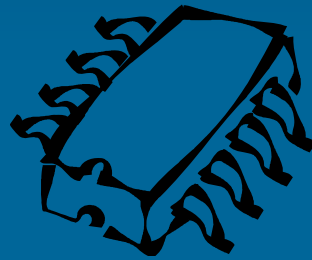


IEEE/CETA Robotics Workshop – Day 4



How to Program the Inputs and Outputs

ACSE/CETA Presentation - 11 November 2011

Dennis Cecic, P. Eng.

Coordinator, Teacher In-Service Program

IEEE Toronto Section

d.cecic@ieee.org



Disclaimer

- The information in this workshop is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor IEEE shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

IEEE/CETA Robotics Workshop

- Day 1. How to Program in C – Day 1
- Day 2. How to Program in C – Day 2
- Day 3. How to Build the Robot Platform
- Day 4. How to Program the Inputs and Outputs
- Day 5. How to Control the Robot

Learning Objectives (Day 4)

- Understand the PIC24 Architecture and Programmer's Model, and how to create embedded C programs for it, using the MPLAB development environment.
- Build the basic electronic control platform for the line-following robot, and understand the trade-offs that have been made in it's design.
- Understand and be able to write a program to drive the DC motors (the "Outputs").
- Understand sensor basics, and be able to modify the program to accept Optical Sensor "Inputs" to detect a line.

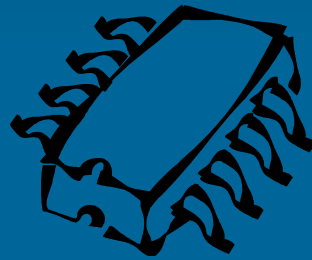
Agenda

- The Robot Platform
- Processing
 - Firmware Development
 - Lab 1 – Wiring the Board
 - 16-bit PIC[®] Architecture & Programmer's Model
 - MPLAB[®] C for PIC24 Compiler Overview and C-Language Extensions for the PIC24
 - Lab 2 – Creating PIC24 C Projects in MPLAB[®]
- Robot Outputs
 - Working with Digital Input and Output Ports
 - Basics of Brushed DC Motors and Their Control
 - PIC24 Timer & Pulse-Width Modulation Peripherals
 - The System Control Loop
 - Lab 3 – Motor Control using the PWM Peripheral

Agenda (continued)

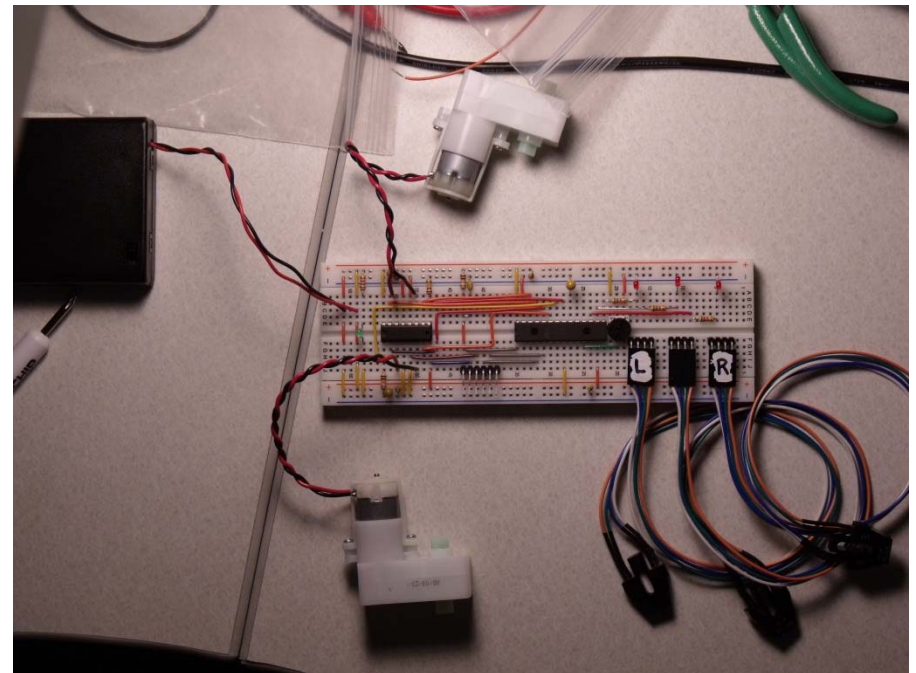
- Robot Inputs
 - What is a Sensor ?
 - Optical Sensors used in Line Following Robots
 - Analog-to-Digital Converter Peripheral
 - Lab 4 –Line Detection Using the ADC Peripheral
- Summary

The Robot Platform

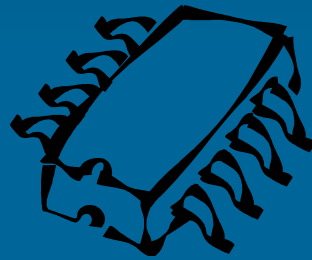


Robot Platform

- Autonomous, Self-contained robot
- Inputs
 - 3-Line sensing Opto emitter/detectors
- Processing
 - PIC24FV Microcontroller
- Outputs
 - 2-GM8 DC Motor
- Simple chassis (not shown)
 - 1-battery pack (4xAA)
 - Solderless Breadboard



Processing

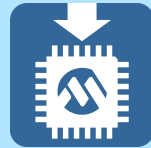


- **Firmware Development**
- **16-bit PIC® Architecture & Programmer's Model**
- **MPLAB® C for PIC24 Compiler Overview and C Language Extensions for the PIC24**

Firmware Development Platform



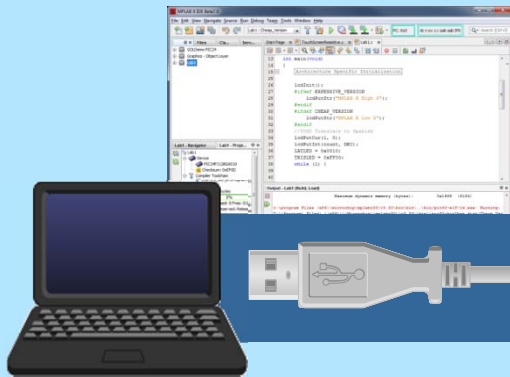
Integrated
Development
Environment



Programmer
Debugger



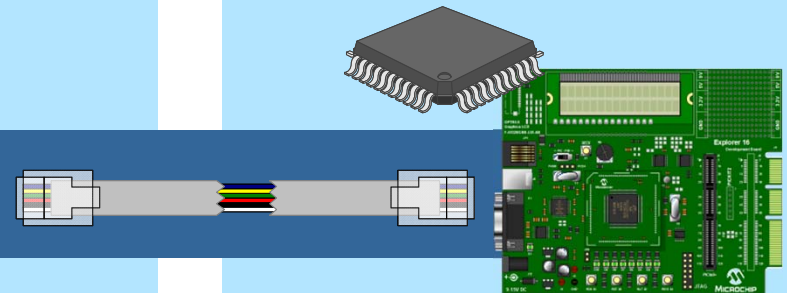
Target
Hardware



MPLAB® 8 IDE
MPLAB C for PIC24F
Compiler/Assembler

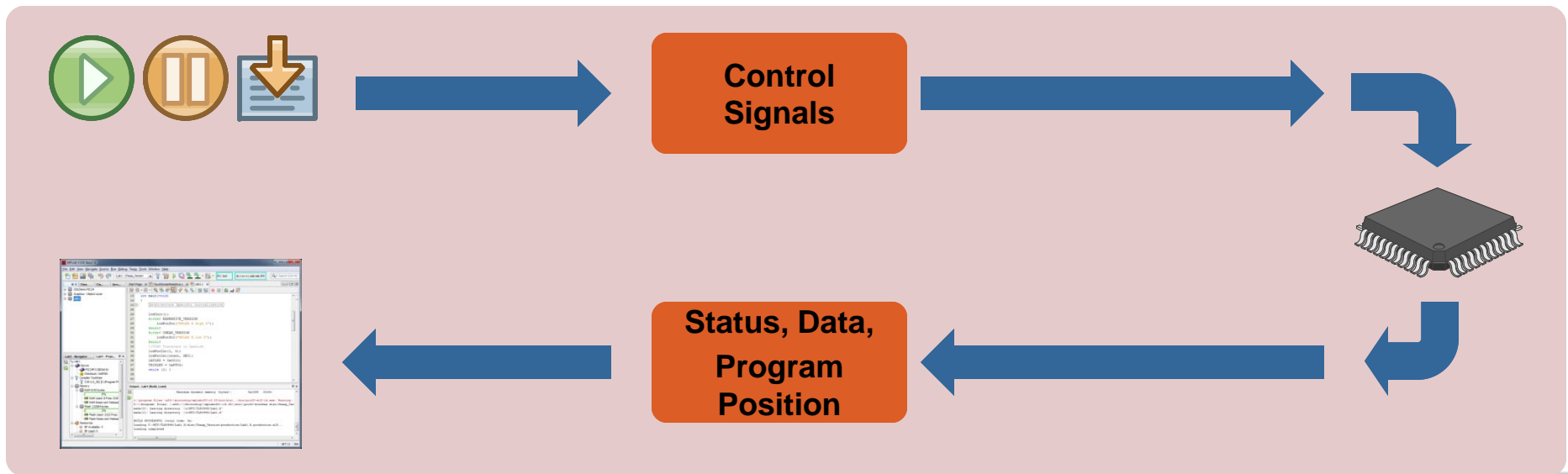


MPLAB PICKit™ 3



Your hardware

What are Programmer Debuggers?



PICKit™ 3



- USB (Full speed)
- Built-in over-voltage/short circuit monitor
- Firmware upgradable
- Supports 2.0v to 6.0v
- Program up to 512K byte flash with the Programmer-to-Go

PG164130 – PICKit™3 Programmer/Debugger - \$44.95



MPLAB® 8 IDE and Components

MPLAB®

Integrated Development Environment

**Programmer's
Editor**

**Source Level
Debugger**

**Project
Manager**

Languages

Simulators

Emulators

Programmers

**Assemblers
Linkers
Librarians**

**MPLAB®
SIM Simulator**

PICKit™ 2 / PICKit™ 3

MPLAB® ICD 2

MPLAB® ICD 3

MPLAB® REAL ICE™

**MPLAB®
& HI-TECH
C Compilers**

3rd Party

3rd Party

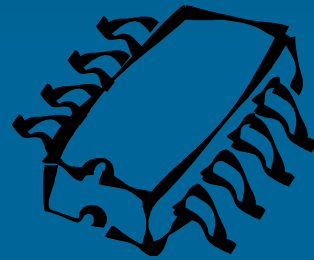
MPLAB® PM 3

MPLAB® 8 IDE Installation

- Supported on Windows XP, Vista, 7 (32- and 64-bit)
- Includes C32 and PICC lite version compilers
 - Does NOT include PIC24F compiler.
 - www.microchip.com/mplab

Lab 1

Wiring The Board



Lab 1

Wiring the Board



Purpose

To understand the basic fabrication procedure for the line-follower robot control circuitry. The completed circuit will be tested using a pre-built .hex file. You should be able to control both motor's speed using the potentiometer, as well as have the LEDs turn on when a line is detected.



