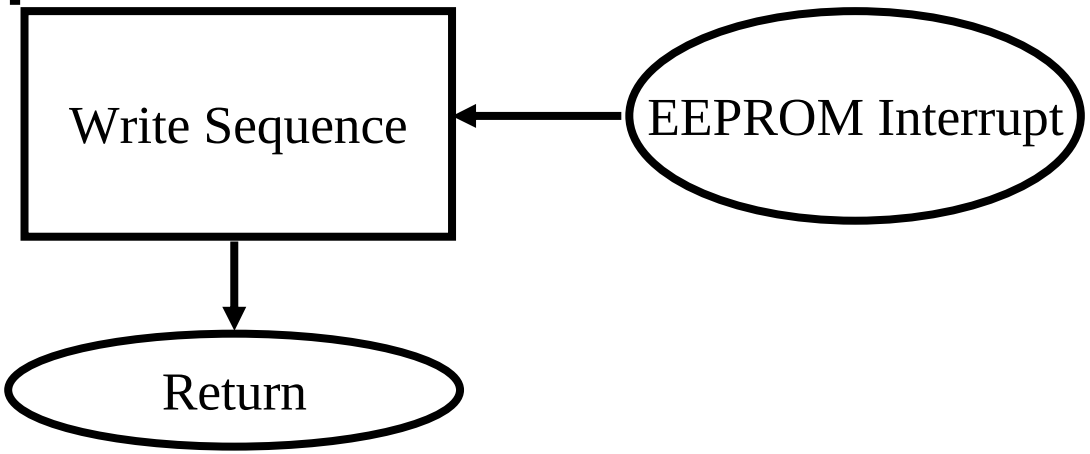
- **Polling**



- Pros:
  - Bytes are written as soon as last byte is complete
  - Minimal code space
- Cons:
  - No other code can be executed when polling

Optimizing Multiple Byte Writes

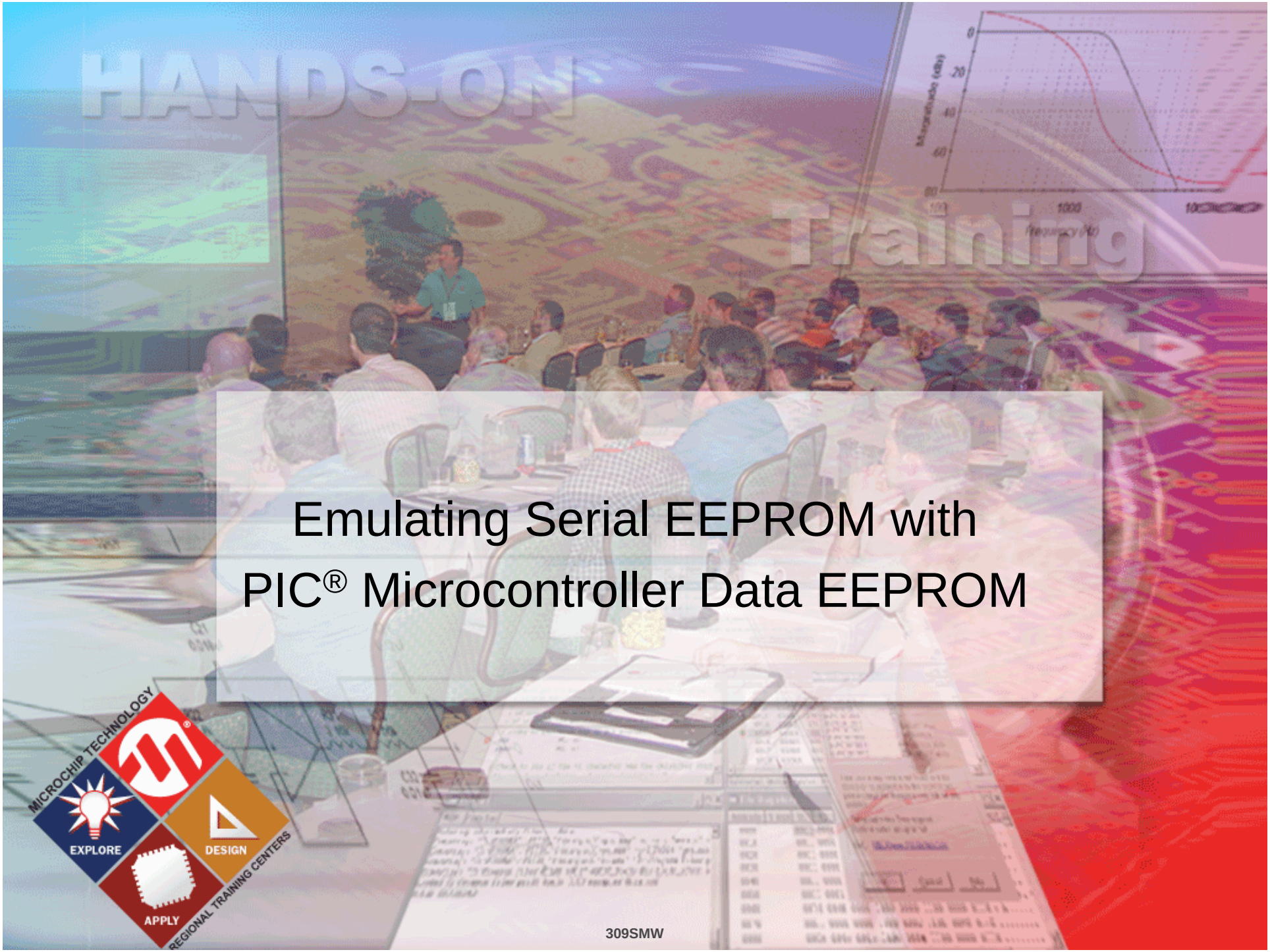
- **Interrupts**



- Pros:
  - Data can be written as soon as last byte is complete
  - Code can execute while Write is in progress
- Cons:
  - ISR processing time

Data EEPROM Summary

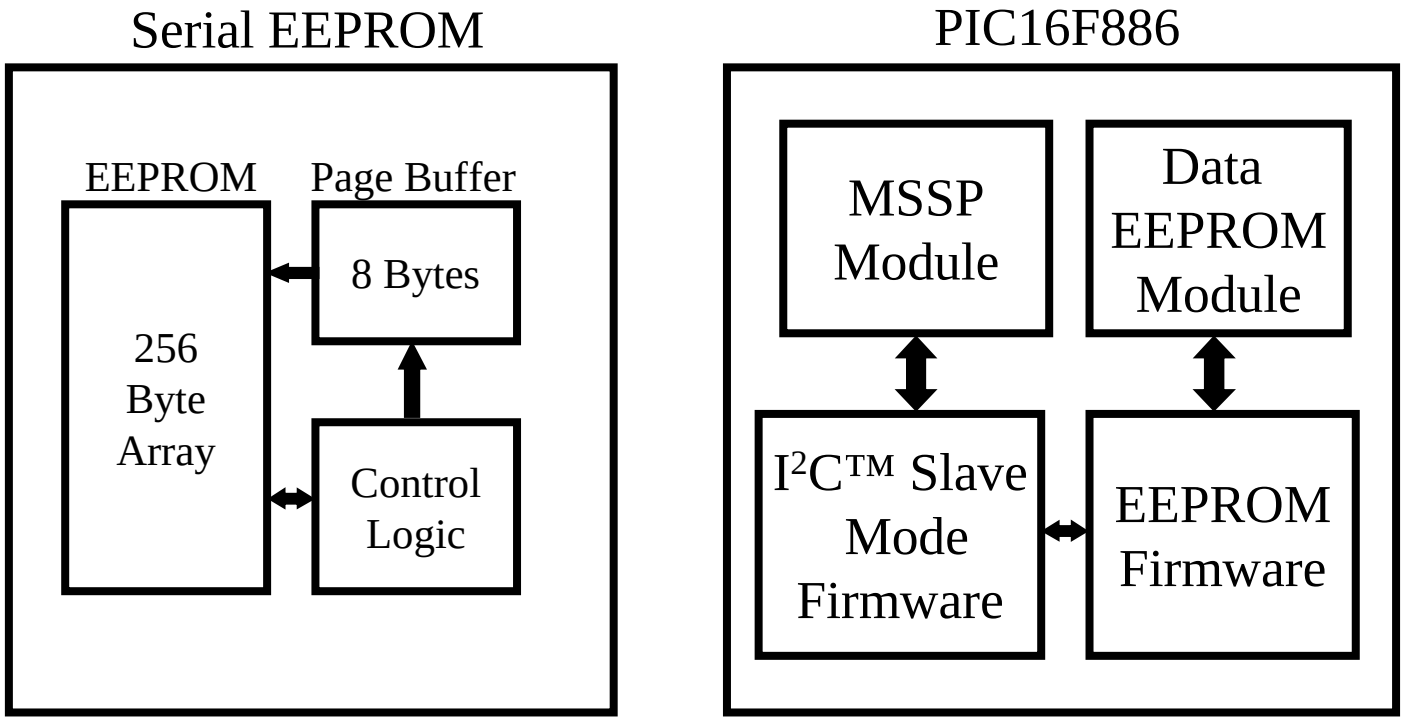
- 256 bytes of nonvolatile memory
- Addressed through Special Function Registers
- Specific Read/Write Sequences
- EEPROM Data Write Interrupt



Emulating Serial EEPROM with PIC® Microcontroller Data EEPROM

309SMW

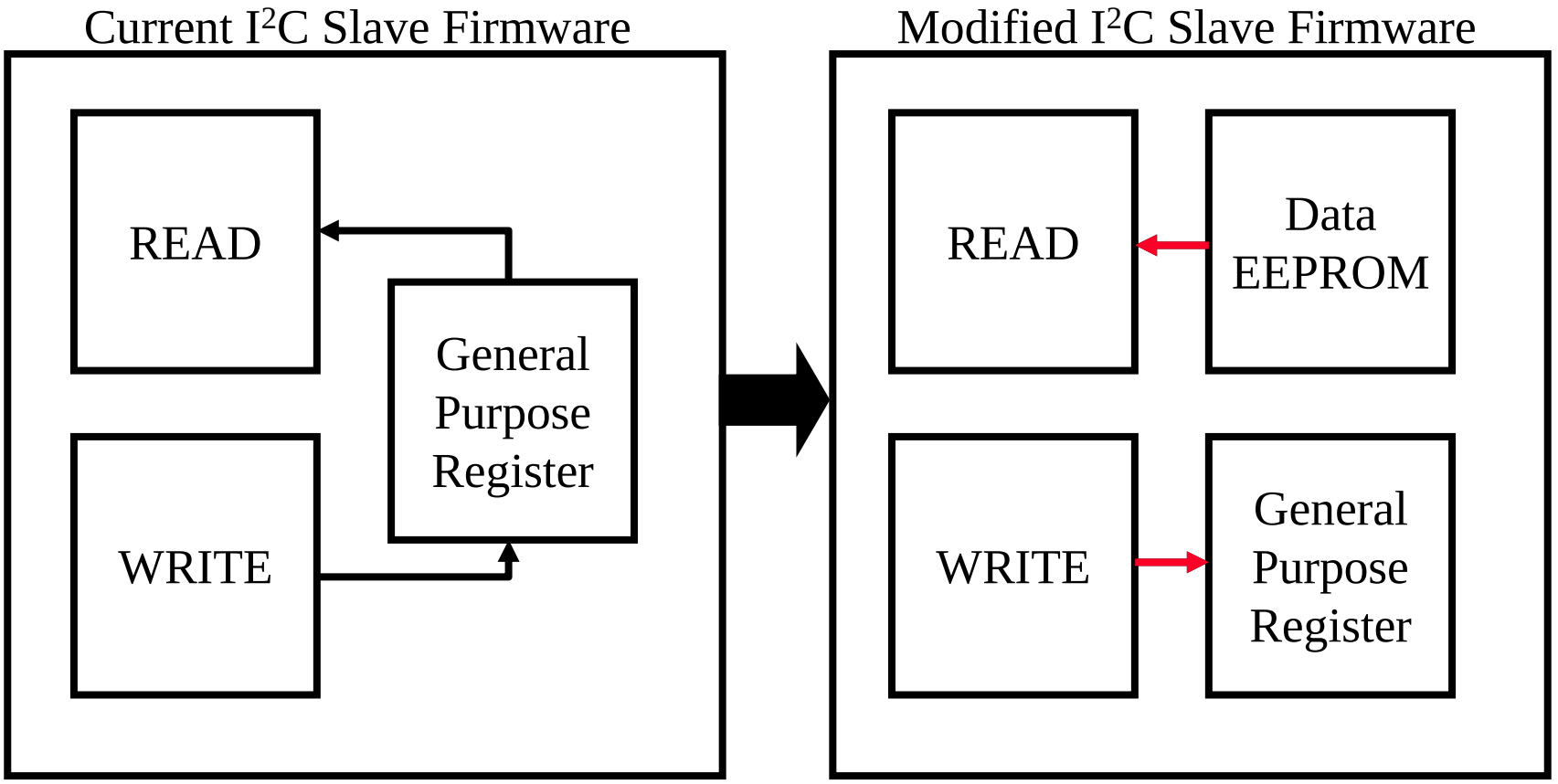
Device Comparison



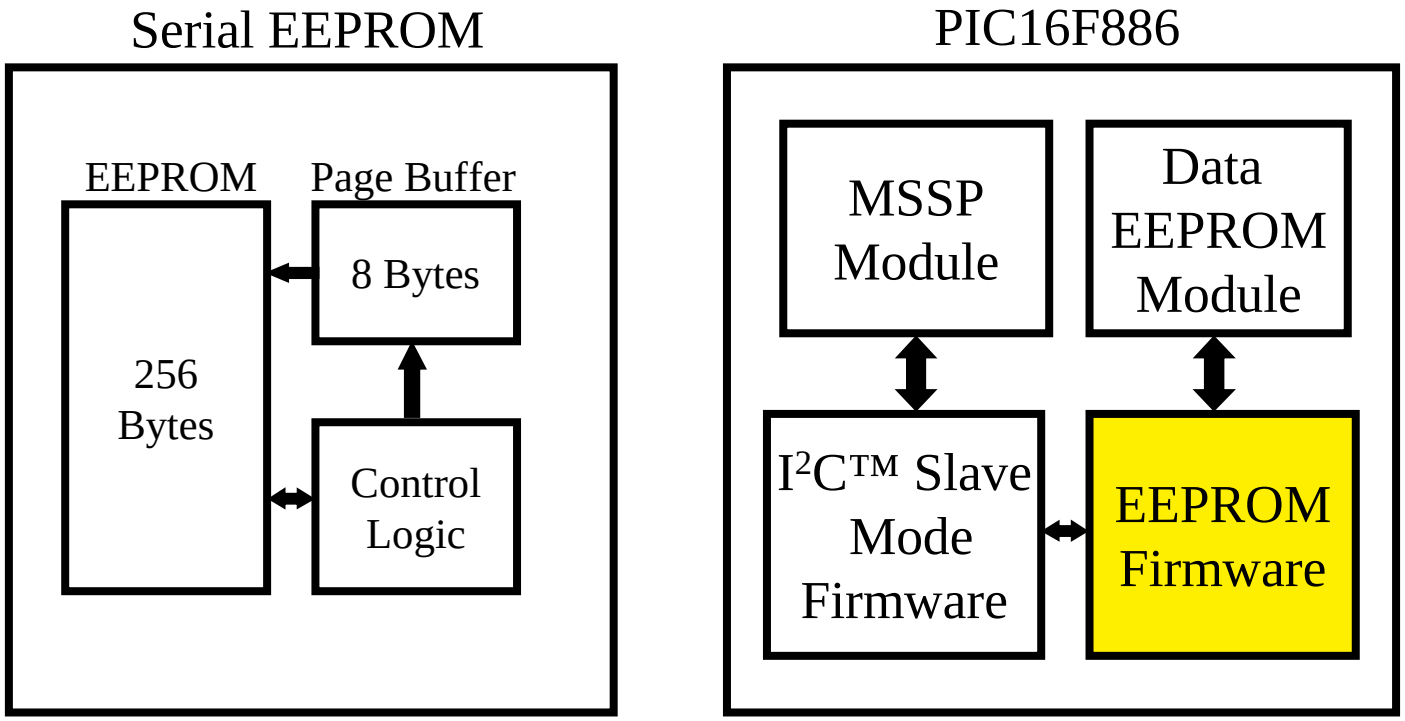
- Serial EEPROM Structure
- PIC16F886 Peripherals and Firmware

I²C™ Slave Mode Firmware

- I²C Firmware must be modified to access data EEPROM and Page Buffer



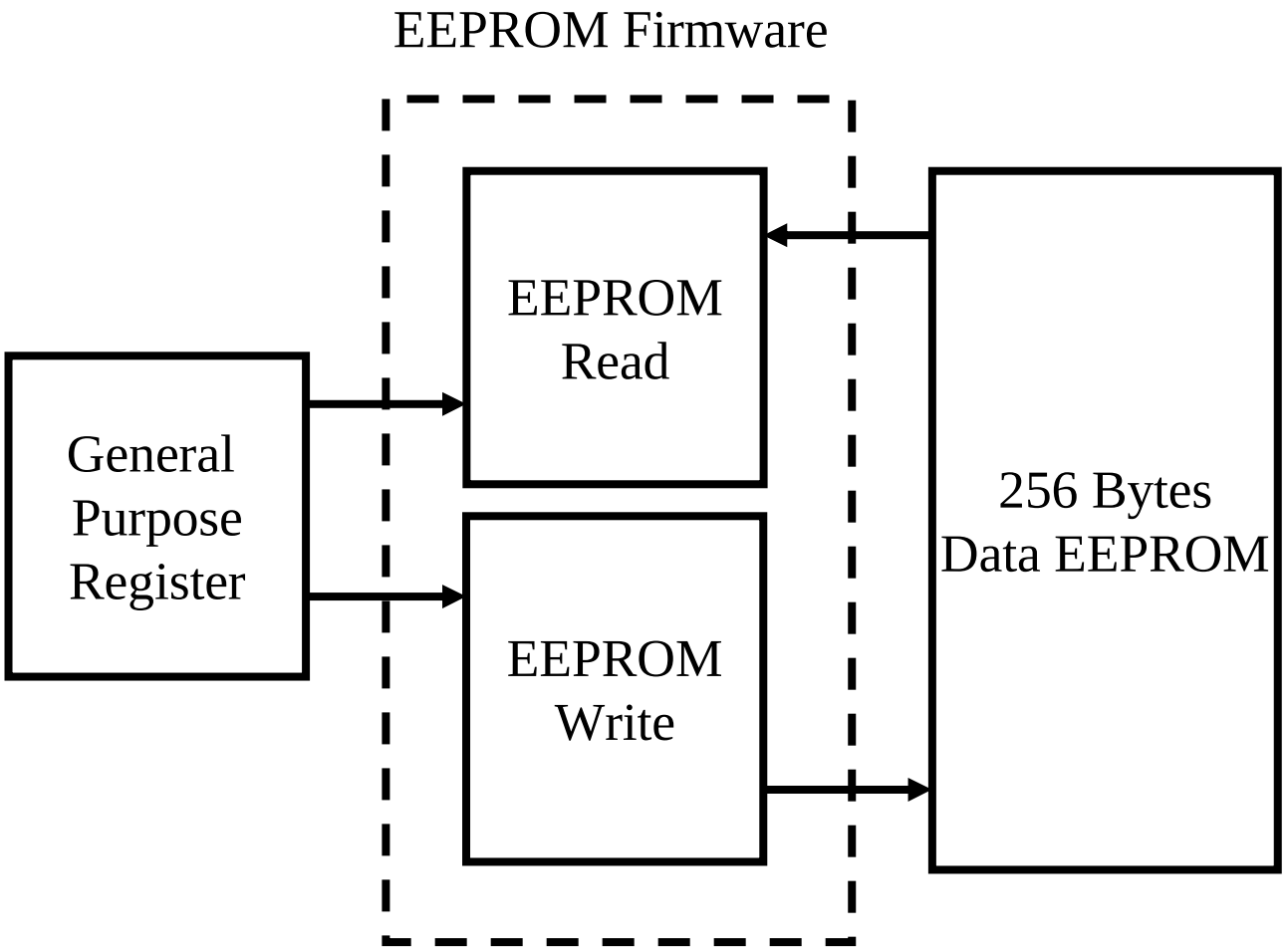
Device Comparison



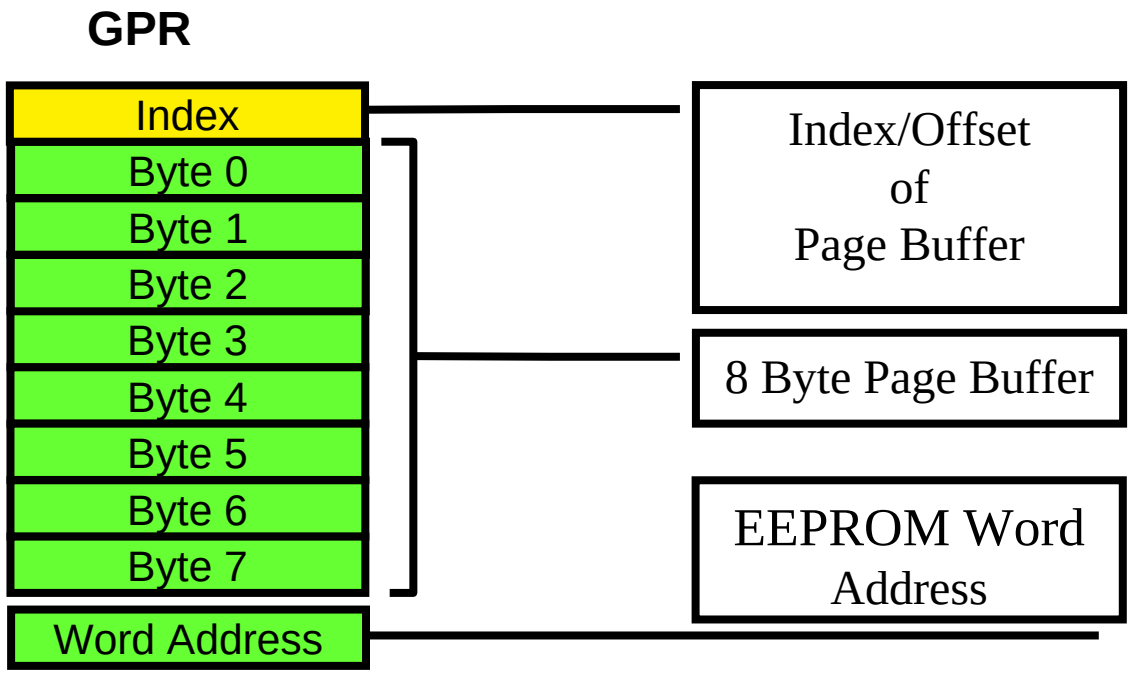
- Serial EEPROM Structure
- PIC16F886 Peripherals and Firmware

EEPROM Firmware

- Purpose: Read and Write EEPROM

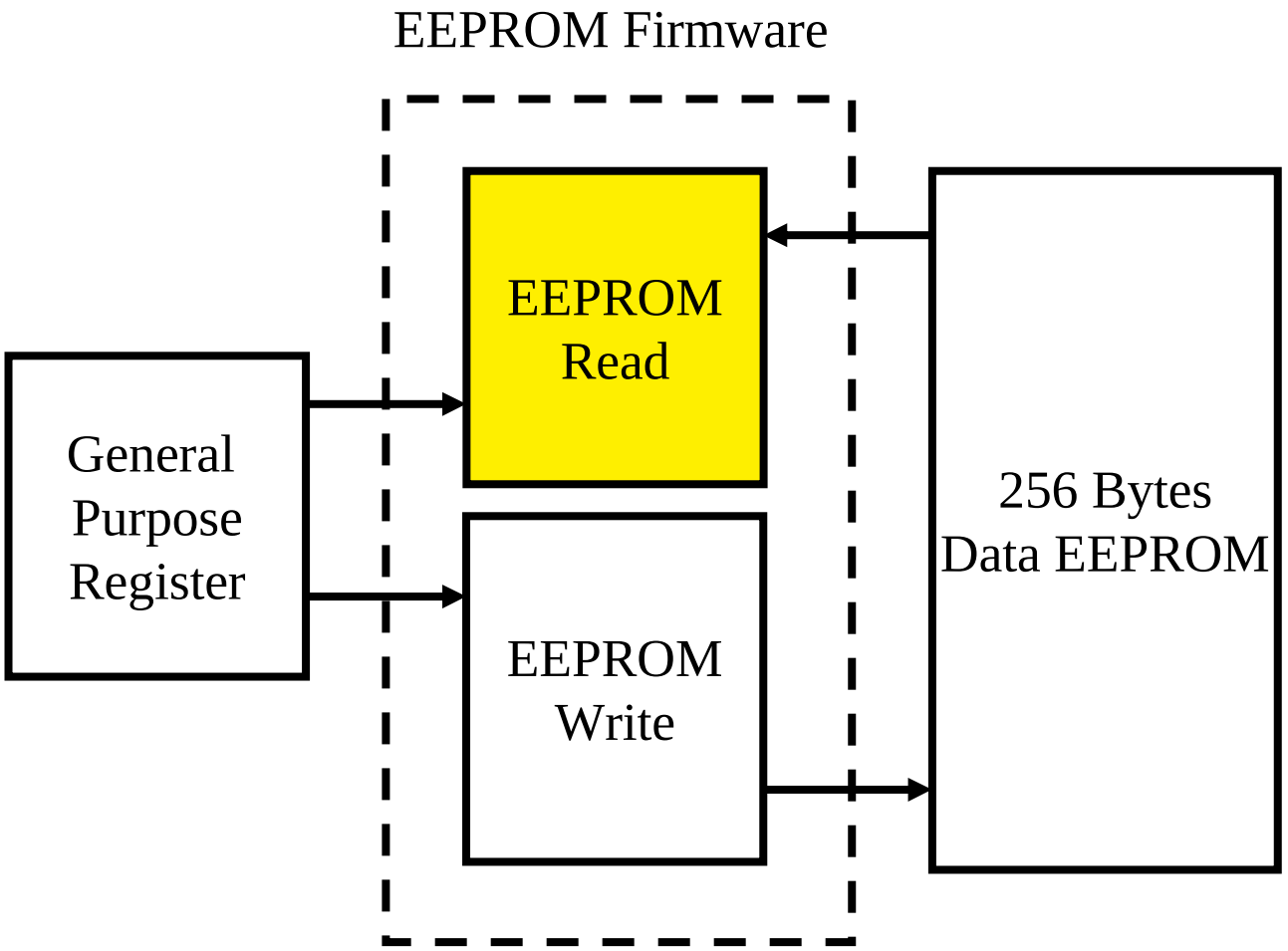


General Purpose Registers



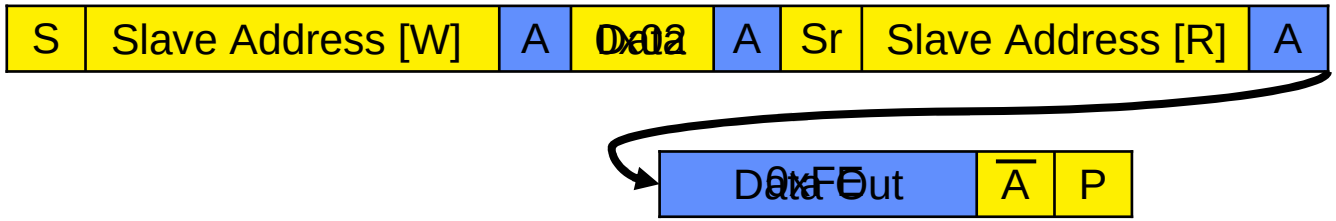
EEPROM Firmware

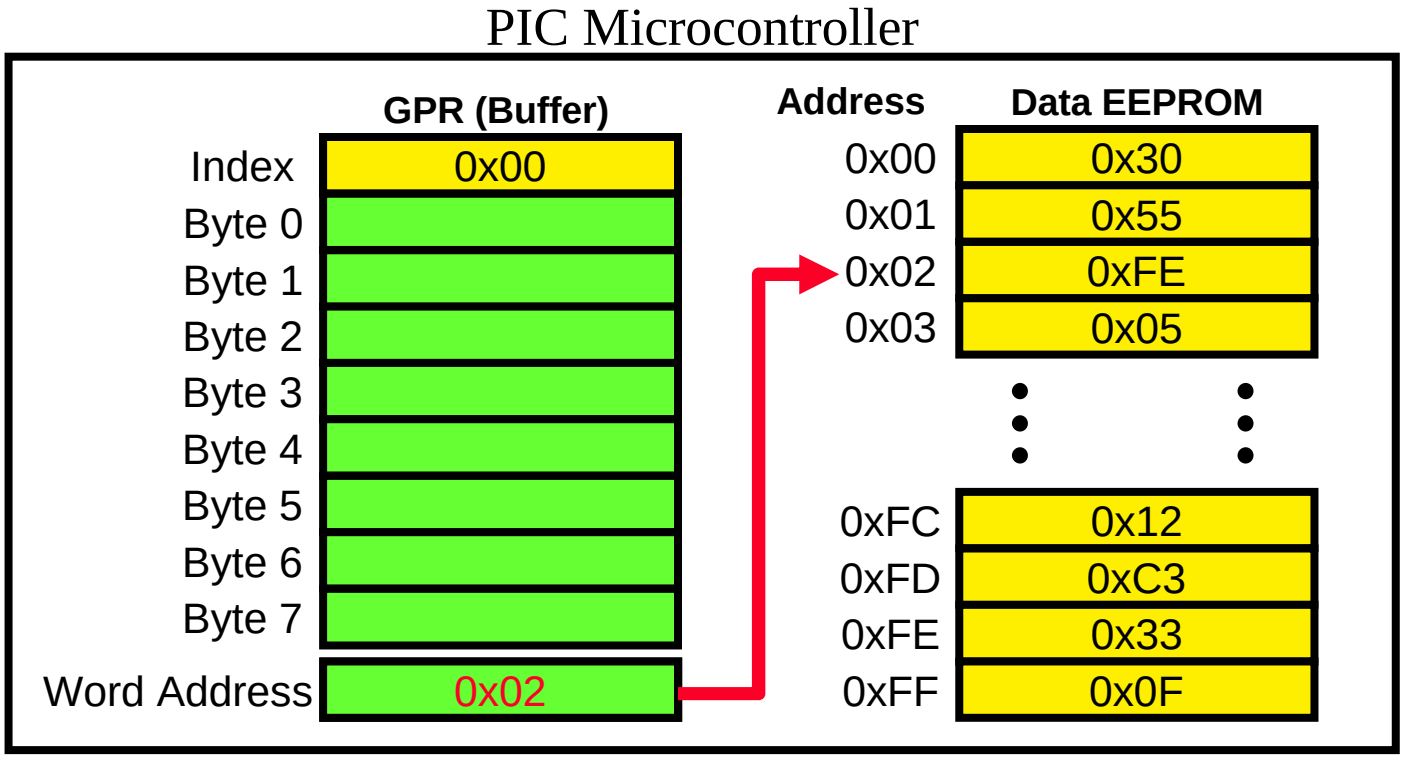
- EEPROM Read:



EEPROM Firmware: MSSP Interrupt

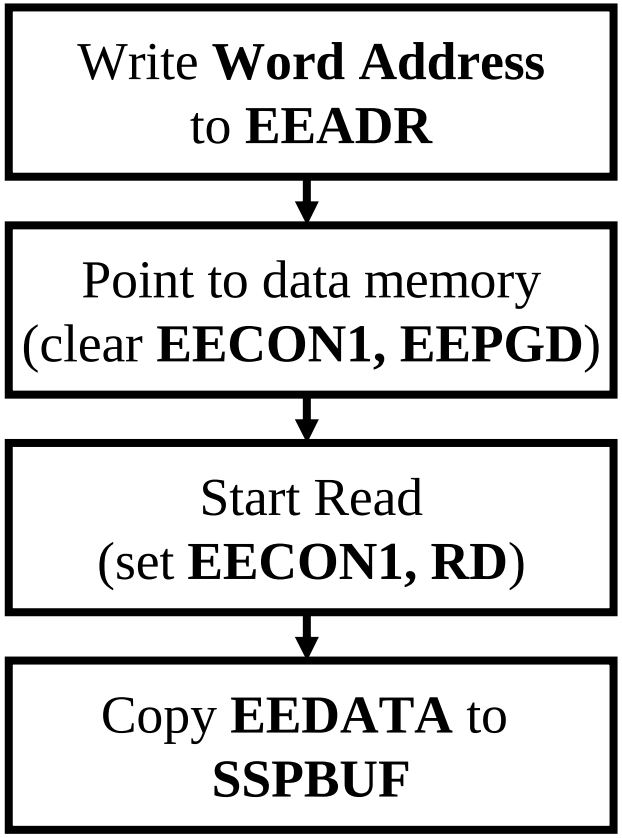
- EEPROM Read:





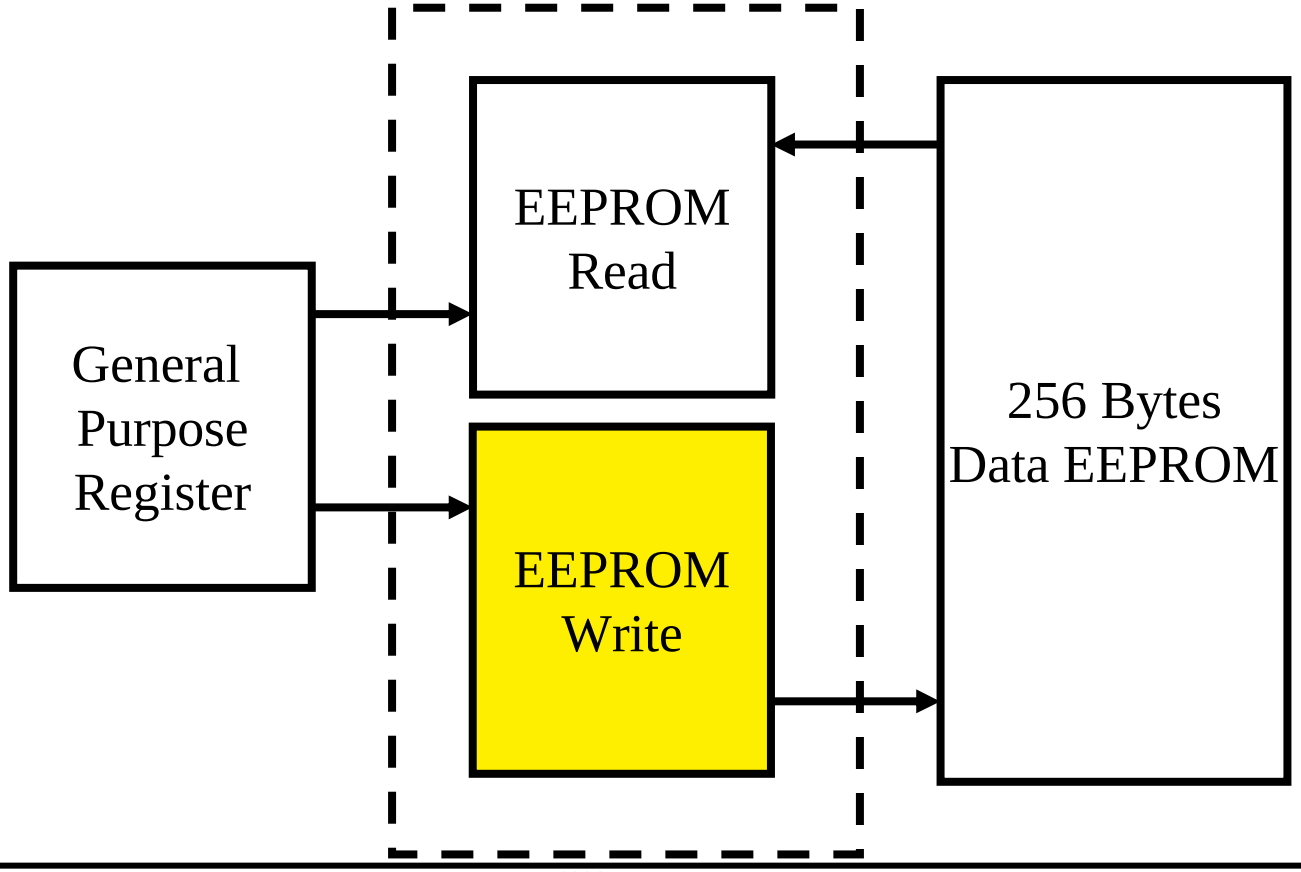
EEPROM Firmware

- EEPROM Read:



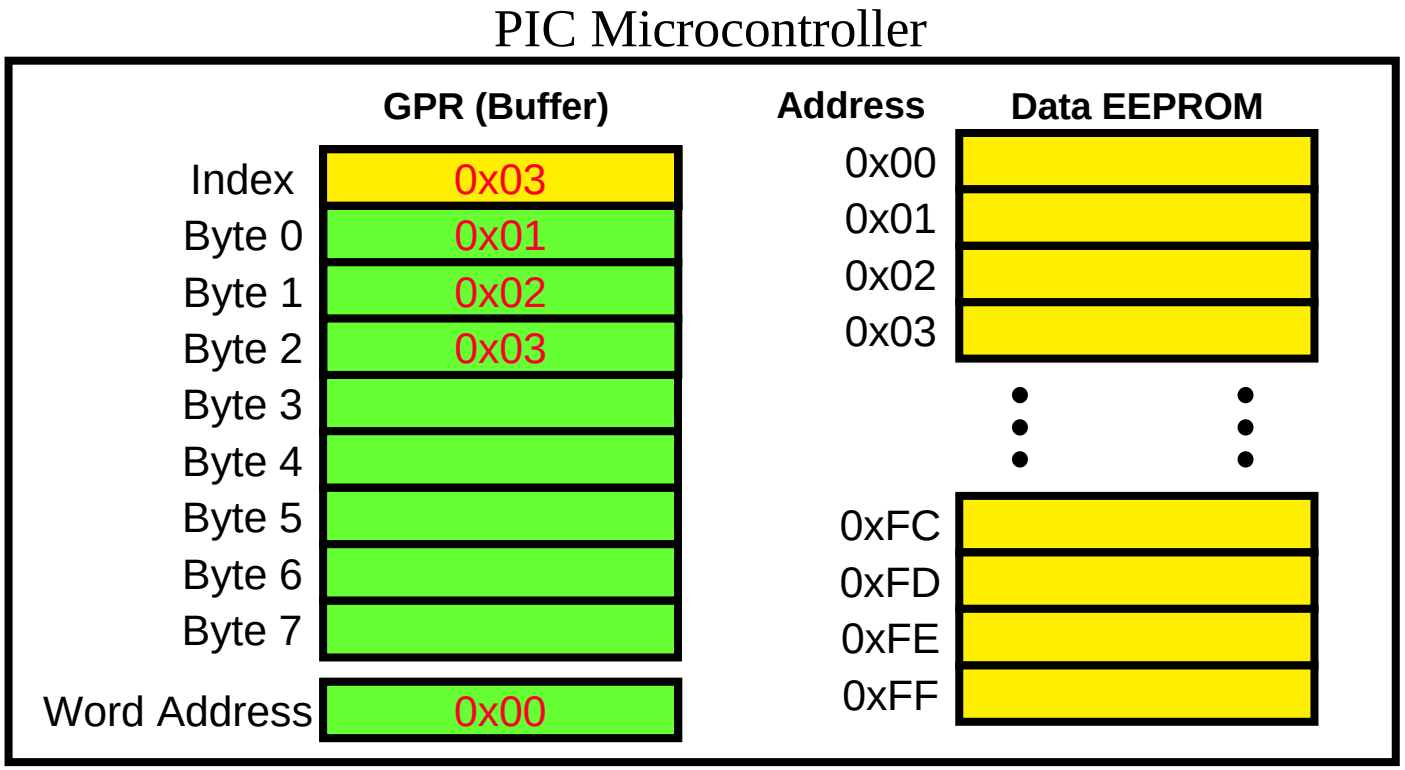
EEPROM Firmware

- EEPROM Write:
  - 3 Step Process:
  - 1) MSSP Interrupt 2) Mainline Code 3) EEPROM Interrupt