Procedure

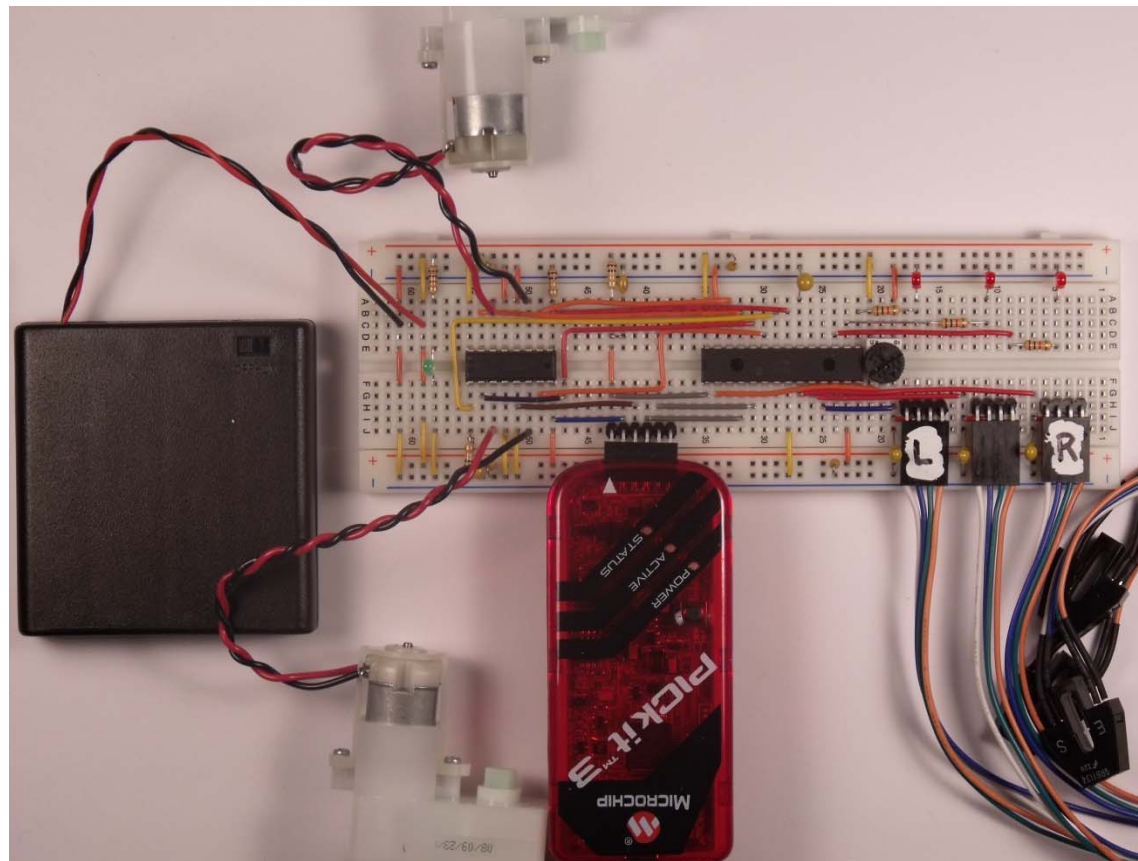
Follow the instructions in the lab manual

Lab 1

Wiring the Board



Results



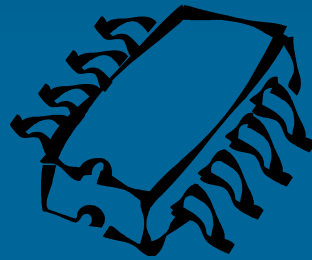
Lab 1

Wiring the Board

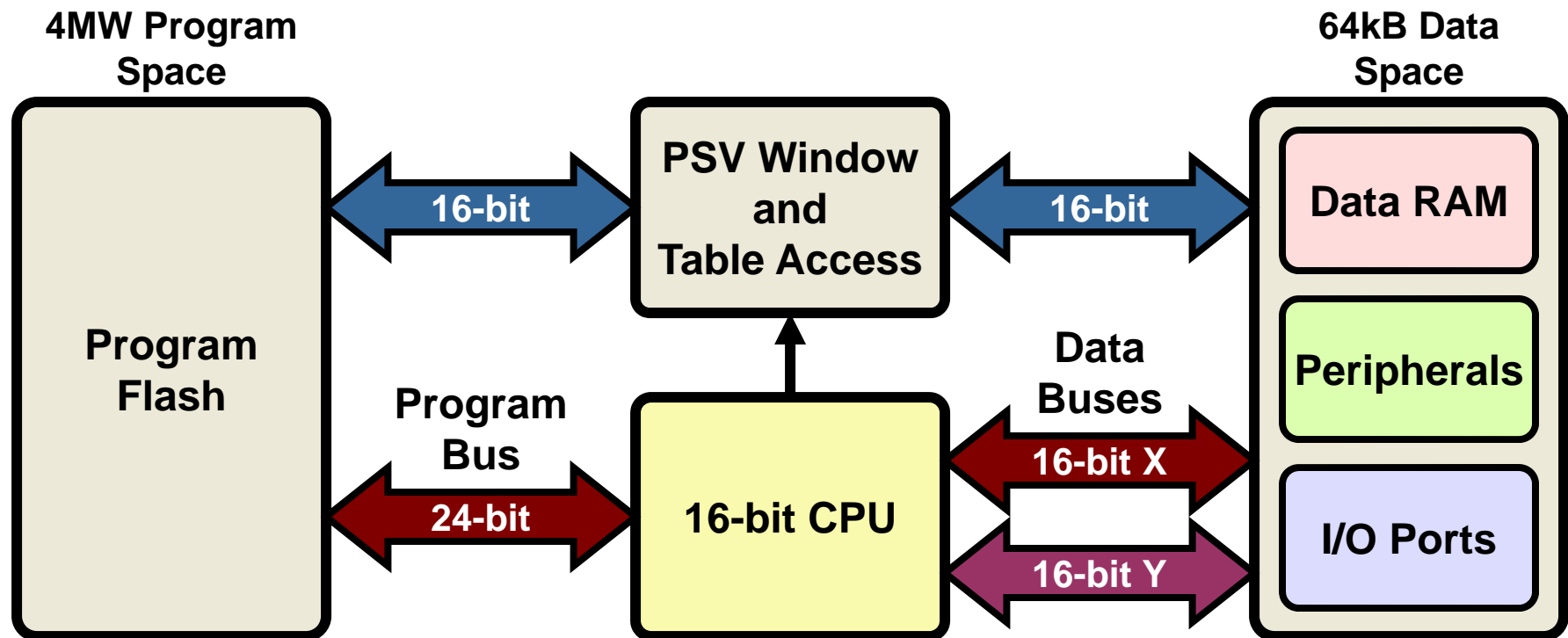


Conclusions

16-bit PIC[®] Architecture and Programmer's Model



Simplified Block Diagram 16-bit PIC[®] Architecture



The Y data bus is only available on the dsPIC30 and dsPIC33 families.

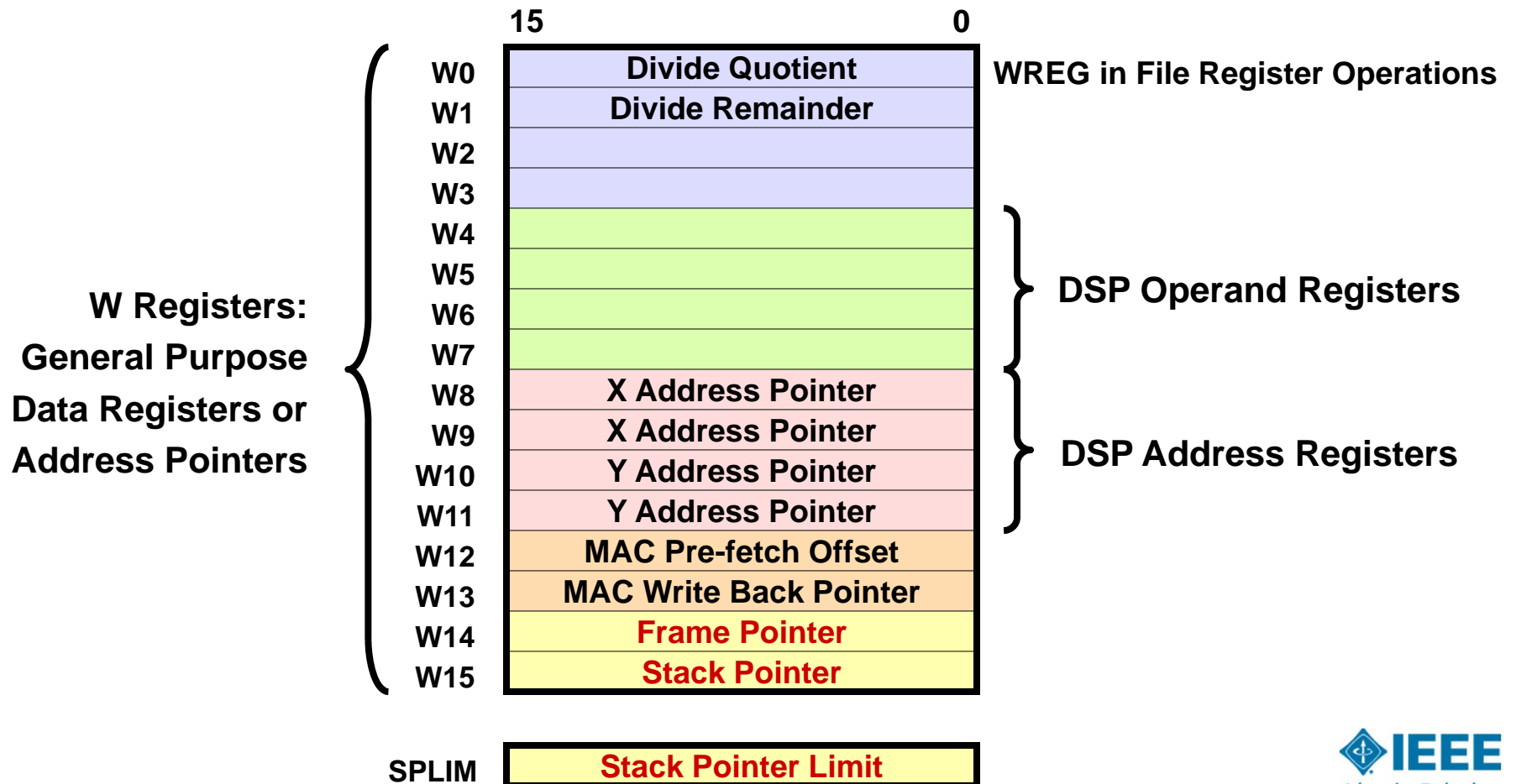


PIC24[®] Architecture

- Harvard Architecture
 - 24-bit wide instruction word
 - 16-bit wide data byte
- Instruction Set Features
 - 76 instructions
 - Most instructions are one cycle
 - Most instructions are one word
 - 2 clocks per instruction cycle
 - 32 MHz (16 MIPS) - PIC24F
 - 80 MHz (40 MIPS) - PIC24H
- 16 x 16-bit Working (W) Registers
- Software stack
- Program Space Visibility
- Hardware Multiply and Divide support