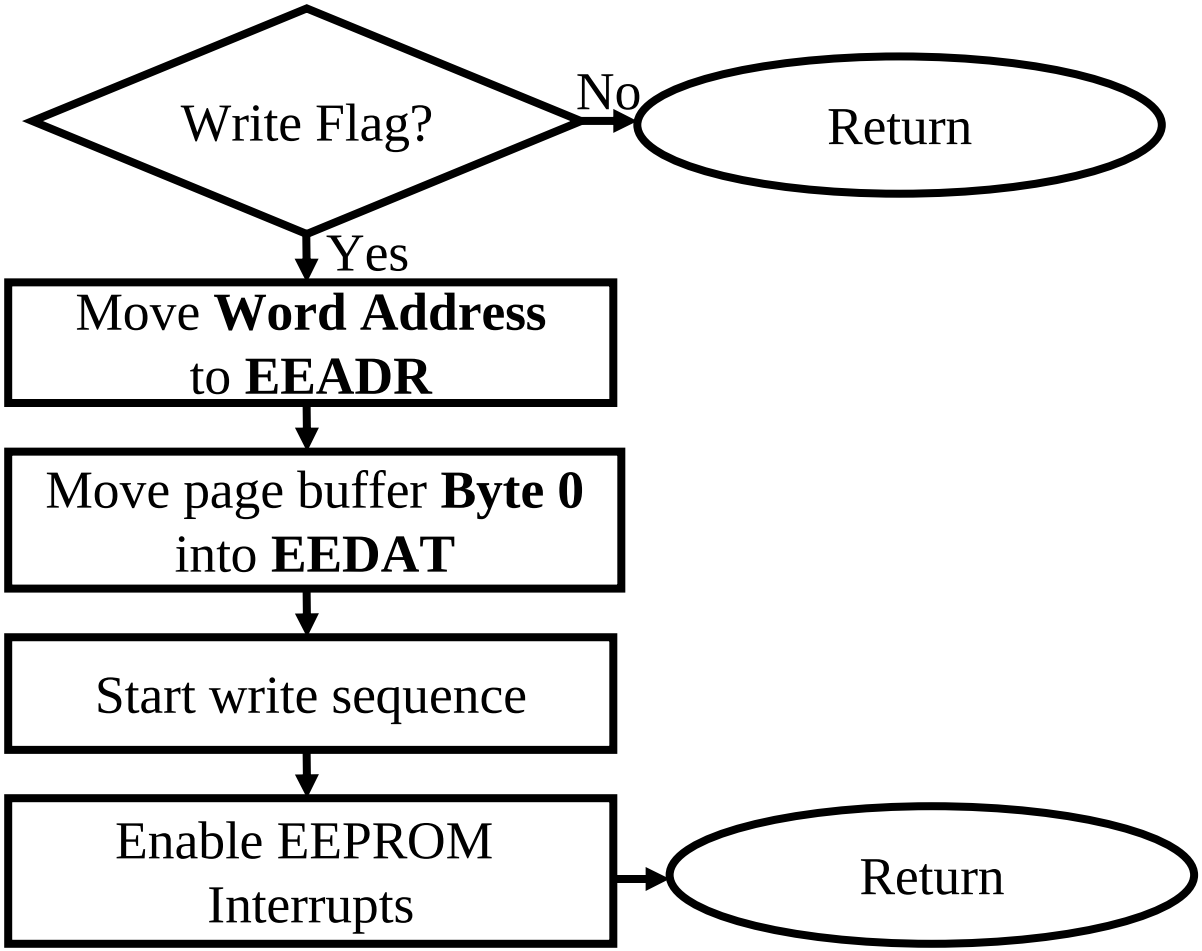
EEPROM Firmware: 1) MSSP Interrupt

- Writing to data buffer



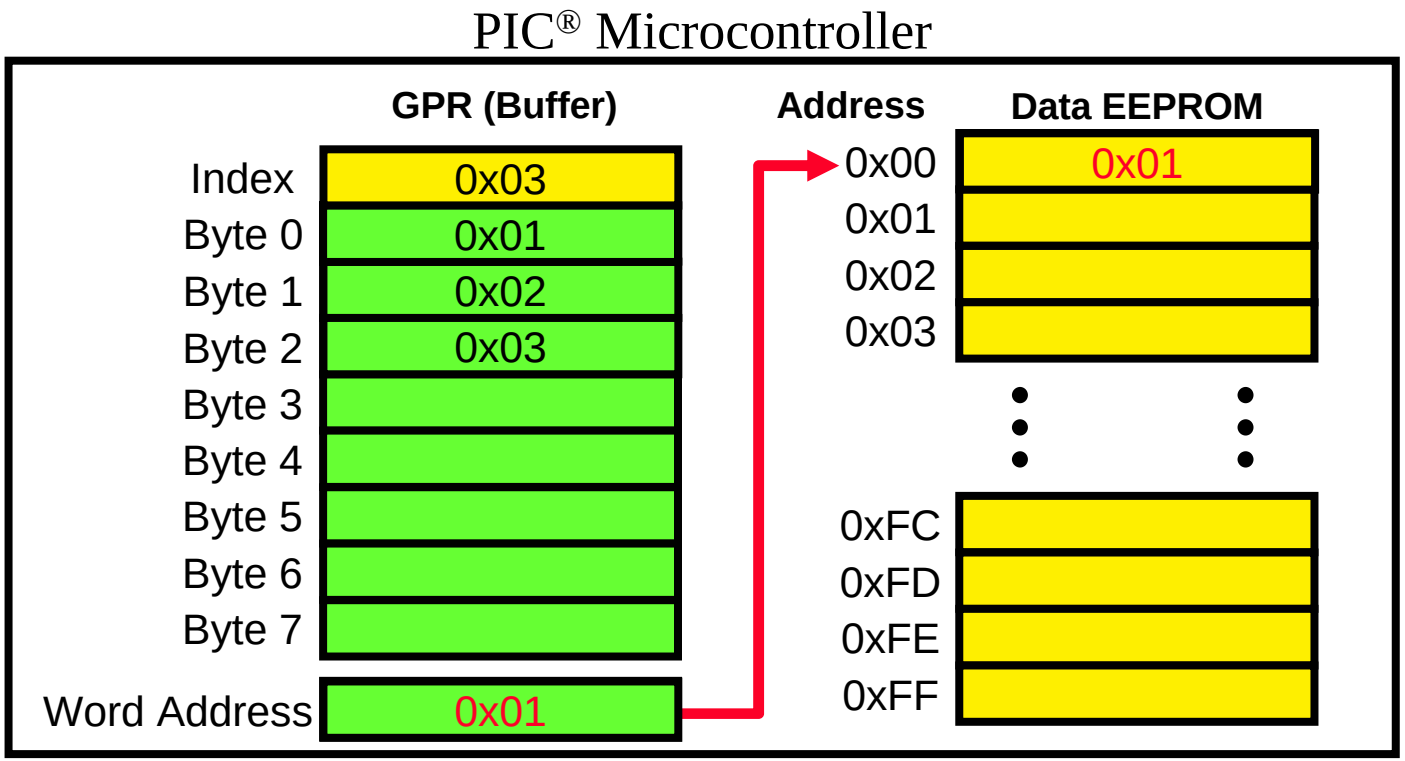
EEPROM Firmware: Mainline Code

- EEPROM Write:



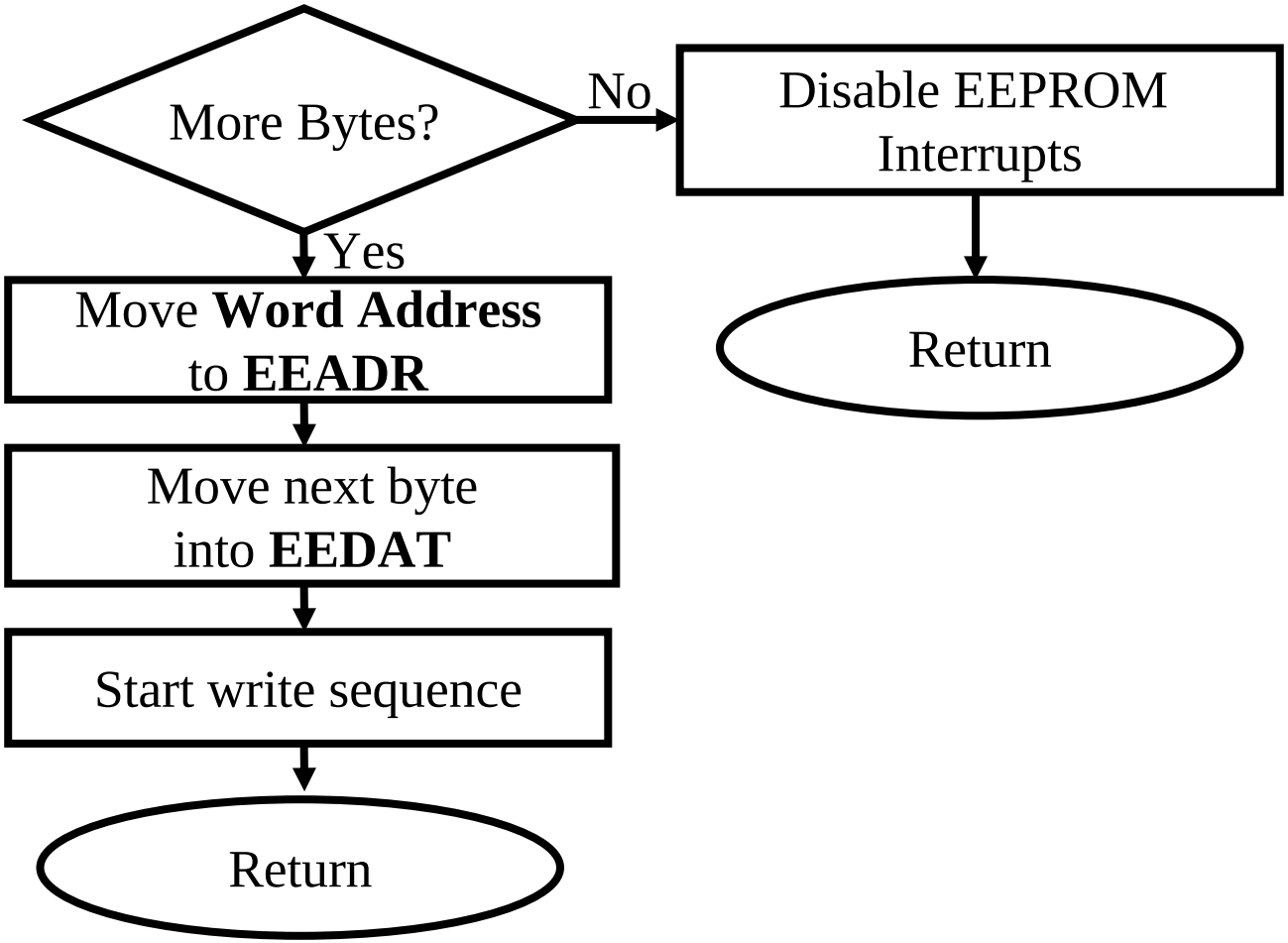
EEPROM Firmware: Mainline Code

- Initiates write to data EEPROM
  - EEPROM Write Flag
- Enables EEPROM Interrupts



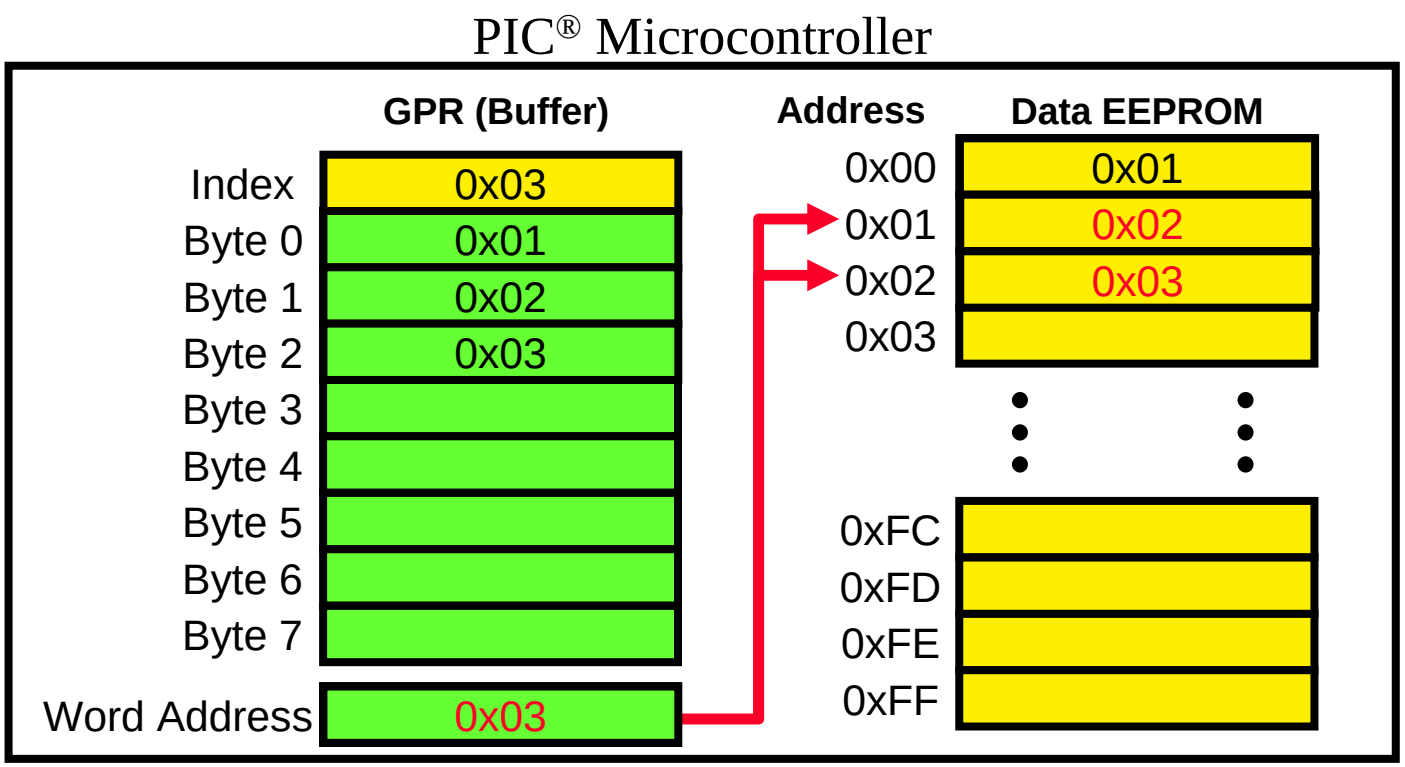
EEPROM Firmware: EEPROM ISR

- EEPROM Write:



EEPROM Interrupts

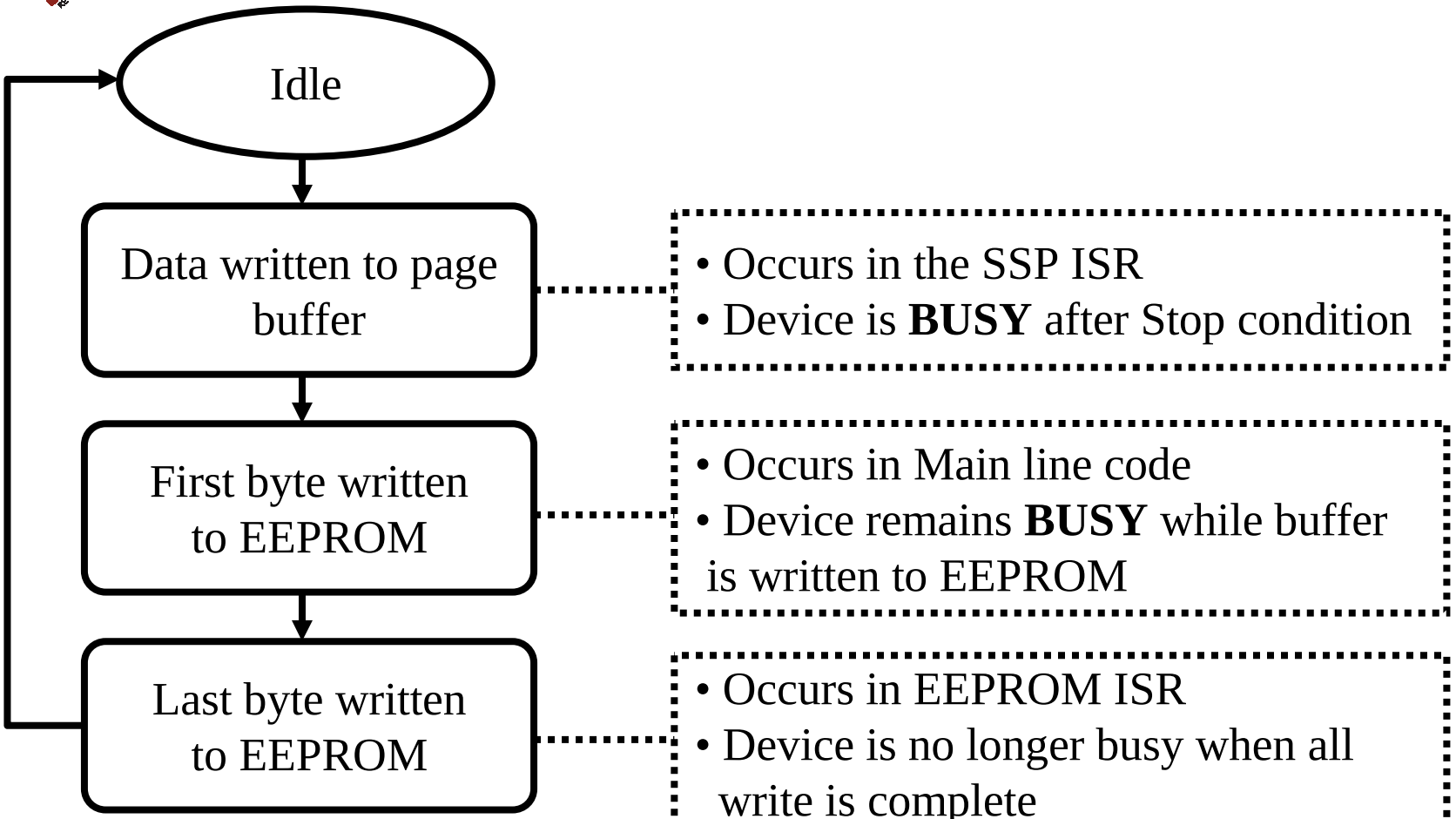
- Writes remaining bytes to EEPROM
- Disables EEPROM Interrupts when complete



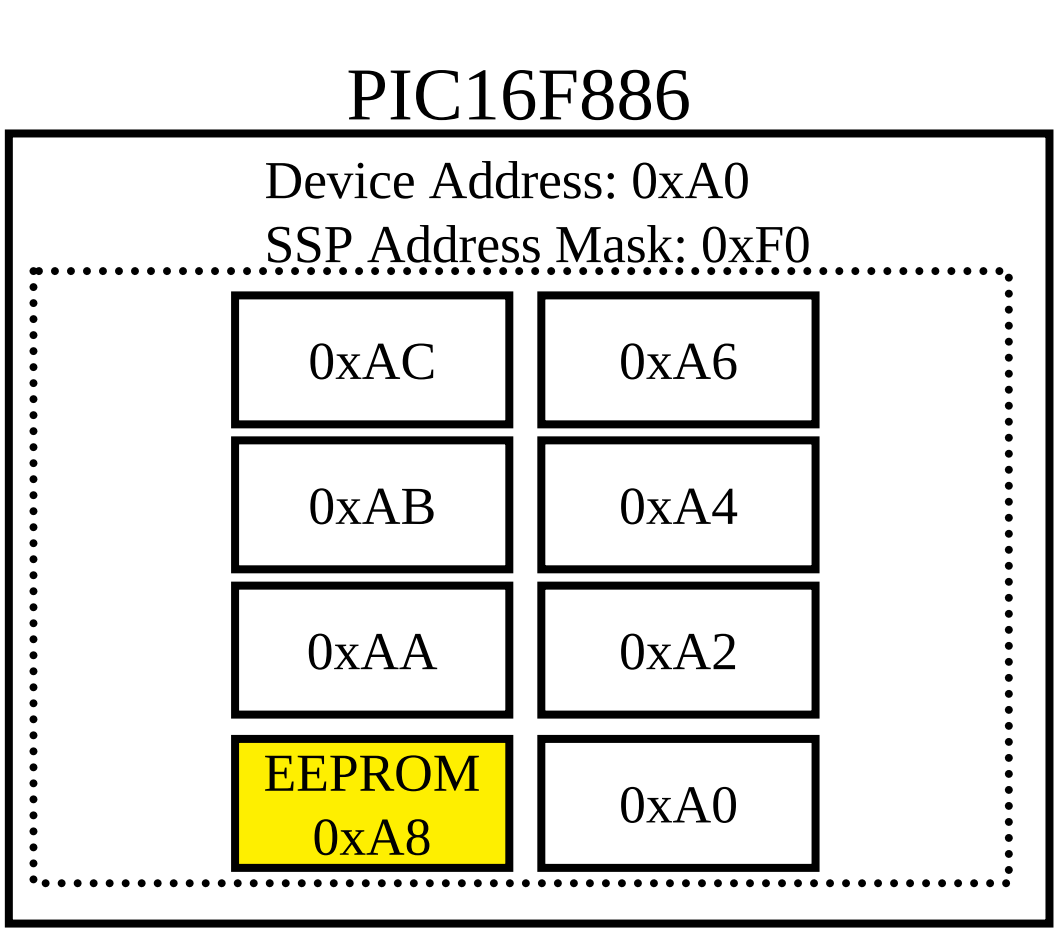
Functionality Comparison

- Serial EEPROM
  - Data write and read
  - Page Buffer
  - Hardware defined device address
  - No acknowledge when busy
- PIC16F886 Serial EEPROM
  - Data write and read
  - Page Buffer
  - Software defined device address
  - **Must implement no acknowledge when busy**

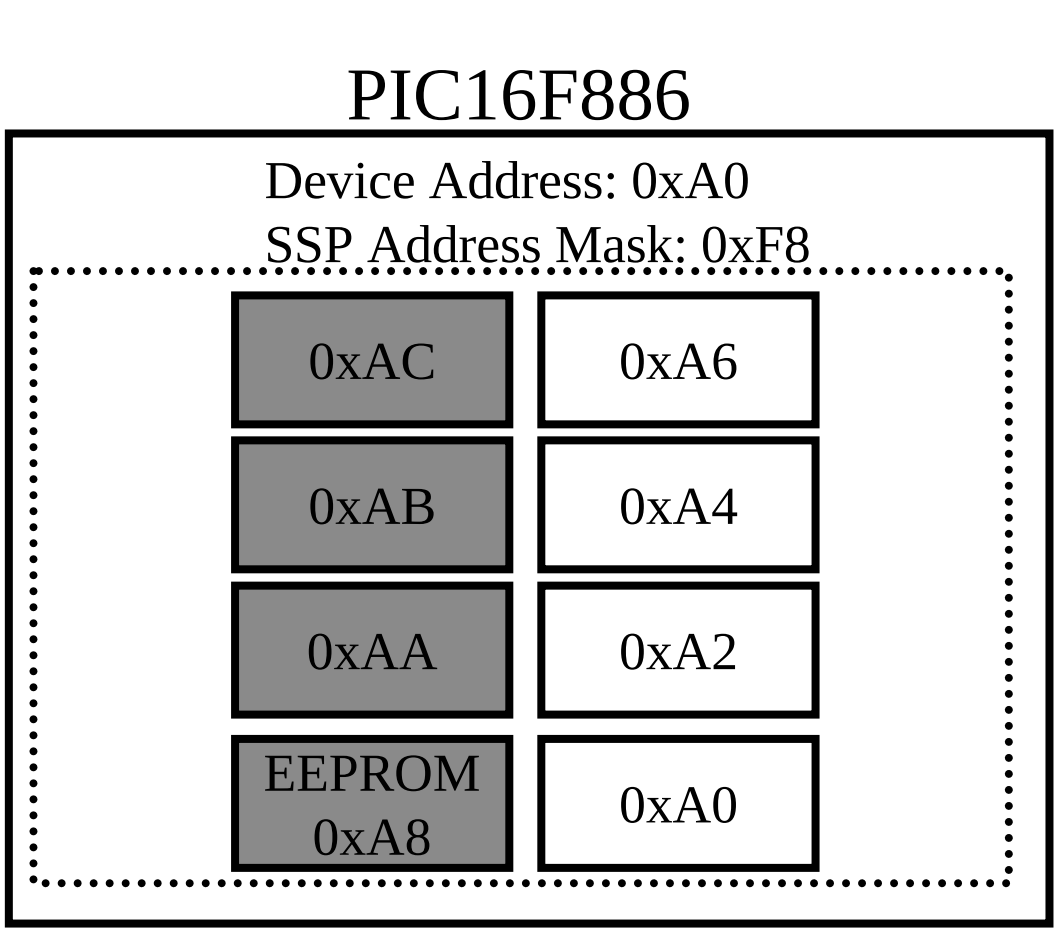
When is the device busy?



Acknowledge Polling – Address Masking



Acknowledge Polling – Address Masking (BUSY)



Summary of Comparison

- ☑ 256B Data EEPROM
- ☑ 8-byte Page Buffer
- ☑ Acknowledge Polling
- ☑ Hardware Defined Control Byte
- ☑ I^2C™ Communications
- **Throughput**
  - **8X more time to write page**
  - **Tradeoffs?**

LAB 2: Serial EEPROM Lab

Summary

- Serial EEPROM Basics
- PIC Data EEPROM
- PIC® MCU Serial EEPROM Implementation
- Lab Demonstration

- Questions?

I²C™ Real-Time Clock Module

309SMW

Module Learning Objectives

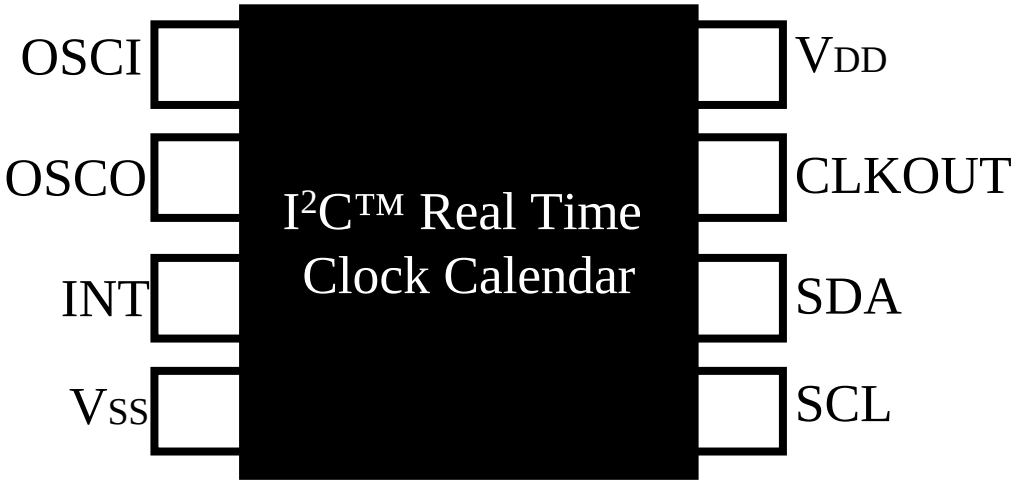
- Learning Objectives:
  - Explain how I²C™ Real-Time Clocks work
  - Emulate a real-time clock using the PIC16F886 Timer and MSSP peripherals
  - Implement a battery backup circuit
- Prerequisites
  - Timer 1 Peripheral
  - Serial Synchronous Port Module
  - I²C Slave Mode Firmware

Module Agenda

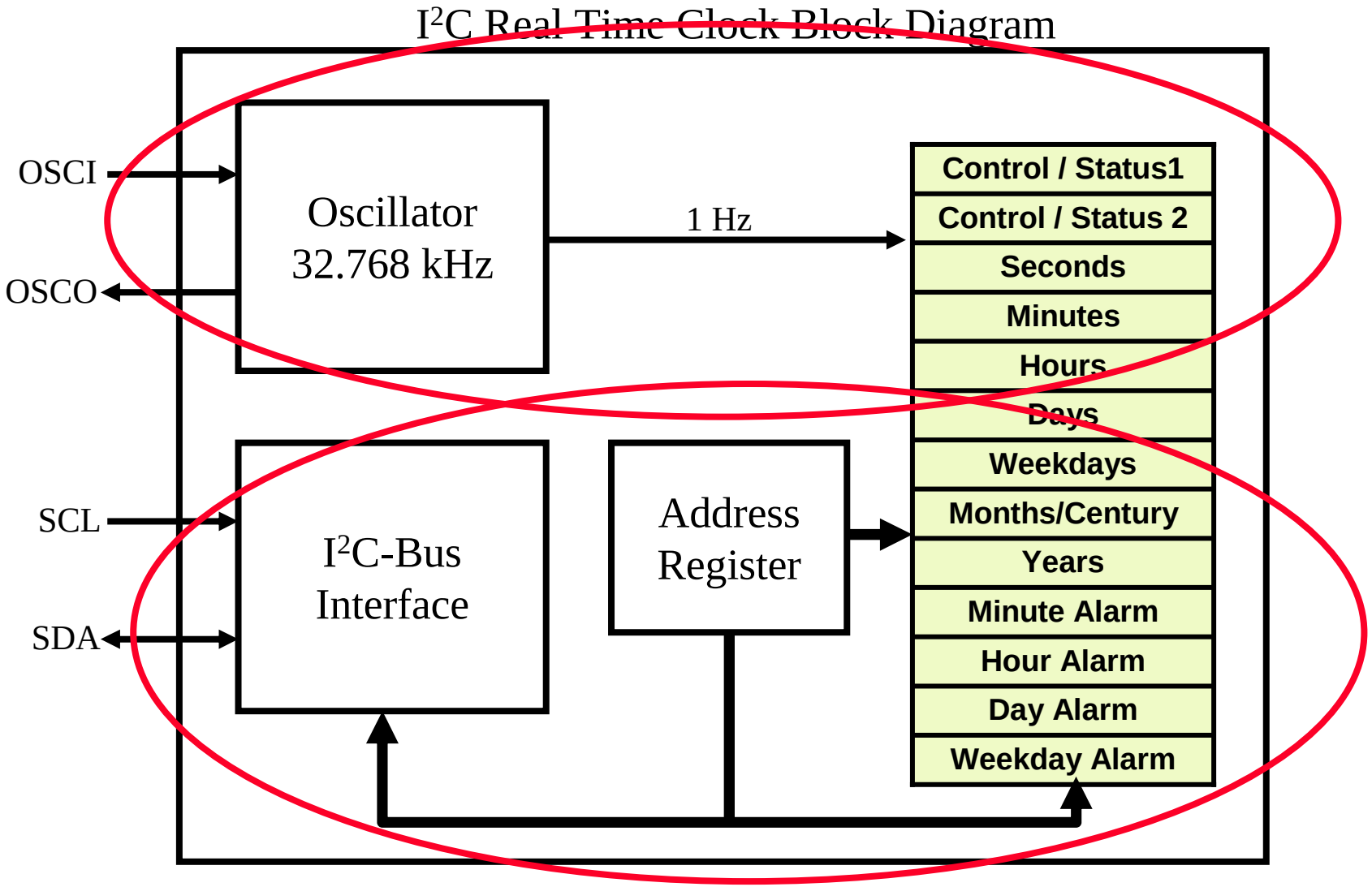
- Introduction to the Real-Time Clock
- Emulating I²C™ Real-Time Clock
- Battery Backup
- Lab Demonstration

Introduction to the I²C™ Real-Time Clock

- Provides Accurate Time
- Serial Communications
- Applications: System Management, Data Logging



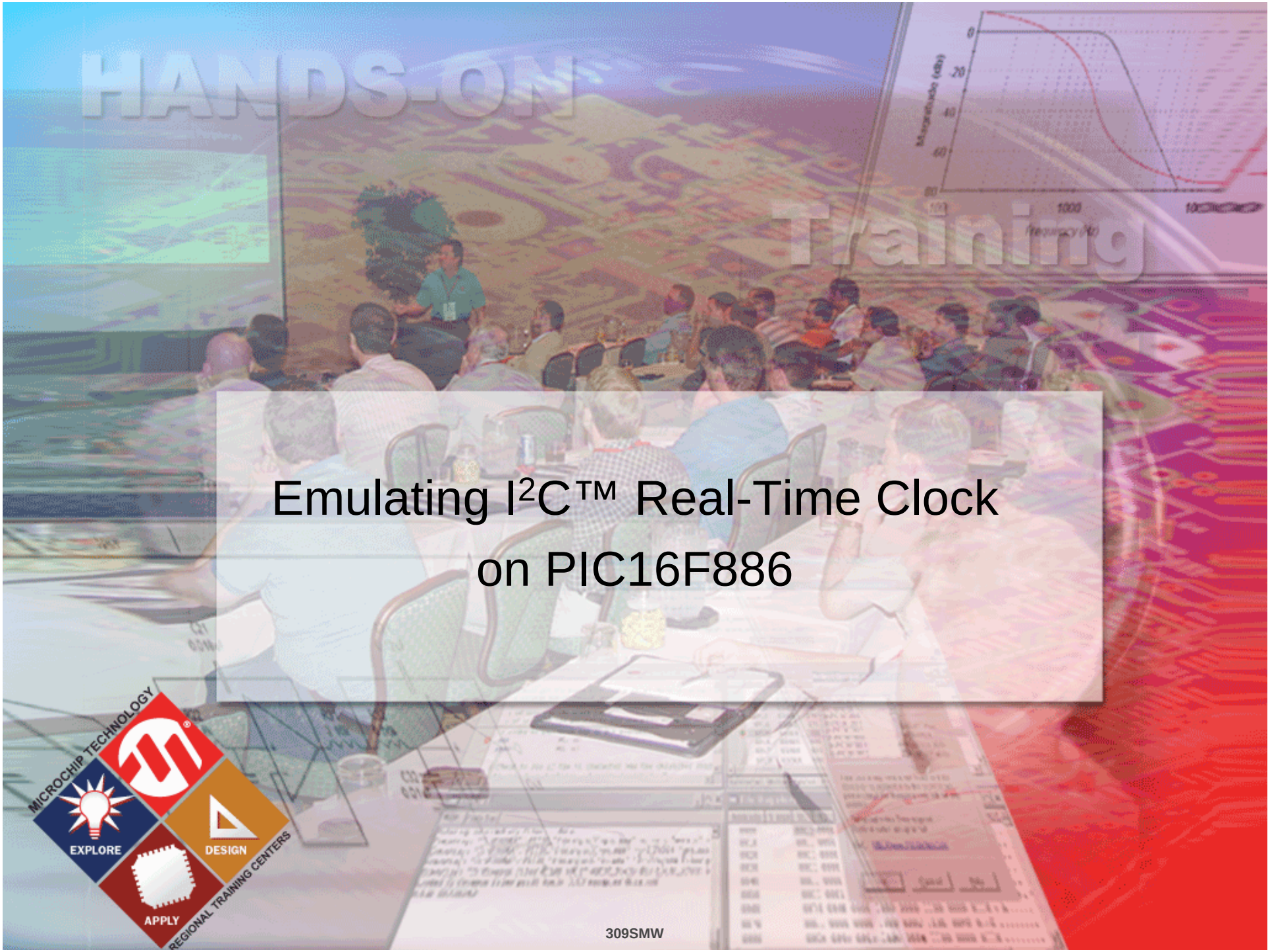
Introduction to the I²C™ Real-Time Clock



Real-Time Clock/Calendar Register Map

- Control and Status
  - Operational Modes
  - Alarm Modes
  - Binary formatted
- Incremented Registers
  - Time Data
  - Alarm values
  - BCD formatted

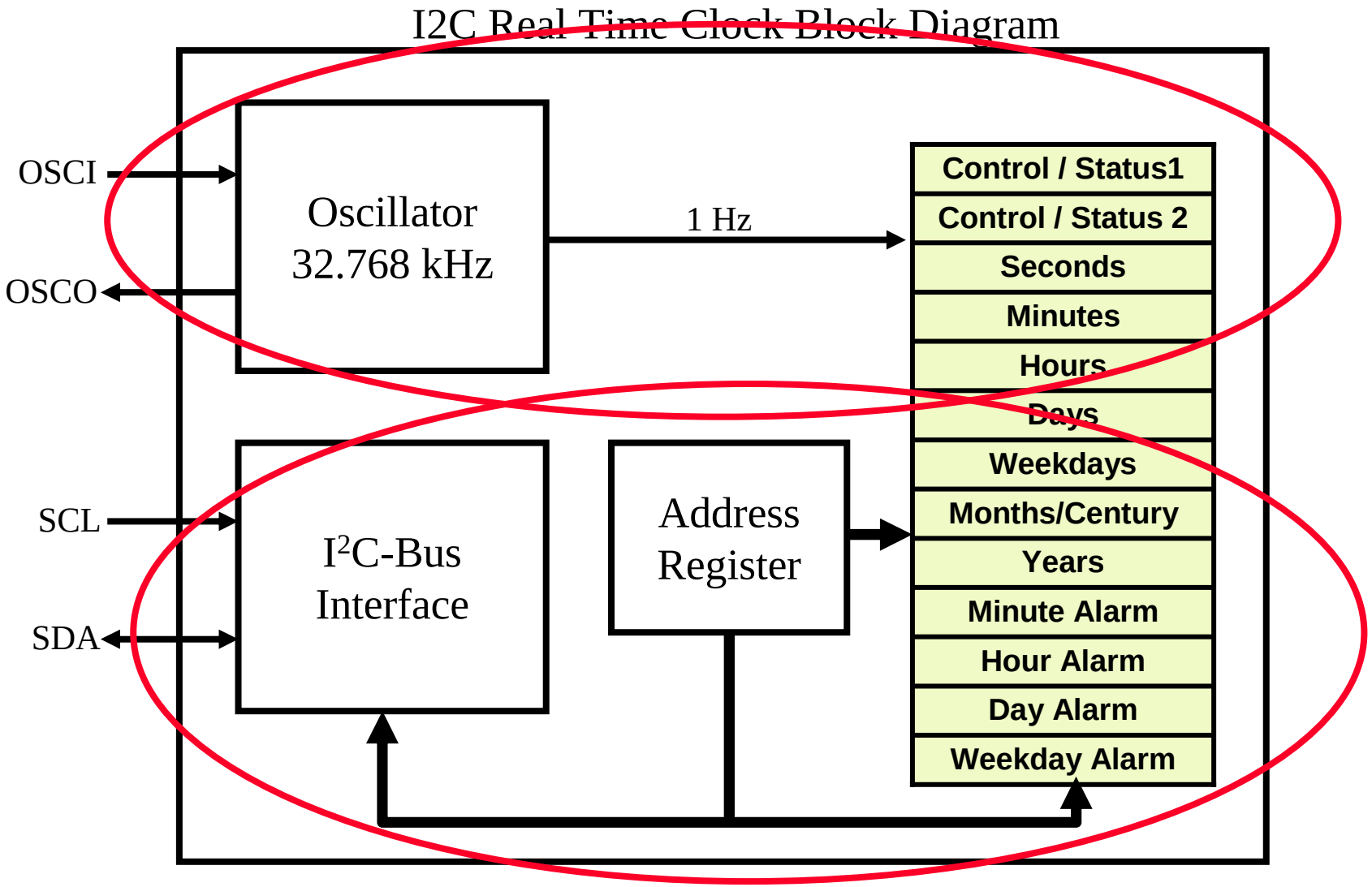
Register Name	Address
Control / Status1	00h
Control / Status 2	01h
Seconds	02h
Minutes	03h
Hours	04h
Days	05h
Weekdays	06h
Months/Century	07h
Years	08h
Minute Alarm	09h
Hour Alarm	0Ah
Day Alarm	0Bh
Weekday Alarm	0Ch



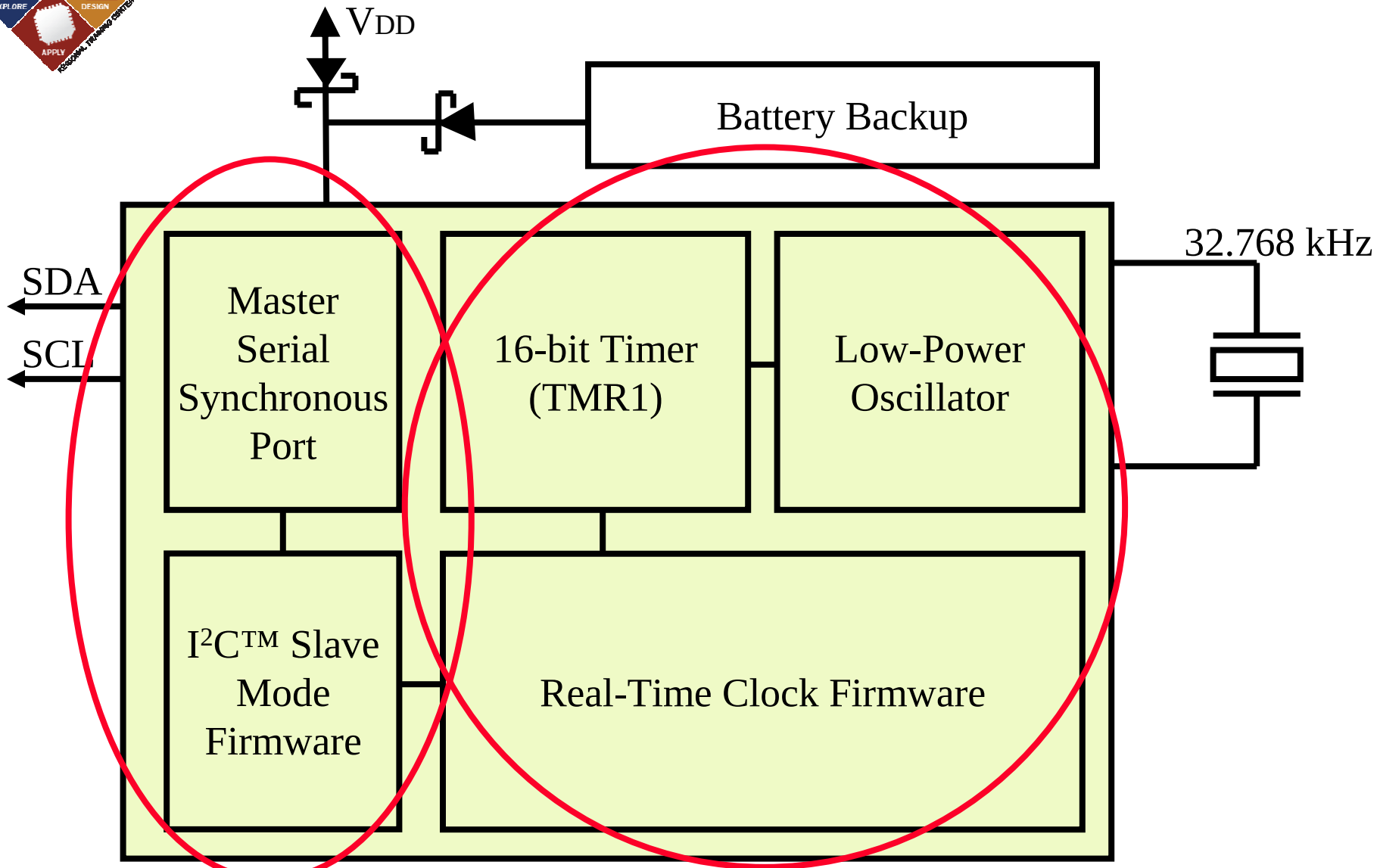
Emulating I²C™ Real-Time Clock on PIC16F886