Programmer's Model

16-bit PIC[®] Architecture



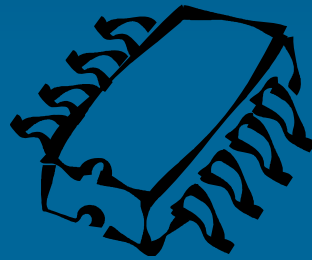
PIC24[®] Peripherals

- 16 and 32-bit timers
- Input capture
- Output compare/PWM
- SPI with deep buffer
- I²C™ with address masking
- UART
- Real-Time Clock and Calendar (RTCC)
- Parallel Master Port (PMP)
- Charge Time Measurement Unit (CTMU)
- Comparators
- Programmable CRC Engine
- Analog-to-Digital Converter
 - 10-bit, 500 ksps (PIC24F)
 - 10-bit, 1.1 Msps or 12-bit, 500 ksps (PIC24H)
- USB Host/Device (PIC24F)
- DMA (PIC24H)
 - CPU-independent data transfers
- Enhanced CAN (PIC24H)
 - CAN 1.2, 2.0A and 2.0B

PIC24FJ vs. PIC24FV vs. PIC24HJ (28-pin DIP)

PIC24FJ32GA002	PIC24FV32KA302	PIC24HJ32GP302
32kB Flash, 8kB Ram	32kB Flash, 2kB Ram, 512 Bytes EEPROM	32kB Flash, 4kB Ram
VDD: 2.0-3.6V	VDD: 2.0-5.5V or 1.8-3.6V	VDD: 3.0-3.6V
8MHz internal oscillator	8MHz internal oscillator	7.37MHz internal oscillator
16MIPs top speed	16MIPs top speed	40MIPs top speed
Fixed 10-bit ADC	Configurable 10/12-bit ADC	Configurable 10/12-bit ADC
-	-	Has DMA Engine with 1kB DMA RAM
Digital I/O can source/sink 16mA	Digital I/O can source/sink 25mA	Digital I/O can source/sink 4mA
-	Lowest run/sleep currents	-

MPLAB[®] C for PIC24 Compiler Overview



MPLAB[®] C for PIC24 and dsPIC

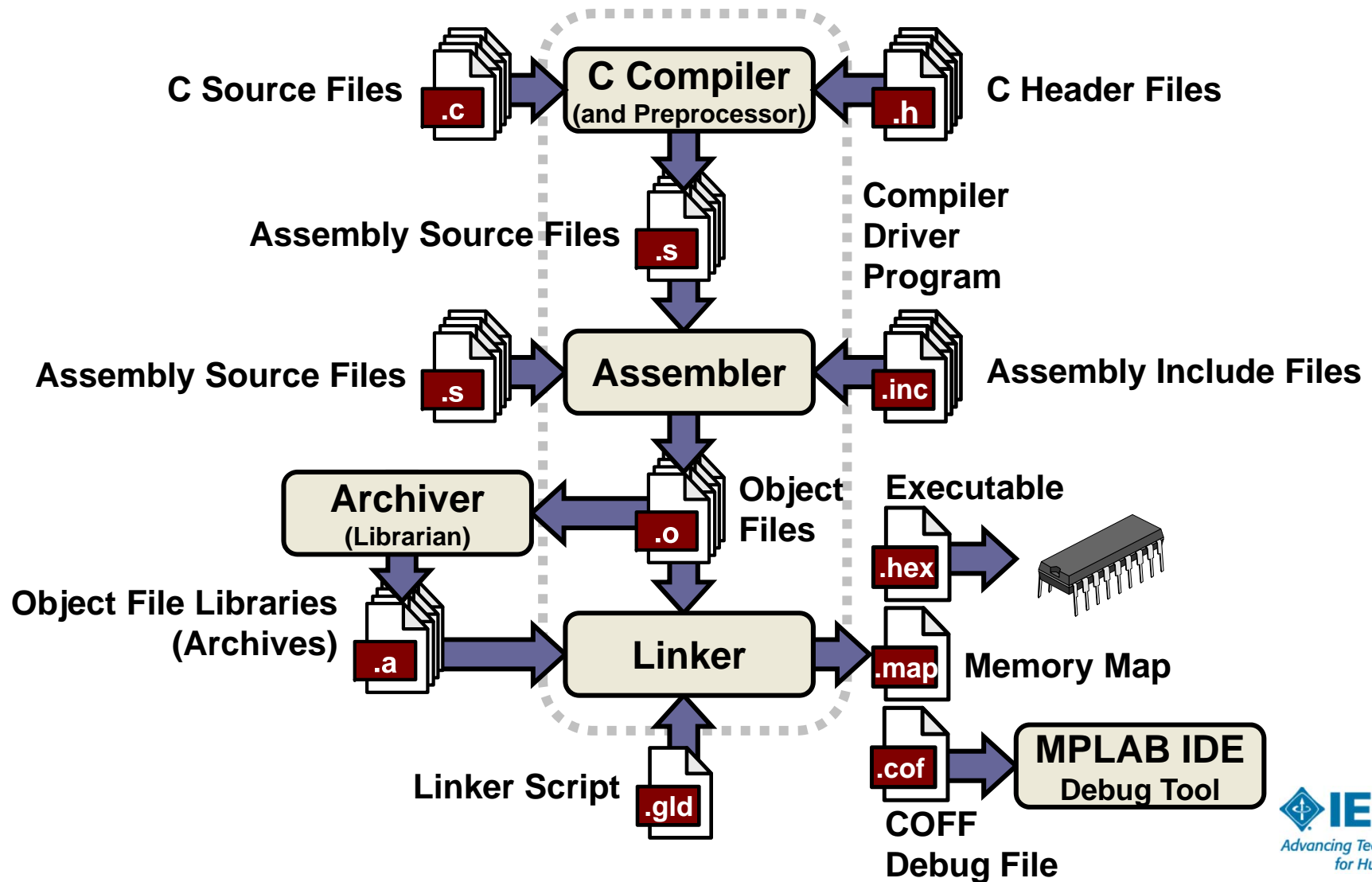
Overview

- ANSI x3.1989 compliant
- Optimizing compiler
- Includes language extensions for Microchip's 16-bit MCUs
- Ported from GCC (GNU) compiler from the Free Software Foundation
- Works as a component of MPLAB[®] IDE
- Standard installation path:
 - **C:\Program Files\Microchip\mplabc30**



MPLAB[®] C for PIC24 and dsPIC (Lite) is available for free from www.microchip.com/mplabc

Development Tools Data Flow



MPLAB[®] C for PIC24 and dsPIC

Header Files

Header files are included as part of the MPLAB C30 installation and are located in the following directory:

`C:\Program Files\Microchip\mplabc30\<version>\Support\PIC24F\h`



`p24FV32KA302.h`

- One header file per device:
 - Provides access to registers as C variables
 - Defines labels for bit manipulation
 - Defines macros to utilize instructions not normally accessible from C
 - Copy it to your project folder

MPLAB[®] C for PIC24 and dsPIC

Linker Scripts

Linker Script files are included as part of the MPLAB C30 installation and are located in the following directory:

`C:\Program`

`Files\Microchip\mplabc30\<version>\Support\PIC24F\gld`



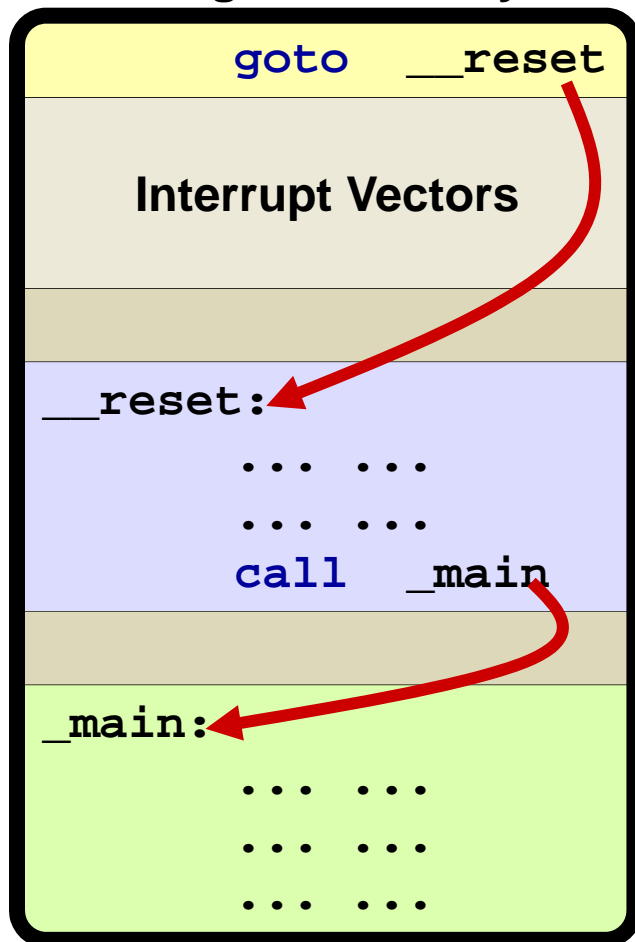
`p24FV32KA302.gld`

- One linker script file per device:
 - Defines memory sections and boundaries
 - Associates ISR names with interrupt vectors
 - Equates register variables with addresses
 - Copy it to your project folder


MPLAB[®] C for PIC24 and dsPIC

Startup and Initialization (“The only assembly”)

Program Memory



← **Reset Vector** (Address 0x000000)
Populated automatically by LINK30
Linker
Calls runtime environment setup code in
`crt0.o` (`__reset` label)

←  **`crt0.o` or `crt1.o`** (from `libpic30.a`)
C Runtime Environment Setup Code
Inserted automatically by LINK30 Linker
(Source files: `crt0.s` and `crt1.s`)

←  **`main.c`**

Your C code's `main()` routine.
Included by you in your project and
placed in memory by LINK30 Linker

MPLAB[®] C for PIC24 and dsPIC

Data Representation

- Multibyte quantities are stored in "little endian" format:
 - LSB is stored at lowest address
 - LSb is stored at lowest numbered bit position