309SMW

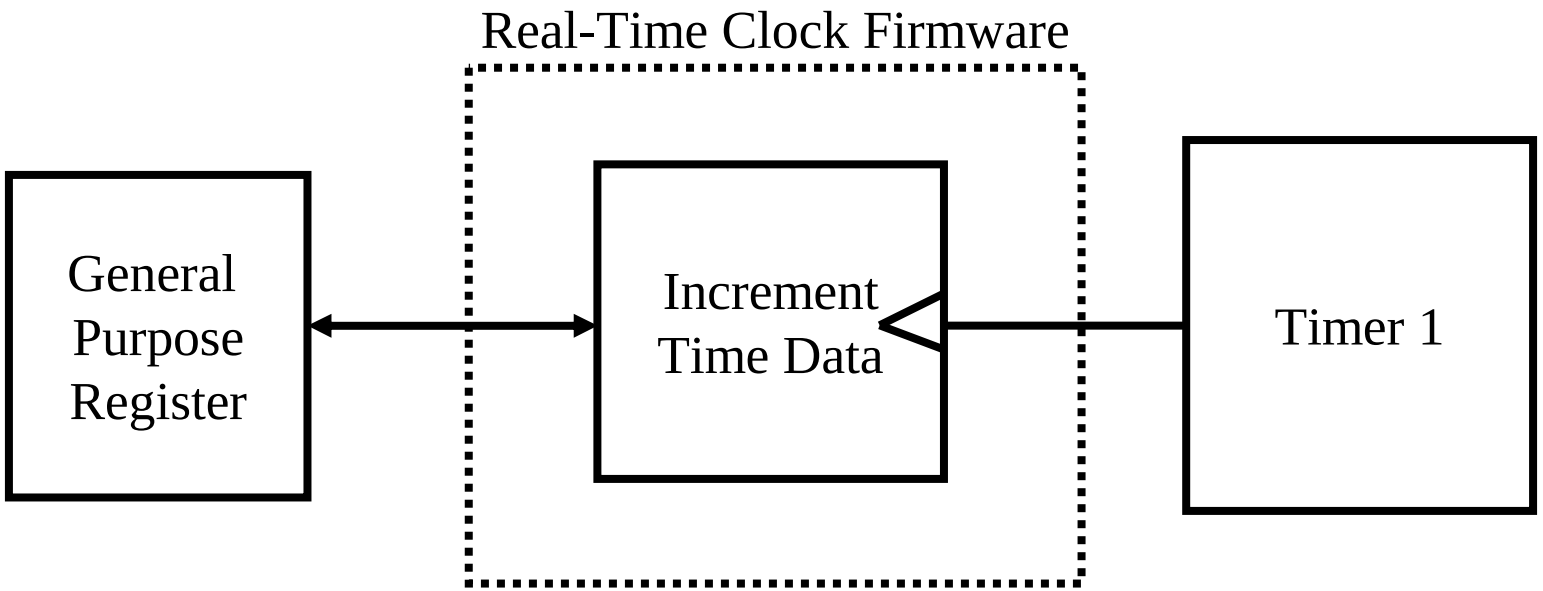
Introduction to the I²C™ Real-Time Clock



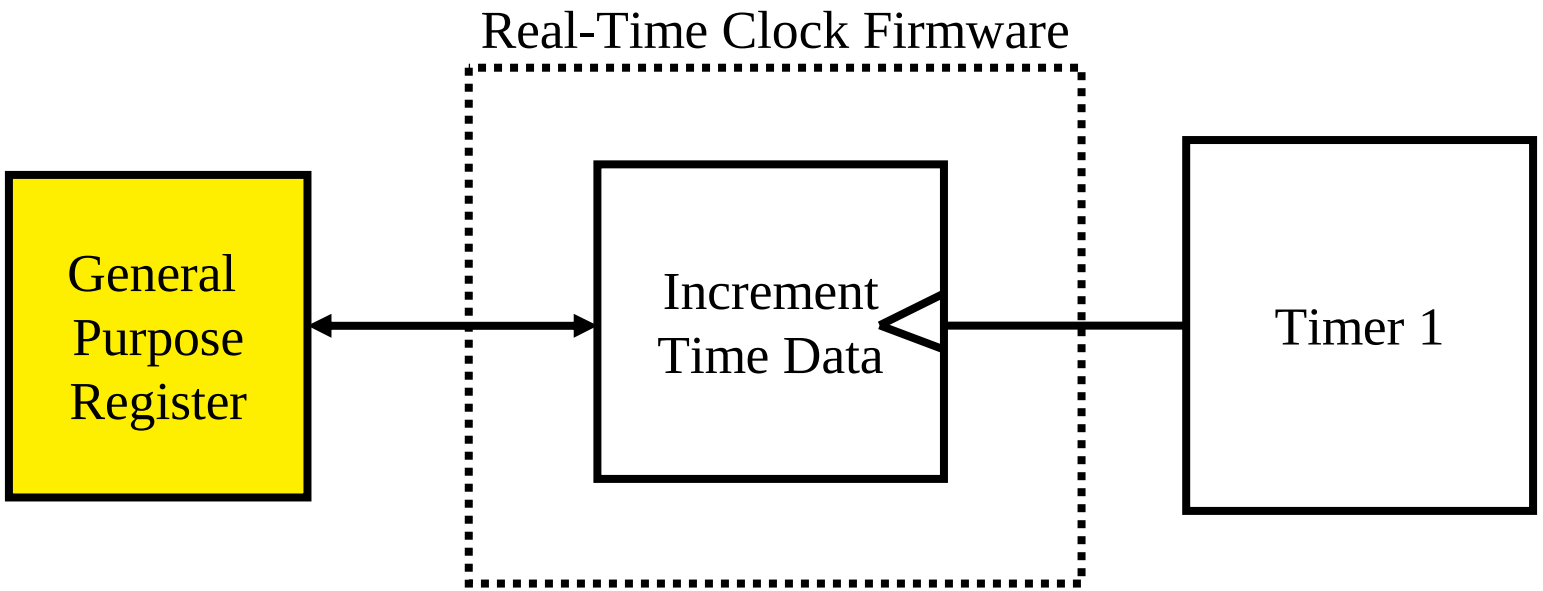
Real-Time Clock Implementation



Real-Time Clock Firmware

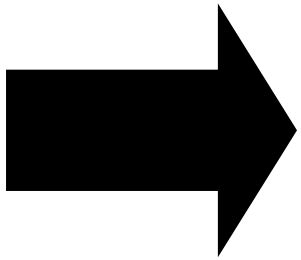


Real Time Clock Firmware



Structure RAM to function within I²C™ Firmware

Register Name	Address
Control / Status1	00h
Control / Status 2	01h
Seconds	02h
Minutes	03h
Hours	04h
Days	05h
Weekdays	06h
Months/Century	07h
Years	08h
Minute Alarm	09h
Hour Alarm	0Ah
Day Alarm	0Bh
Weekday Alarm	0Ch



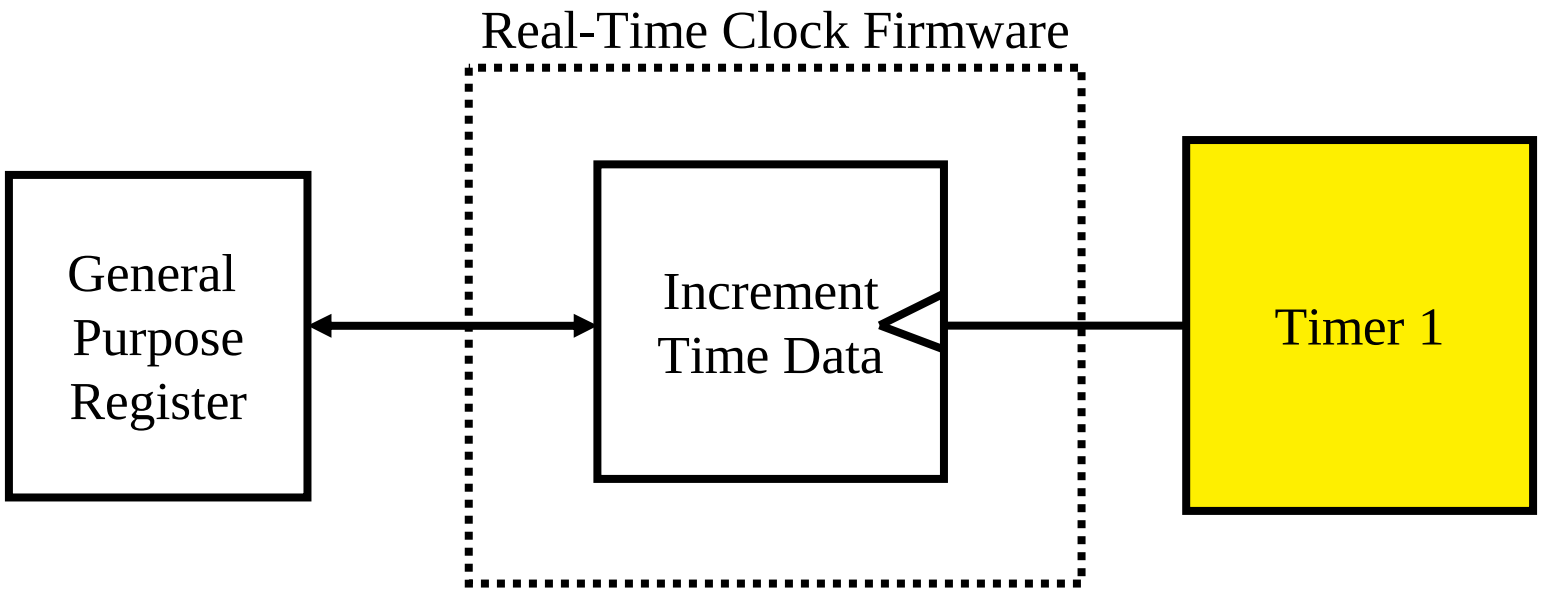
General Purpose Registers	Offset
Offset / Index	00h
Control / Status1	01h
Control / Status 2	02h
Seconds	03h
Minutes	04h
Hours	05h
Days	06h
Weekdays	07h
Months/Century	08h
Years	09h
Minute Alarm	0Ah
Hour Alarm	0Bh
Day Alarm	0Ch
Weekday Alarm	0Dh

RAM Definition

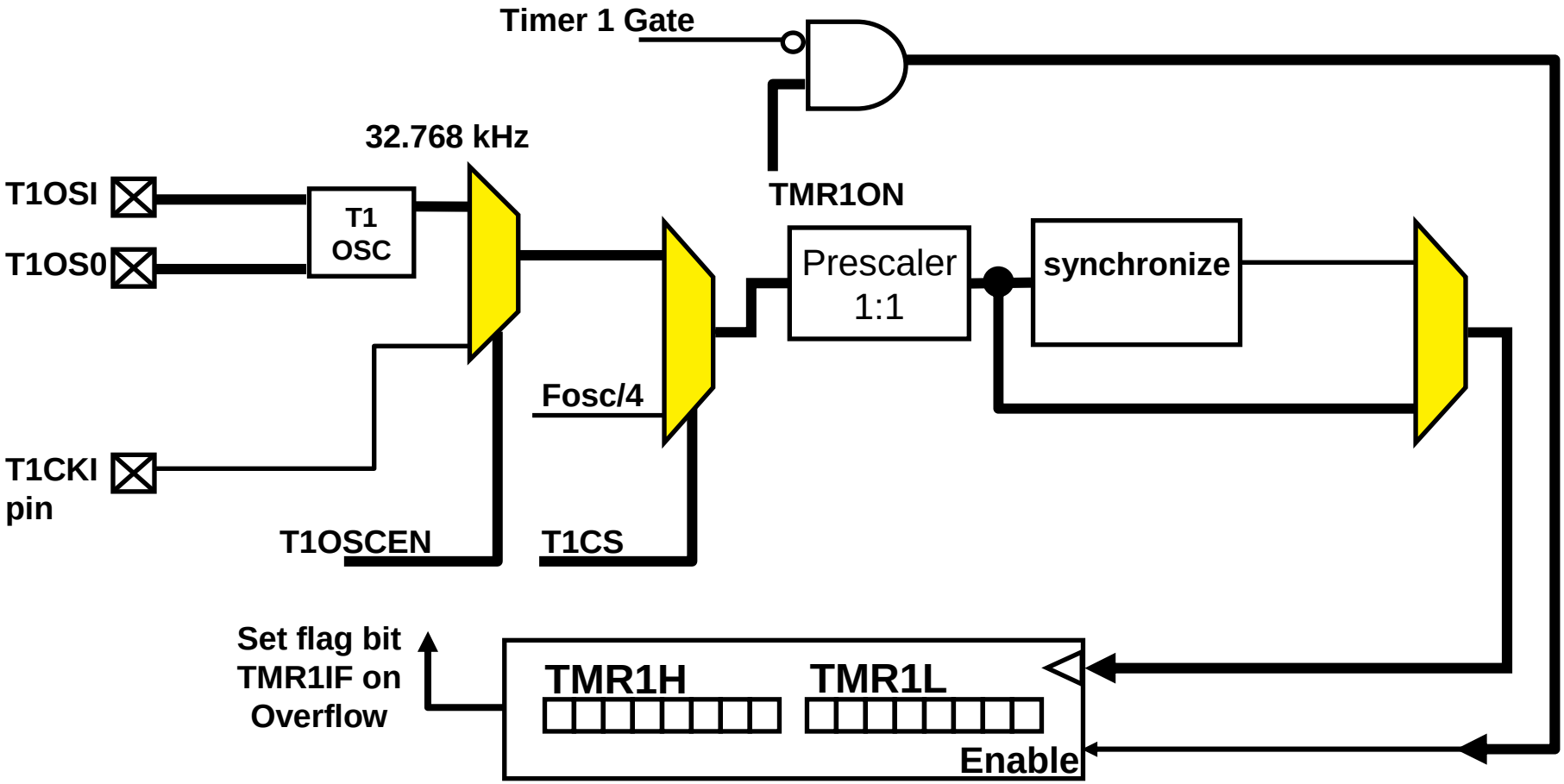
- Located in rtc.inc

```
;RTC_Buf is defined in rtc.asm
;  as 'RTC_buf res RTC_BUF_LEN'

#define RTC_BUF_LEN .14        ; RTC Buffer Length

#define RTC_Configure_1        (RTC_Buf + 0x01)
#define RTC_Configure_2        (RTC_Buf + 0x02)

#define RTC_Seconds            (RTC_Buf + 0x03)
#define RTC_Minutes            (RTC_Buf + 0x04)
#define RTC_Hours              (RTC_Buf + 0x05)
#define RTC_Weekdays           (RTC_Buf + 0x06)
#define RTC_Days               (RTC_Buf + 0x07)
#define RTC_Months             (RTC_Buf + 0x08)
#define RTC_Years              (RTC_Buf + 0x09)

#define RTC_Min_Alarm          (RTC_Buf + 0x0A)
#define RTC_Hr_Alarm           (RTC_Buf + 0x0B)
#define RTC_Day_Alarm          (RTC_Buf + 0x0C)
#define RTC_Week_Alarm         (RTC_Buf + 0x0D)
```

Real-Time Clock Firmware



Timer 1



Calculating Time Between Interrupts

- Equation:

Interrupt Interval (sec) = (Prescaler / Clock Frequency) * 2^{16}

- Preloading Timer 1 Equation:

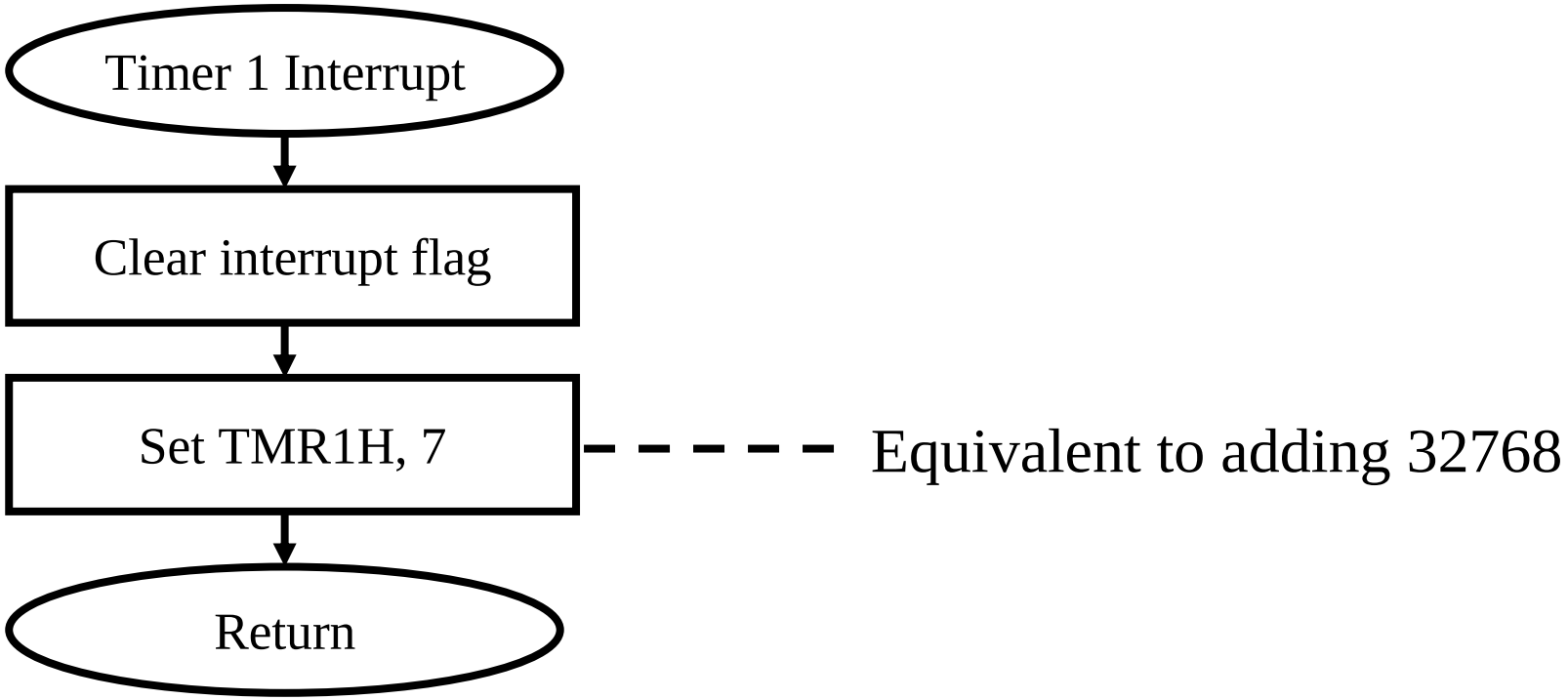
Interrupt Interval (sec) = (Prescaler / Clock Frequency) * **(2^{16} – Preloaded Value)**

Generating Accurate Time Base

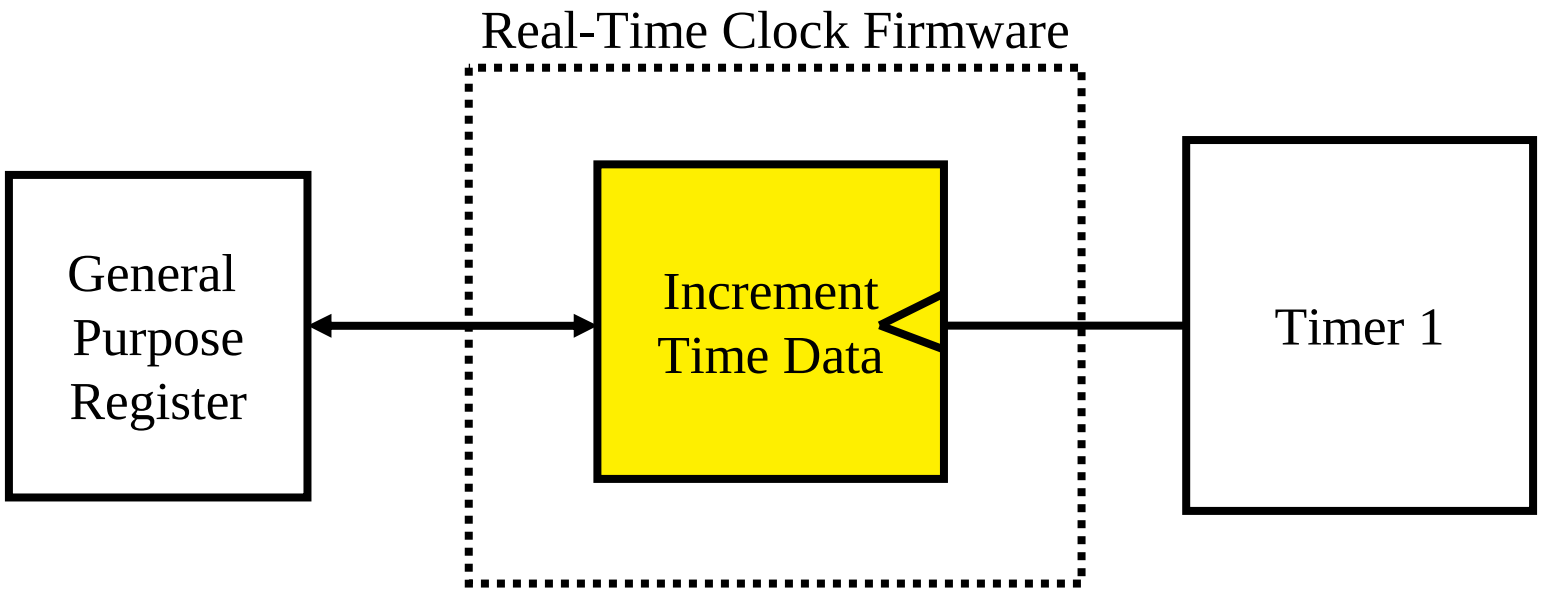
- Example: 1 second interrupts using external 32.768 kHz crystal

1 Second = (1 / 32768) * (2^{16} – Preloaded Value)

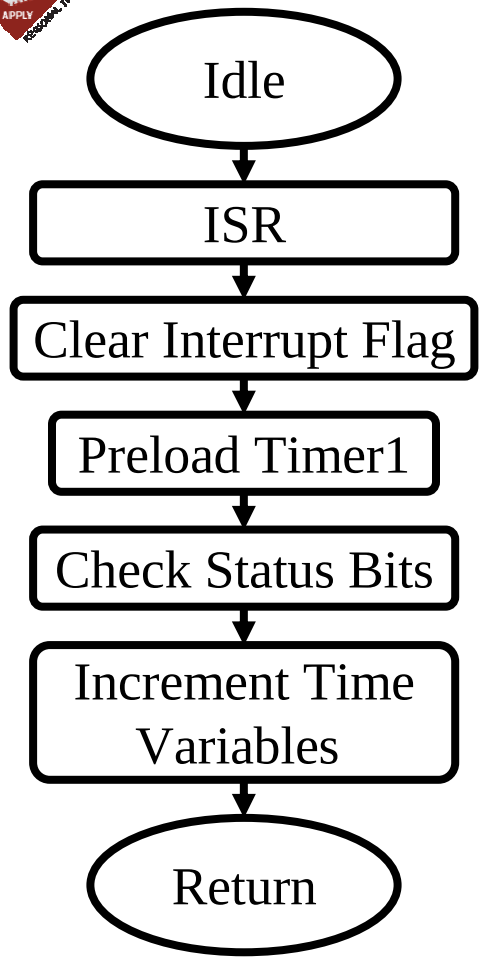
Preloaded Value = 32768 = 0x8000



Real-Time Clock Firmware

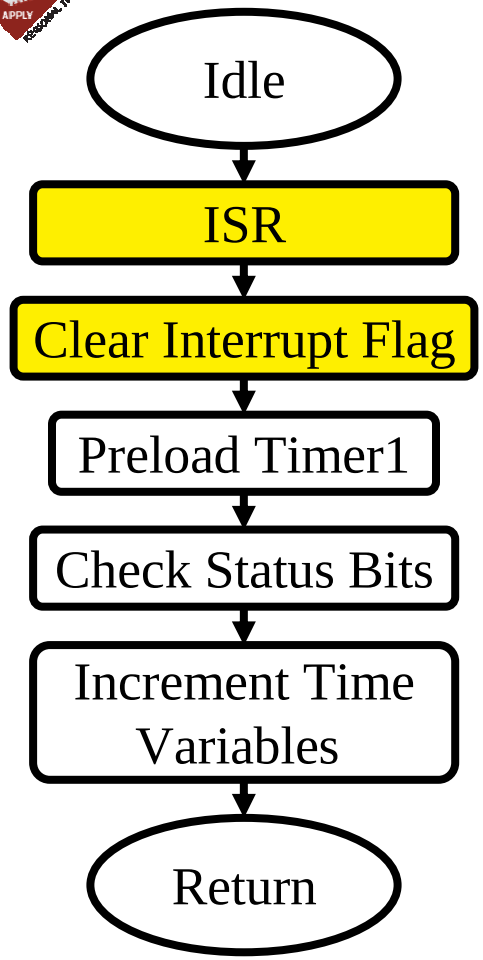


Real Time Clock Firmware

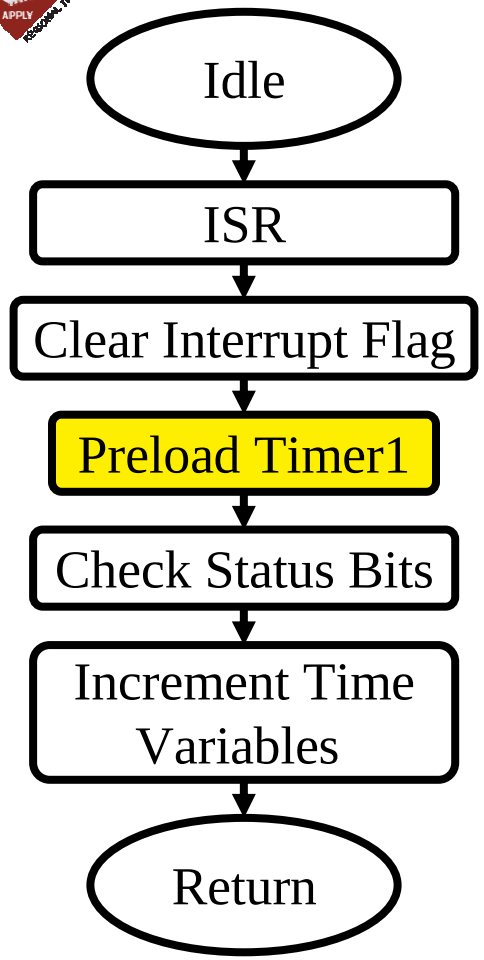


- Software Basics:
  - Located in ISR
  - Executes on Timer 1 interrupts each second

Real-Time Clock Firmware

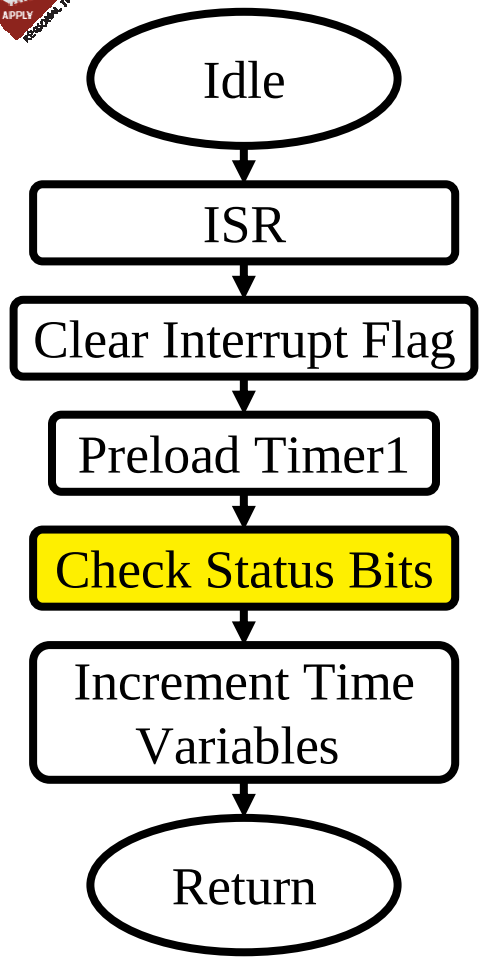


Real Time Clock Firmware



- Set Most Significant bit of TMR1H

Real Time Clock Firmware



Configure 1 (0x00)

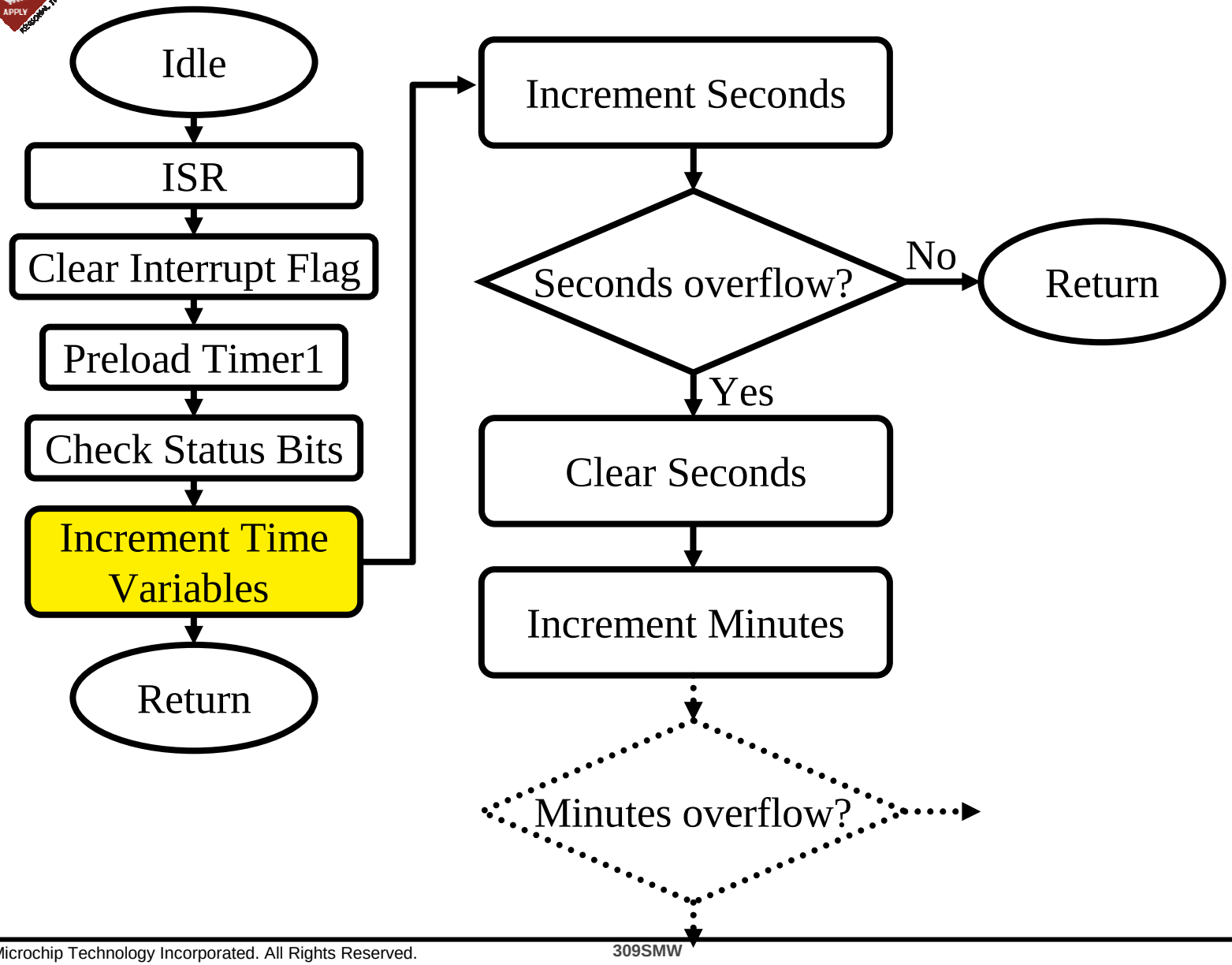
7	6	5	4	3	2	1	0
X	X	STOP	X	X	X	X	X

Configure 2 (0x01)

X	X	X	X	X	Alarm Flag	Alarm On	X
7	6	5	4	3	2	1	0

- Two Configuration Registers
  - RAM structure is being emulated from existing device
- Determine Modes of Operation
  - On/Off
  - Alarm On/Off

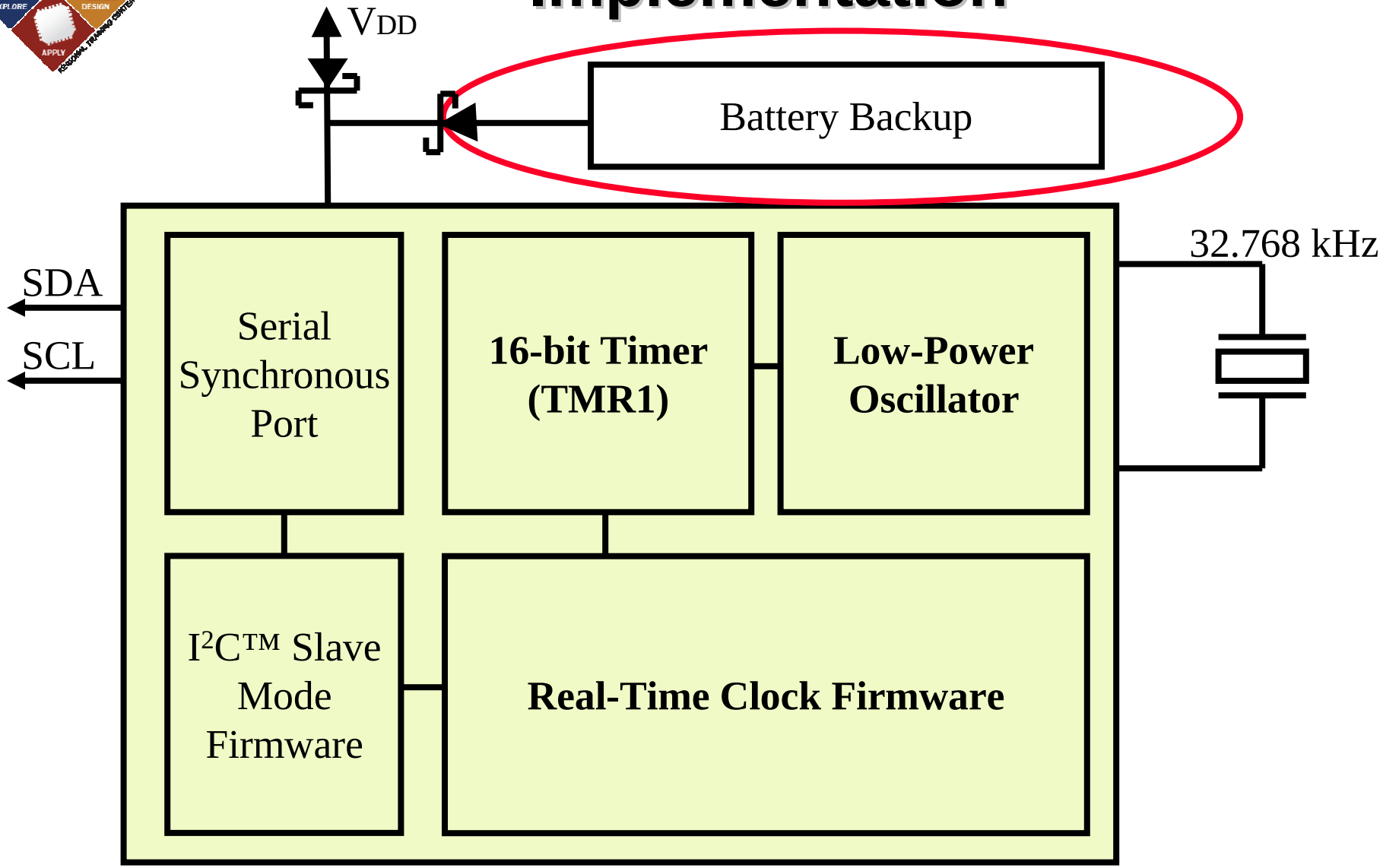
Real-Time Clock Firmware



Firmware Summary

- Firmware:
  1. Executes at a set time base
  2. Checks status bits
  3. Increments time variables
  4. Stores results to GPR
- Note:
  - Check month and year when incrementing days
    - Number of days in month differs
    - Leap years have an extra day in February

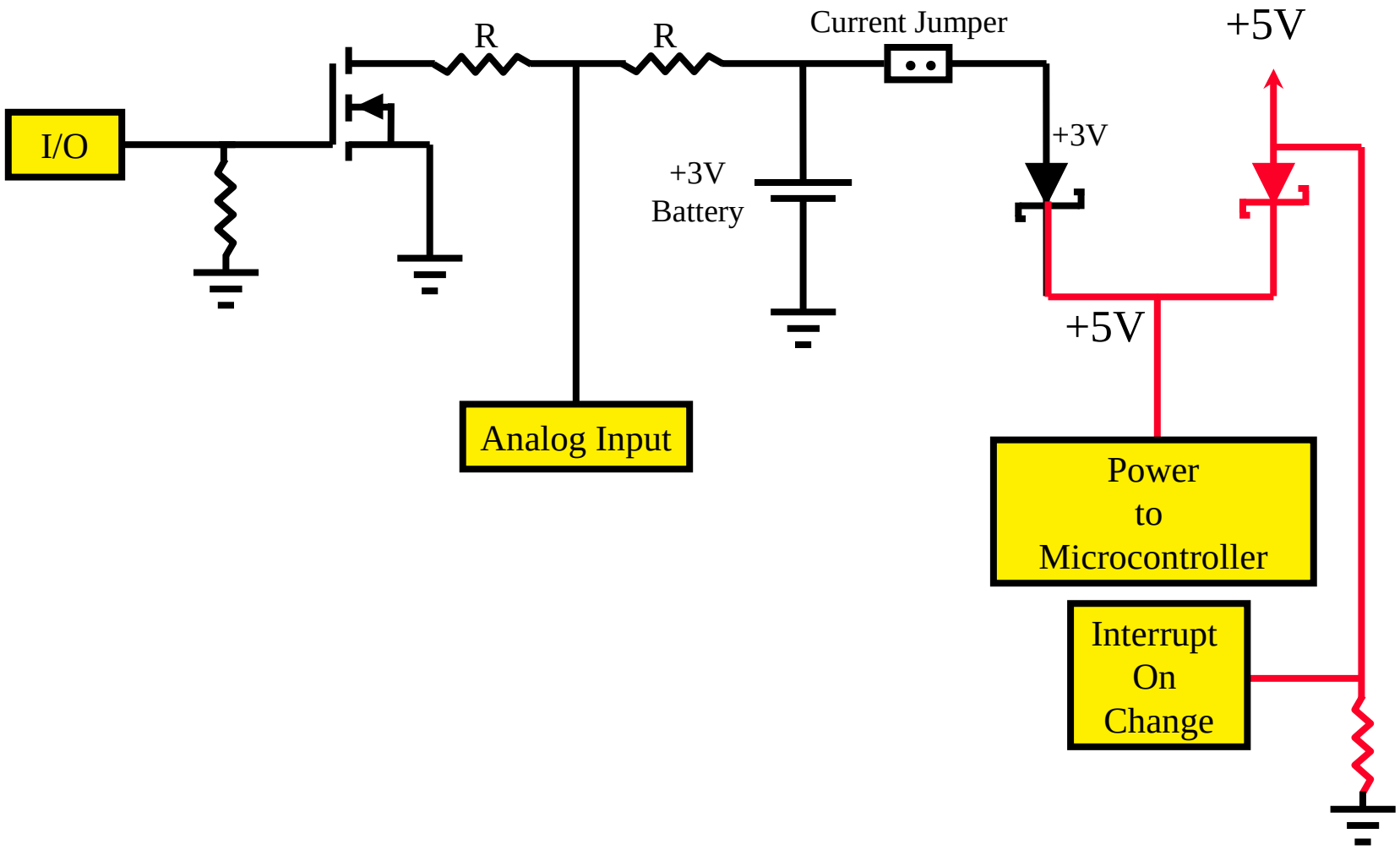
Real-Time Clock Microcontroller Implementation



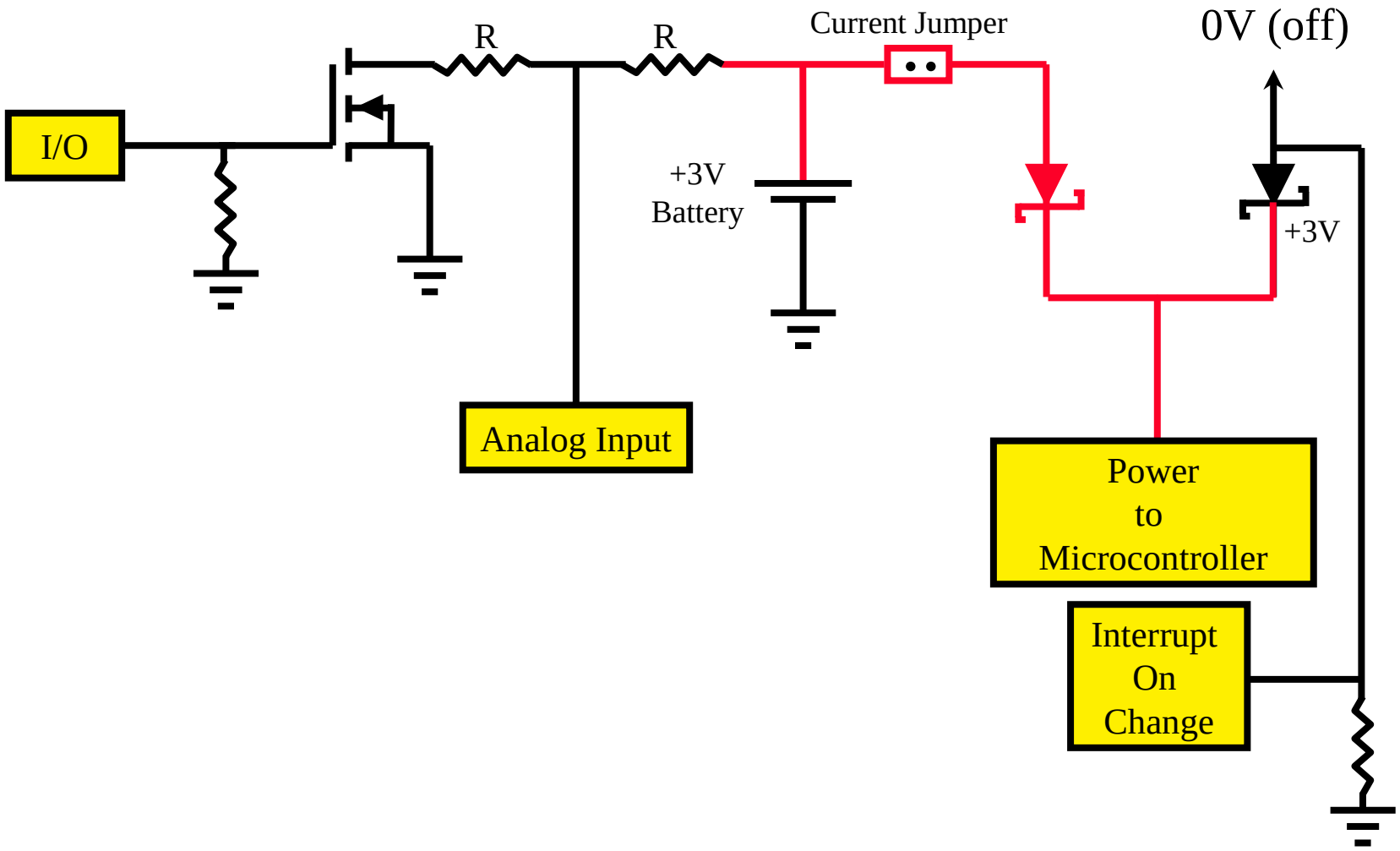
Battery Backup

- Automatically switches to battery power
- Power source can be determined
  - Interrupt on change
- Provides battery voltage measurement
  - Internal voltage reference