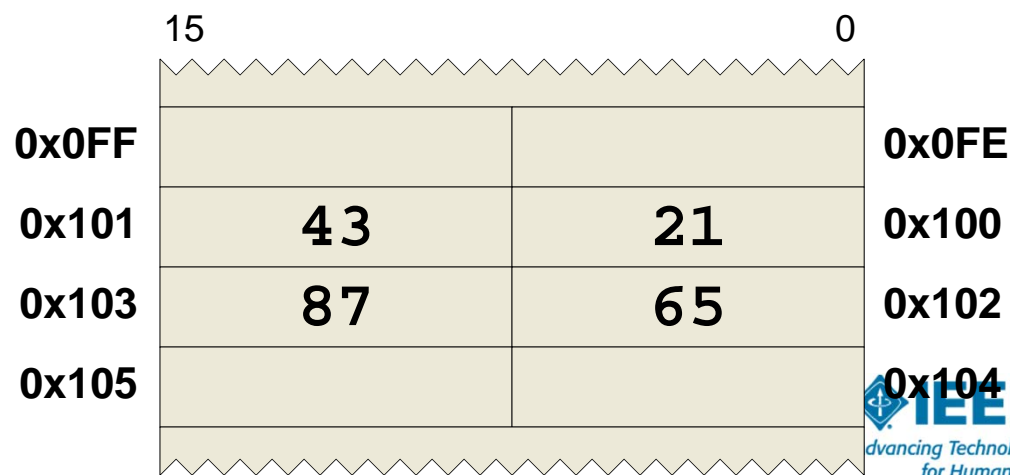
How the value 0x87654321 is stored

In Working Registers W4 & W5

W4	4321
W5	8765

In Data Memory (RAM) @ 0x100

	15		0
0x0FF			0x0FE
0x101	43	21	0x100
0x103	87	65	0x102
0x105			0x104



MPLAB[®] C for PIC24 and dsPIC

Integer Data Types

Type	Bits	Min	Max
<code>char</code> , signed <code>char</code>	8	-128	127
<code>unsigned char</code>	8	0	255
<code>short</code> , signed <code>short</code>	16	-32768	32767
<code>unsigned short</code>	16	0	65535
<code>int</code> , signed <code>int</code>	16	-32768	32767
<code>unsigned int</code>	16	0	65535
<code>long</code> , signed <code>long</code>	32	-2^{31}	$2^{31} - 1$
<code>unsigned long</code>	32	0	$2^{32} - 1$
<code>long long</code> , signed <code>long long</code>	64	-2^{63}	$2^{63} - 1$
<code>unsigned long long</code>	64	0	$2^{64} - 1$

MPLAB[®] C for PIC24 and dsPIC

Floating Point Data Types

Type	Bits	E Min	E Max	N Min	N Max
float	32	-126	127	2^{-126}	2^{127}
double *	32	-126	127	2^{-126}	2^{127}
long double	64	-1022	1023	2^{-1022}	2^{1023}

E = Exponent

N = Normalized (approximate)

* double is equivalent to long double if `-fno-shortt-double` command line option is used

Type	Bits	Min	Max
float	32	1.175494×10^{-38}	$3.40282346 \times 10^{38}$
double *	32	1.175494×10^{-38}	$3.40282346 \times 10^{38}$
long double	64	$2.22507385 \times 10^{-308}$	$1.79769313 \times 10^{308}$

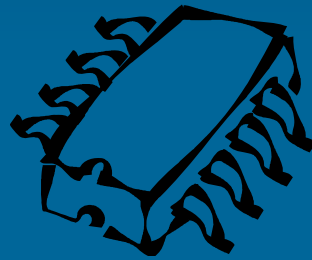
Technology
for Humanity

MPLAB[®] C for PIC24 and dsPIC

Pointers

- All pointers are 16-bits in size, regardless of whether they are data pointers or program pointers
- Constants in flash viewed via PSV
- Jump tables sometimes used for program pointers (handles)

MPLAB[®] C for PIC24 C-Language Extensions



MPLAB[®] C for PIC24 and dsPIC

C Language Extensions

- Program Memory Variables
- Special Function Register (SFR) Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Special Function Register (SFR) Access

- Special Function Registers can be accessed directly in C code:
 - Word access: `PORTA = 0xFFEC;`
 - `SFRNAMEbits.BITNAME` can be used to access bits and bit fields
 - e.g. `PORTAbits.RA1 = 1;` or
`IPC0bits.INT0IP = 0;`
 - `_SFRNAME` and `_BITNAME` can also be used to reference an SFR or bit
 - e.g. `_PORTA` or `_RA1`
- Must use standard header and linker script files

MPLAB[®] C for PIC24 and dsPIC

C Language Extensions

- Program Memory Variables
- Special Function Register (SFR) Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Inline Assembly

Simple Form – Single Line

Syntax

```
asm ( "instruction" )
```

- Only a single string can be passed
- *Generally* used for instructions that take no operands or take immediate operands
- For ANSI compliance use `__asm__` instead of `asm`

Examples

```
asm ( "nop" );           // One Cycle Delay
asm ( "clrwdt" );       // Clear Watchdog Timer
asm ( "pwrsav #0" );    // Sleep mode
asm ( "pwrsav #1" );    // Idle mode
```

Inline Assembly

Simple Form – Multiple Lines

Syntax

```
asm ( "instruction_1"  
      "instruction_2"  
      ...  
      "instruction_n" )
```

- Only one "asm" keyword is required
- Include each instruction within double quotes
- Put each instruction on a separate line for better readability
- One set of parentheses encloses the entire list of instructions within the asm statement

Inline Assembly Macros

Inline Assembly Macro Definitions

```
#define Nop()      {__asm__ volatile ("nop");}  
#define ClrWdt() {__asm__ volatile ("clrwdt");}  
#define Sleep()  {__asm__ volatile ("pwrsav #0");}  
#define Idle()   {__asm__ volatile ("pwrsav #1");}
```

- Provide an easy way to execute specific assembly language instructions that have no C equivalent (builtins are even better...)

Examples

```
Nop();           // Insert nop instruction  
ClrWdt();       // Clear the watchdog timer  
Sleep();        // Enter SLEEP mode  
Idle();         // Enter IDLE mode
```

MPLAB[®] C for PIC24 and dsPIC

C Language Extensions

- Program Memory Variables
- Special Function Register (SFR) Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Configuration Bits

Definition

Configuration Bits - Configuration bits are special bits in registers that may only be modified at program time. They control setup of "permanent" features of the microcontroller that are generally set once for the life of the application.

- Configures features such as:
 - Oscillator type
 - Watchdog timer
 - Code protection
- See "Special Features of the CPU" in data sheet for more details

How to modify Configuration Bits

- Must be done in code
- Method varies by architecture & compiler
- See device specific header file for syntax and example (e.g. `p24FV32KA302.h`)

Example

PIC24 and dsPIC[®] DSC

```
#include <p24FV32KA302.h>
```

```
_FOSCSEL(FNOSC_FRC & IESO_OFF)
```



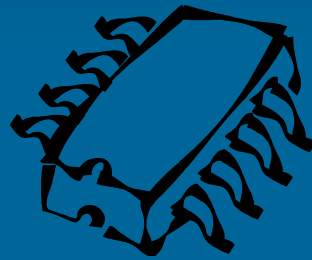
_CONFIG must be
somewhere after
inclusion of header

MPLAB[®] C for PIC24 and dsPIC

- Documentation:
 - Compiler User's Guide
 - `C:\Program Files\Microchip\mplabc30\<version>\docs\hlpMPLABC30.chm`
 - Assembler
 - `C:\Program Files\Microchip\mplabc30\<version>\docs\hlpMPLABASM30.chm`
 - Linker + Librarian
 - `C:\Program Files\Microchip\mplabc30\<version>\docs\hlpMPLABLINK30.chm`
 - Libraries
 - `C:\Program Files\Microchip\mplabc30\<version>\docs\hlpLib30.chm`

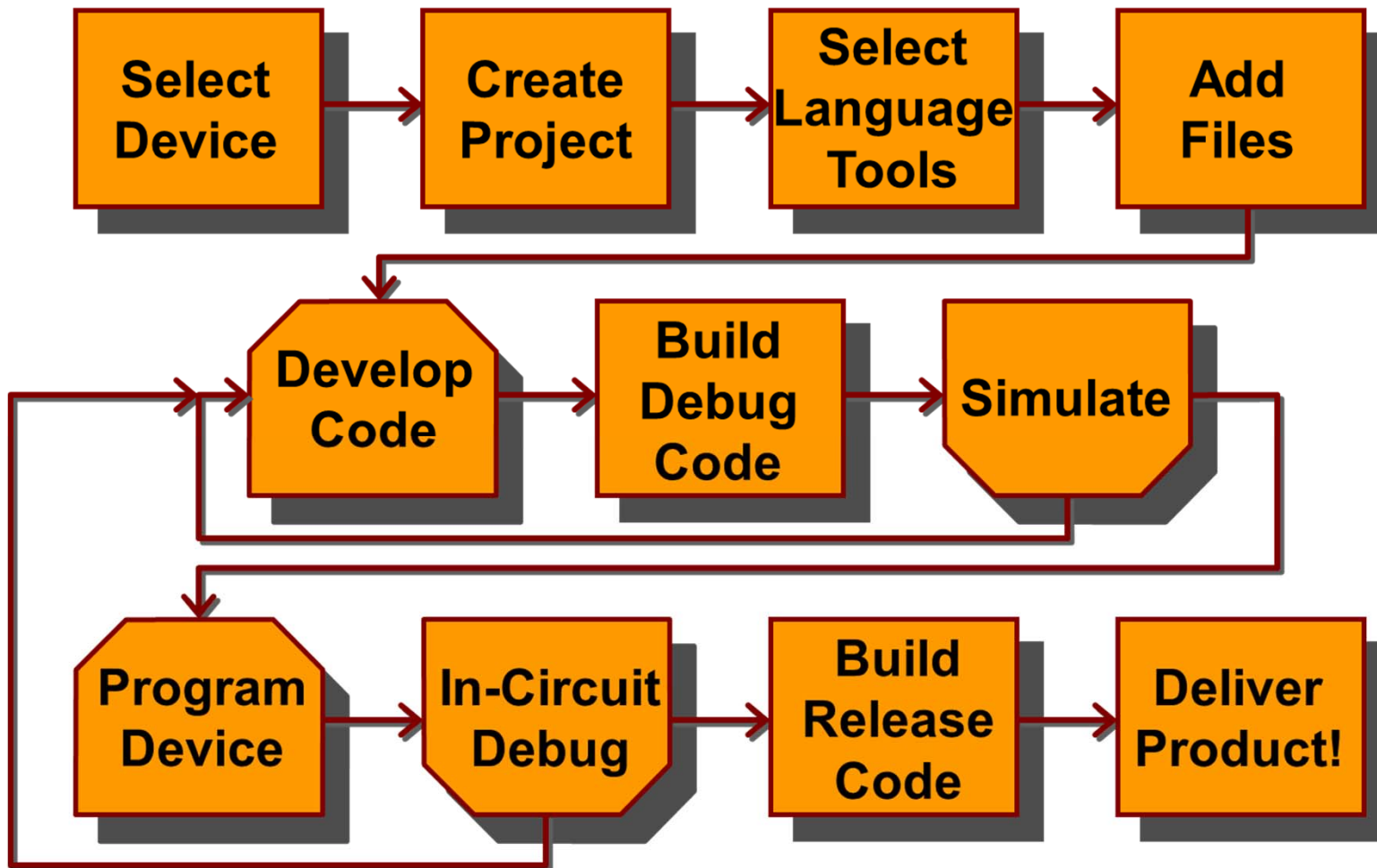
Lab 2

Creating PIC24 C Projects in MPLAB®



Working with MPLAB

- The Basic Development Process -



Lab 2

Creating PIC24 C Projects in MPLAB®



Purpose

The purpose of this lab is to illustrate the steps required to create a C-based project within the MPLAB Integrated Development Environment. You will learn how to select the compiler as the build tool, which files must be included in your project, and finally, how to build, download and run your program on the Robot Platform.