Battery Backup Circuit (Mains Power)

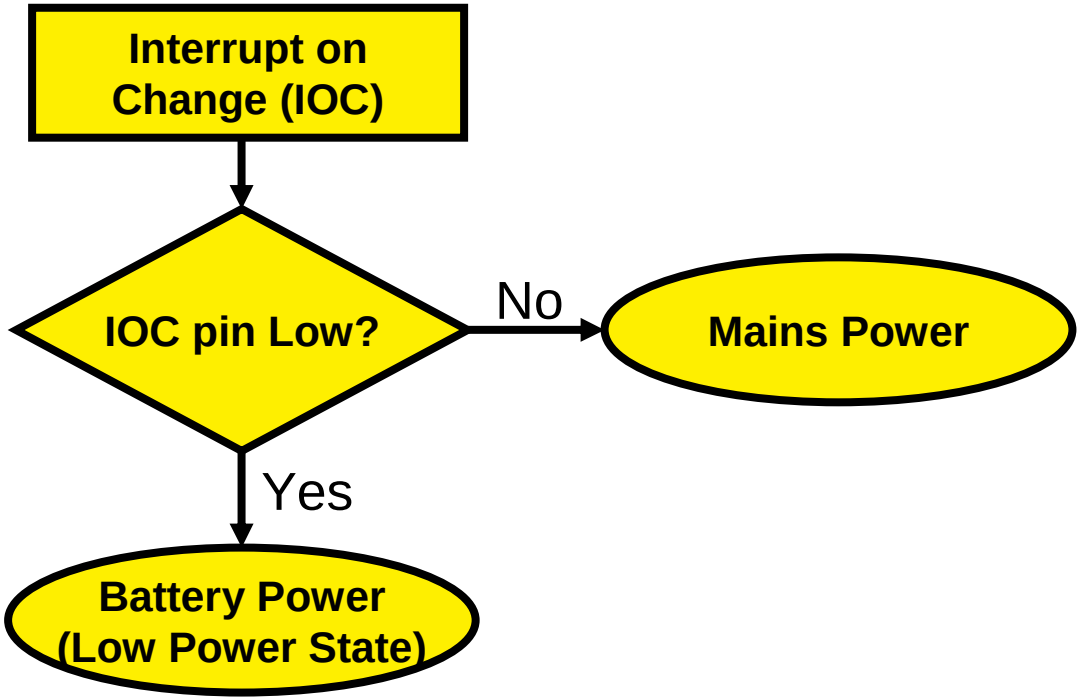


Battery Backup (Battery Power)



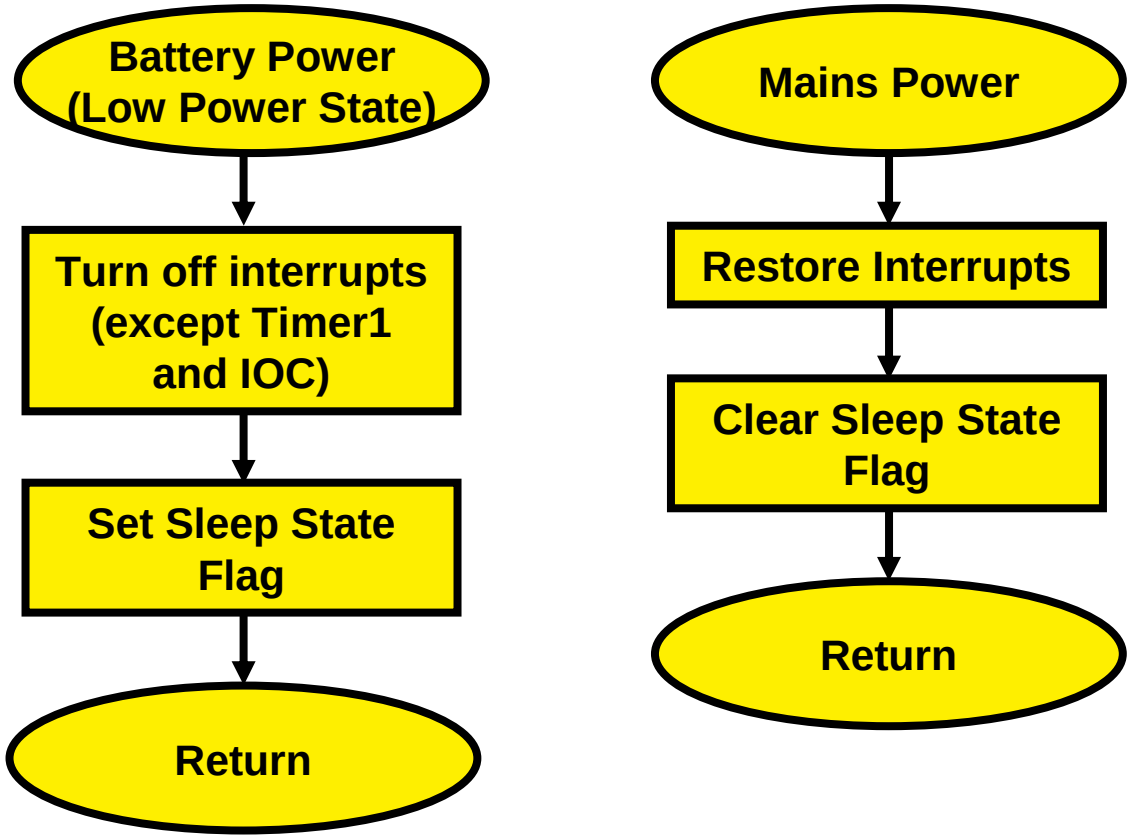
Battery Backup Firmware

- Determining power source

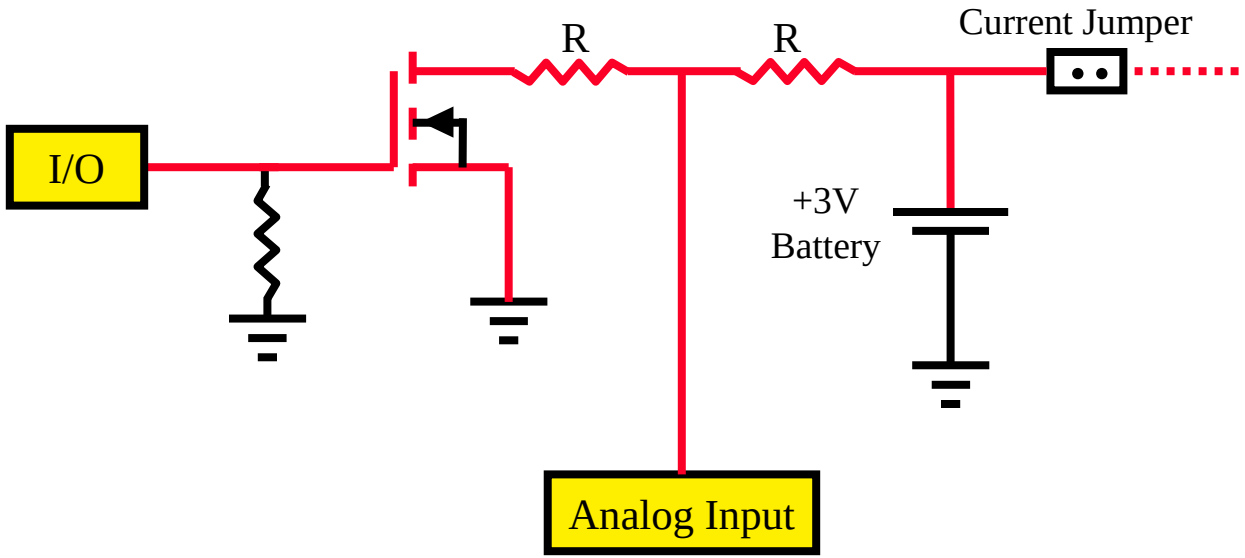


Battery Backup Modes

- Find in main.asm

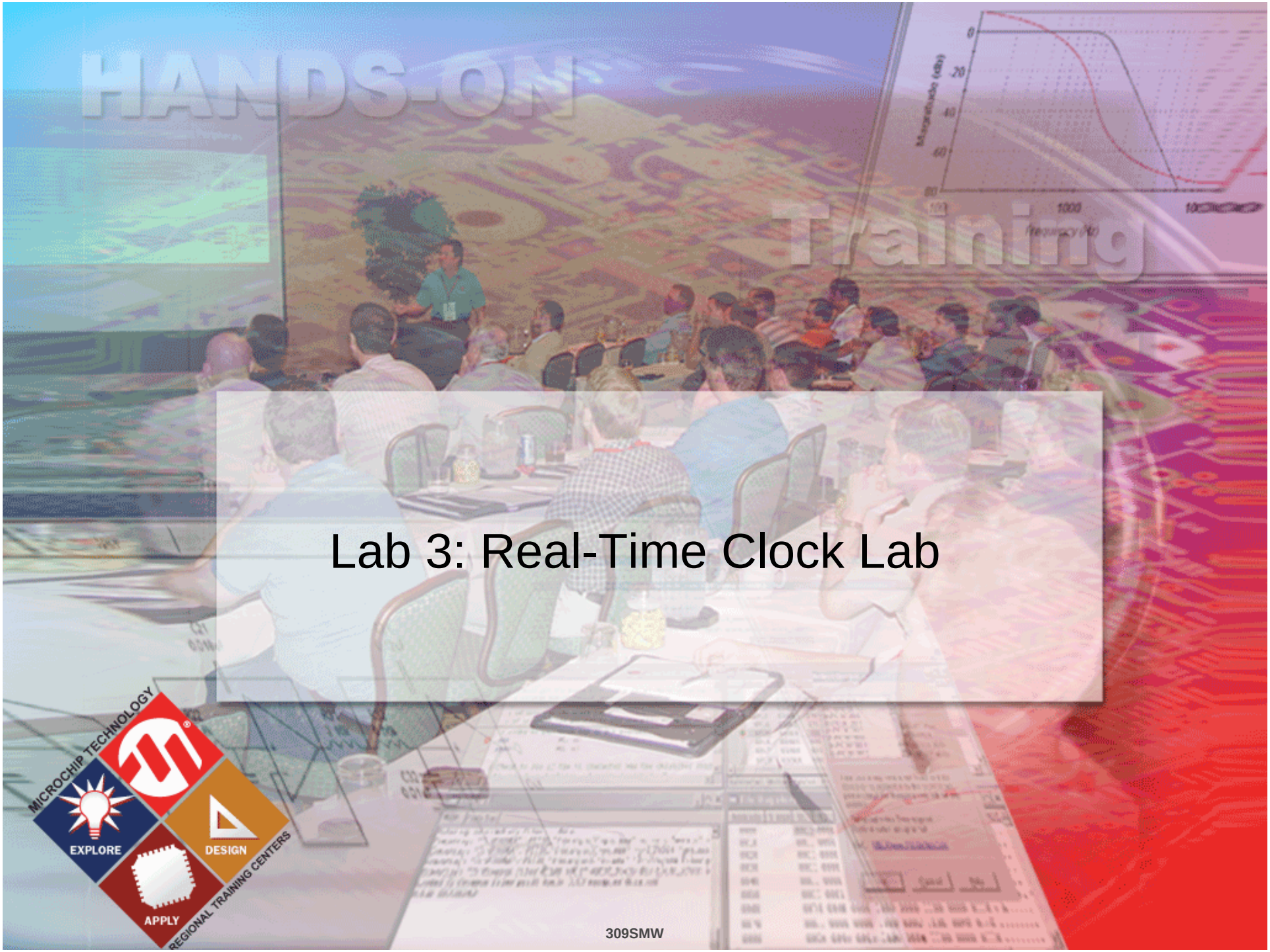


Battery Backup (Battery Measure)



Battery Backup Summary

- The battery backup circuit provides the following:
  - Instant battery power when mains power is off
  - Interrupt notification when power source changes
  - Ability to measure battery voltage


HANDS-ON
Training
Lab 3: Real-Time Clock Lab
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

Summary

- Introduction to the Real-Time Clock
- Emulating I²C™ Real Time Clock
- Battery Backup
- Lab Demonstration

- Questions?


HANDS-ON
Training
Thermal Management Module
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

Module Learning Objectives

- At the end of this class you should be able to:
  - Explain 3-wire fan control
  - Explain how thermal controllers work
  - Implement an I²C™ Thermal Controller using a PIC16F886 Microcontroller

- Prerequisites
  - I²C Module
  - Analog-to-Digital Converter
  - Compare/Capture/PWM
  - Timer 1

Module Agenda

In the next hour we will discuss…

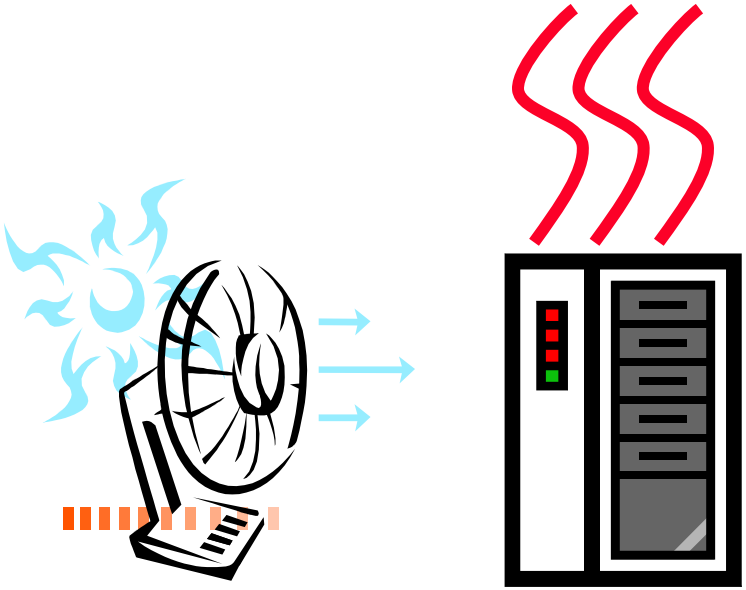
- Thermal Management Basics
- 3 Wire Fans
- Thermal Controllers
- Implementing a Thermal Controller on a PIC16F886
- Lab Exercise

Thermal Management

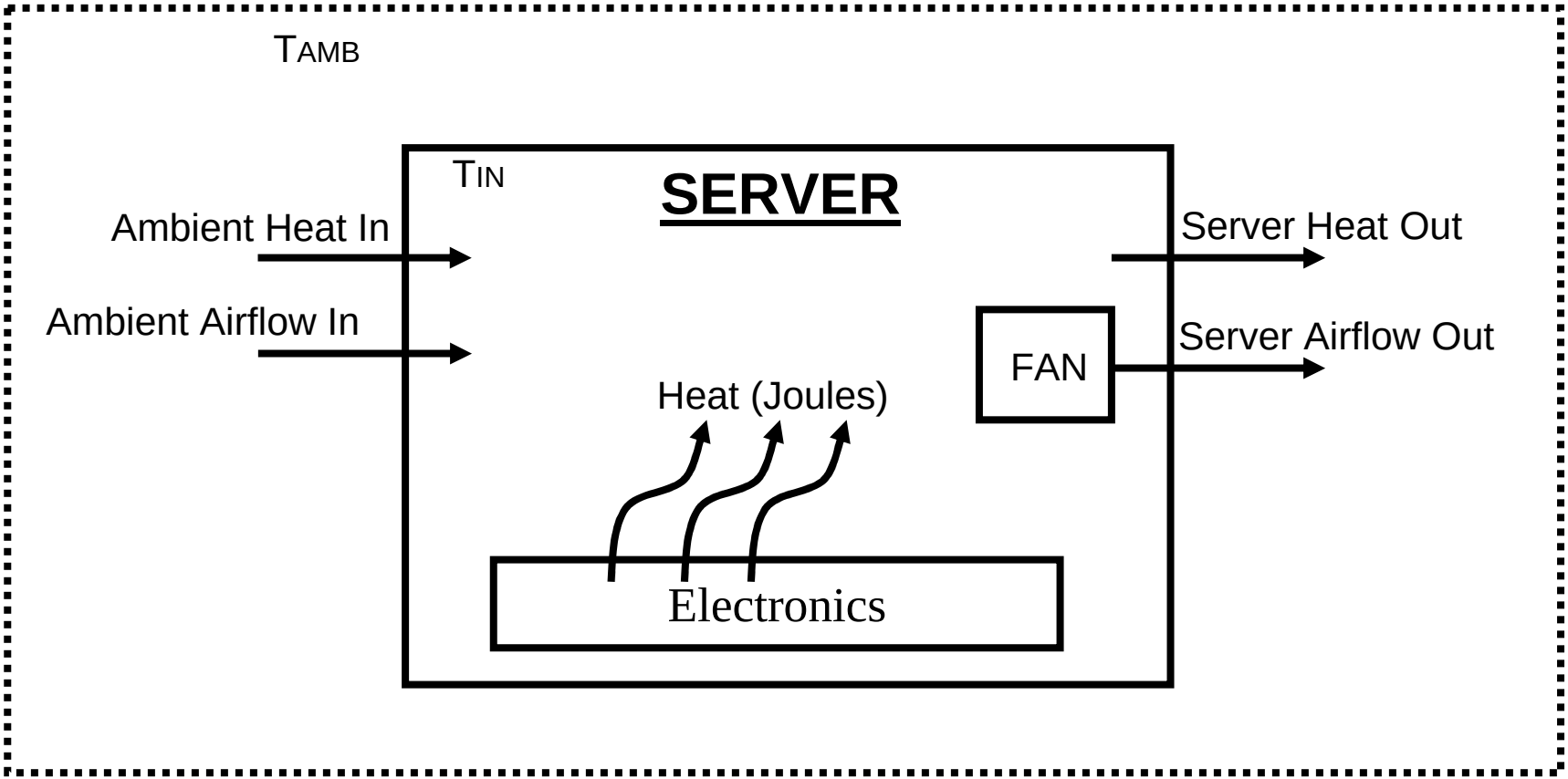
- What is Thermal Management?
  - Monitoring fan status
  - Monitoring system temperature
  - Optimizing the control of temperature through monitoring

Thermal Management

- Why the need for Thermal Management?
  - More transistors running at higher speeds = Heat
  - Reduces system noise
  - Monitor failures
  - Lower power consumption



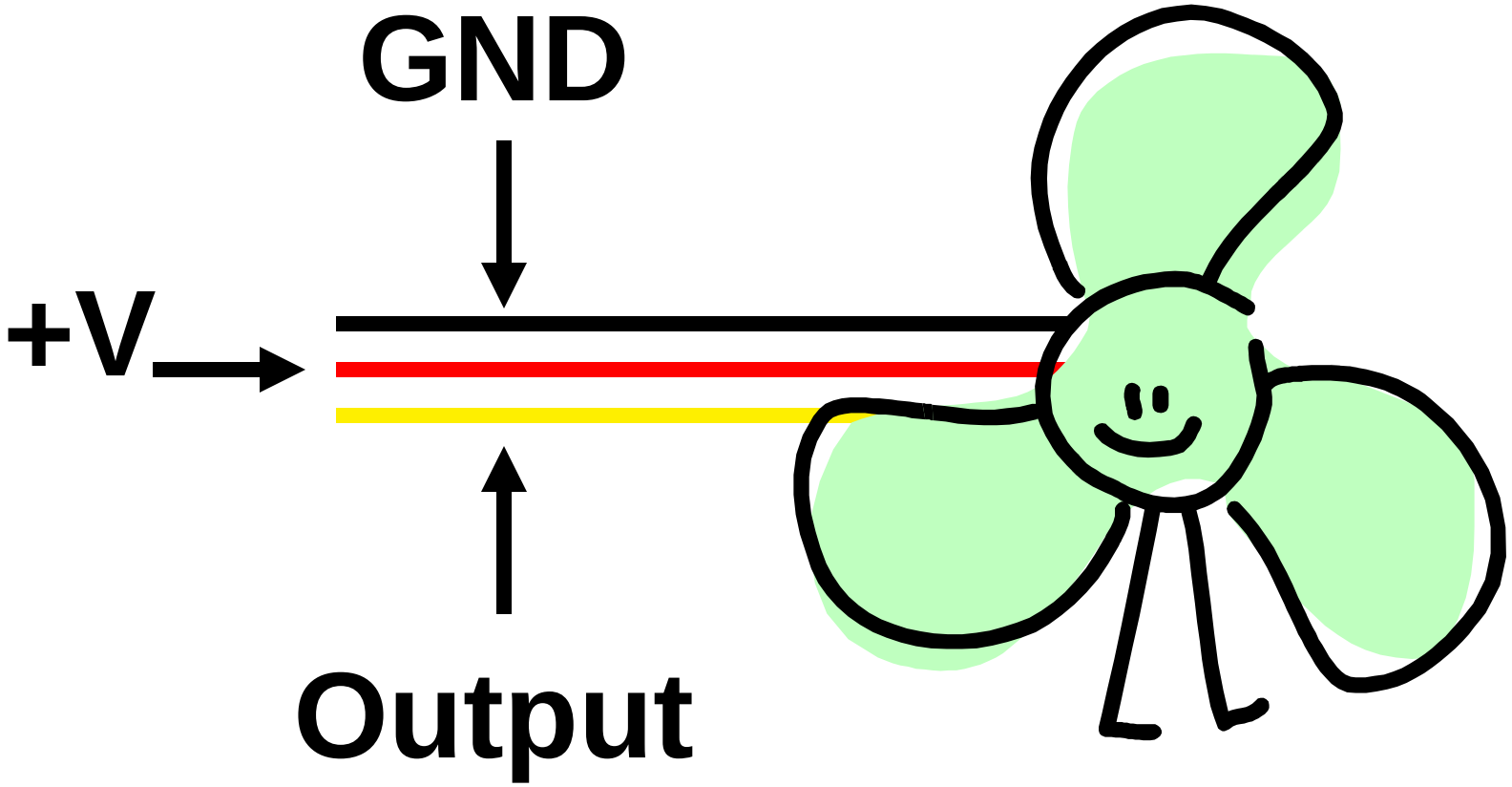
Typical Thermal System




HANDS-ON
Training
EXPLORE
DESIGN
APPLY
MICROCHIP TECHNOLOGY
REGIONAL TRAINING CENTERS

3-Wire Fans

3-Wire Fans



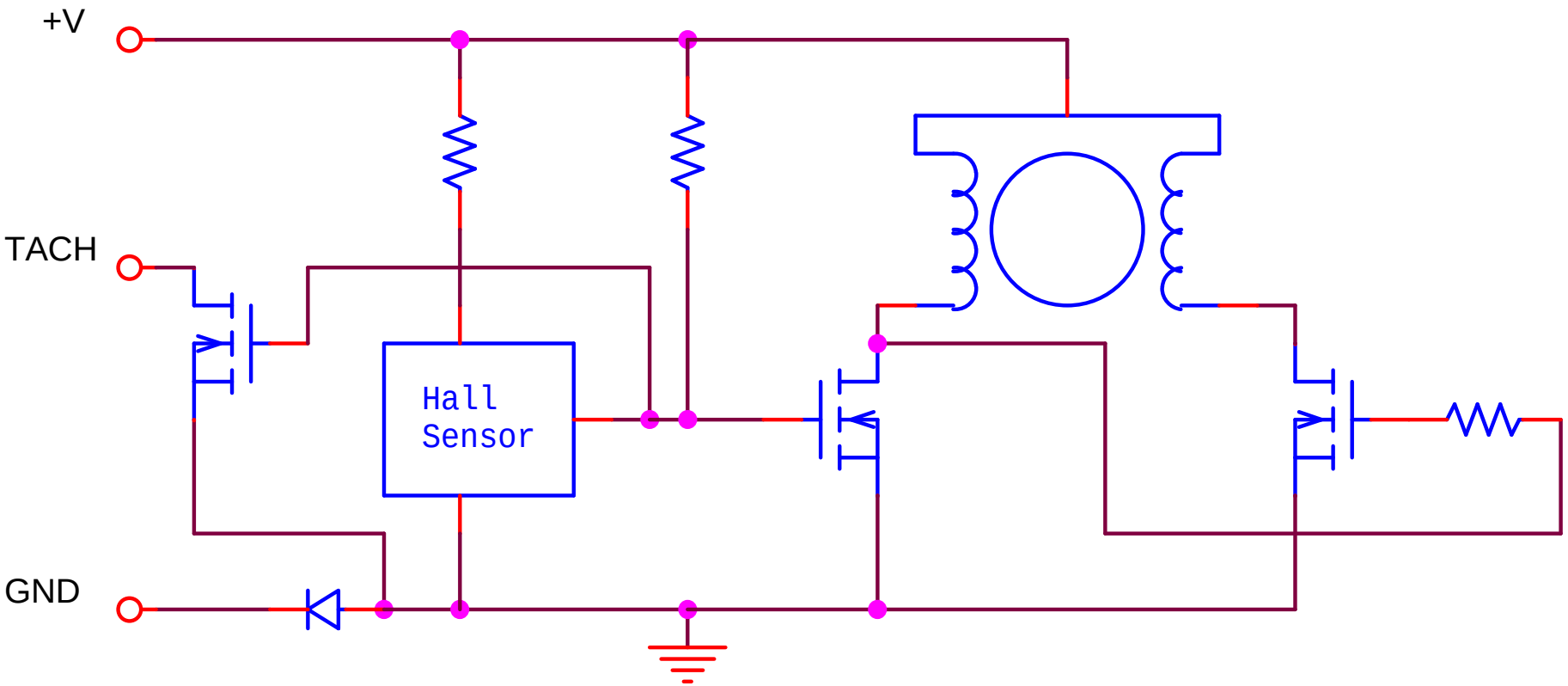
The Third Wire

- Tachometer (most common)
  - 1, 2, or 4 pulses per revolution
  - Open collector output

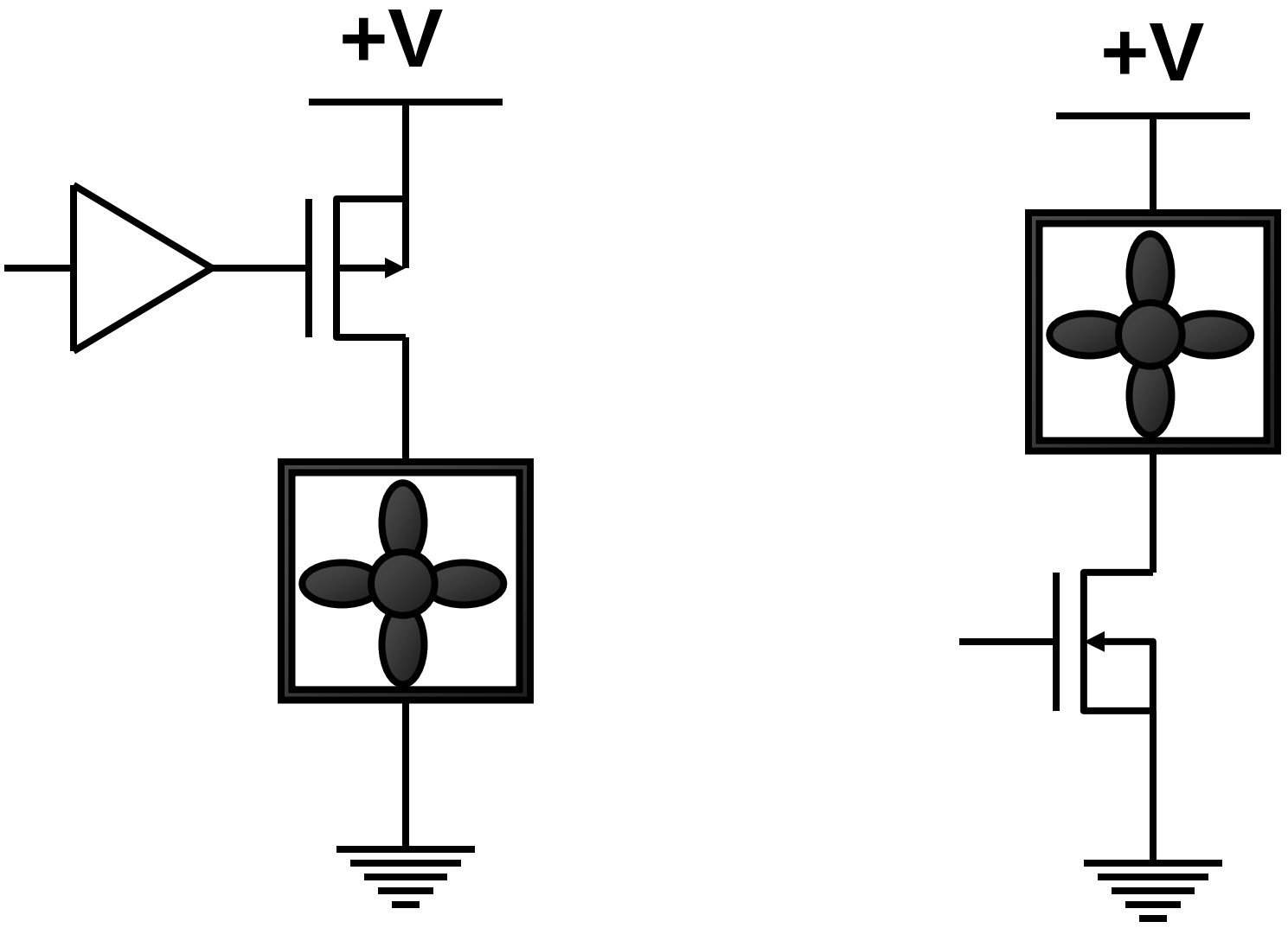
- Alarm output
  - Signals when the fan has failed

3-Wire Fan Schematic

- A look inside



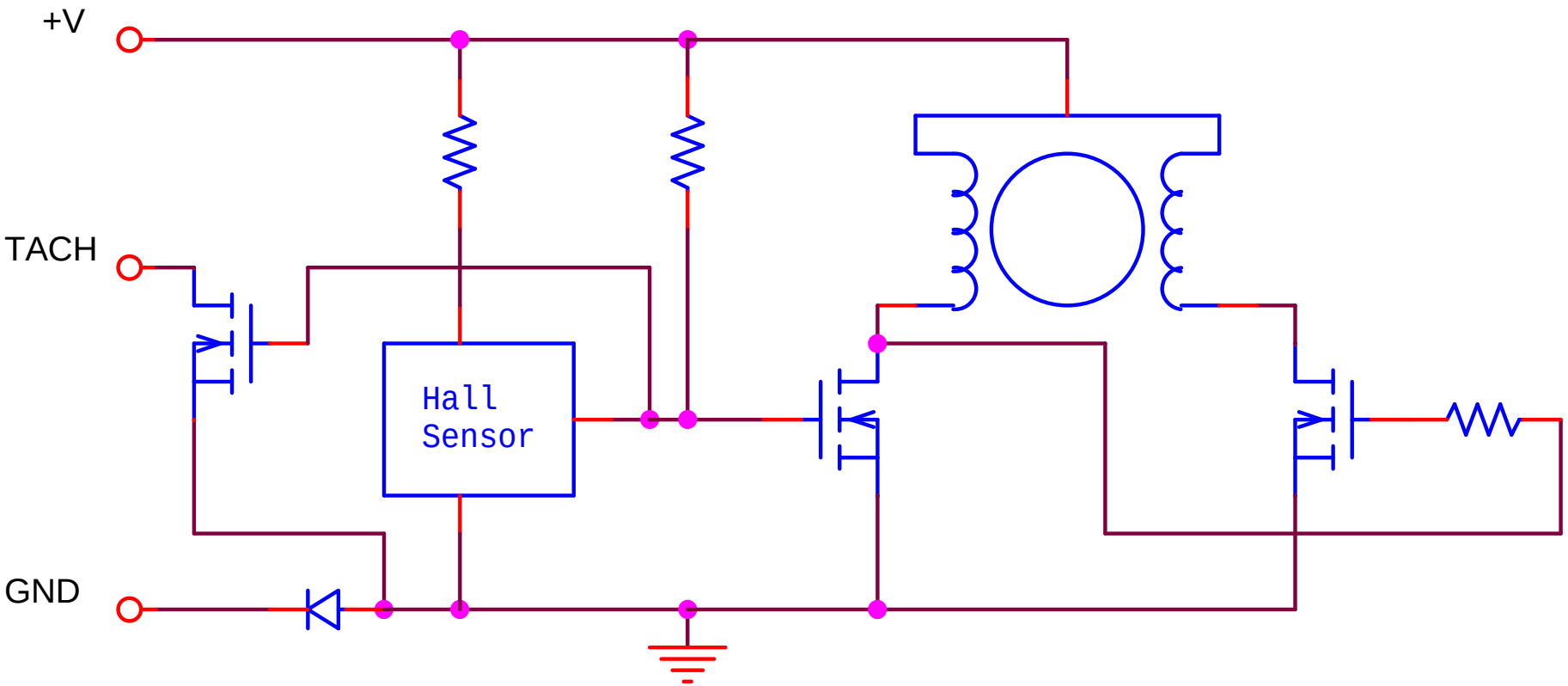
Low Frequency PWM



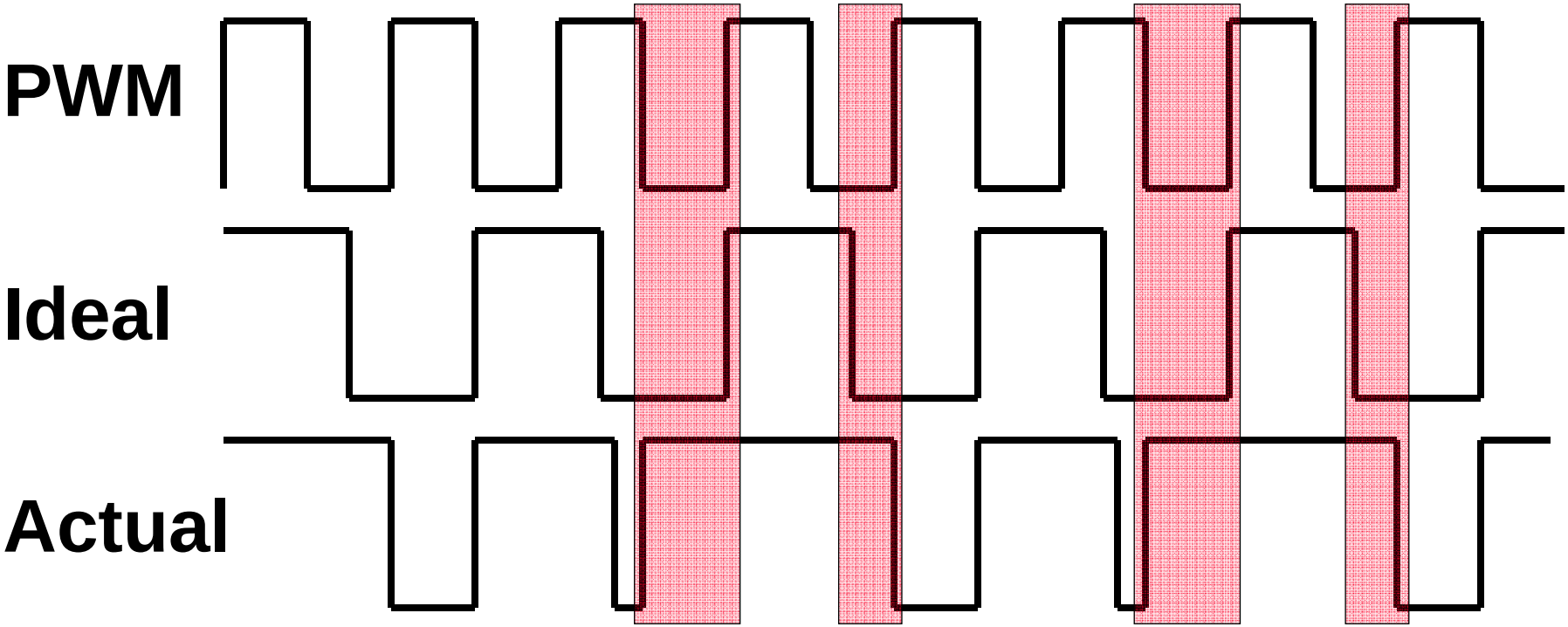
Low Frequency PWM

- Ability to extend operating range down to about 10% of max RPM
- Check with manufacturer if you can PWM the 3-Wire fan
- Tachometer output is no longer valid

3-Wire Schematic

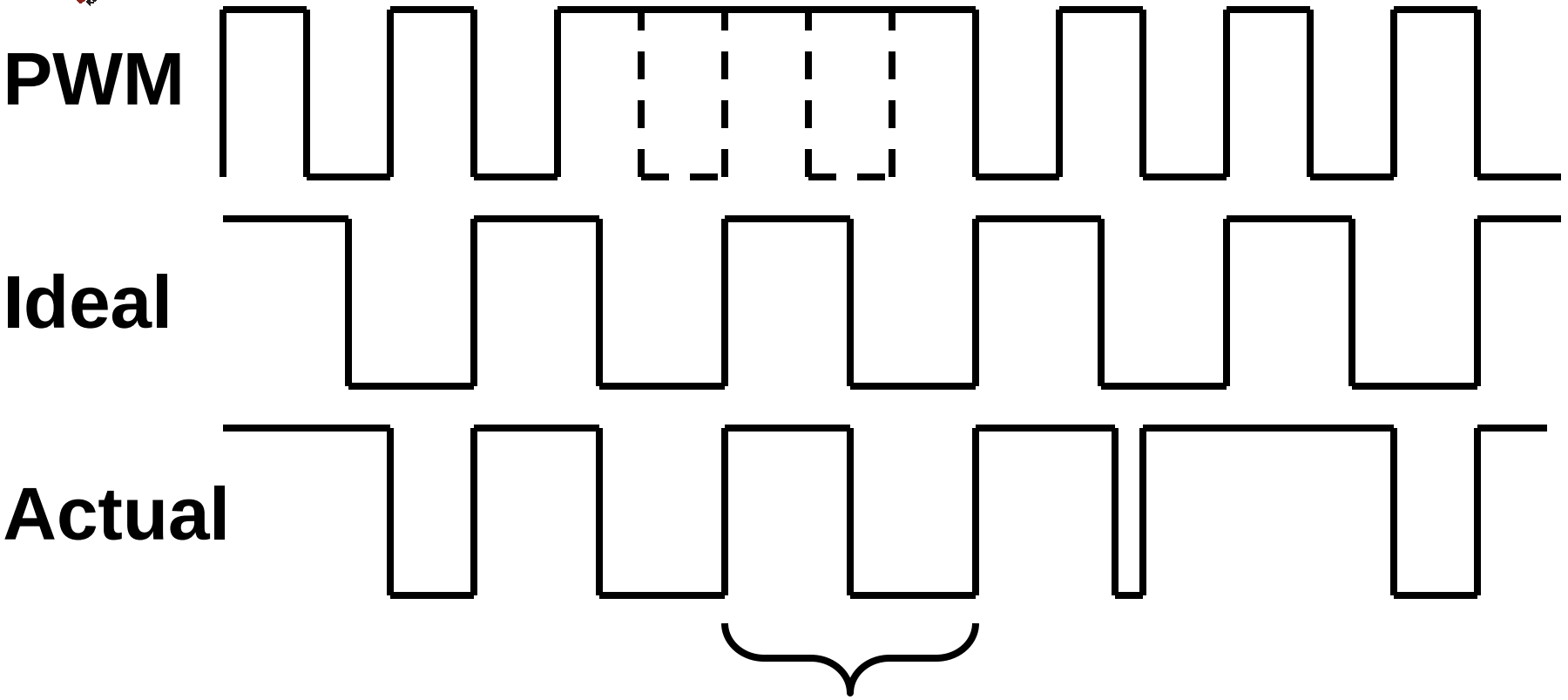


Tachometer Output



- Tachometer output is invalid

Creating a Valid Output



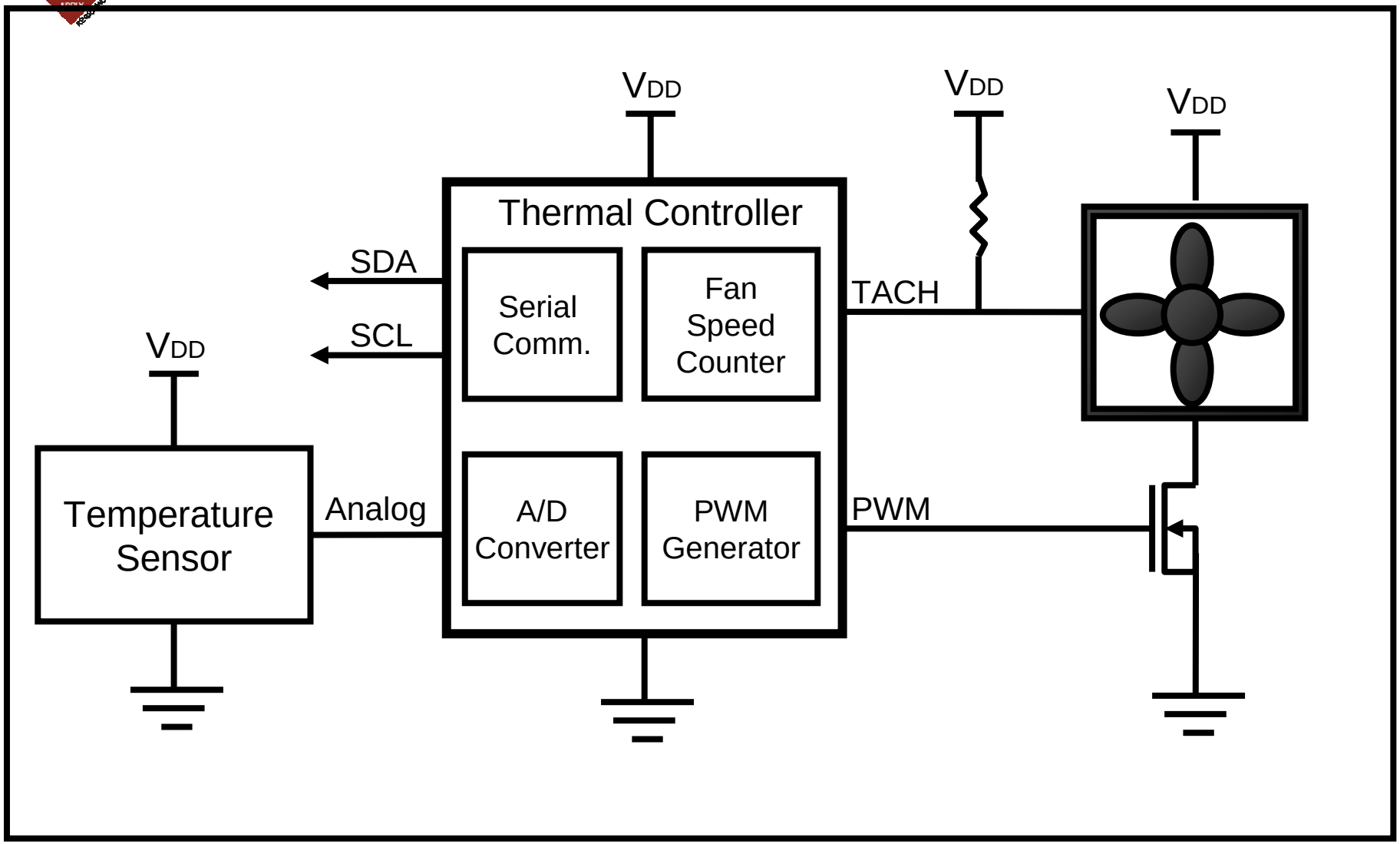
- Pulse stretching can be used to obtain a complete period

Three Wire Fans

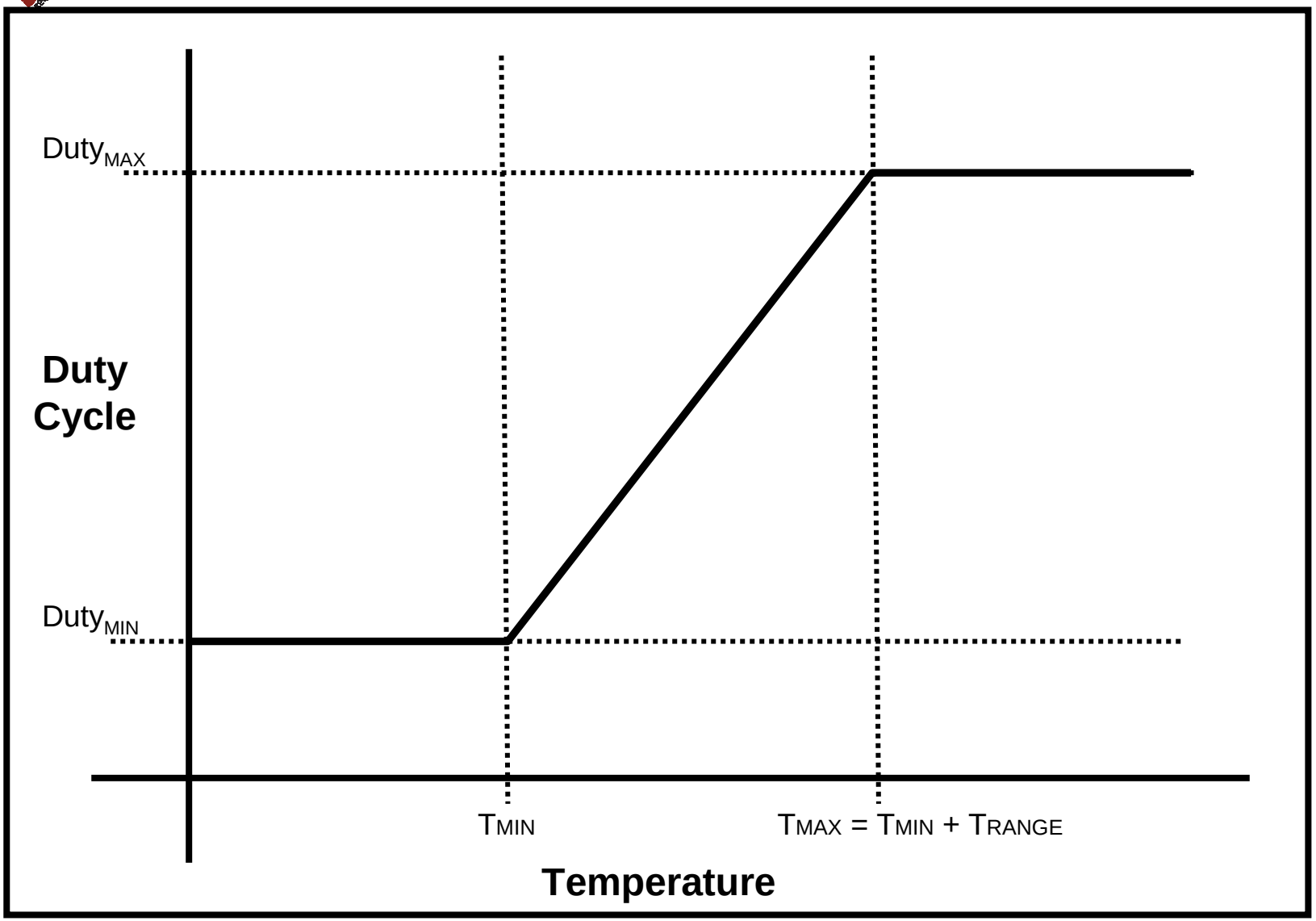
Questions?


HANDS-ON
Training
Thermal Management Controllers
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

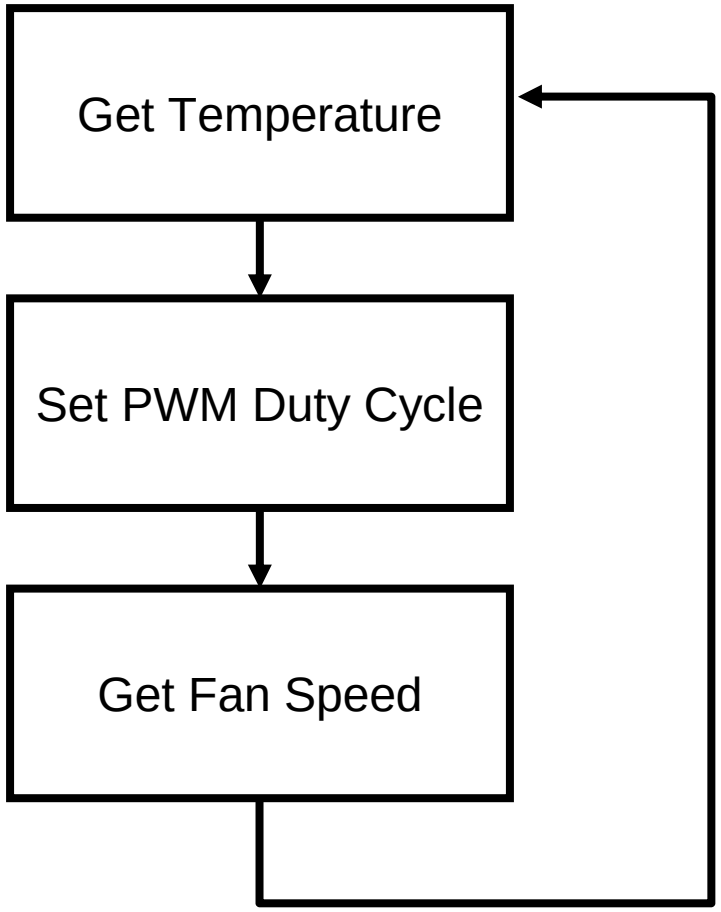
Example Thermal Controller Functions



Fan Temperature Control



Thermal Controller Flowchart



Thermal Controller Configuration

- **Serial Communications**
- Control Registers:
  - Temperatures
  - Duty Cycle
  - PWM Frequency
- Status Registers:
  - Fan Speed
  - Temperature

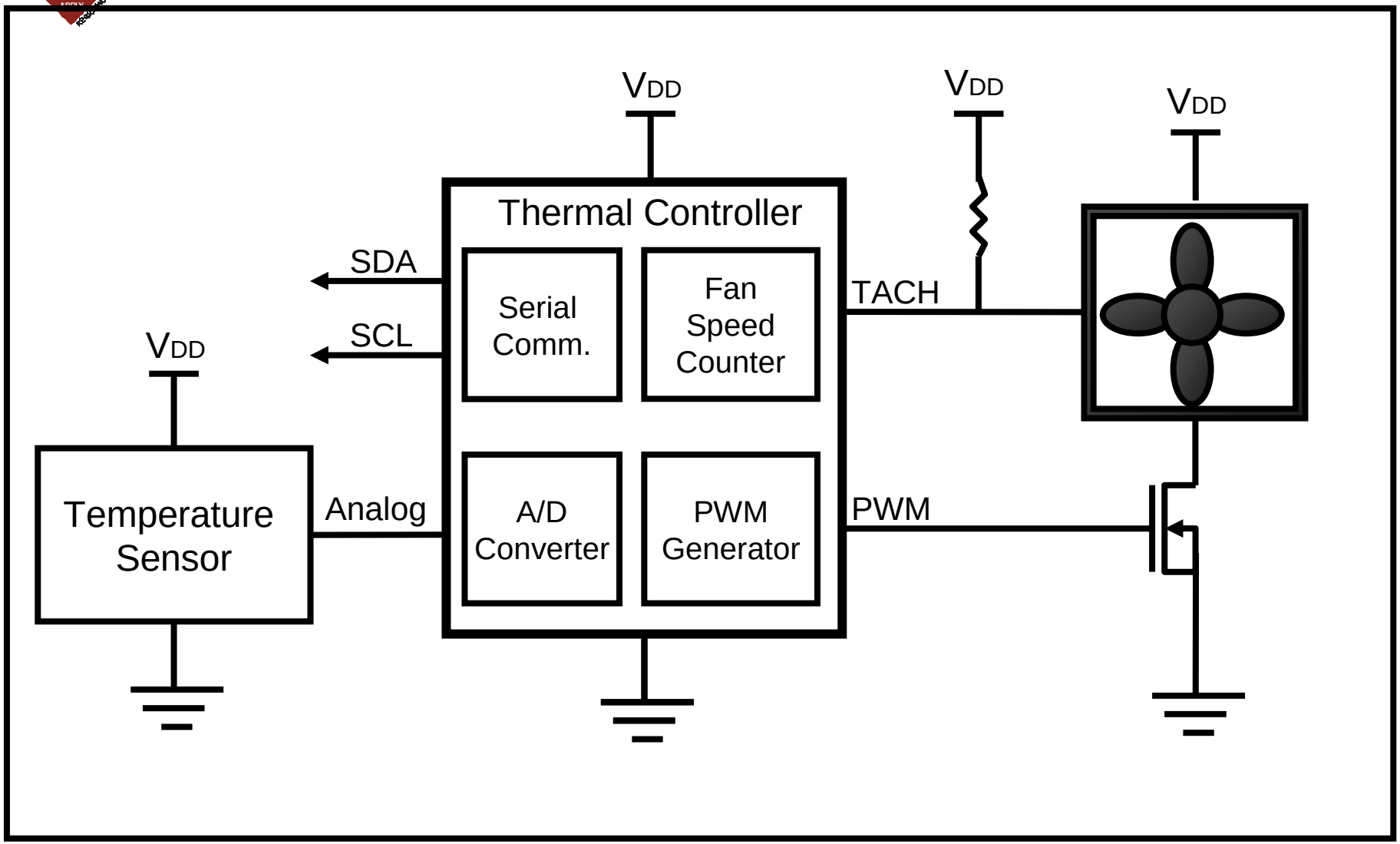
Fan Controller Registers

Configuration
Status Register
PWM Frequency
Local Temperature Value
Local Temperature High
Local Temperature Low
Duty Cycle Maximum
Duty Cycle Minimum
Fan Speed Reading

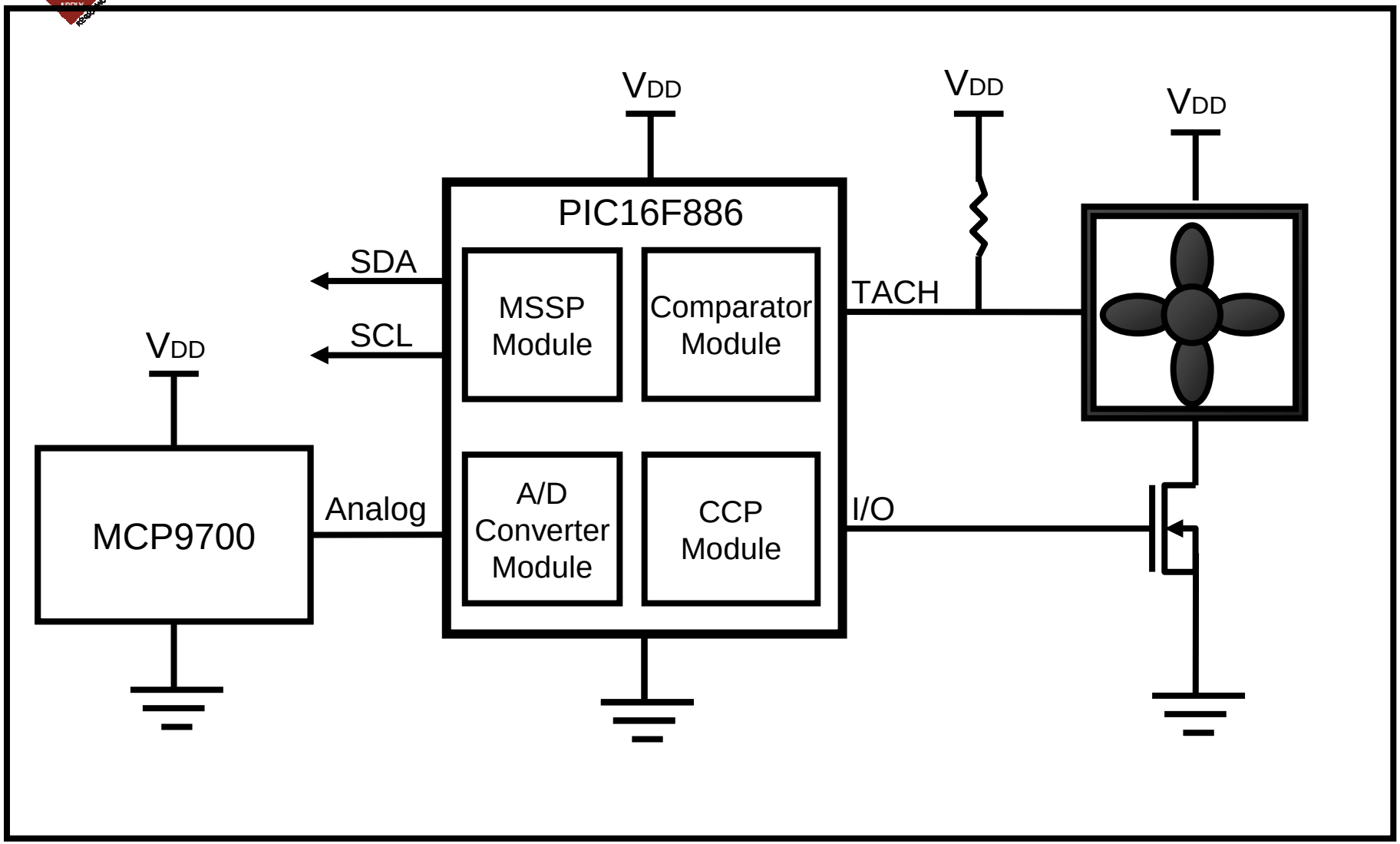
Implementing a Thermal Controller on a PIC16F886

309SMW

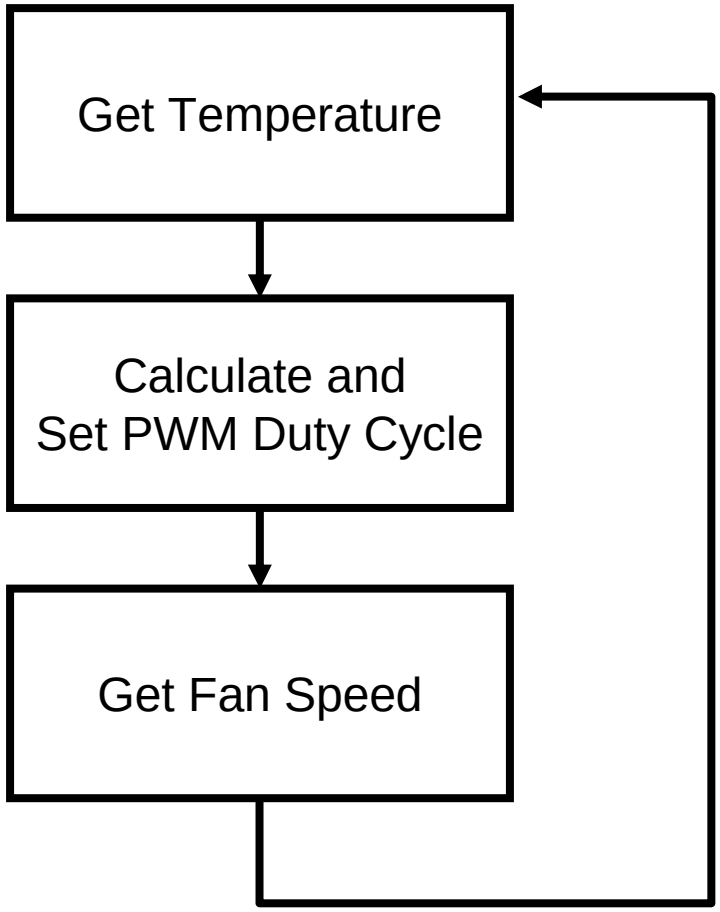
Thermal Controller Implementation



Thermal Controller Implementation



PIC® MCU Thermal Controller Flowchart



Temperature Measurement

- **MCP9700 Temperature Sensor**
- Analog Temperature Sensor

$$V_{OUT} = T_C * T_A + V_{0degC}$$

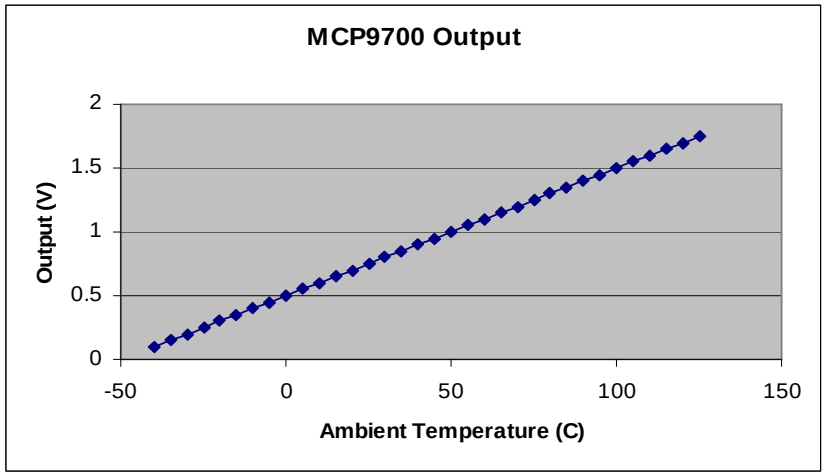
Where:

T_A = Ambient Temperature

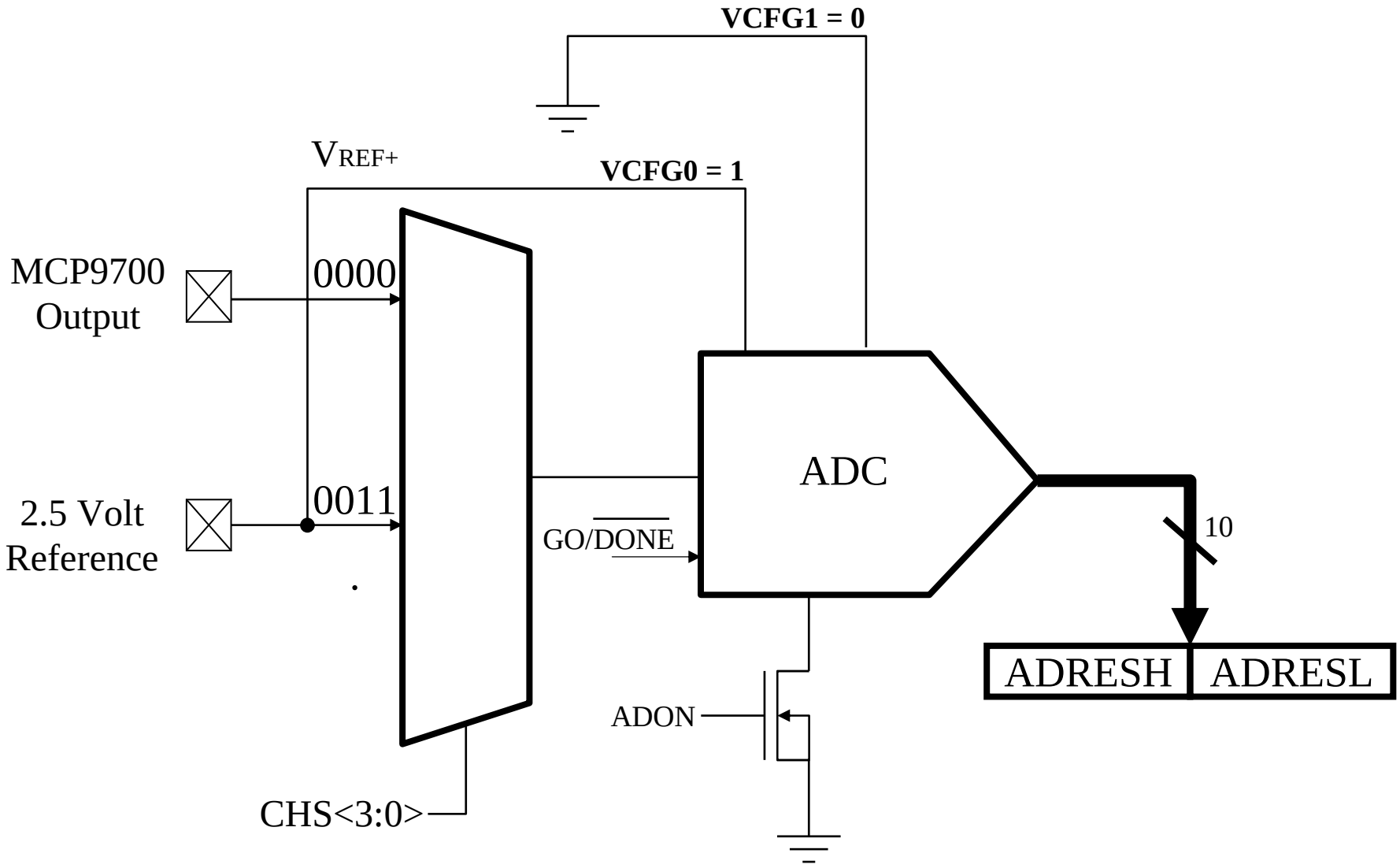
V_{OUT} = Sensor Output Voltage

V_{0degC} = Sensor Output Voltage at 0 degC = 500 mV

T_C = Temperature Coefficient = 10 mV/degC

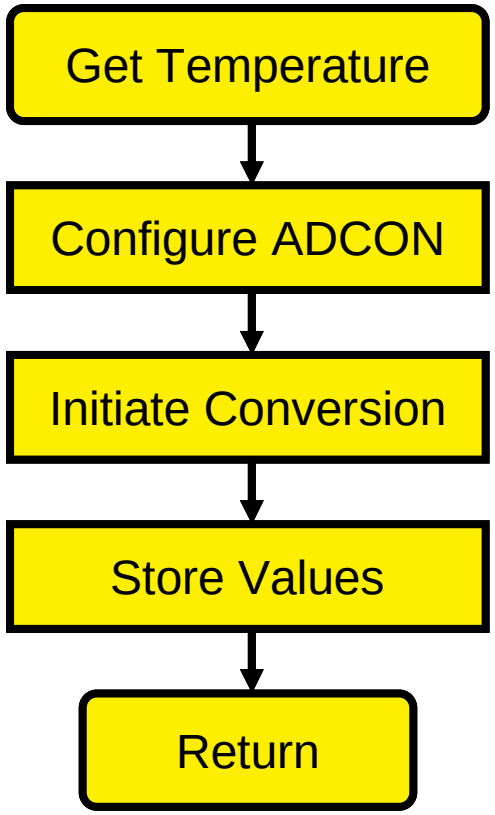


Configuring Analog-to-Digital Converter

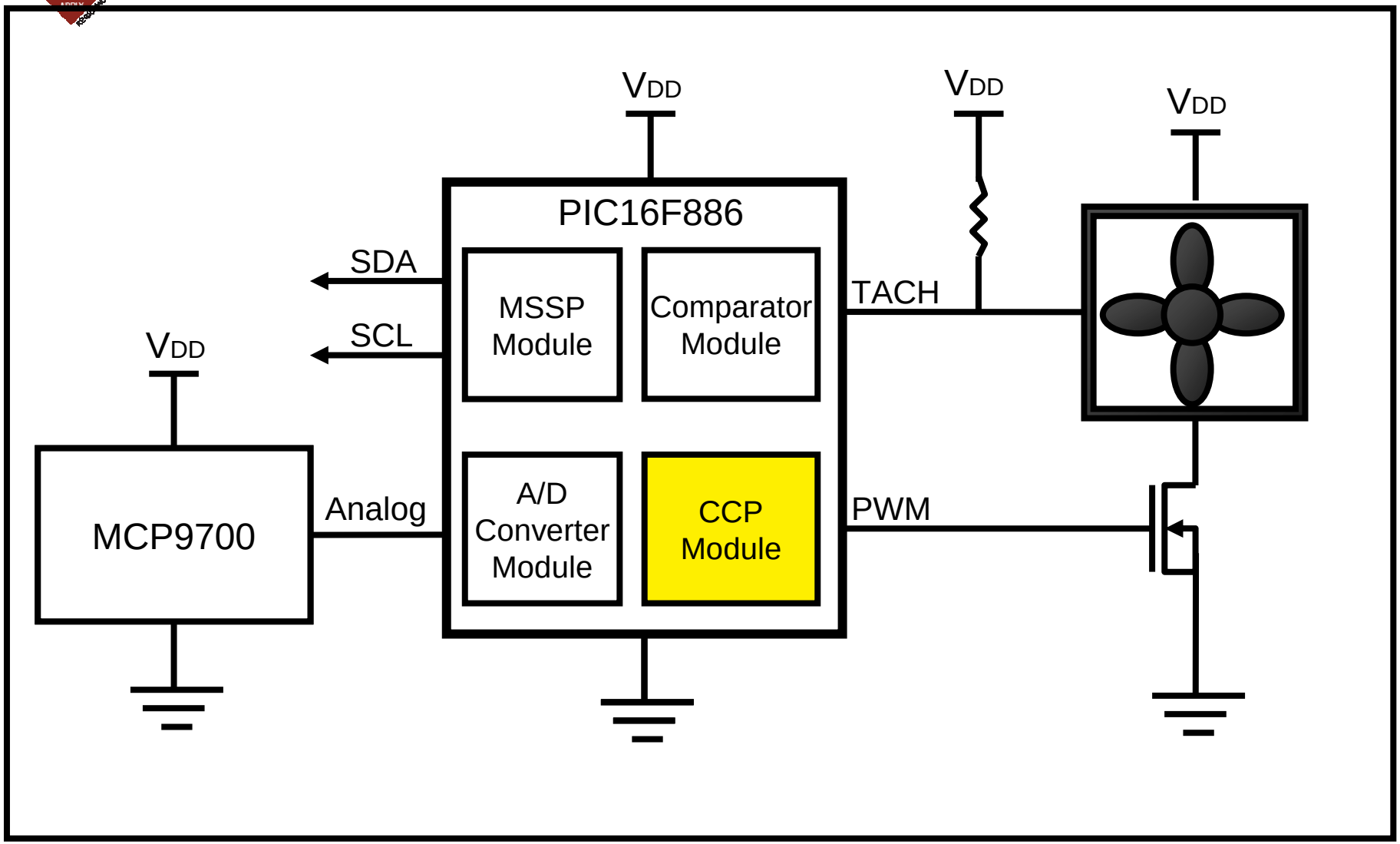


Temperature Measurement

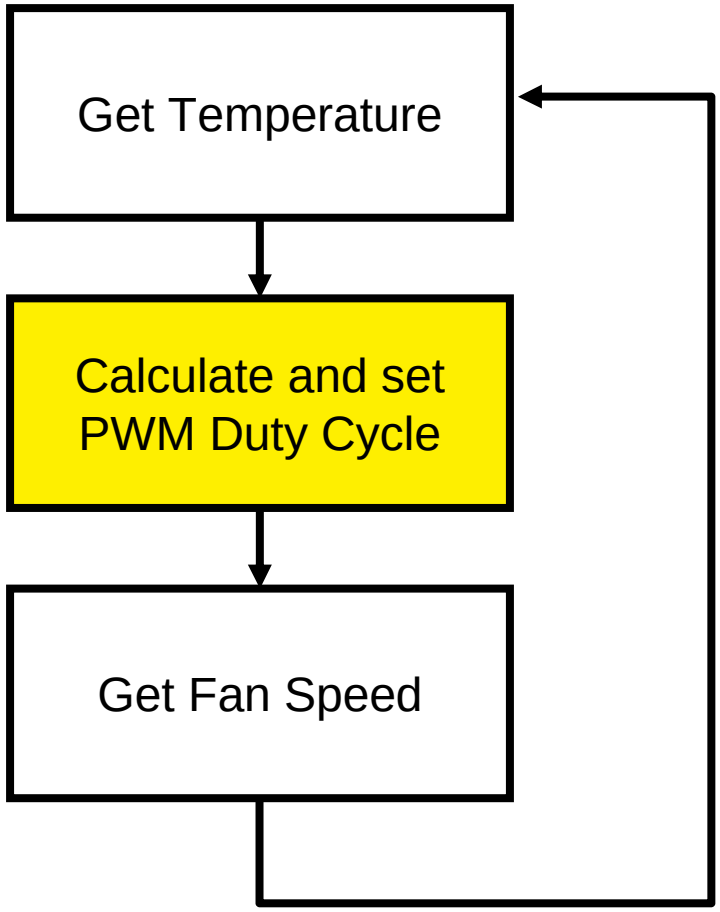
- 10-Bit Analog-to-Digital Conversion
- Store to 2 Byte Temperature Register



Thermal Controller Implementation



PIC® MCU Thermal Controller Flowchart

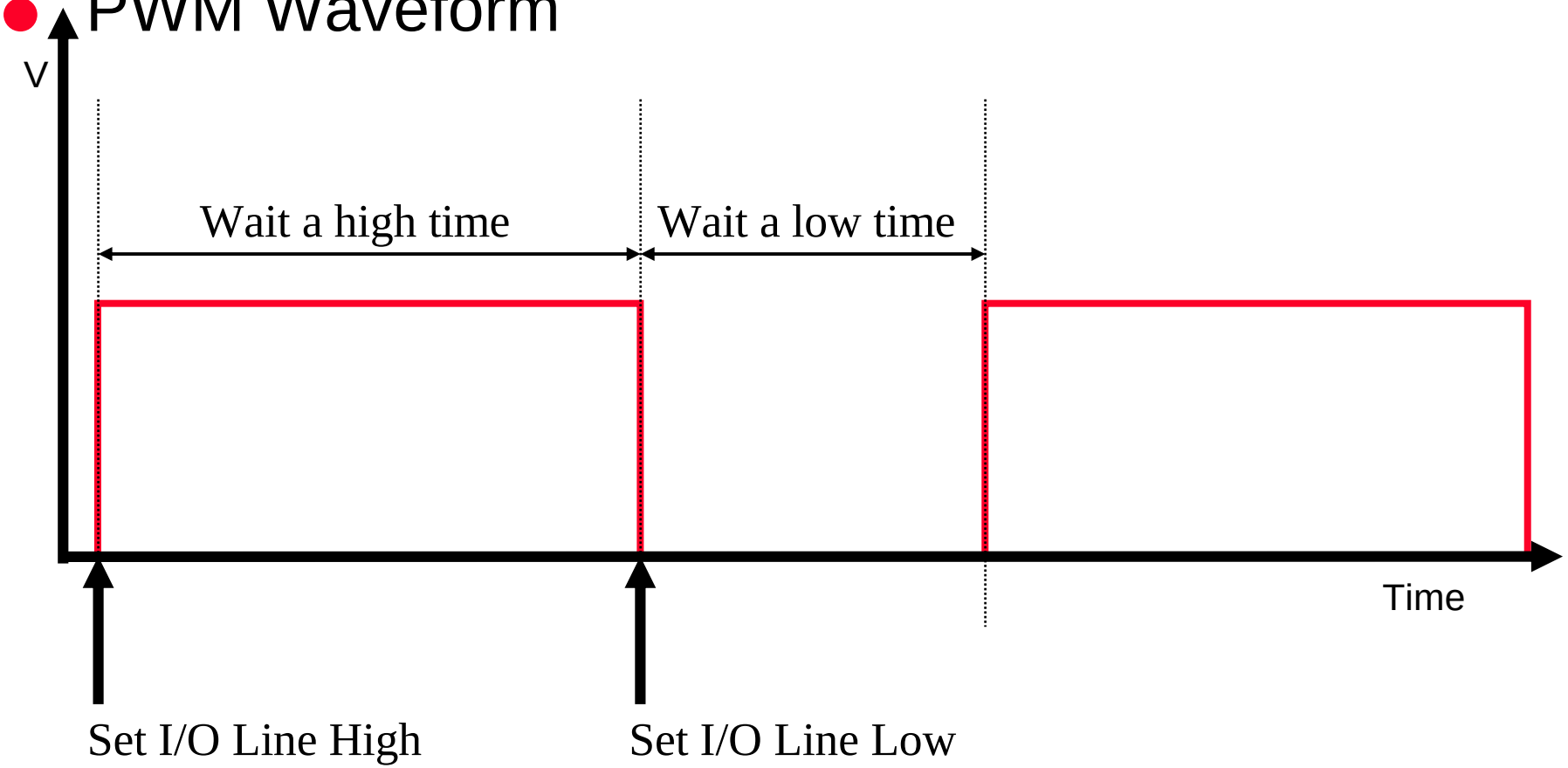


Generating a PWM Signal

- Problem: Low-speed 10-100 Hz signal to control fan speed
- PWM module cannot generate a 10 Hz signal
- Solution: Software PWM

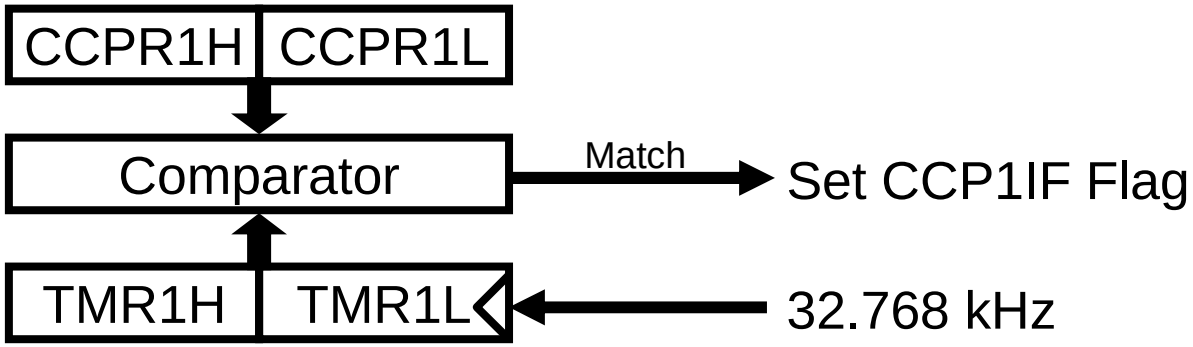
Generating a Software PWM

- PWM Waveform

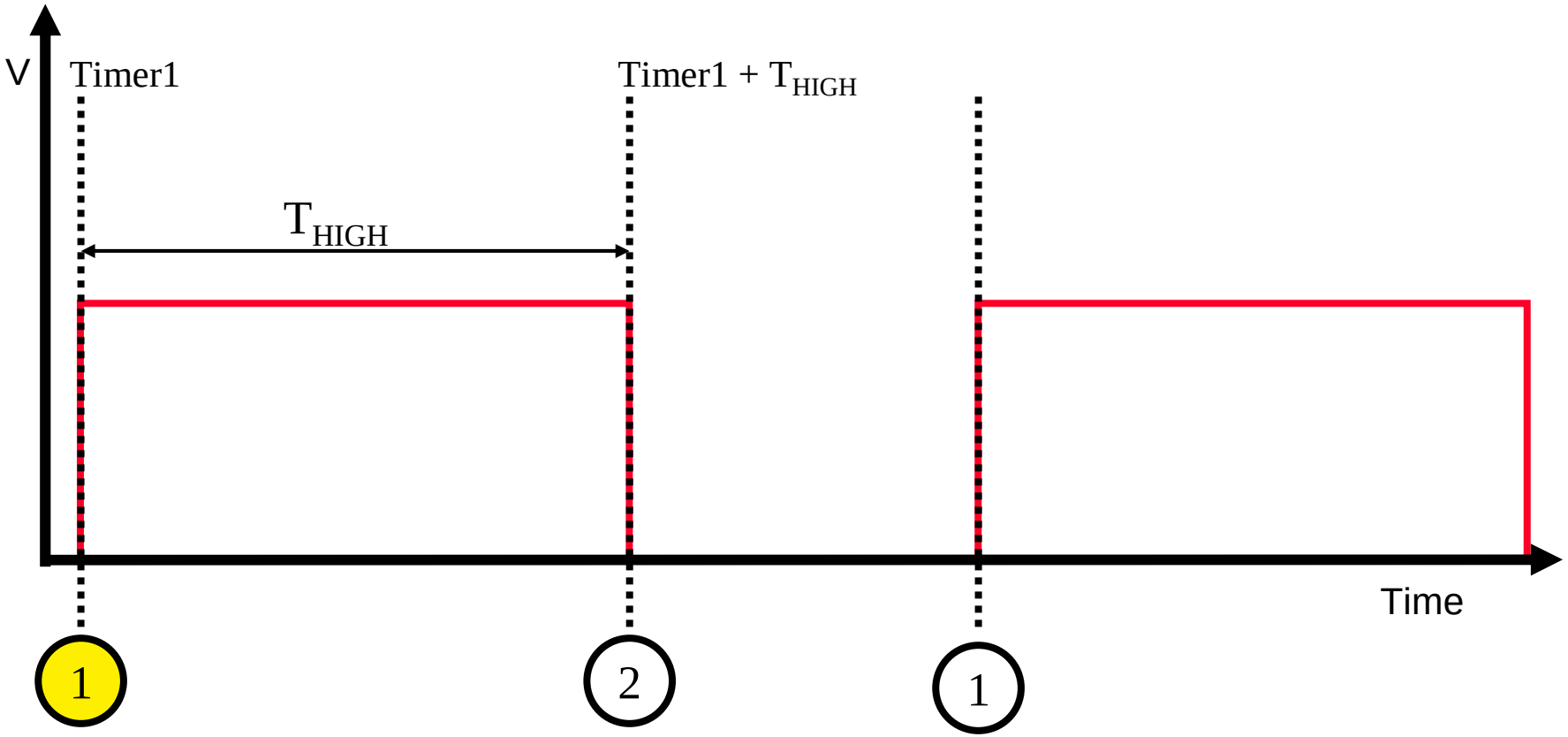


Generating a Software PWM with CCP Compare Mode

- **COMPARE** Mode to time when the I/O Line needs to be toggled
- Interrupt is generated when TMR1 equals CCPR1