Procedure

Follow the instructions in the lab manual

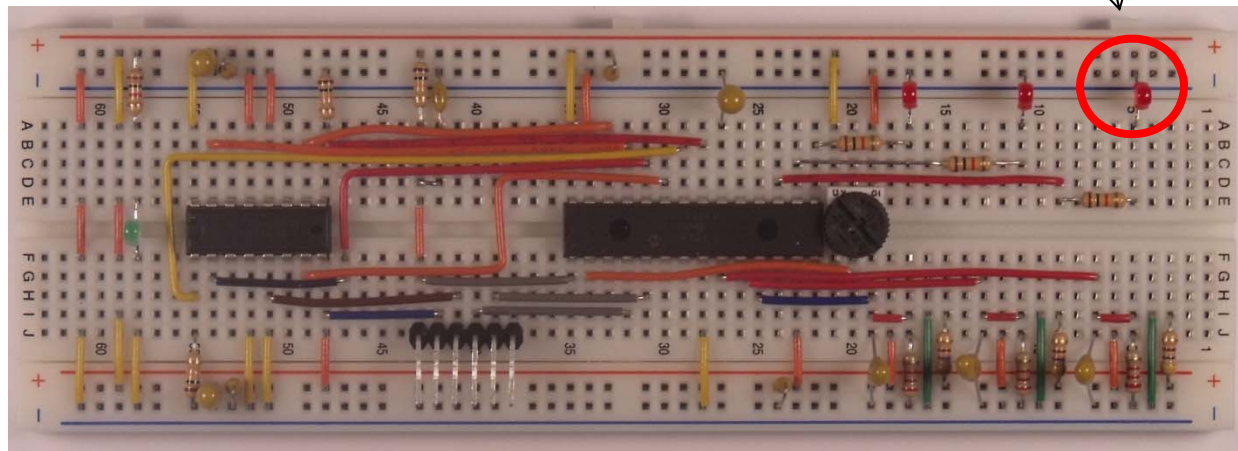
Lab 2

Creating PIC24 C Projects in MPLAB®



Results

LED Blinks!



Lab 2

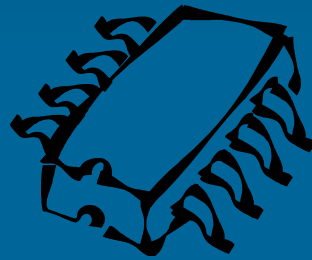
Creating PIC24 C Projects in MPLAB®



Conclusions

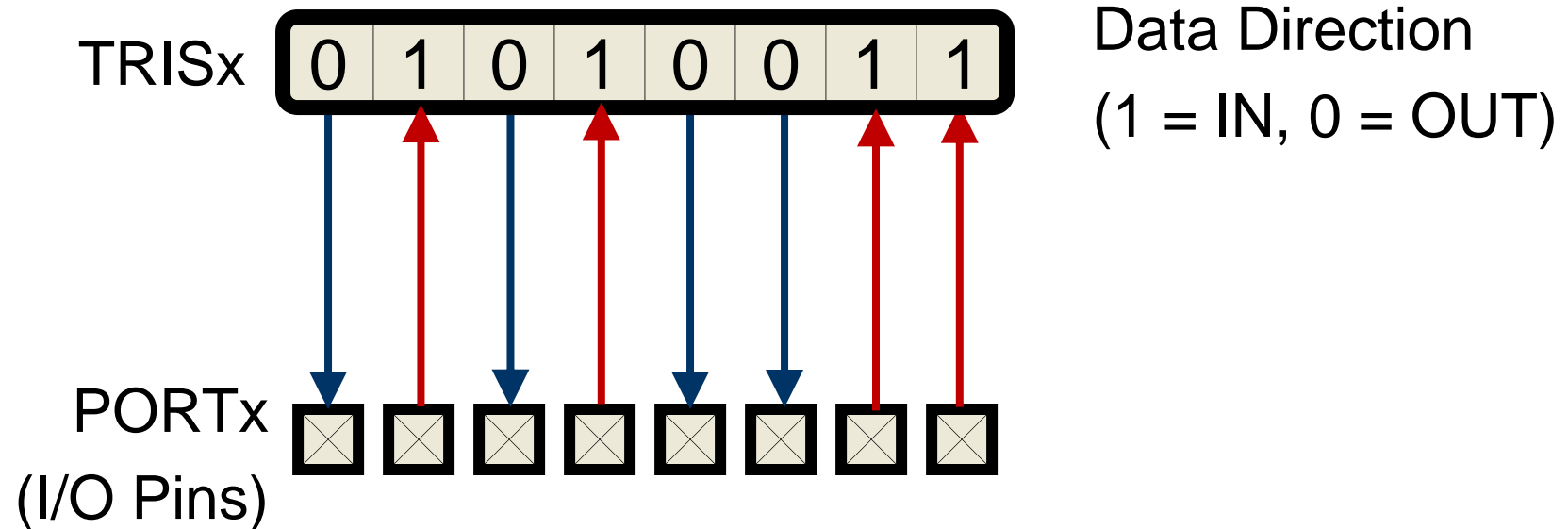
- Minimum steps to setup a C based project:
 - Follow standard MPLAB® project setup steps
 - Select the device
 - Select C30 as the build tool
 - #include device header file in source code
 - Add configuration bits setup in code


Robot Outputs



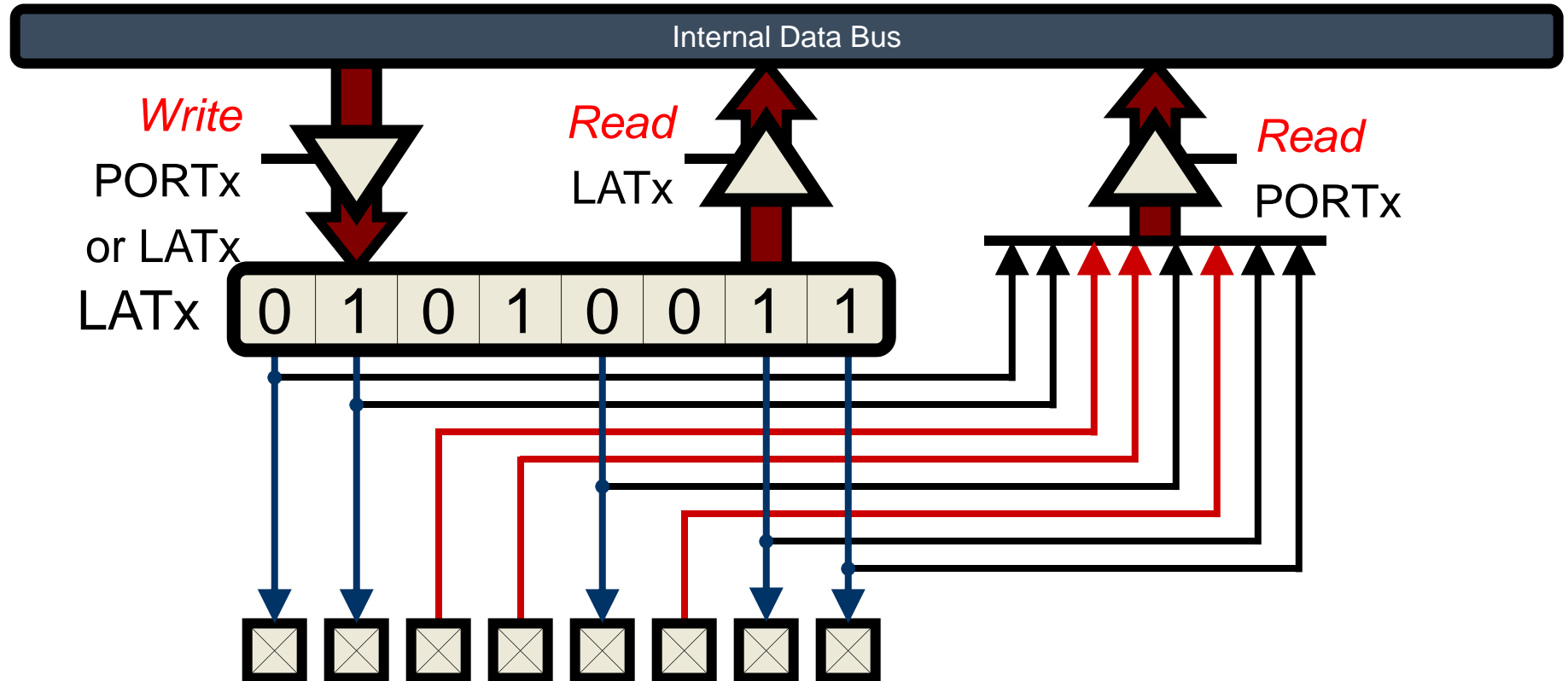
- **Working with Digital Input and Output Ports**
- **Basics of Brushed DC Motors and Their Control**
- **PIC24 Timer & Pulse-Width Modulation Peripherals**
- **The System Control Loop**

TRIS (Data Direction) Registers



 Pins are all inputs by default (TRIS = 0xFFFF)

PORT and LAT Registers



When to use LAT reg?

- For bit (Read-Modify-Write) operations on port pins, use the LATch rather than PORTx:
 - In assembly,
 - `BSET LATA, #0`
 - `BSET LATA, #1`

 - In C,
 - `LATAbits.LATA0 = 0 ;`
 - `LATAbits.LATA1 = 1 ;`

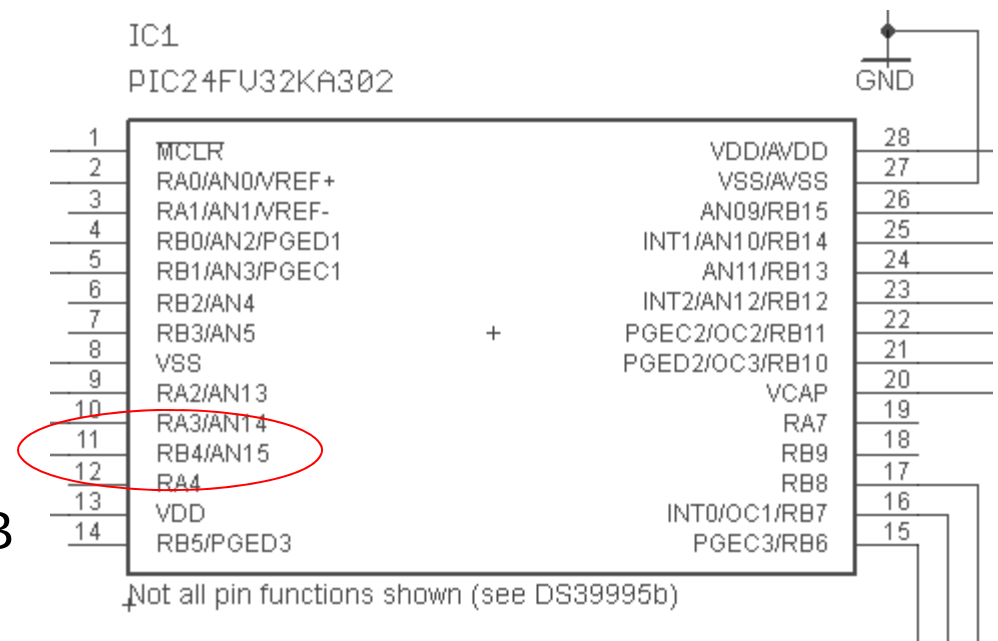
Summary: Digital I/O Ports

Working With TRIS, PORT and LAT Registers

1 Initialize Output Latches to Known State	Write to: LAT <code>LATx = 0;</code> <code>LATxbits.LATxn = 0;</code> <code>LATxn = 0;</code>
2 Configure Data Direction of Pins	Write to: TRIS <code>TRISx = 0x0023;</code> <code>TRISxbits.TRISxn = 1;</code> <code>TRISxn = 1;</code>
3a Write to outputs (bit, byte or word op)	Write to: LAT <code>LATx = 0x00F0;</code> <code>LATxbits.LATxn = 0;</code> <code>LATxn = 0;</code>
3b Read from inputs (bit, byte or word op)	Read from: PORT <code>myVar = PORTx;</code> <code>myVar = PORTxbits.Rxn;</code> <code>myVar = Rxn;</code>