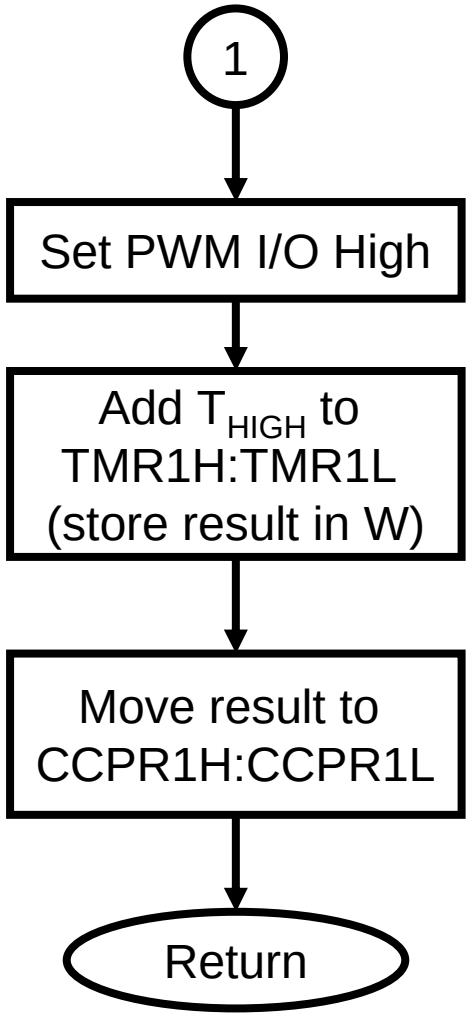


Generating a Software PWM using the CCP Compare Mode

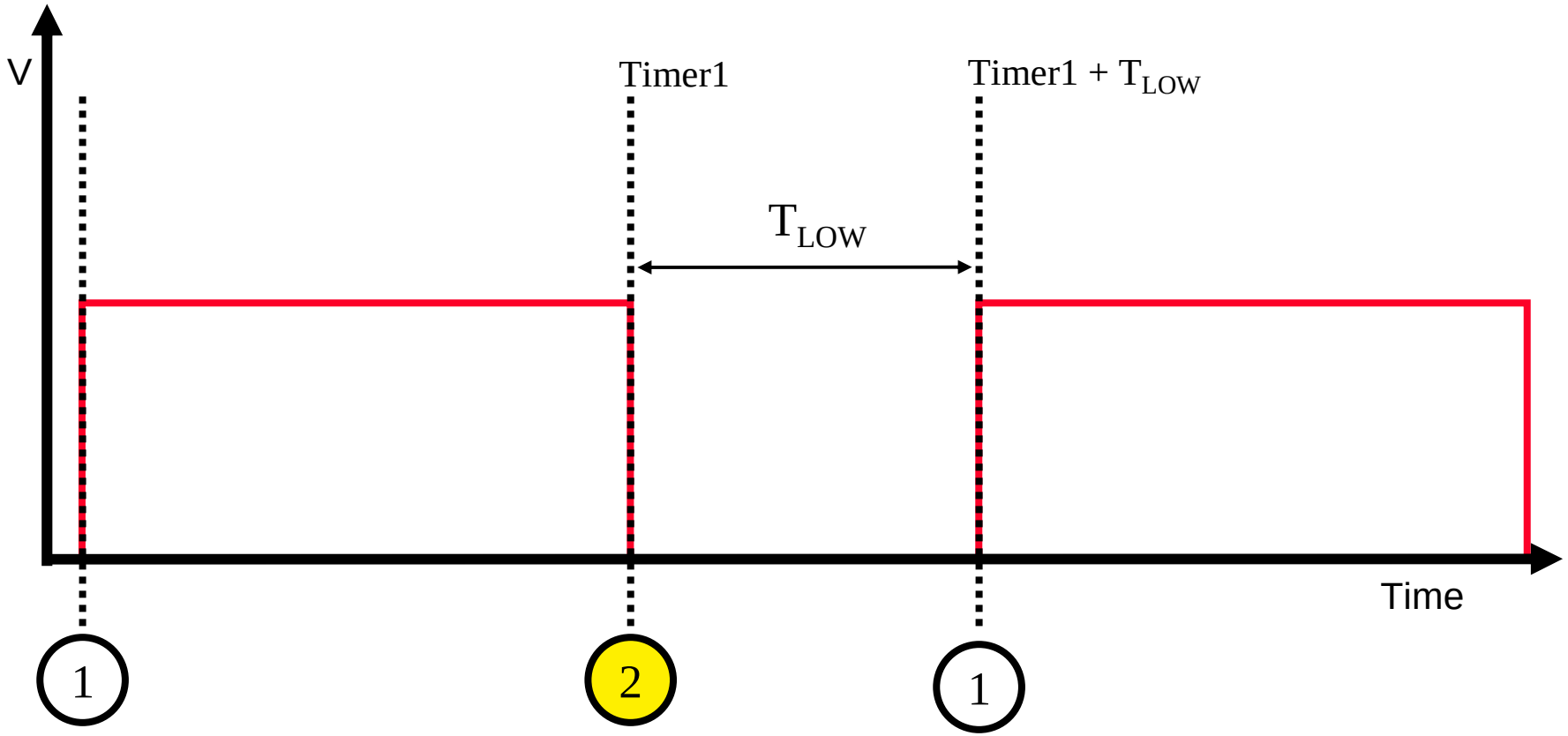


Software PWM Flowchart

- CCP (Compare) Interrupt

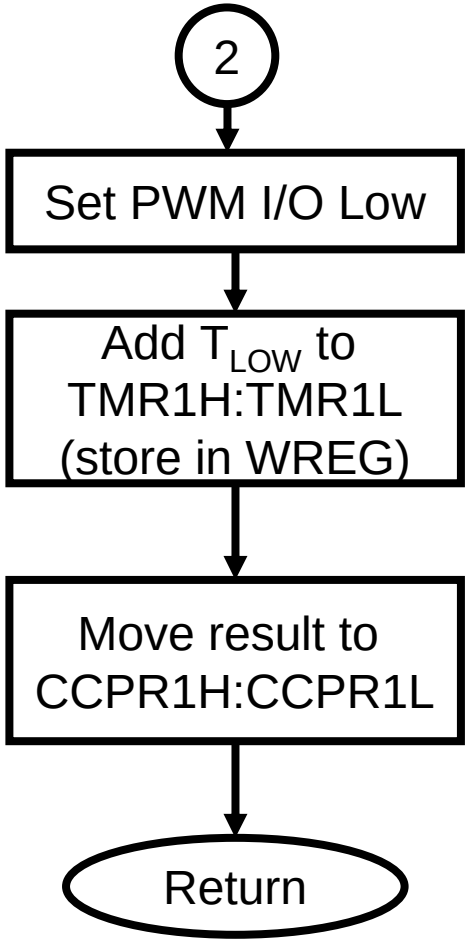


Setting PWM Duty Cycle

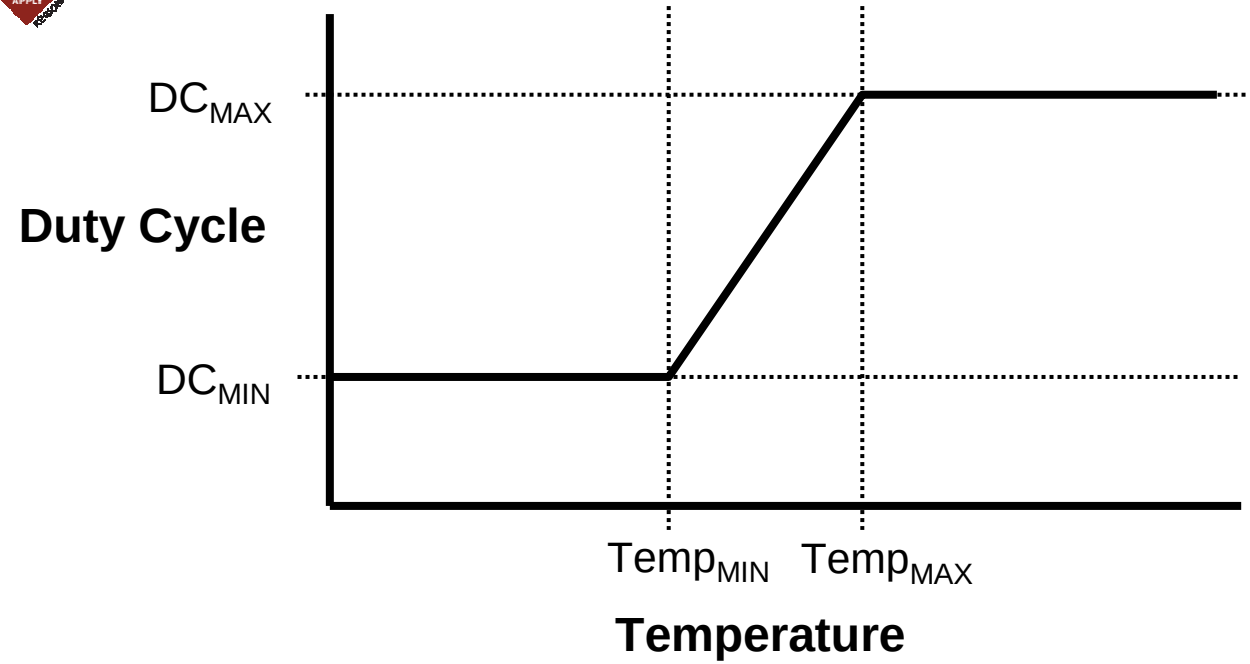


Software PWM Flowchart

- CCP (Compare) Interrupt

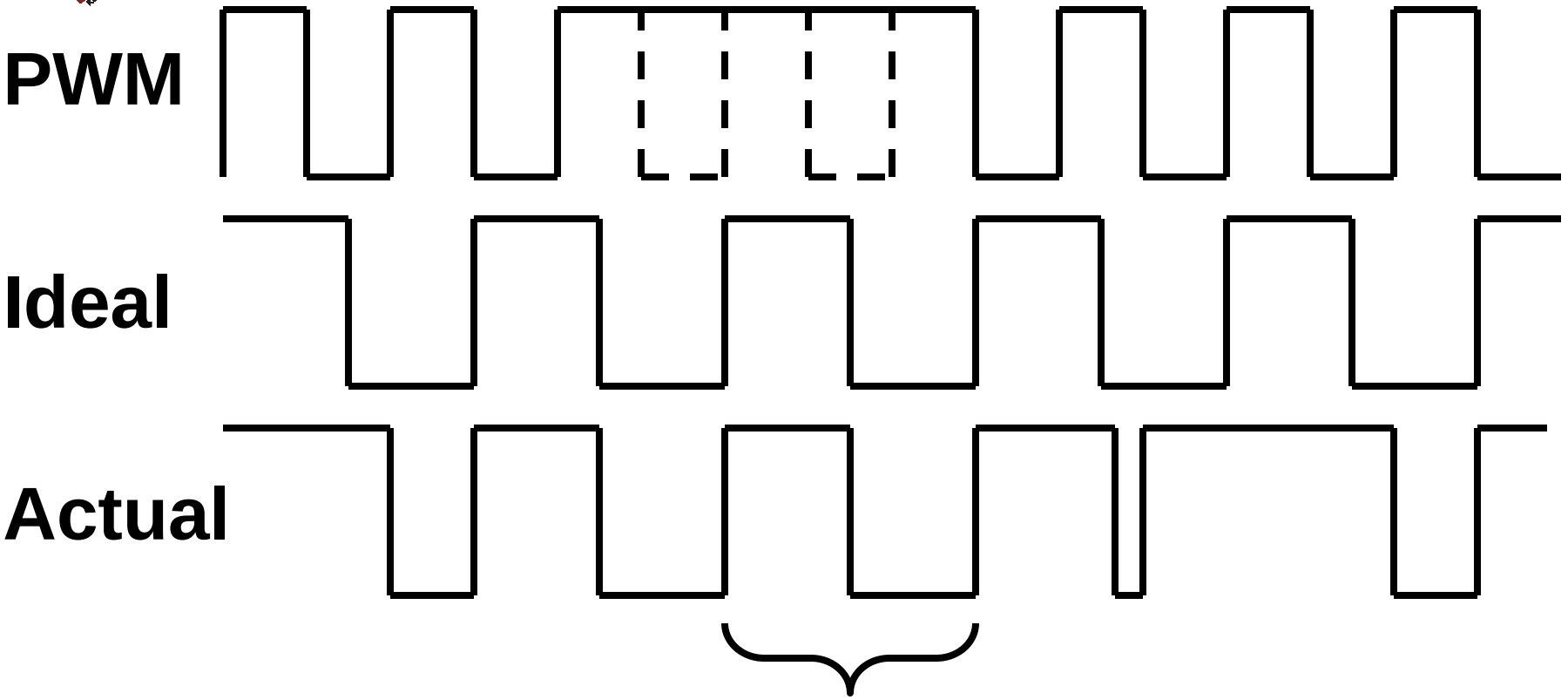


Calculating PWM Duty Cycle



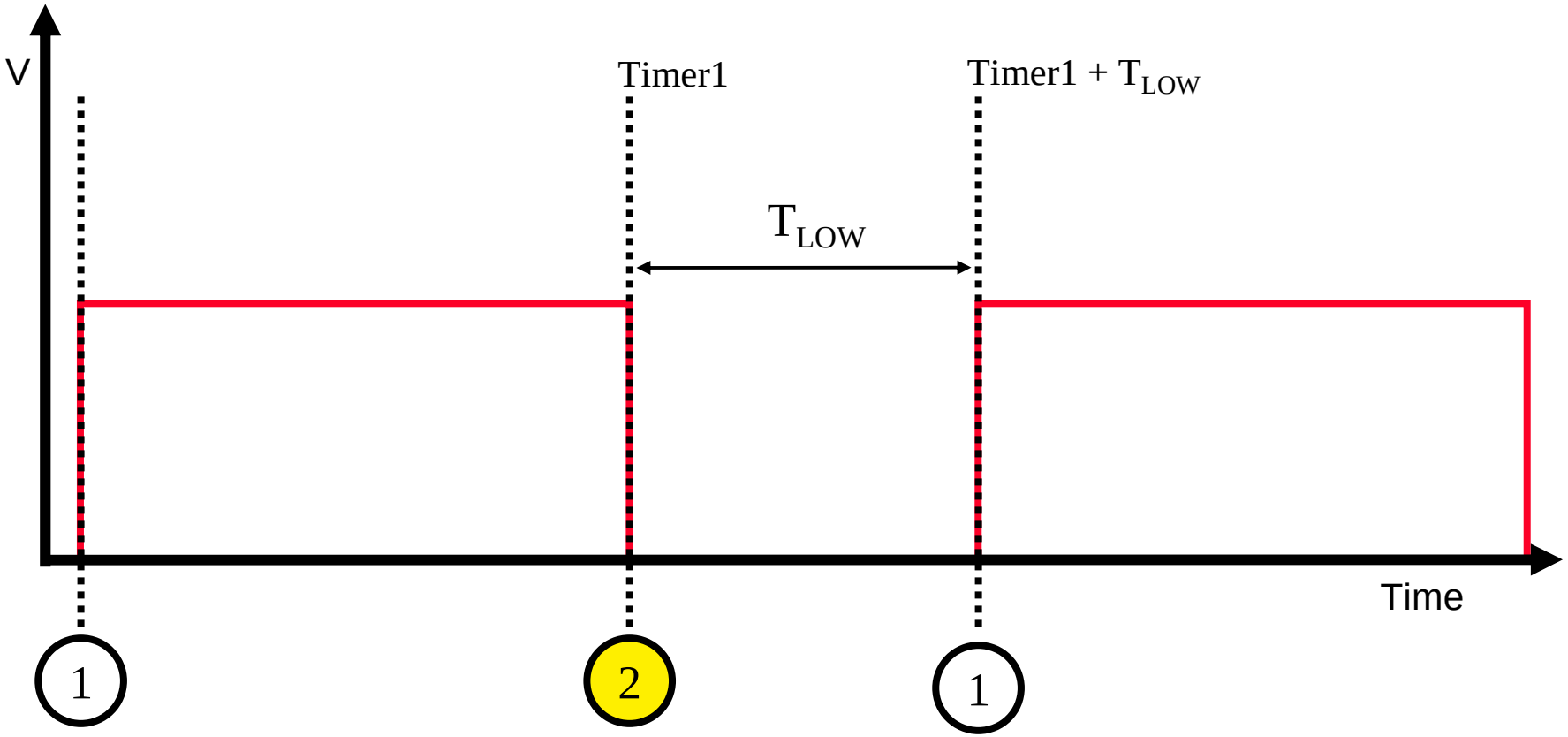
- $T_{HIGH} = \mathbf{T_A} * [(\mathbf{DC_{MAX}} - \mathbf{DC_{MIN}}) / (\mathbf{Temp_{MAX}}\text{-}\mathbf{Temp_{MIN}})]$
- $T_{LOW} = \mathbf{T_{PERIOD}} - T_{HIGH}$

Recall: Pulse Stretching



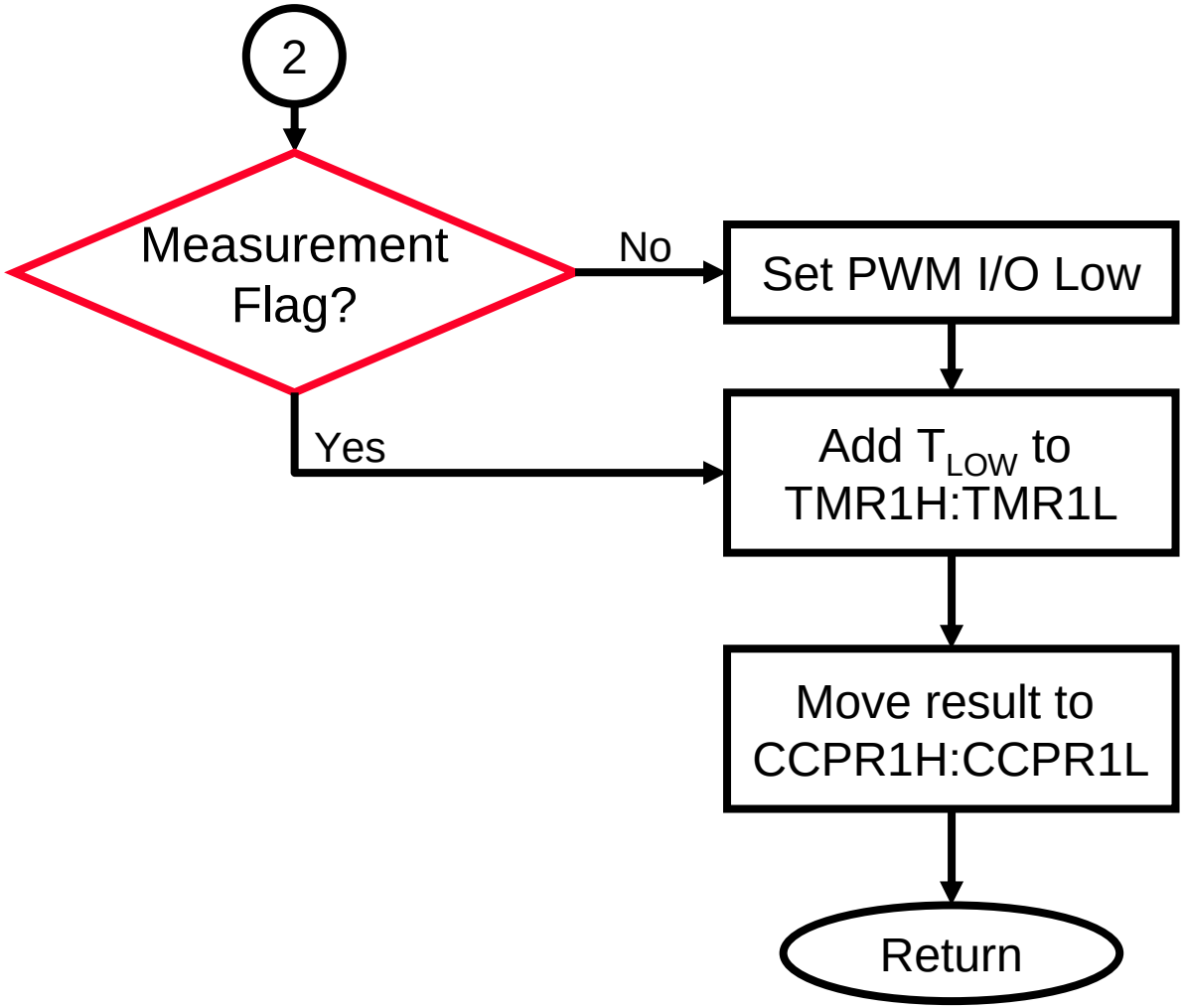
- Must leave the PWM line high if a measurement in in progress
- Solution: Create a flag called **Measurement Flag** to indicate when a measurement is in progress

Setting PWM Duty Cycle



Software PWM Flowchart

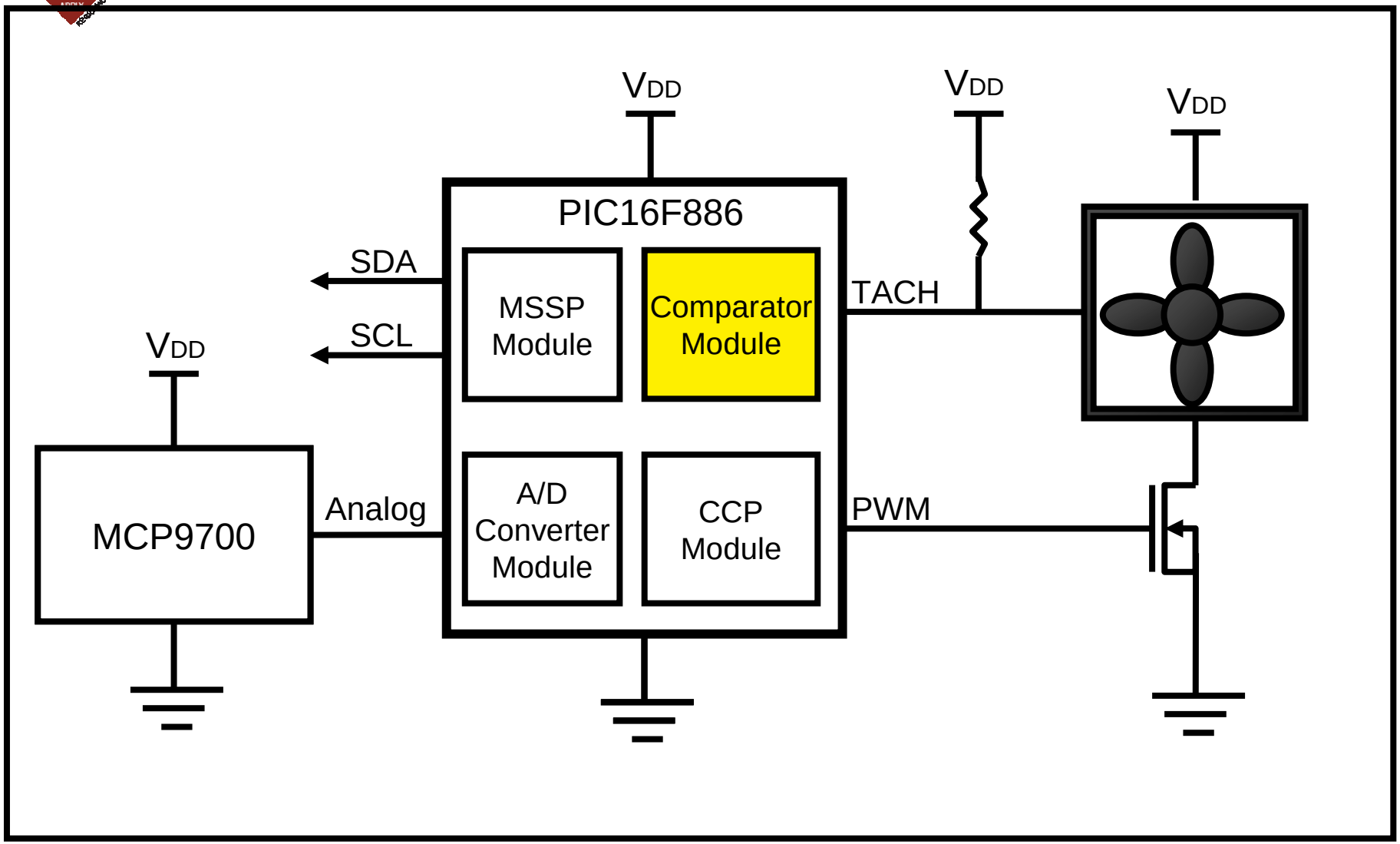
- CCP (Compare) Interrupt



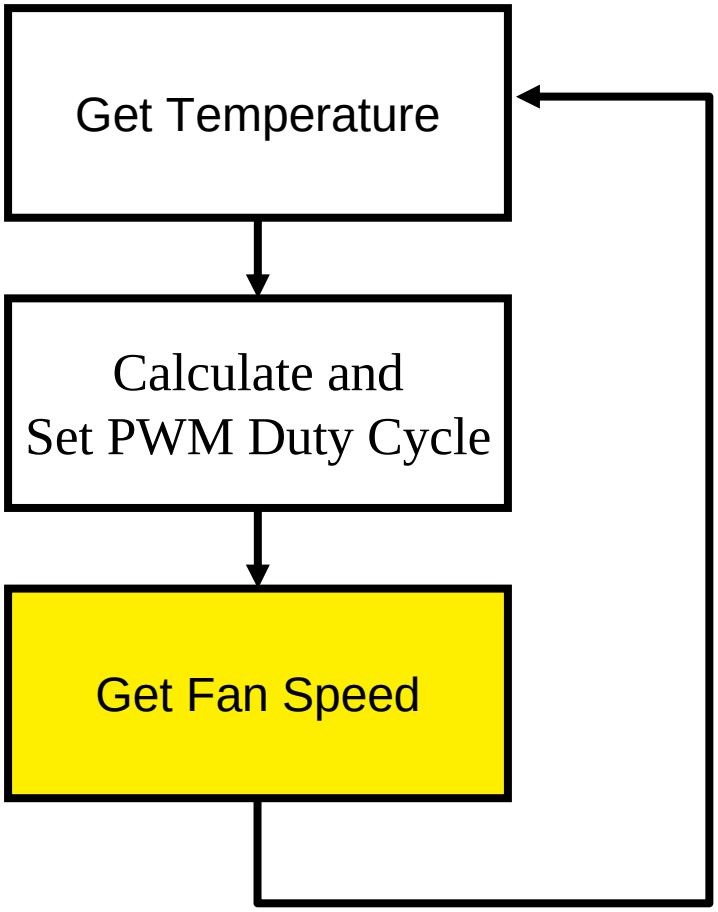
PWM Generation

Questions?

Thermal Controller Implementation



PIC Thermal Controller Flowchart

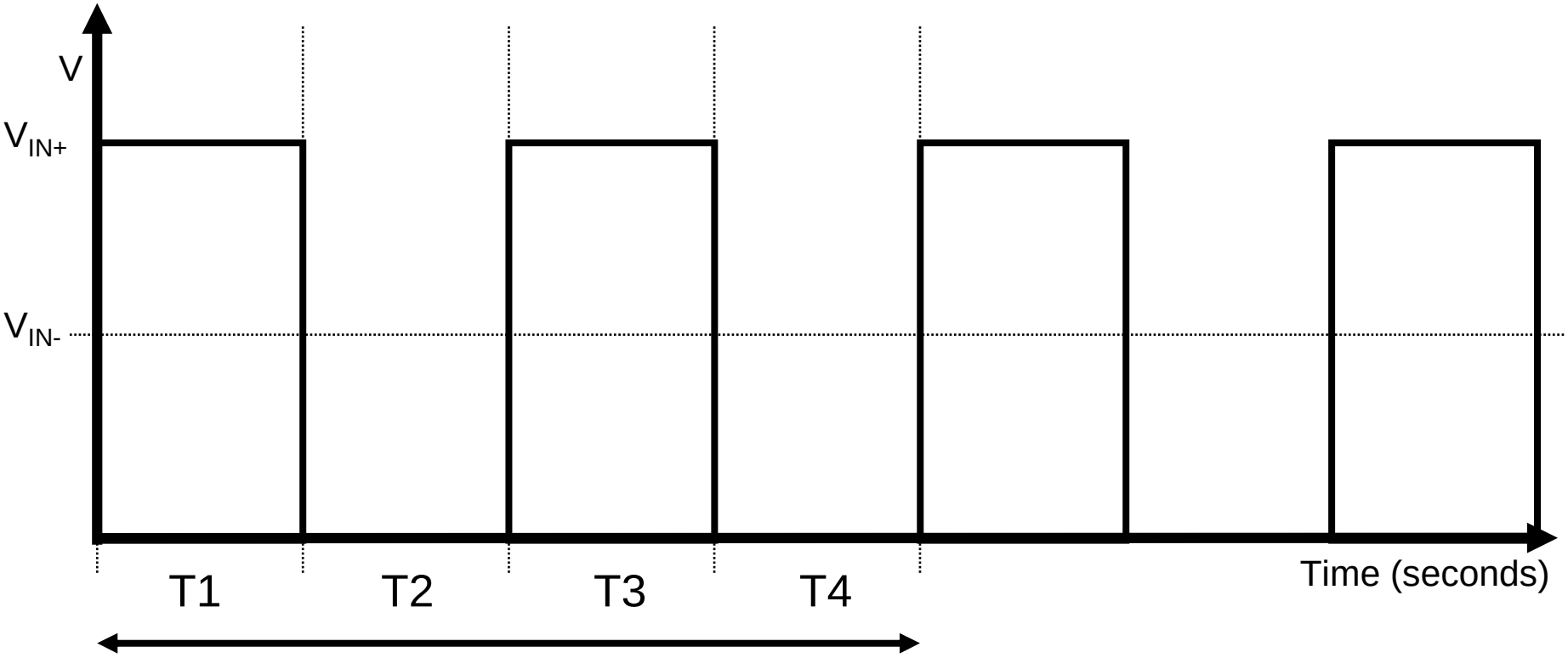


Measuring Fan Speed

- Comparator and timer peripheral

- 4 transitions per revolution on TACH line
  - Fan datasheet specifies TACH line output

Tachometer Output



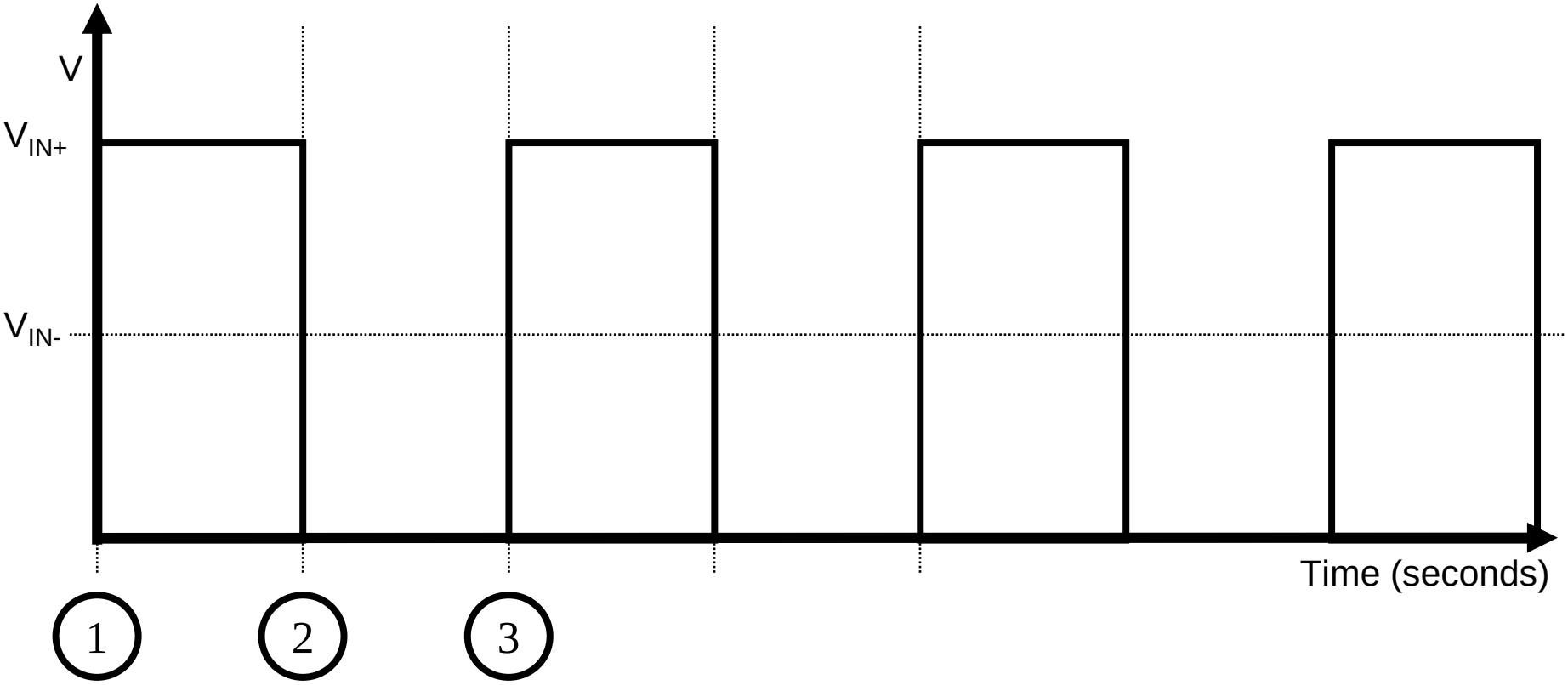
T = 1 Rotation

T = T1 + T2 + T3 + T4

T1 = T2 = T3 = T4 = 60 / (4 x RPM)

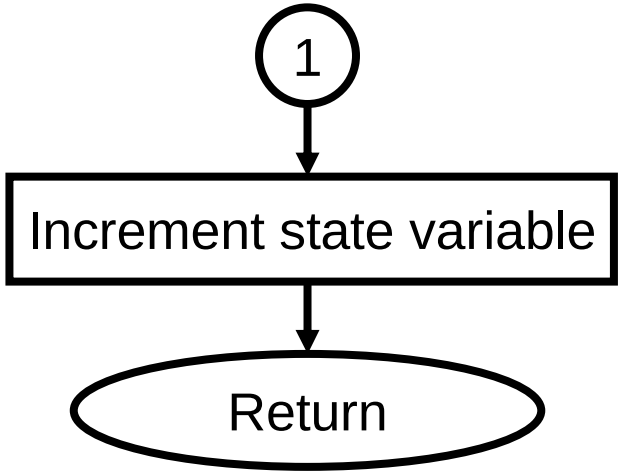
RPM = 60 / (4 x T1)

Comparator Interrupts

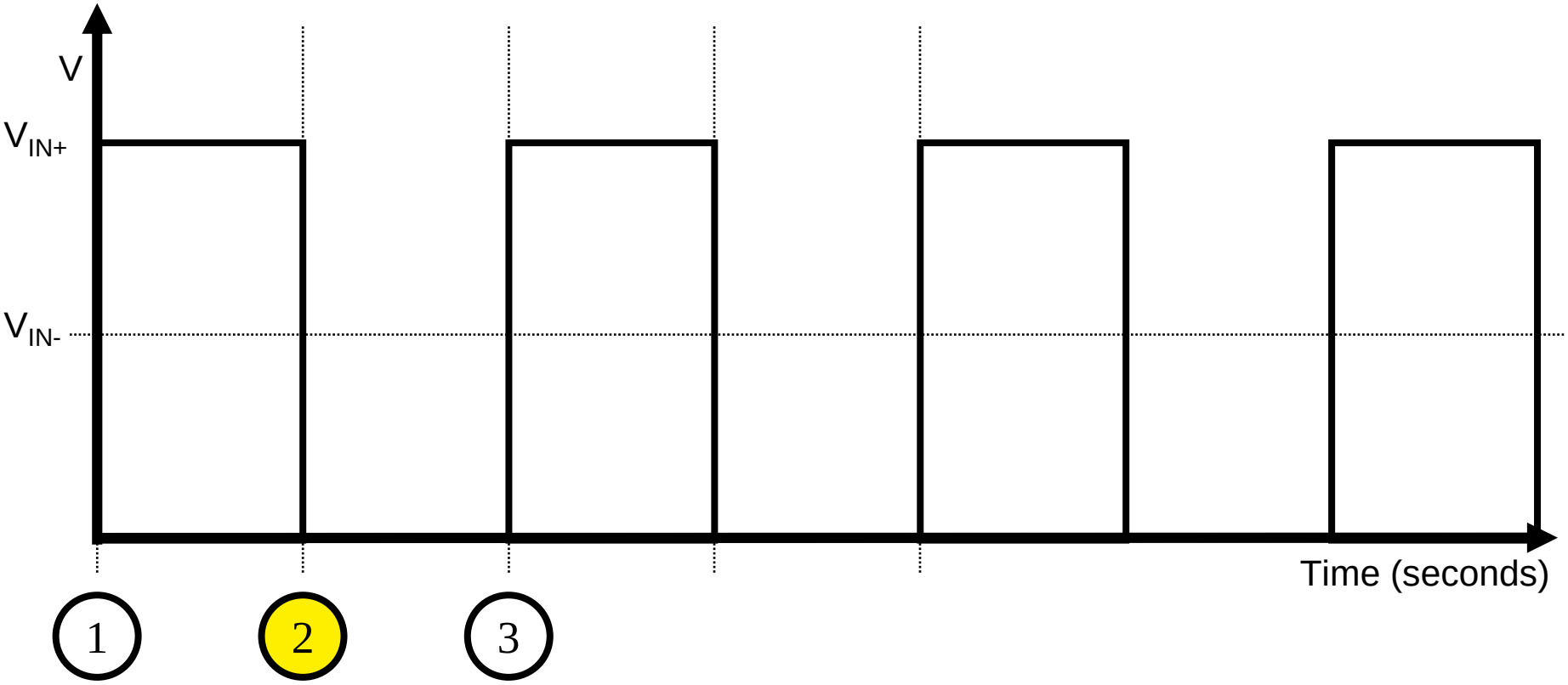


Measuring Fan Speed

- Comparator Interrupt
- Increment the state variable

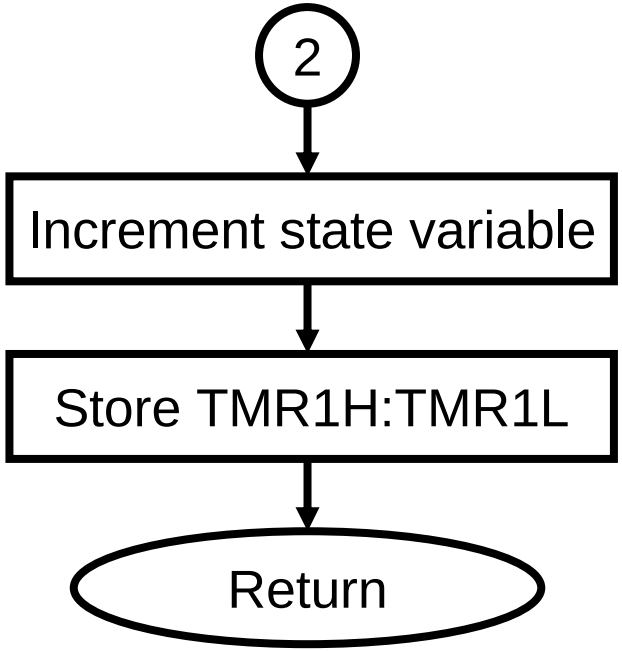


Comparator Interrupt 2

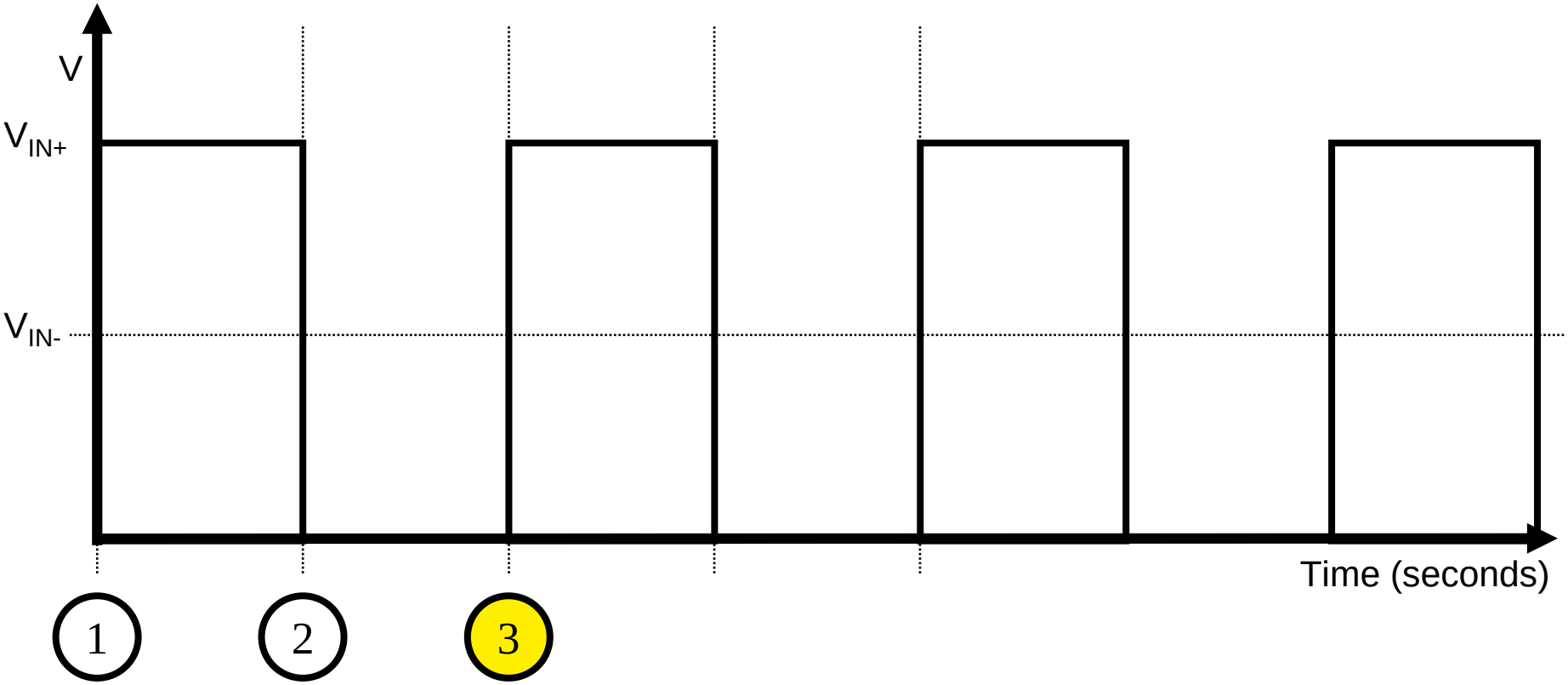


Measuring Fan Speed

- Comparator Interrupt
- Second interrupt stores Timer 1 value

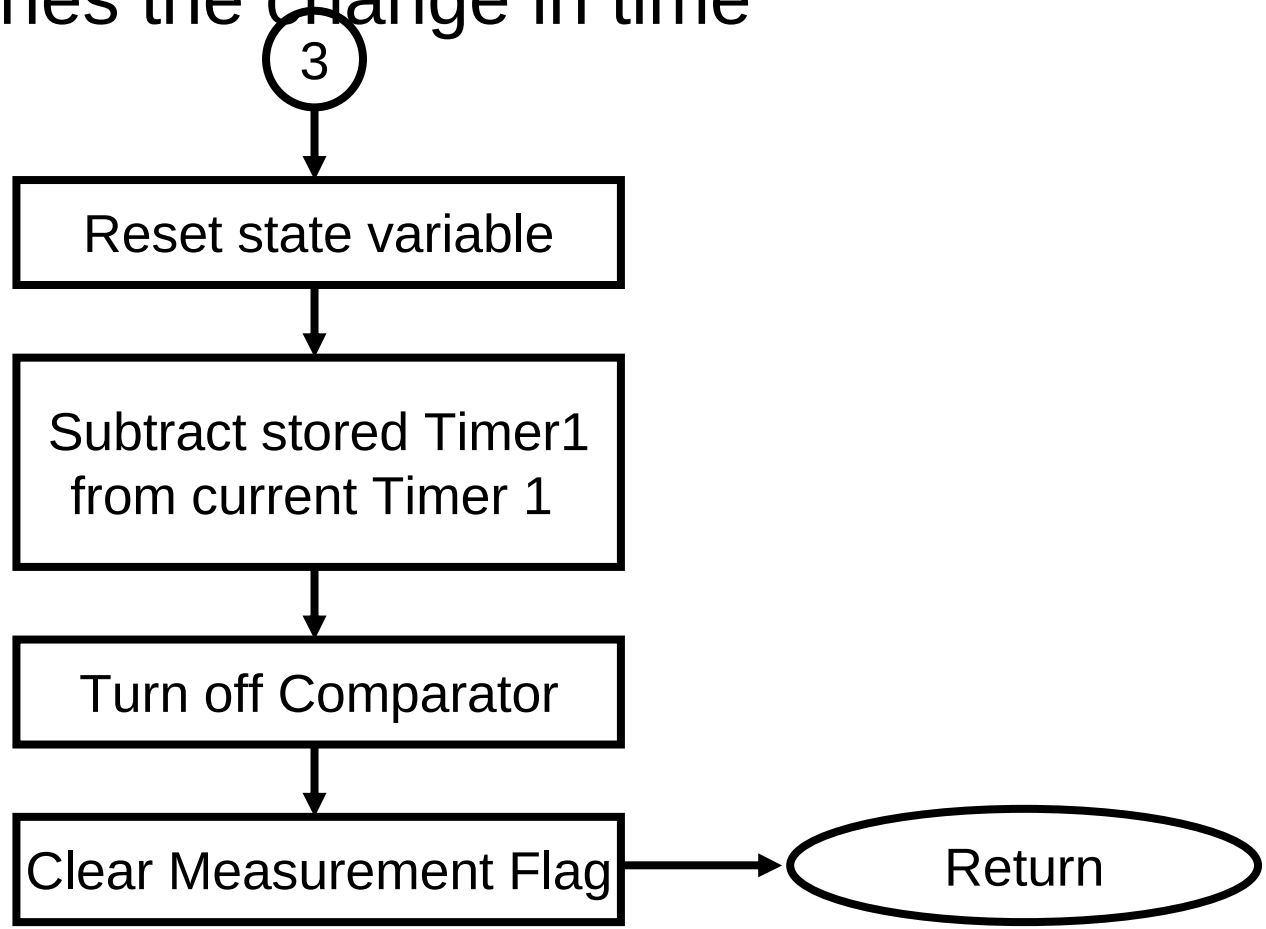


Comparator Interrupt 3

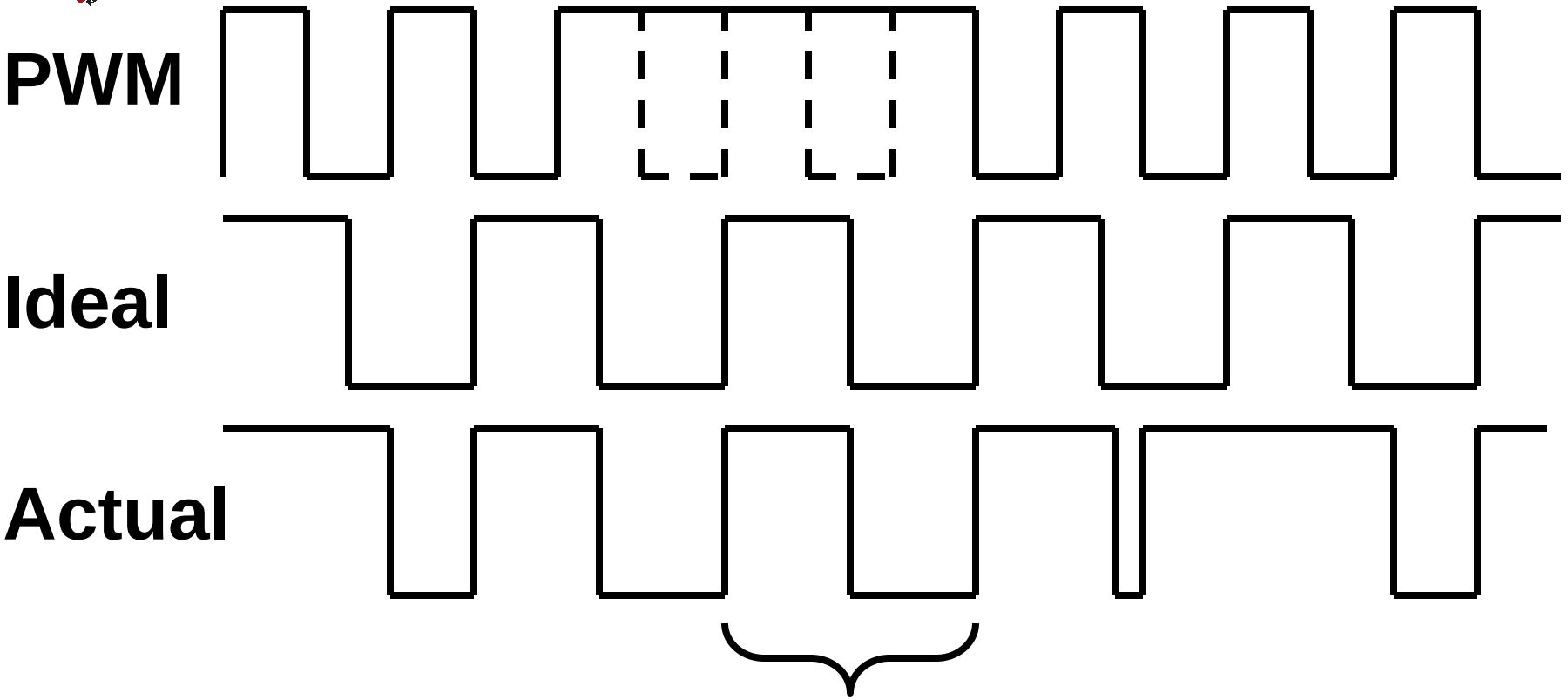


Measuring Fan Speed

- Comparator Interrupt
- Third determines the change in time

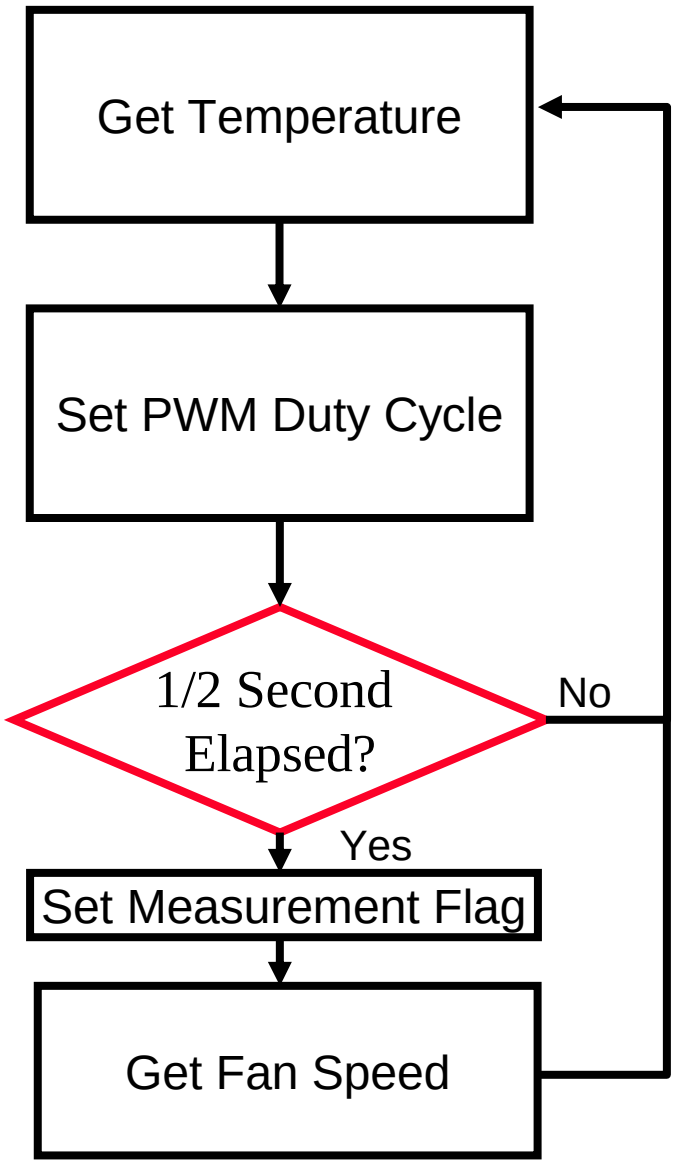


Problem: Measuring too often



- Fan will be full on during measurements
- Measuring too often can change the speed of a fan
- Not measuring enough may yield inaccurate speed measurements
- Solution: Measure every half second

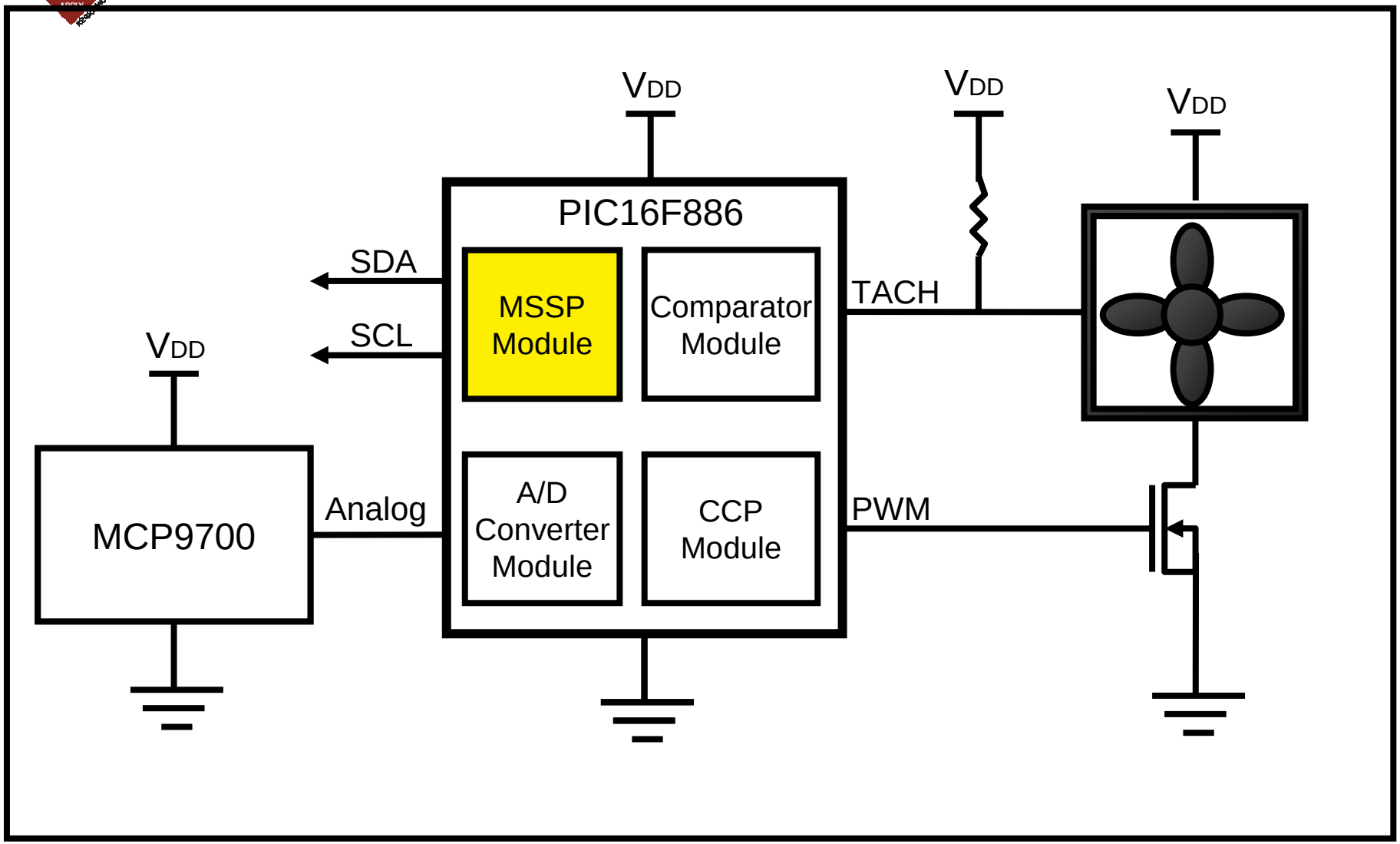
Solution



Measuring Fan Speed

Questions?

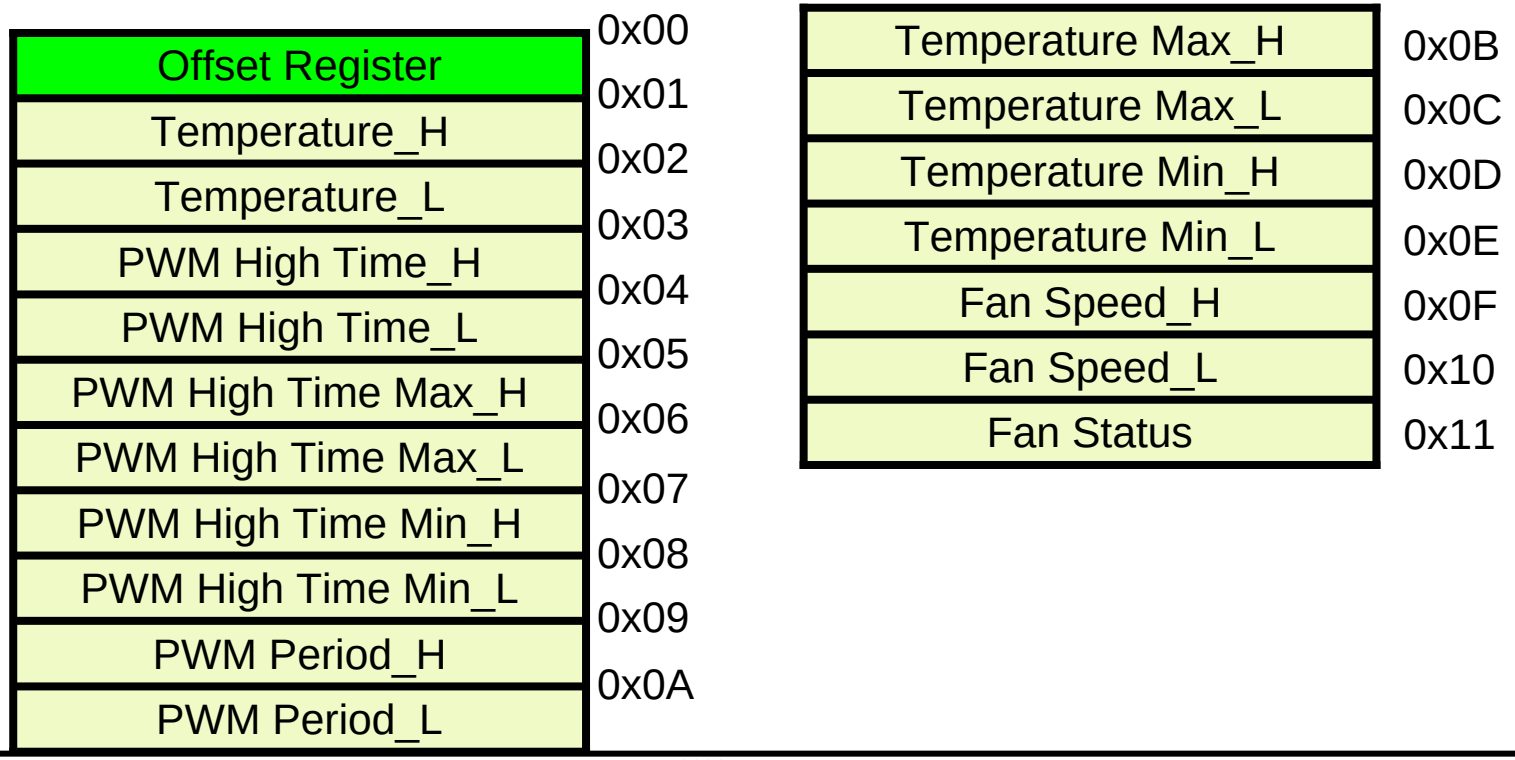
Thermal Controller Implementation



Serial Communications

- Fan Controller writes data to GPR
- Fan Controller data is made read/writable through the I²C™ Slave Mode Firmware

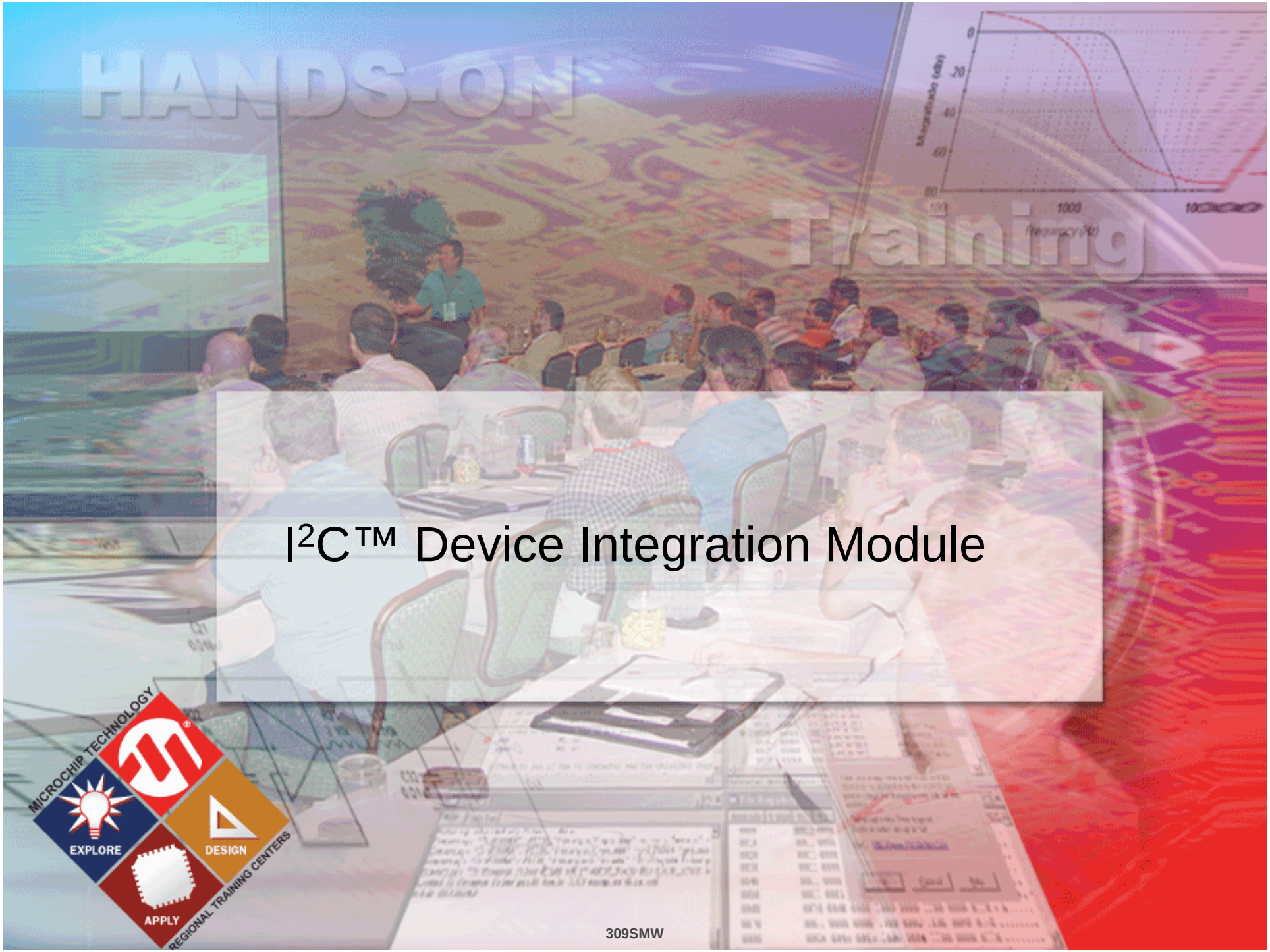
General Purpose Registers




HANDS-ON
Training
Lab 4: Thermal Management Lab
MICROCHIP TECHNOLOGY
EXPLORE
DESIGN
APPLY
REGIONAL TRAINING CENTERS
309SMW

Summary

- Thermal Management Basics
- 3 Wire Fans
- Thermal Controllers
- Implementing a Thermal Controller on a PIC16F886
- Lab Exercise



I²C™ Device Integration Module

309SMW

Module Learning Objectives

- At the end of this class you should be able to:
  - Integrate emulated I²C™ devices on the PIC16F886 using the I²C Slave Mode Firmware
  - Design software that takes multitasking considerations into account

- Prerequisites
  - I²C Module
  - Real-Time Clock Module
  - Serial EEPROM Module
  - Thermal Management Module

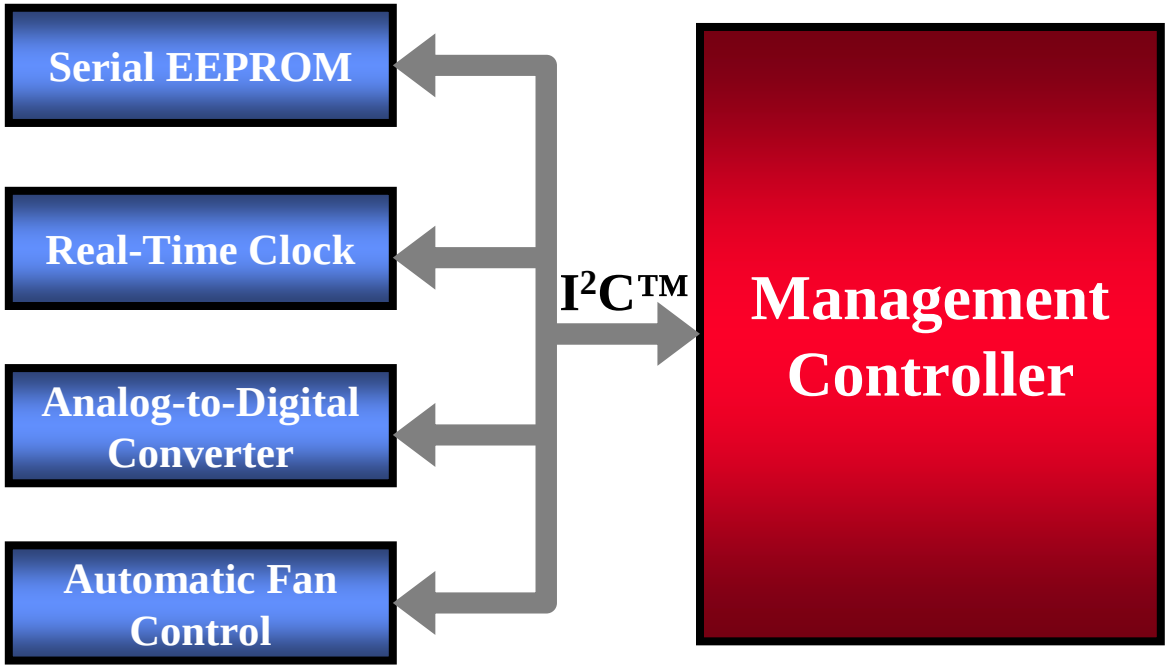
Module Agenda

In the next hour we will discuss…

- Integration Basics
- Integrated I^2C™ Firmware
- Firmware Multitasking Considerations
- Summary

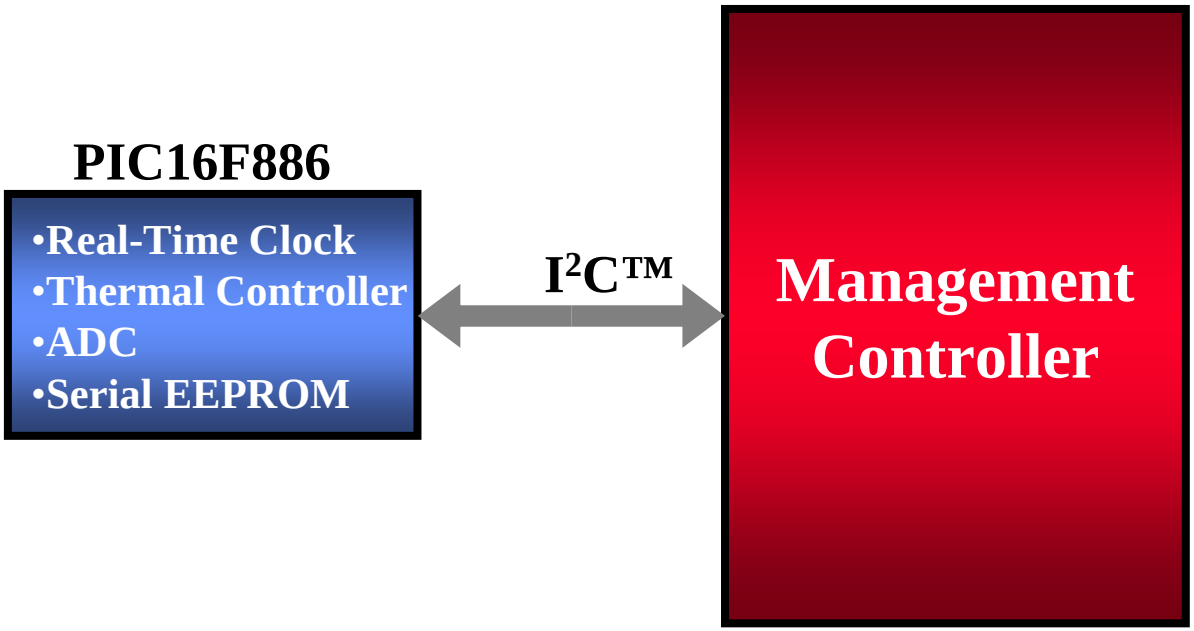
Integration Basics

- Typical Bus Configuration



Integration Basics

- Bus Integration Using PIC16F886



Pros and Cons of Integration with PIC® Microcontroller

PROS

- Cheaper
- Less board space
- Customizable
- Intelligence

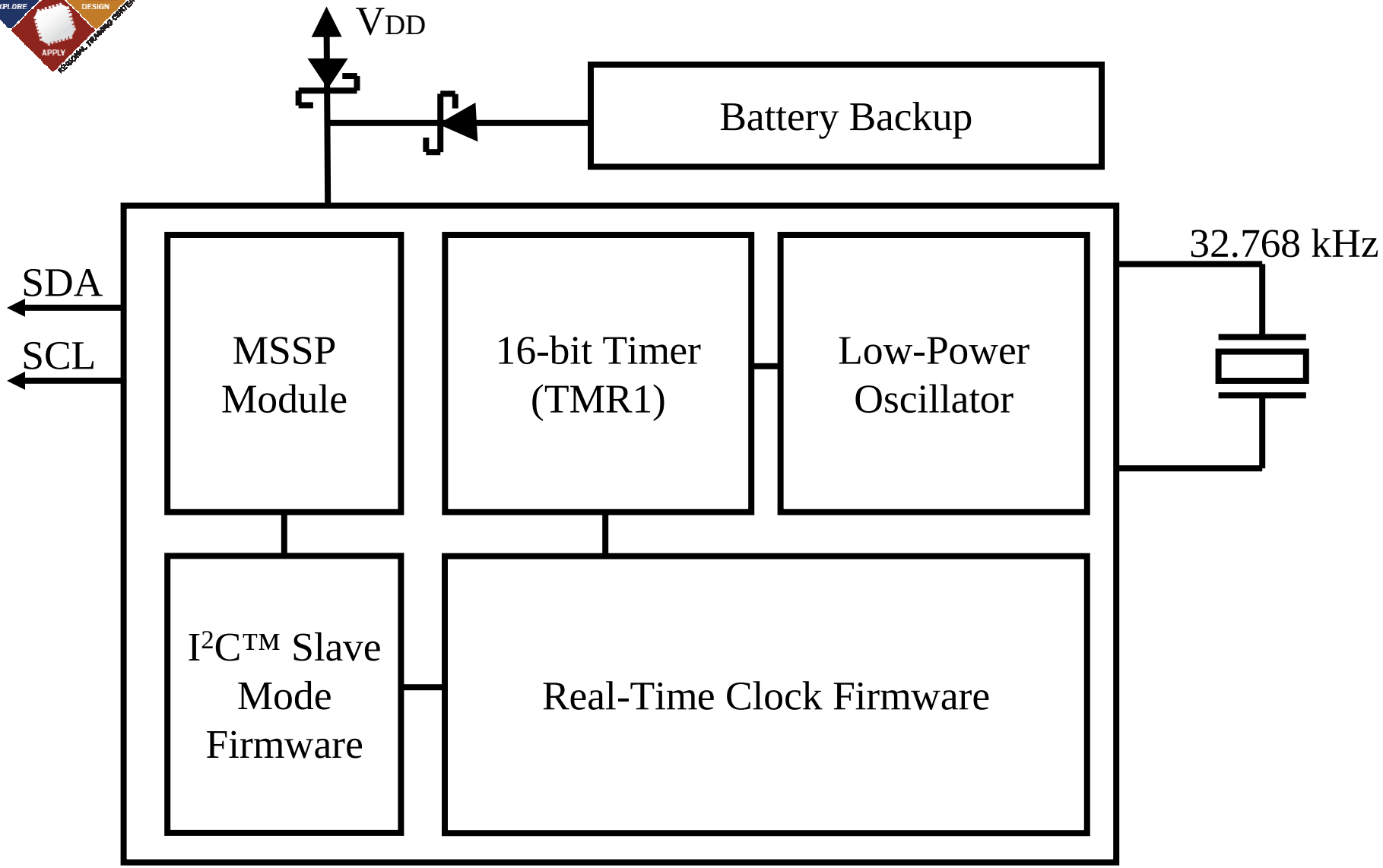
CONS

- Software Complexity
- Increased development time

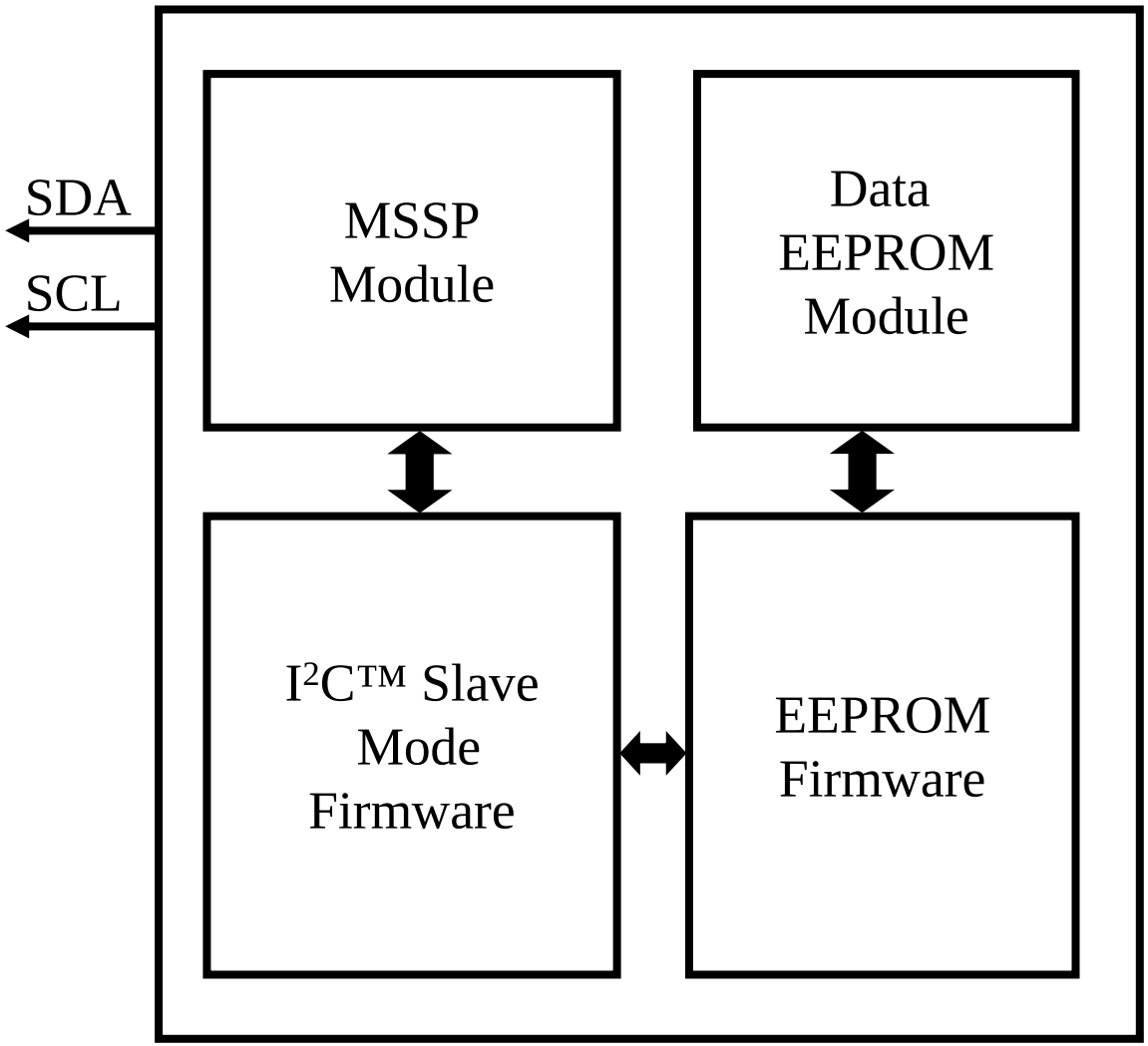
I²C™ Slave Devices

- So far we have created a…
  - Real-Time Clock / Calendar
  - Serial EEPROM
  - Thermal Controller

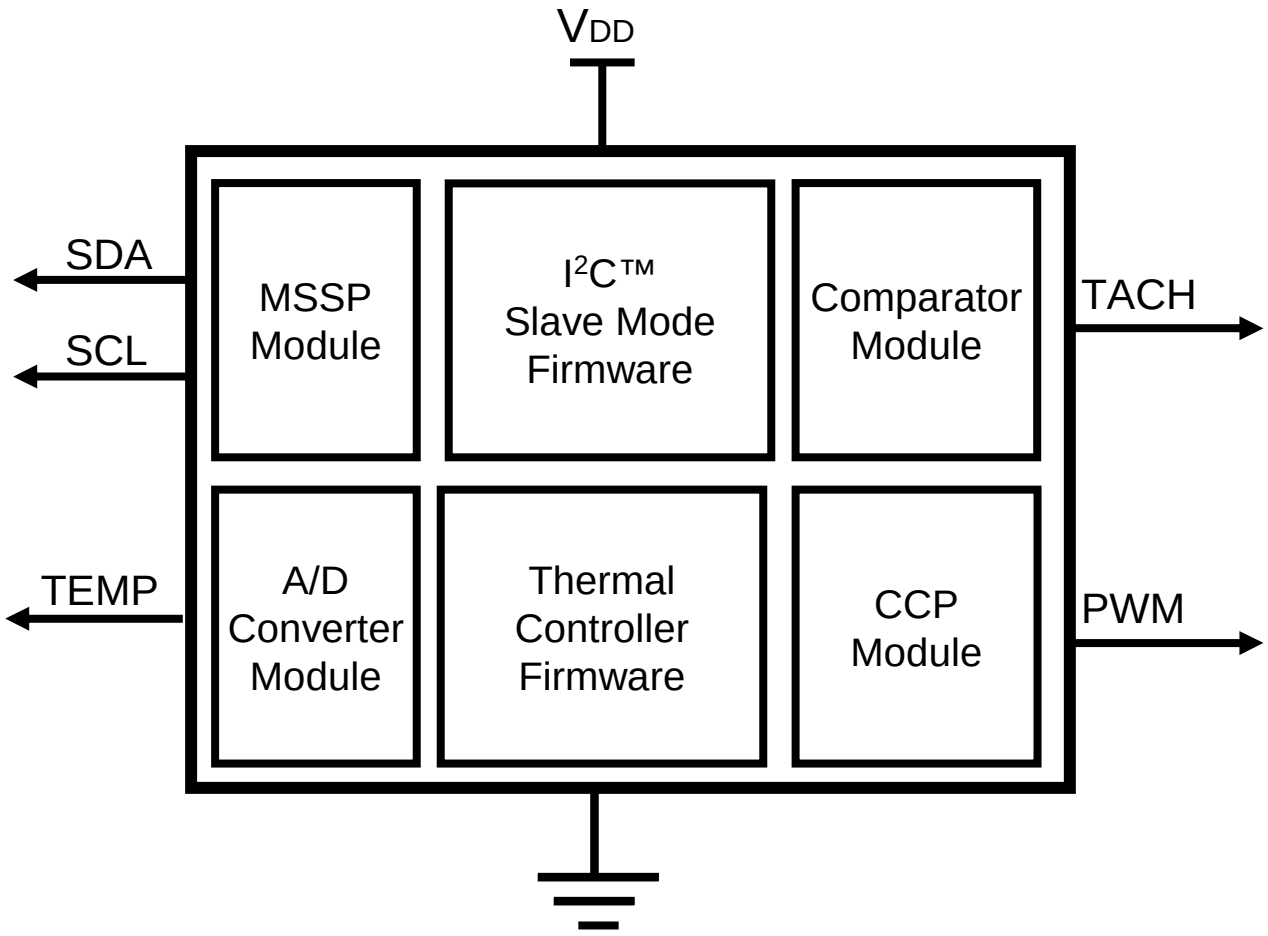
I²C™ Real-Time Clock / Calendar



Serial EEPROM

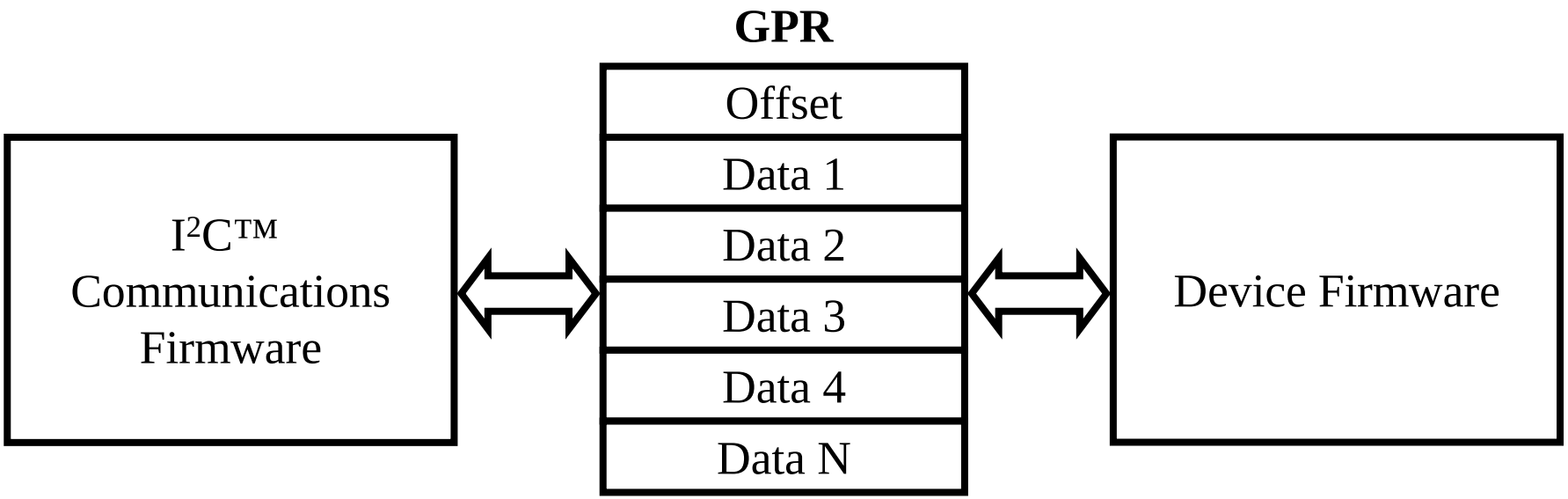


Thermal Controller



I²C™ Slave Mode Firmware

- We have created modular firmware that…
  - Interprets I²C communications across the bus
  - Writes to and reads from general purpose registers (GPR)



I²C™ GPR

Real-Time Clock
Index
Control / Status1
Control / Status 2
Seconds
Minutes
Hours
Days
Weekdays
Months/Century
Years
Minute Alarm
Hour Alarm
Day Alarm
Weekday Alarm

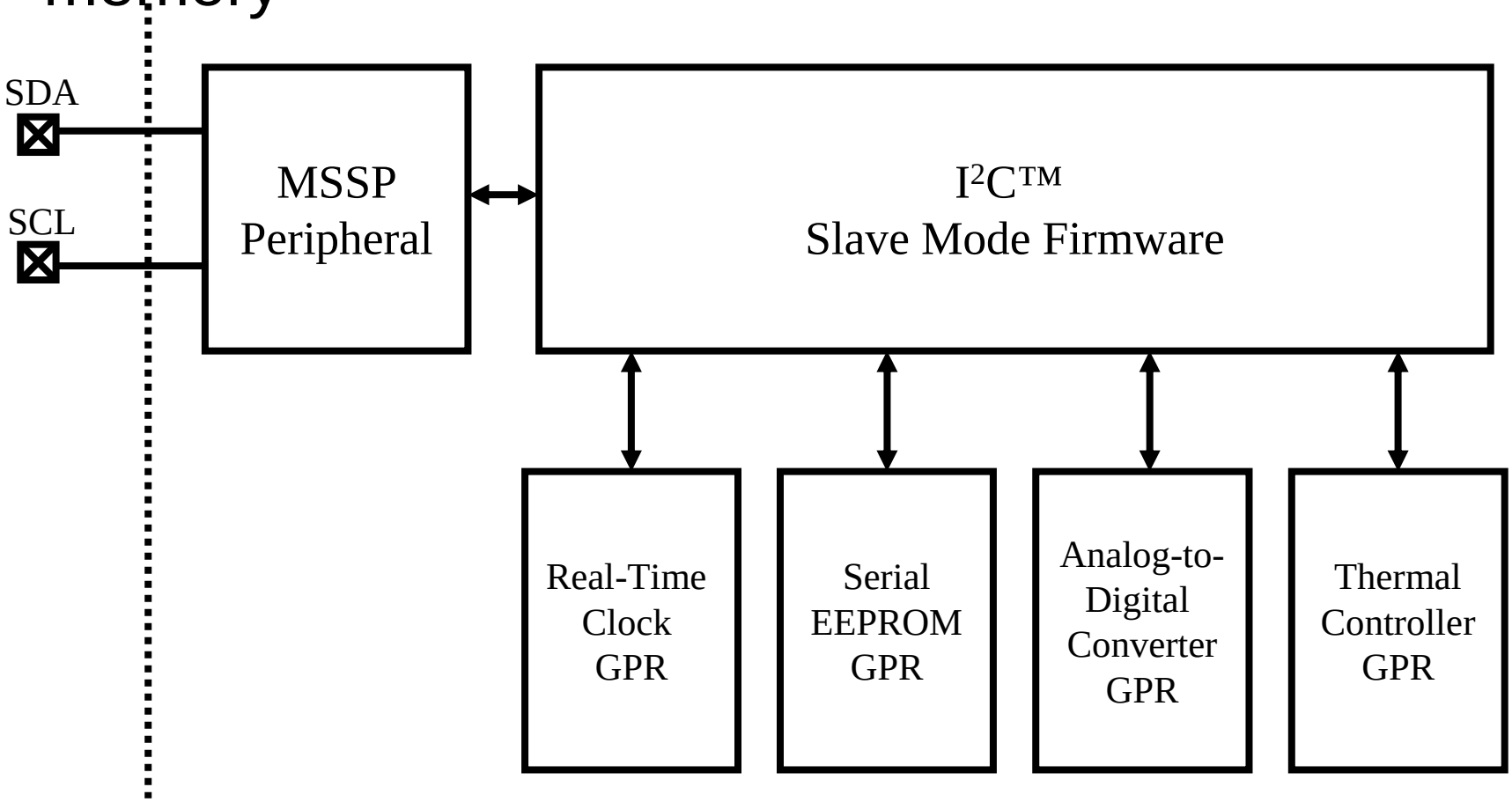
Serial EEPROM
Index
Page Byte 1
Page Byte 2
Page Byte 3
Page Byte 4
Page Byte 5
Page Byte 6
Page Byte 7
Page Byte 8
Word Address

Analog-to-Digital
Index
ADC_High
ADC_Low

Thermal Control
Index
Temp_H
Temp_L
Duty_H
Duty_L
OutMax_H
OutMax_L
OutMin_H
OutMin_L
Period_H
Period_L
TempMax
TempMin
TachSpeed
Status

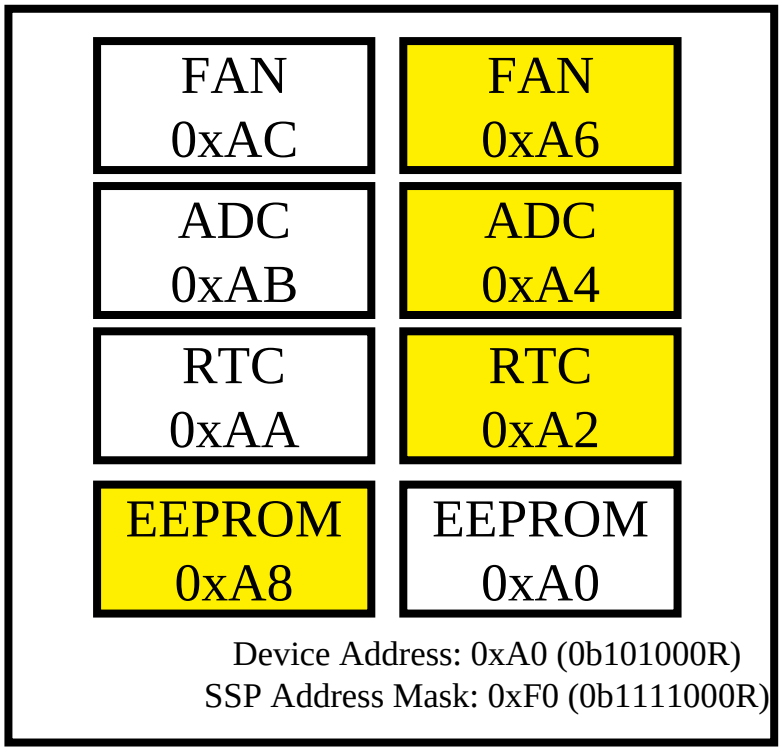
How does I²C™ firmware change?