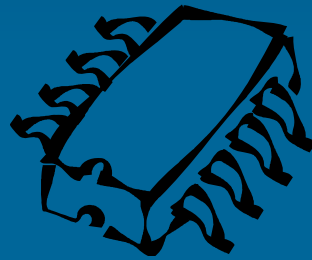
Dealing with the Analog Input Pins

- I/O pins that share functionality with Analog peripheral (ADC or Comparator input) default as “analog” inputs on reset.
- Before you can use it as a digital input, you must program the ANSA, ANSB or ANSC registers



```
ANSBbits.ANSB4 = 0; // RB4/AN15 is a Digital input  
ANSBbits.ANSB4 = 1; // RB4/AN15 is an Analog input
```

Basics of Brushed DC Motors and Their Control



Agenda

- Basic Motor Theory
- Brushed DC Motor Components
- Motor Equivalent Circuit
- Important Motor Parameters
- Driving Miss. Motor
 - 4 Modes of a Motor
 - PWM
 - TB6612 Driver IC
 - L293D Driver IC

Basic Motor Theory

- What is a Motor?

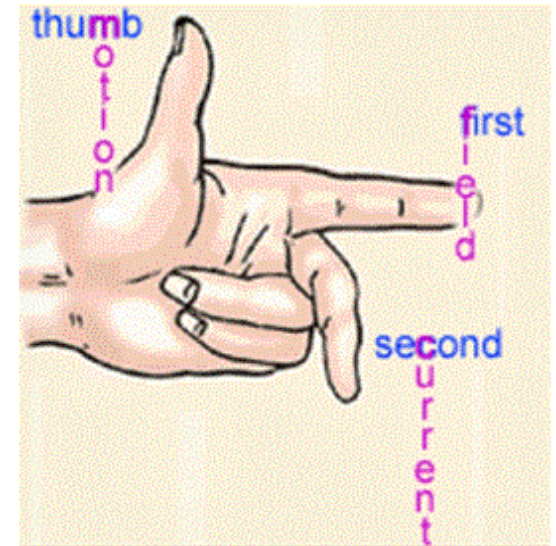
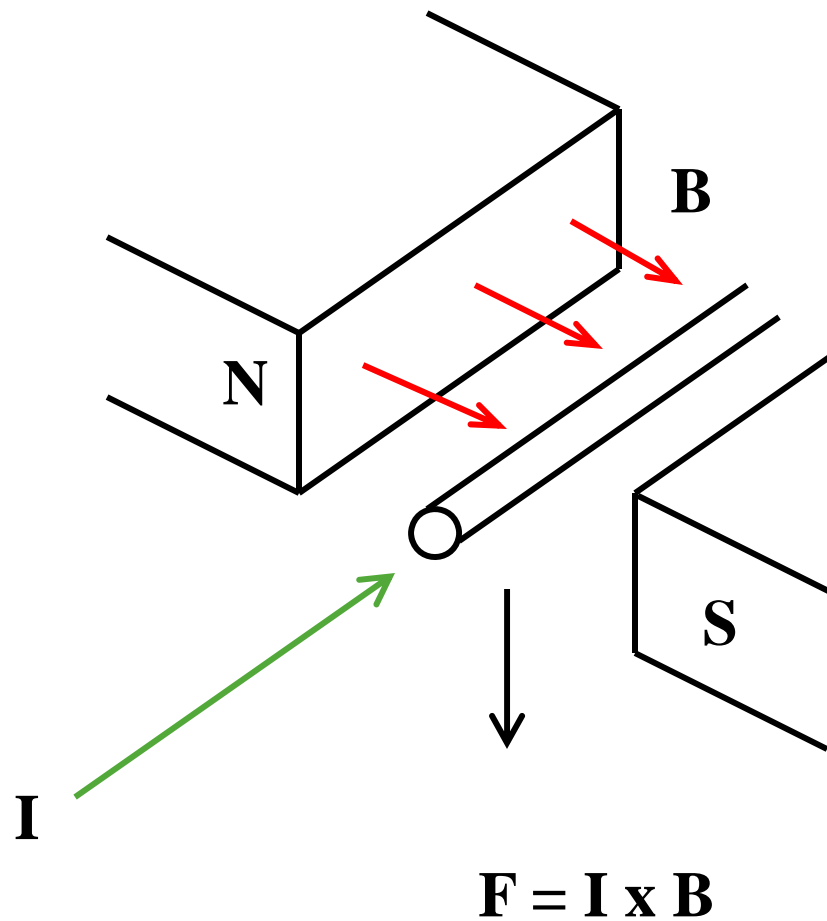
A Motor Converts Electrical Energy to Mechanical

How?

Force is developed when charge moves through a magnetic field

$$F = I \times B$$

Left Hand Rule

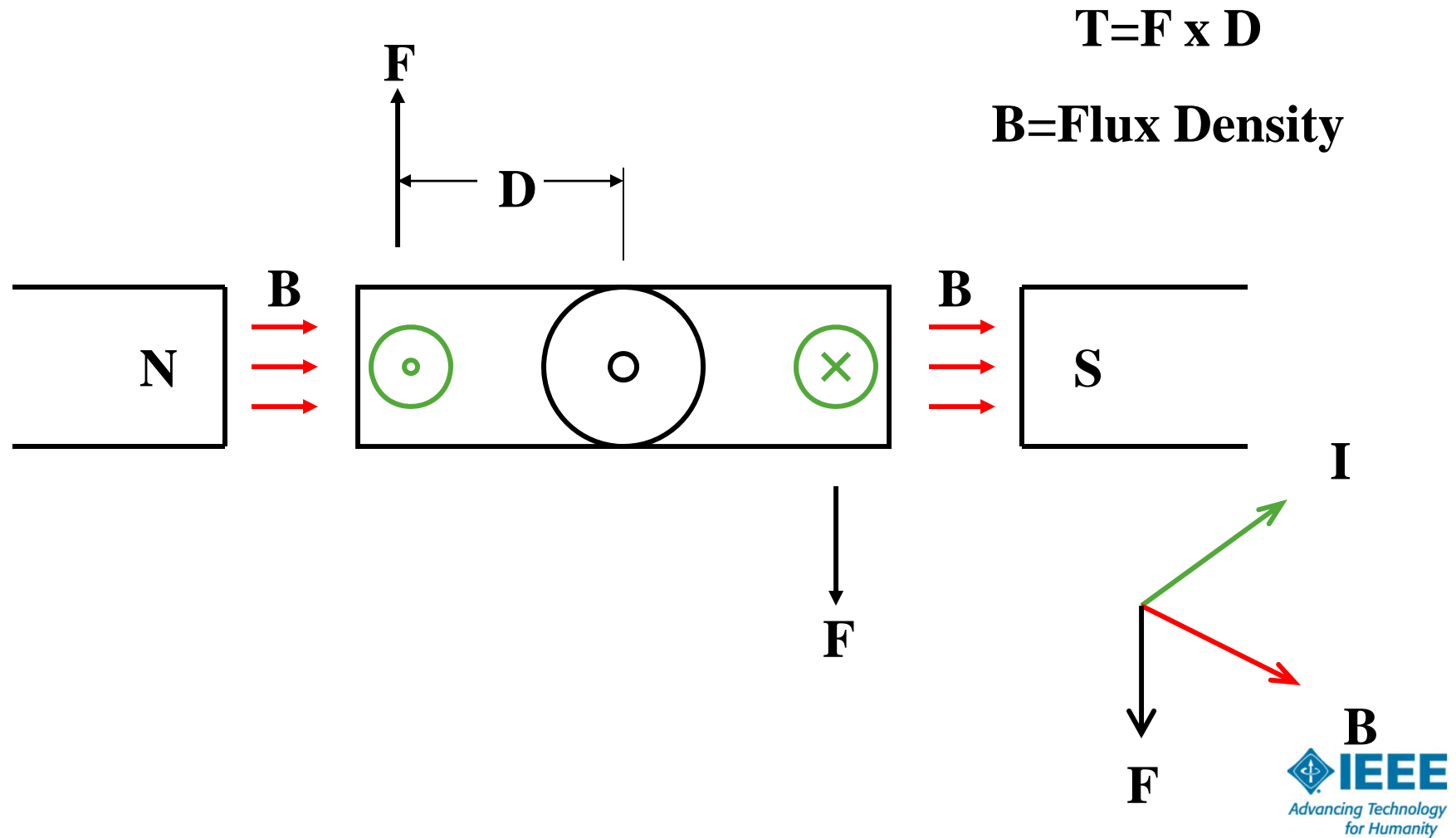


I = Current
(2nd finger)

B = Field
(1st finger)

F = Force/Motion (Thumb)

Motor Torque



DC Motor Torque

- Summary

- Torque = Force * Distance

- $F = I \times B$

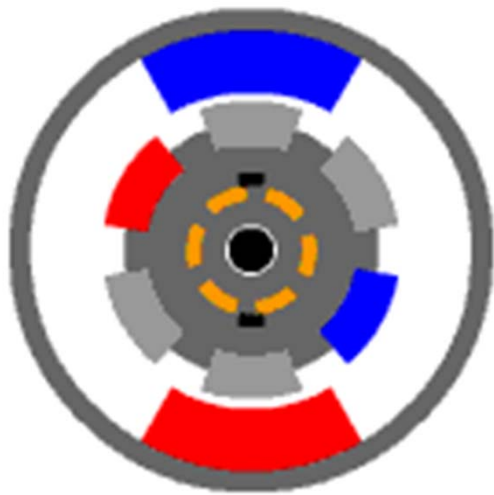
- $T = (I \times B) * D$

- When B and D are constant $T = K * I$

- When field is wound $B = K * I_F$

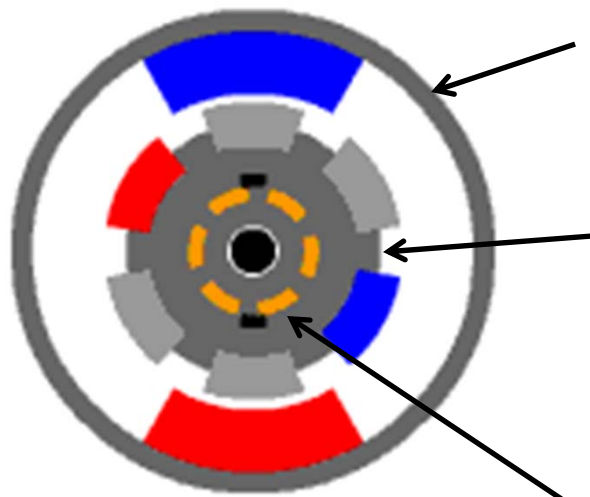
- In wound DC motors Torque (T) and Flux (B) can be controlled independently