- Must be able to address multiple address and memory



I²C™ Slave Mode Firmware

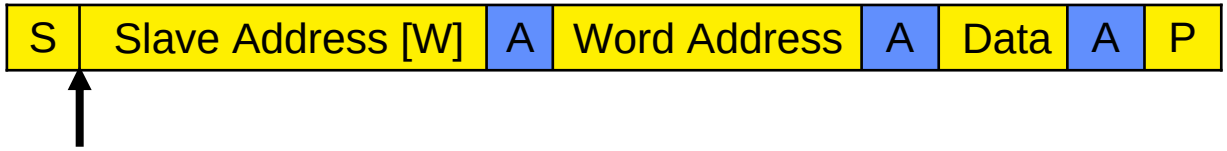
- Address masking allows each device to have a unique Address

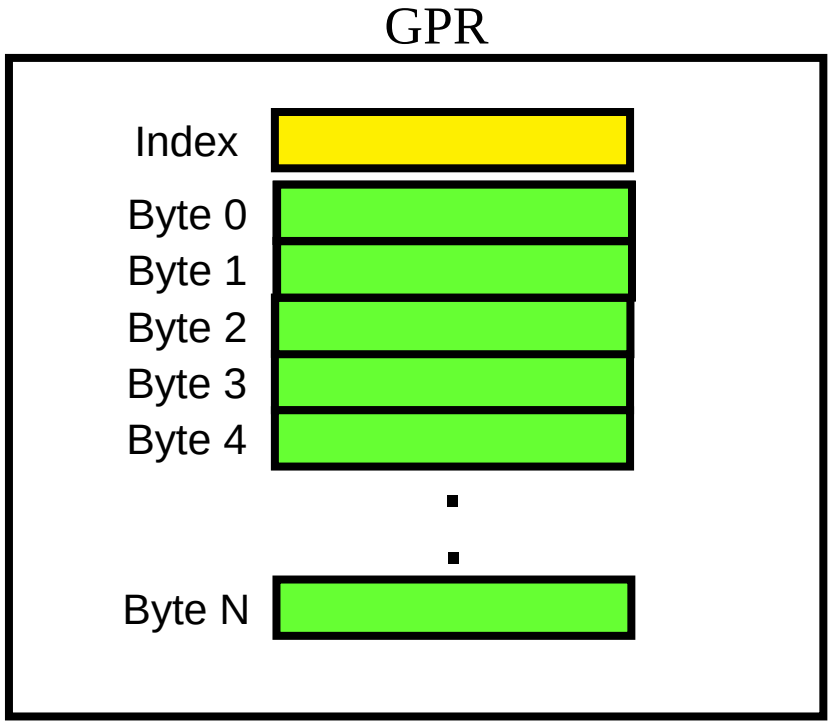


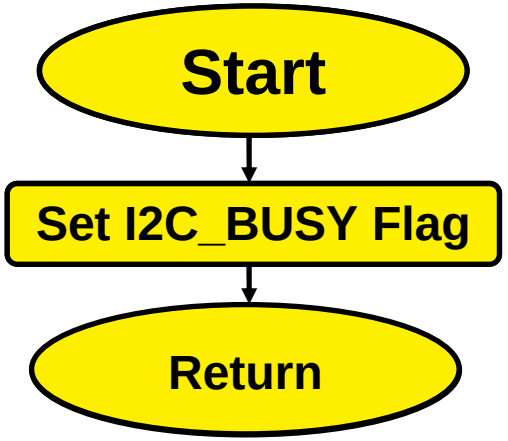
- Use a **software flag** to indicate which device memory to operate on

Review: Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

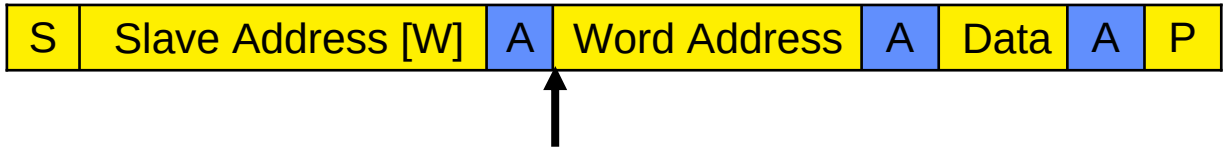


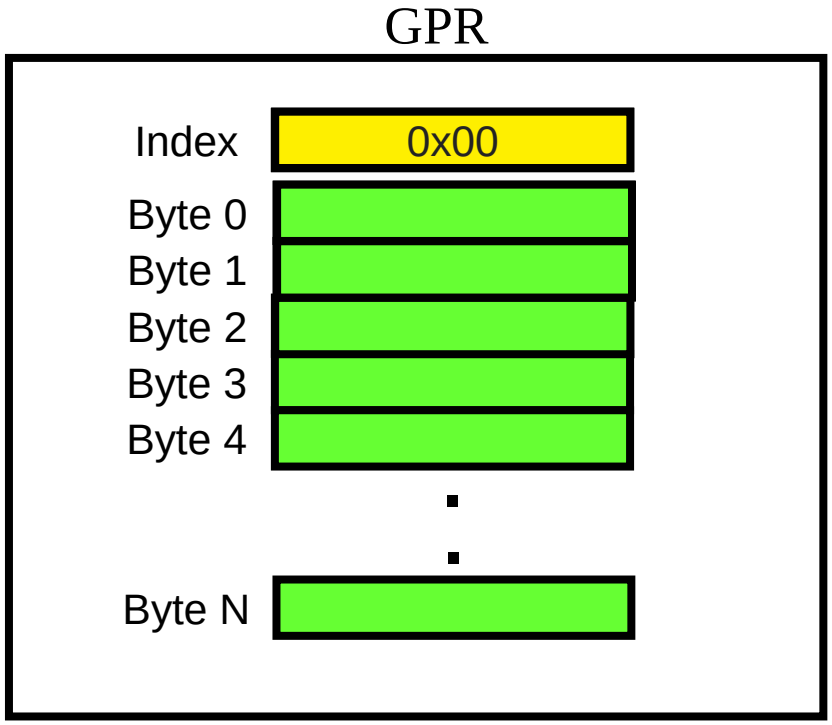


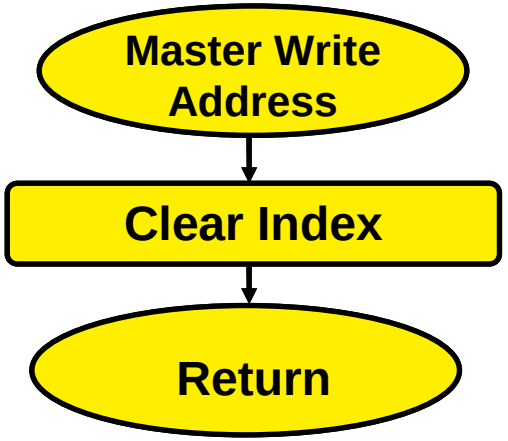


Review: Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

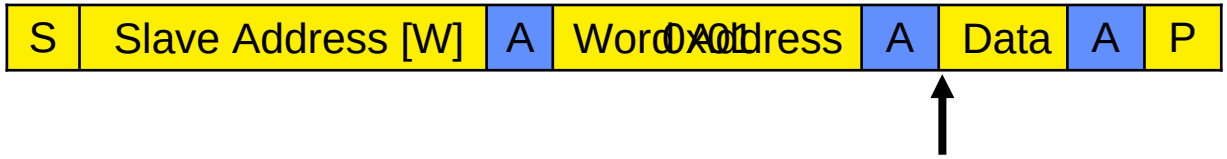


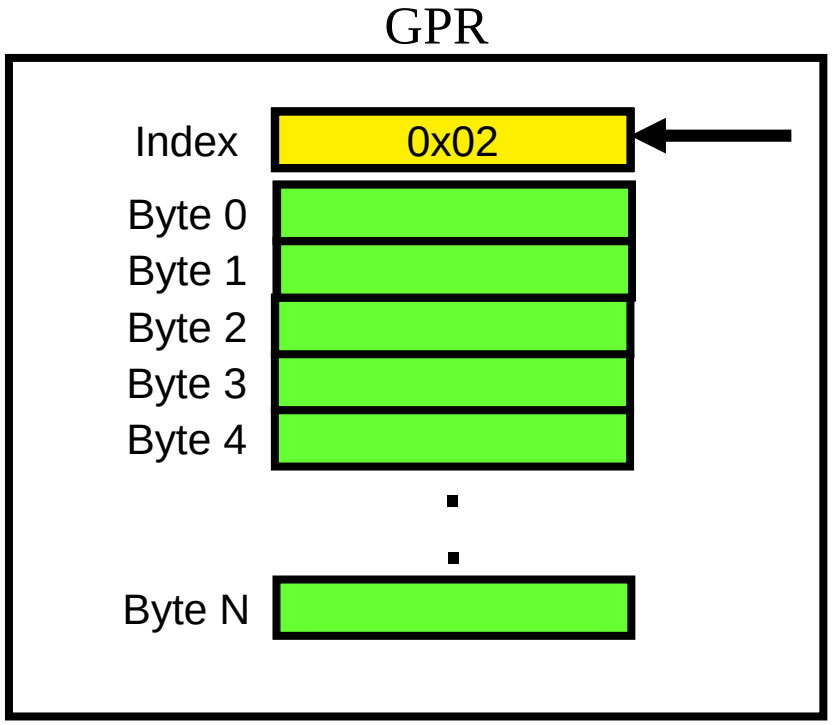


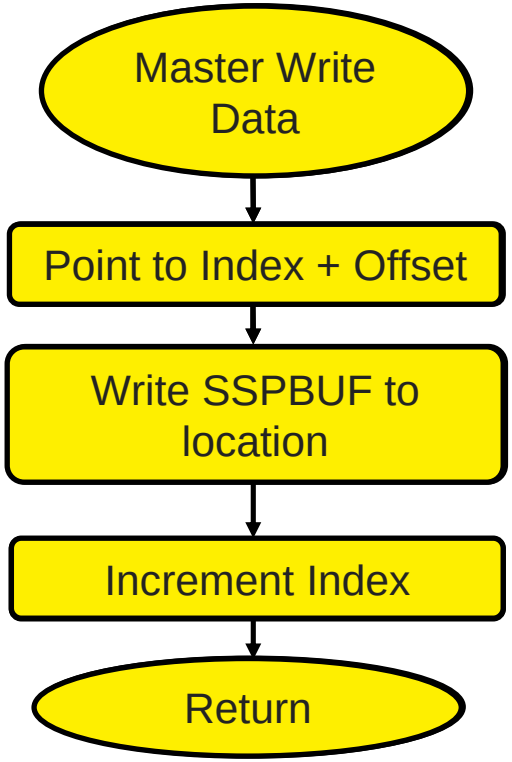


Review: Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

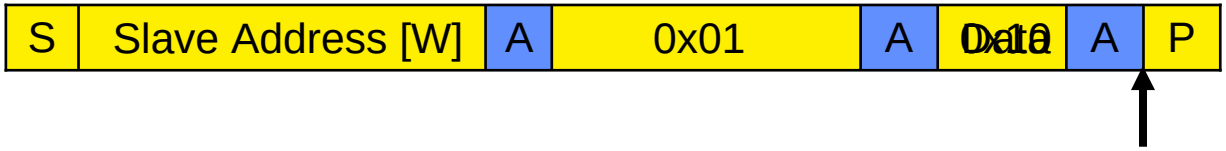


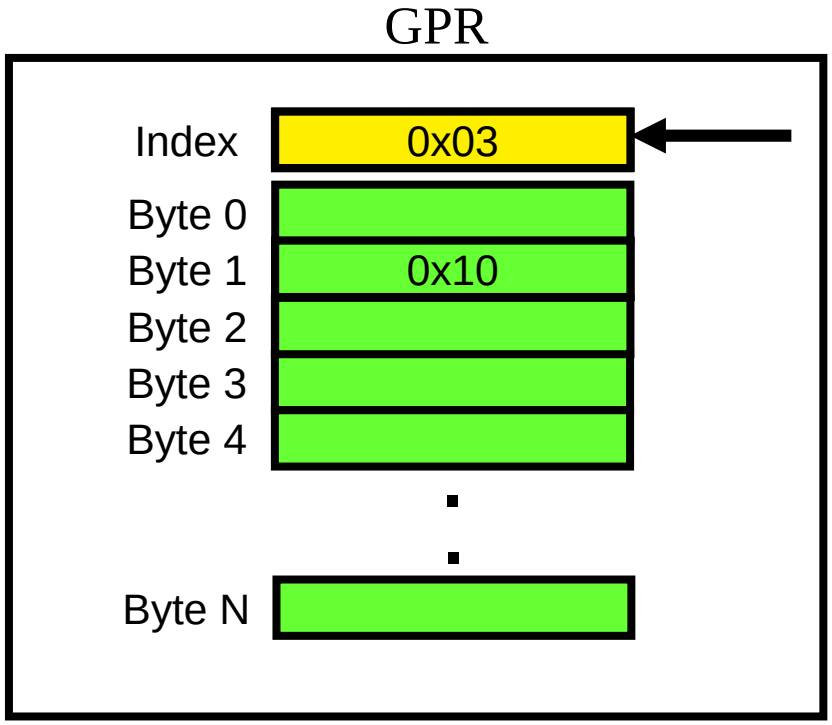


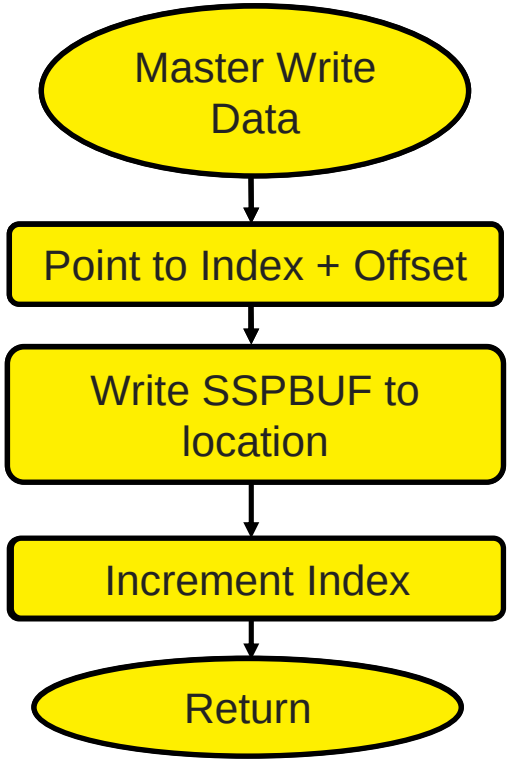


Review: Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

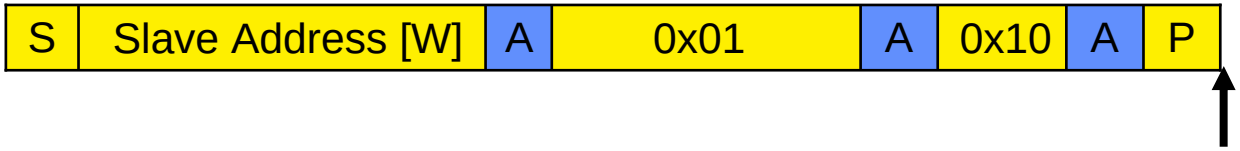


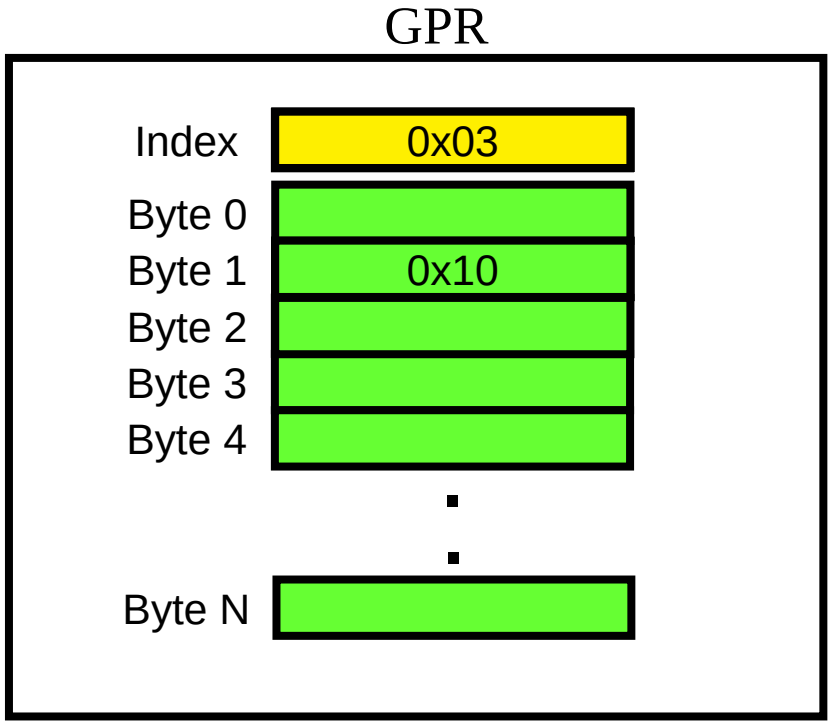


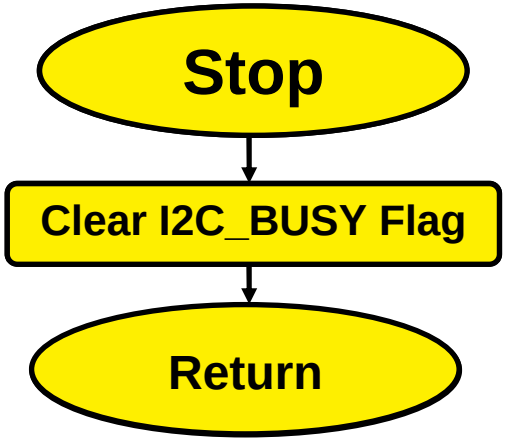


Review: Handling I²C™ Write Messages

- Master writes data to I²C Slave Device

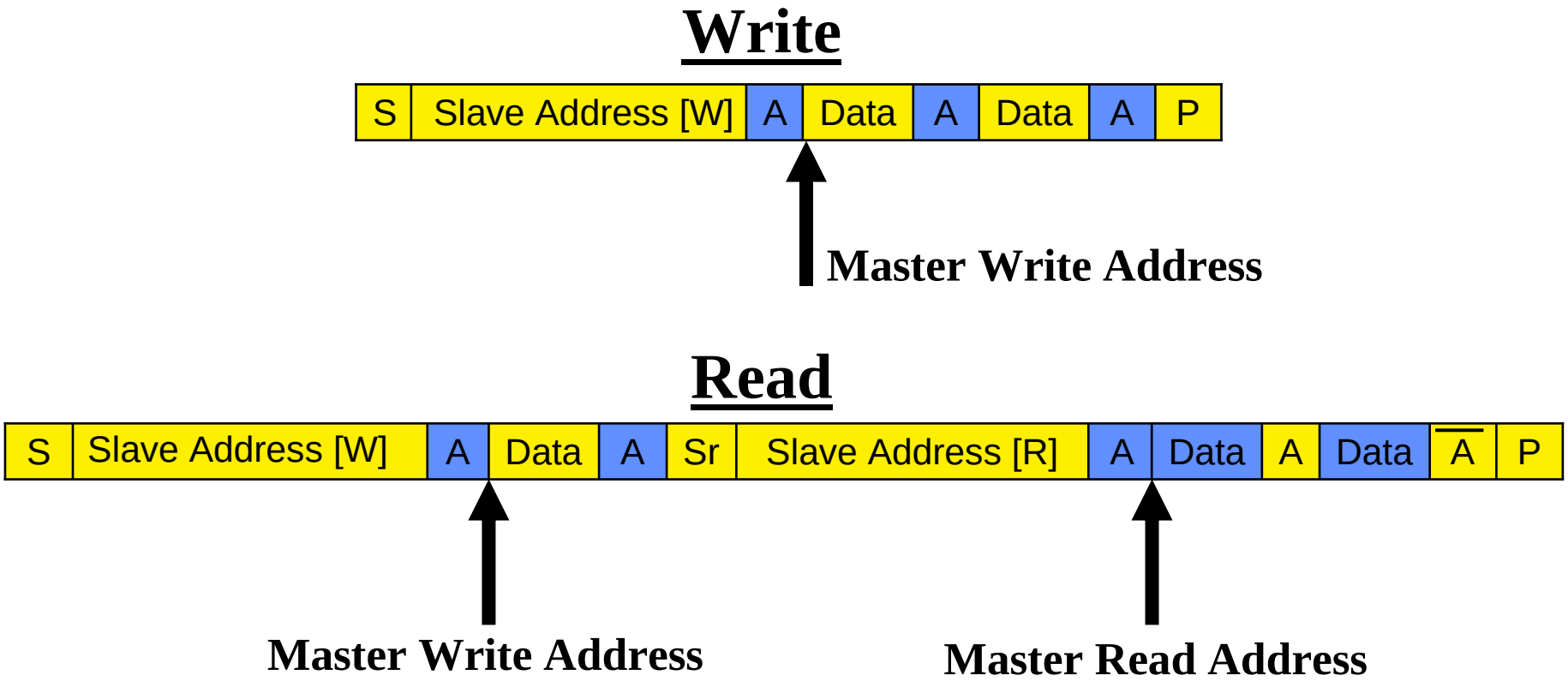






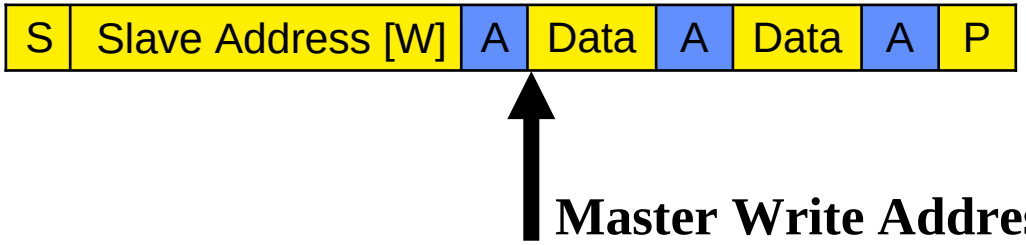
I²C™ Message Format

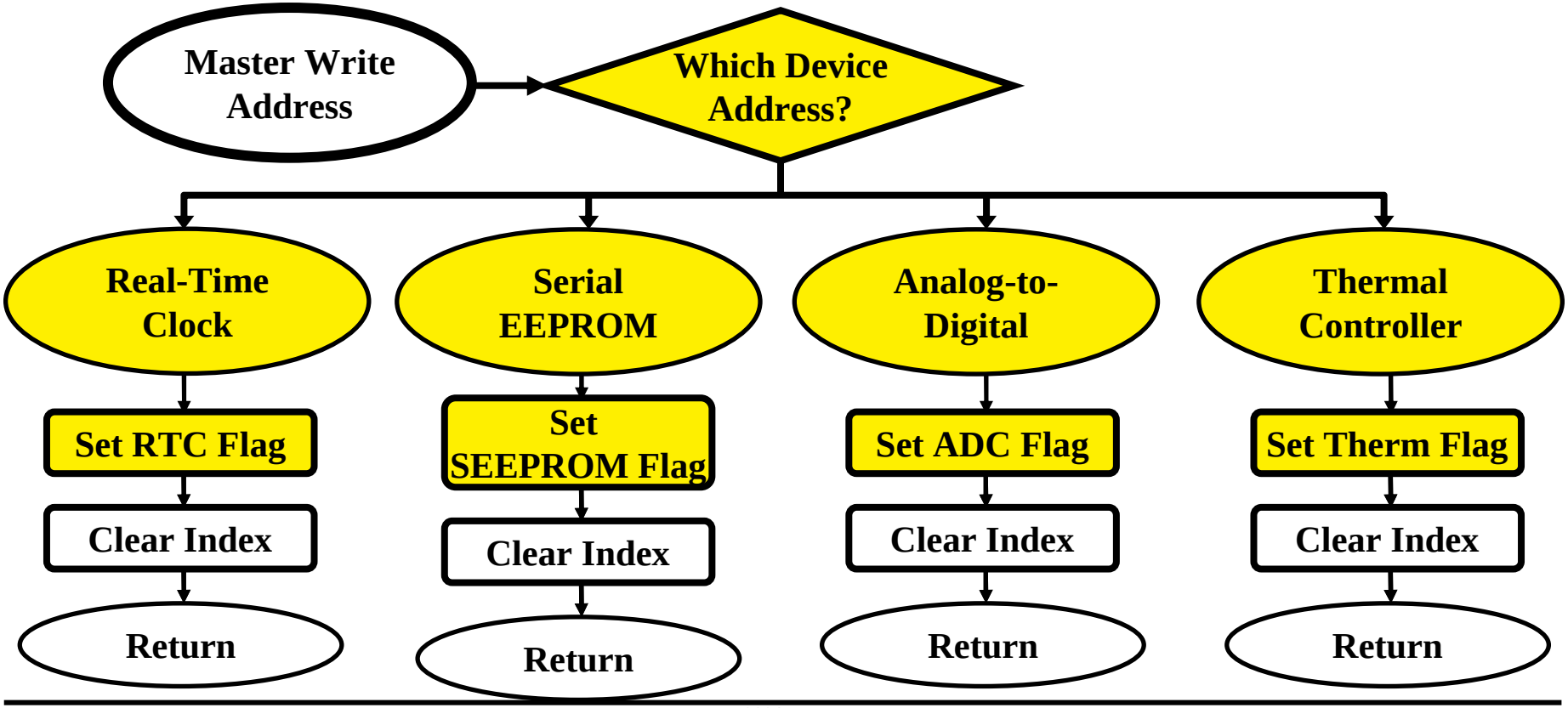
- Locations where I²C slave addresses must be read and corresponding device flag should be set



Master Write Address

Write

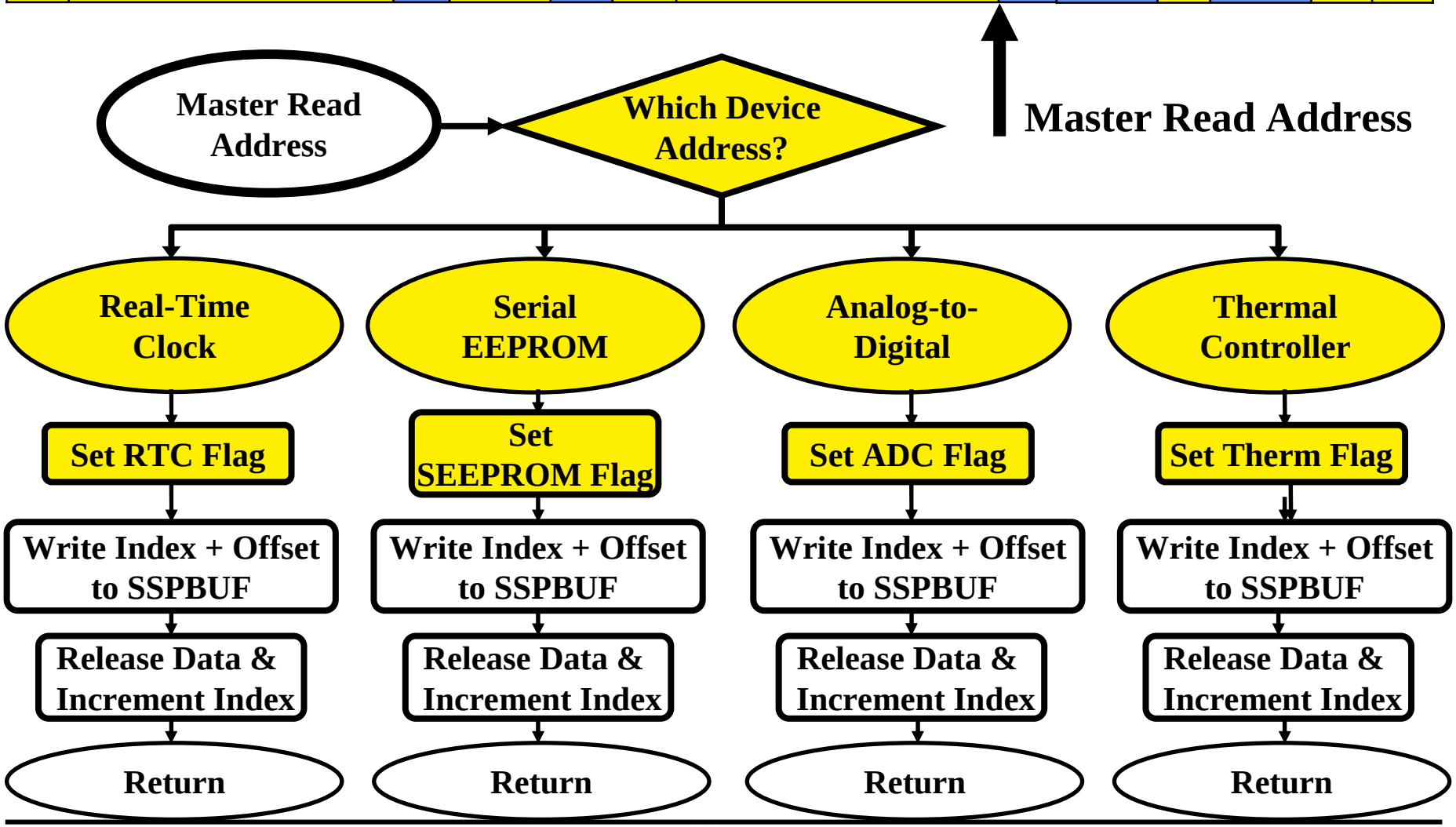




Master Read Address

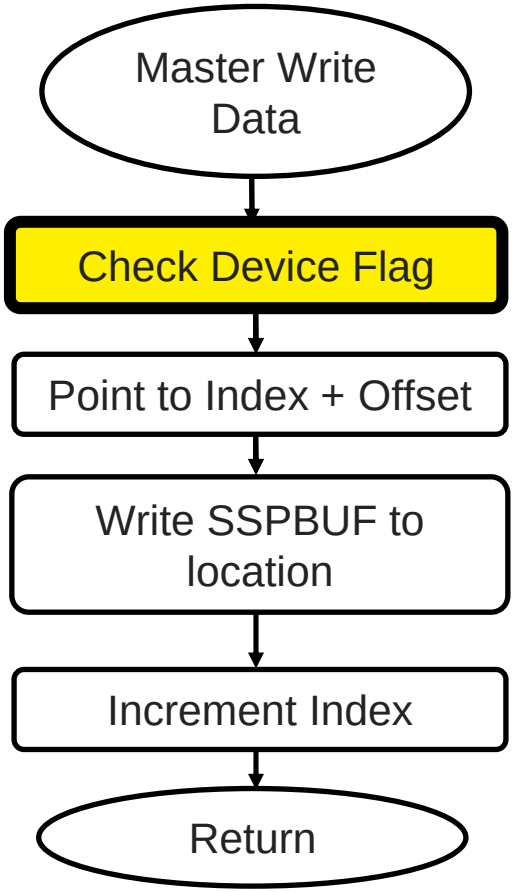
Read

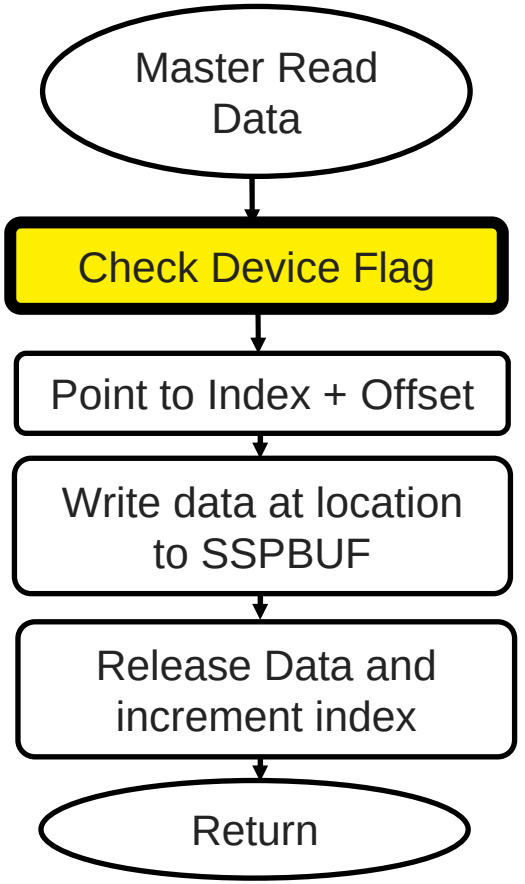
S	Slave Address [W]	A	Data	A	Sr	Slave Address [R]	A	Data	A	Data	$\overline{A}$	P



Master Write/Read Data

- In data states, check device flags to determine which memory to Read/Write





Reviewing so far: I²C™ Integration

- I²C Slave Mode Firmware is modular
- Can be modified to accommodate multiple devices
- Multiple device requirements:
  - Decode device addresses when addresses are received
  - Use flags to indicate which memory to read/write

- Questions?

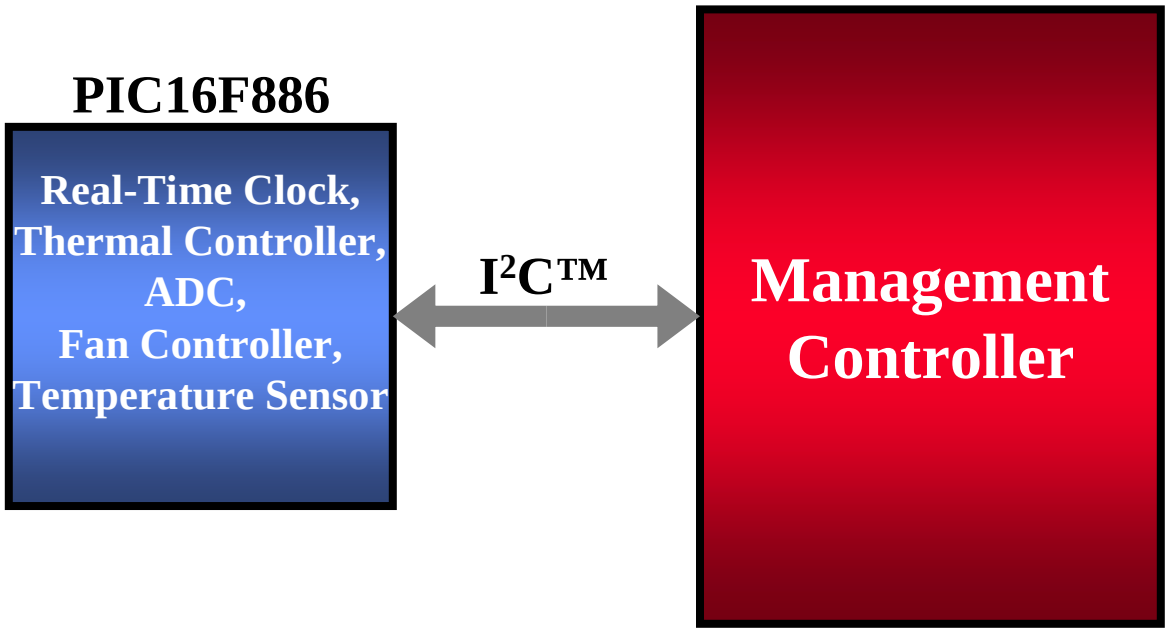
Module Agenda

In the next hour we will discuss…

- Integration Basics
- Integrated I²C™ Firmware
- Firmware Multitasking Considerations
- Summary

Firmware Multitasking Considerations

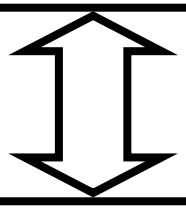
- There are a lot of processes running!
  - Serial Communications
  - Time Keeping
  - Temperature Measurements
  - Fan Speed, Software PWM



Firmware Multitasking Considerations (INTOSC=8MHz)

Interrupt Service Routine

- Fan Speed Measurement (Comparator) **(10 us)**
- Fan PWM Generation (CCP) **(16 us)**
- Real-Time Clock Update (Timer 1) **(17 us)**
- I^2C™ Communications (MSSP) **(120 us)**
- Serial EEPROM Writes (Data EEPROM) **(10 us)**

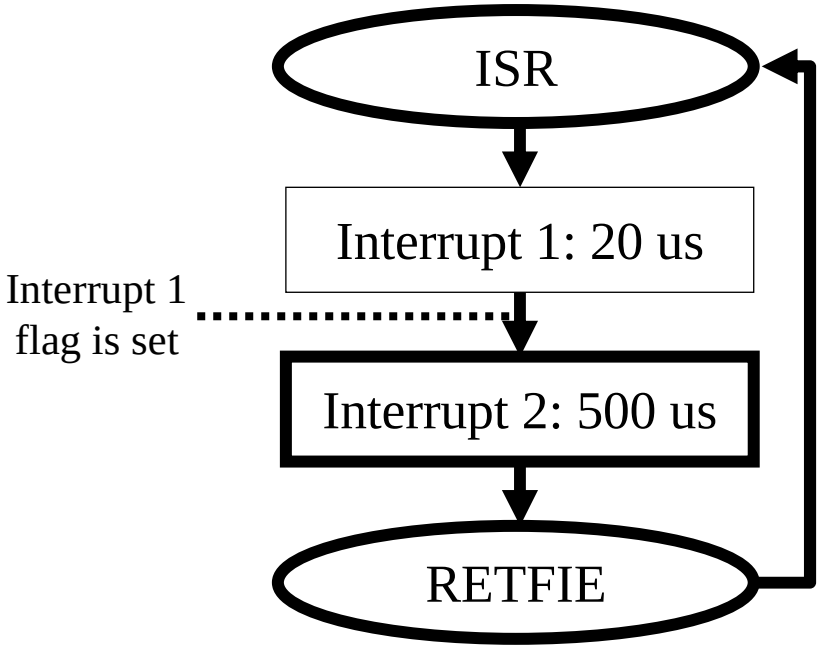


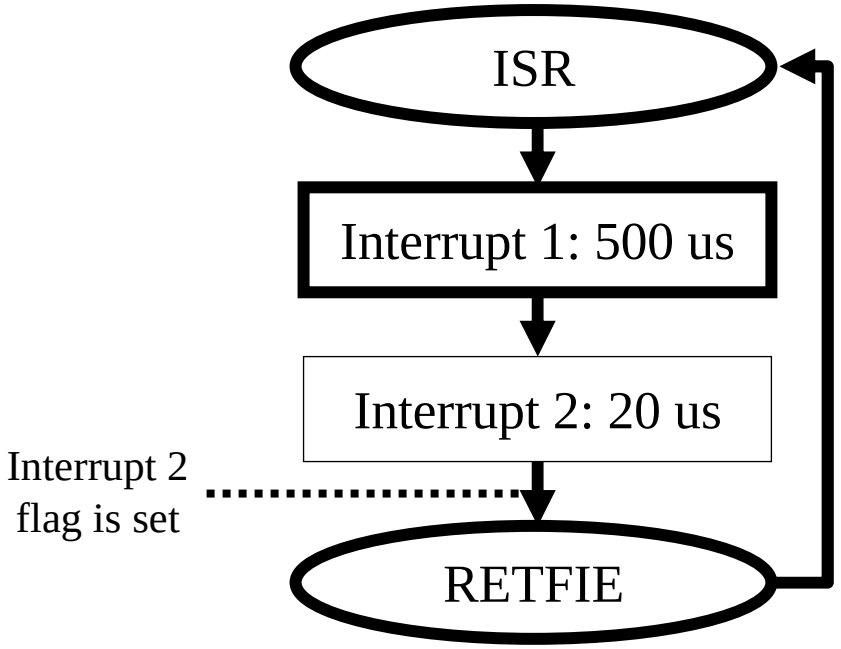
Mainline

- Temperature Measurement **(66 us)**
- PWM Duty Cycle Calculation **(386 us)**
- Initiate Fan Speed Measurement **(10 us)**

ISR Structure

- Does the order you test interrupts matter?
  - No, as long as interrupts are independent.
- The worst case scenario (in bold) shown both cases below is the same





Firmware Multitasking Review

- We chose to…
  - Use interrupts whenever possible
    - Event driven tasks with critical timing
  - Leave processor intensive tasks in Mainline code
    - Math routines and non-time critical tasks (I.E. temperature measurements)
  - Write Interrupt Service code as efficiently (execution time) as possible

Summary

- Integration Basics
- I²C™ Firmware
- Firmware Multitasking Considerations

- Questions?