Brushed DC Motor: Operation



- Red is North Polarization
- Blue is South Polarization
- Rotor will rotate until North is aligned with South
- Just before alignment, commutator brushes contact next set of contacts
- Spark is generated when the brushes change windings

Brushed DC Motor: Major Components

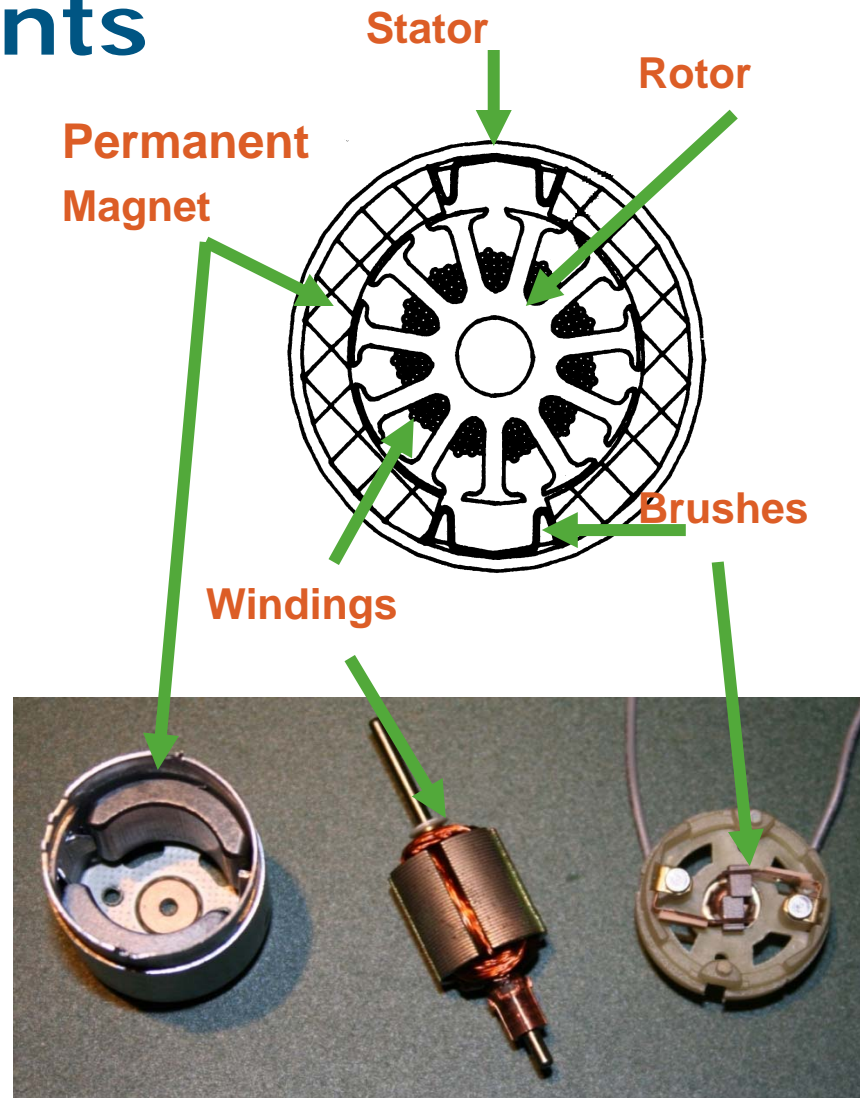


Stator: Stationary Part that contains the permanent magnets

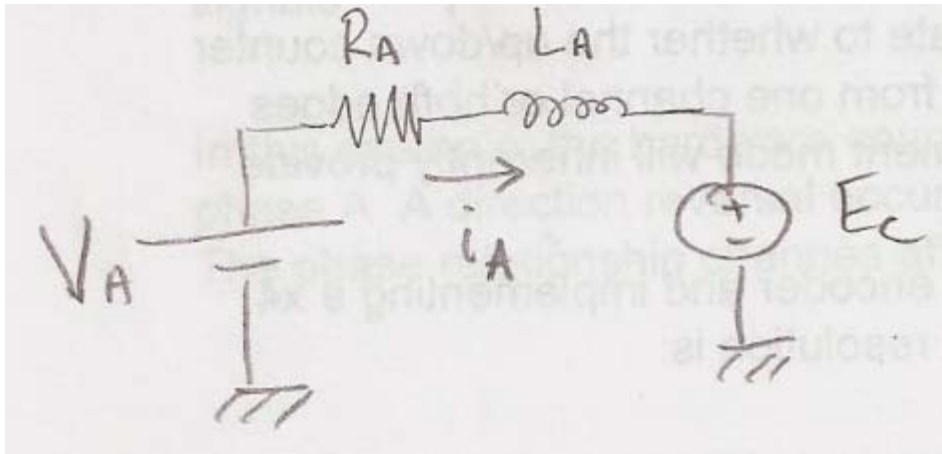
Rotor: Rotating Part that carries the torque-producing current (also called the **Armature**)

Commutator & Brushes: Part that directs the Motor supply voltage to the Armature circuit

Brushed DC Motor: Major Components



Brushed DC Motor: Equivalent Circuit



- A DC motor is simultaneously a DC generator

R_a: Armature Resistance

L_a: Armature Inductance

V_a: Armature Voltage (you apply)

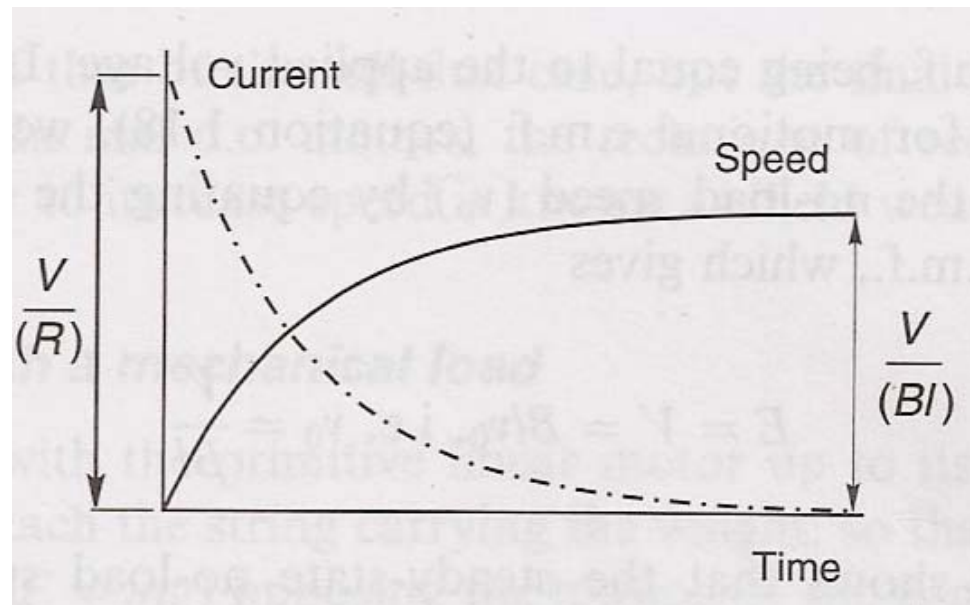
E_c: Generated Voltage (\propto speed)

i_a: Resultant Torque-producing Armature Current

Motor Action (Torque) $\propto i_a$

$$i_a = \frac{V_a - E_c}{R_a}$$

Current (Torque)-Speed Graph



- Torque=max @ Speed=0, since $E_c = 0!!$
- Final speed proportional to V_a

What is the Final Speed?

- In open loop control (we are doing this)
 - DC Motor “self-adjusts” speed based on load and applied armature voltage, V_a
- In closed loop control (advanced)
 - Control system measures motor speed and varies V_a accordingly (“PID” control)

GM8 Gear Motor Key Specifications

GM8 Plastic Geared Motor

Offset/inline output shaft

Ratio: 143:1

Dimensions: 53.8 x 47.8 x 22.9mm

Weight: 31.5g



1-866-276-2687

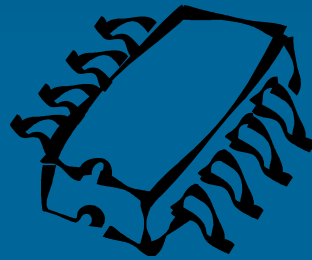
www.solarbotics.com

GM8 with Default RM3 motor

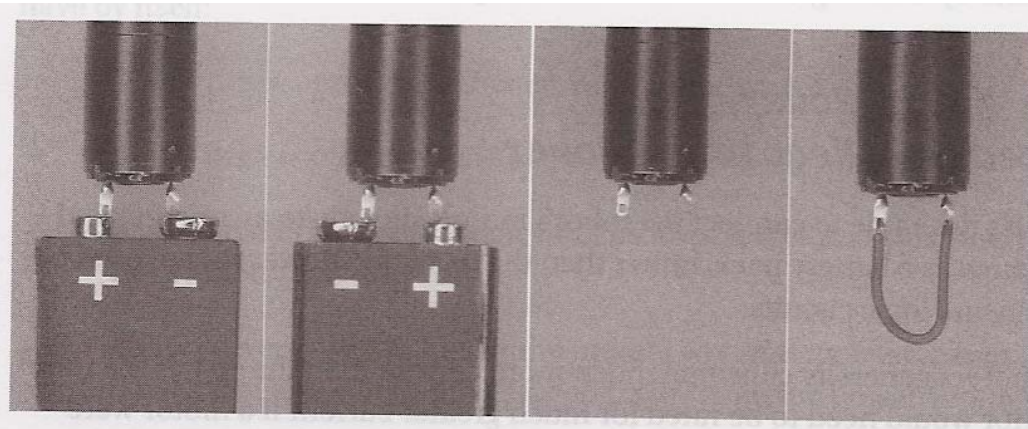
Test Voltage (V)	Unloaded RPM	Unloaded Current (mA)	Stall Current (mA)	Stall Torque	
				(gm*cm)	(oz*in)
3	40	50	400	3200	44.44
6	78	52	700	5500	76.38
9	110	62	1000	5700	79.16
12	135	76	1250	5850	81.24

- Gears trade raw motor RPM for torque
- $R_a = 7.3$ ohm (measured)
- $L_a = 4.1$ mH (measured)

Driving Miss Motor



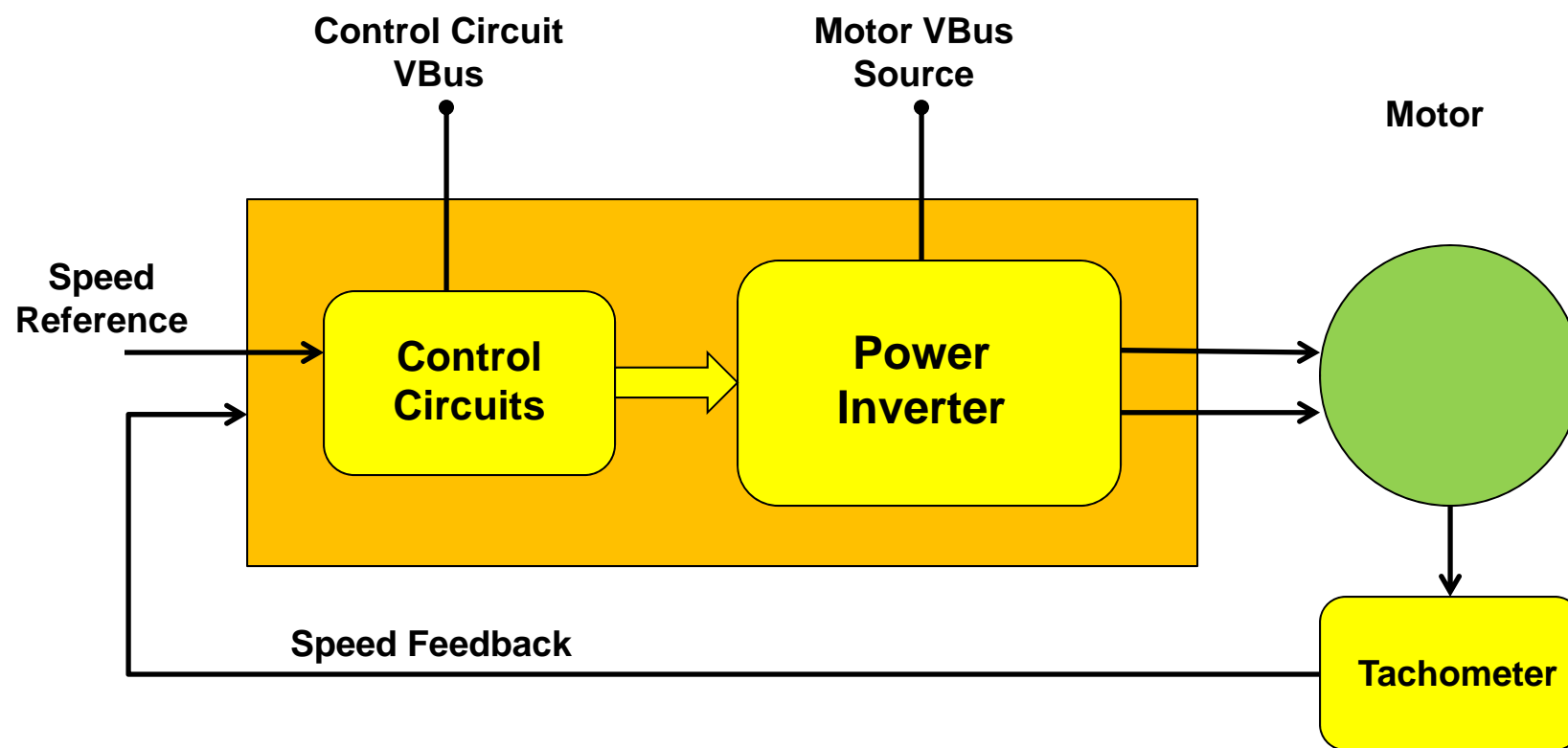
The 4 Modes of a Motor



- Rotating Clockwise
- Rotating Counter-Clockwise
- Coasting
- Braking

Speed-Controlled Motor (“Closed Loop”)

- General system diagram of a speed-controlled motor



Why Two Supplies?

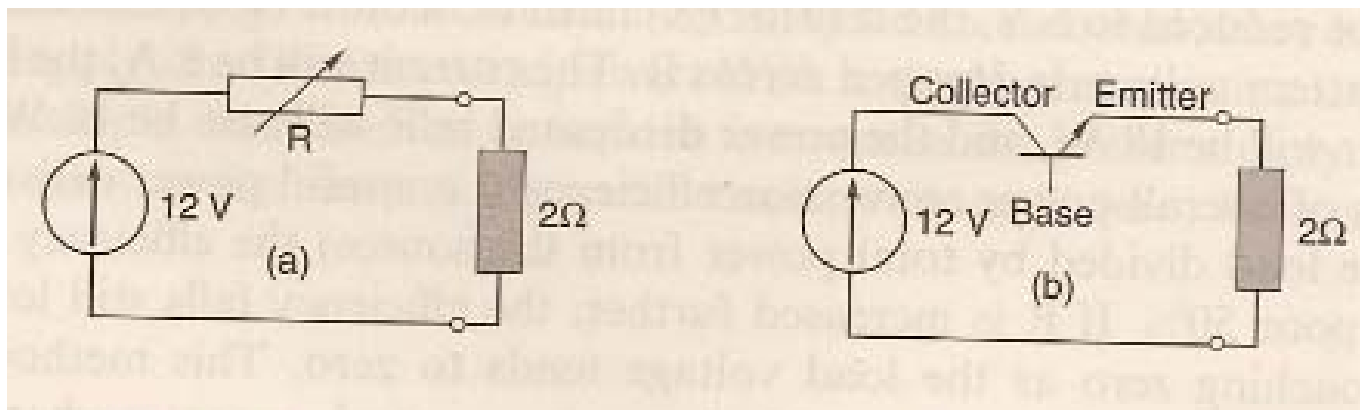
- During operation (start/stop), motors produce noise, which can disrupt the digital brains (logic chips and MCU)
- Motors produce more torque at higher voltages (6.4v is better than 3.3v)
 - Digital circuits work at low voltages
- Batteries can provide larger, less-resistant current flow when the motor starts

Creating a Variable DC Voltage from a Fixed DC Supply

- To control speed, you must control V_a
- We start from 4xAA (6.4V)
- How to create variable motor voltage?

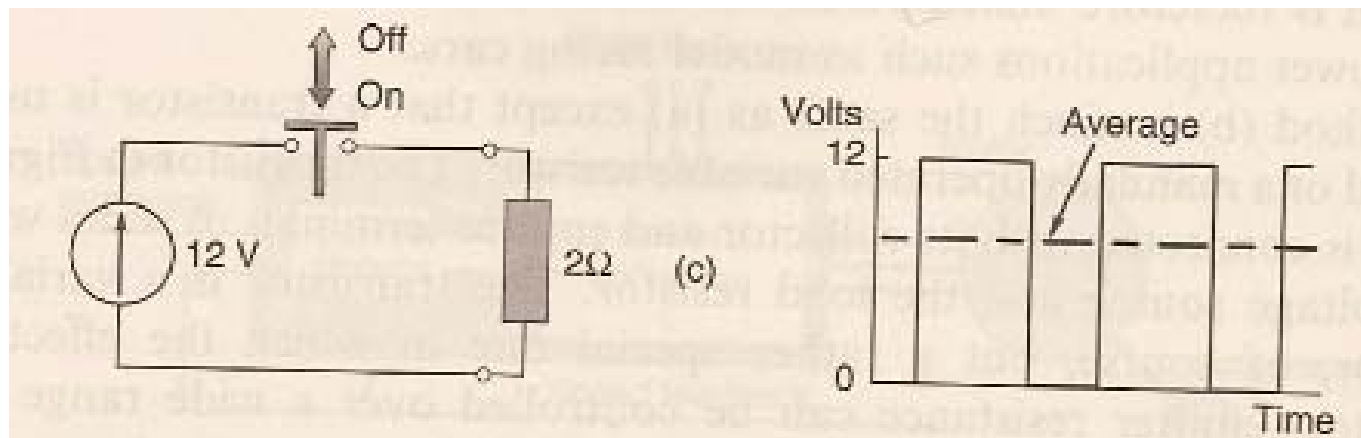
Creating a Variable DC Voltage from a Fixed DC Supply

- “Variable Resistor” Method
- Wastes power!
- Produces smooth load voltage



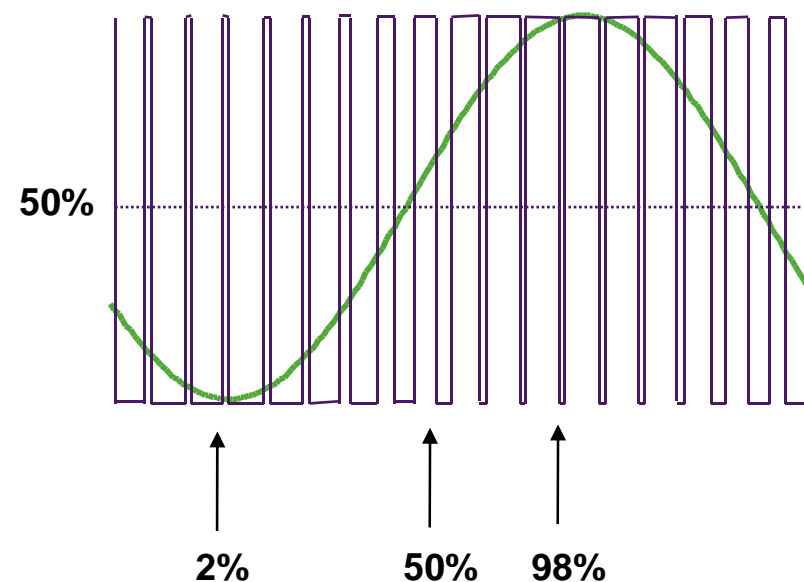
Creating a Variable DC Voltage from a Fixed DC Supply

- “Switching Control”
- Control duty cycle to get desired avg. voltage
- No power is wasted!
- Load voltage is “choppy”, but a Motor’s inductance smoothes out the current

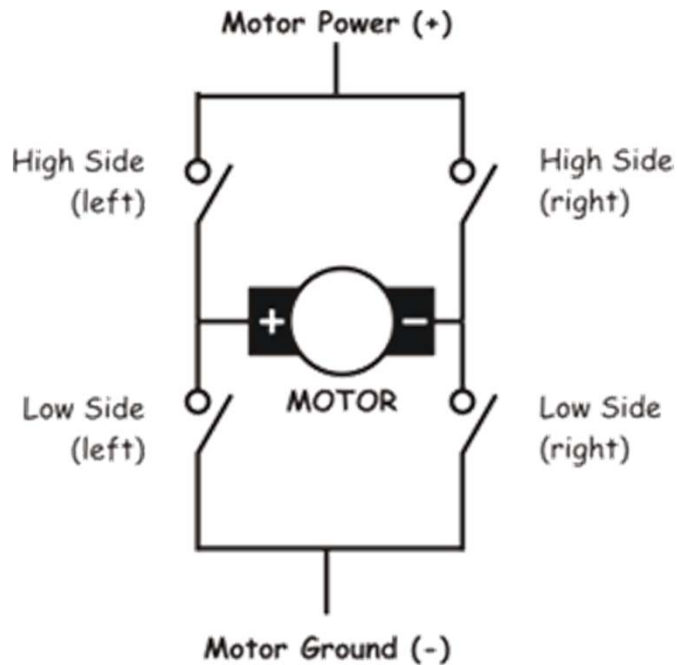


Pulse Width Modulation (PWM)

- Allows fixed DC Input, AC output.
- Output voltage is PWM
- Motor integrates PWM voltage and produces (in this example) a sinusoidal current with small ripple at carrier frequency
- Minimal power loss in power transistors



H-Bridge Equivalent Circuit

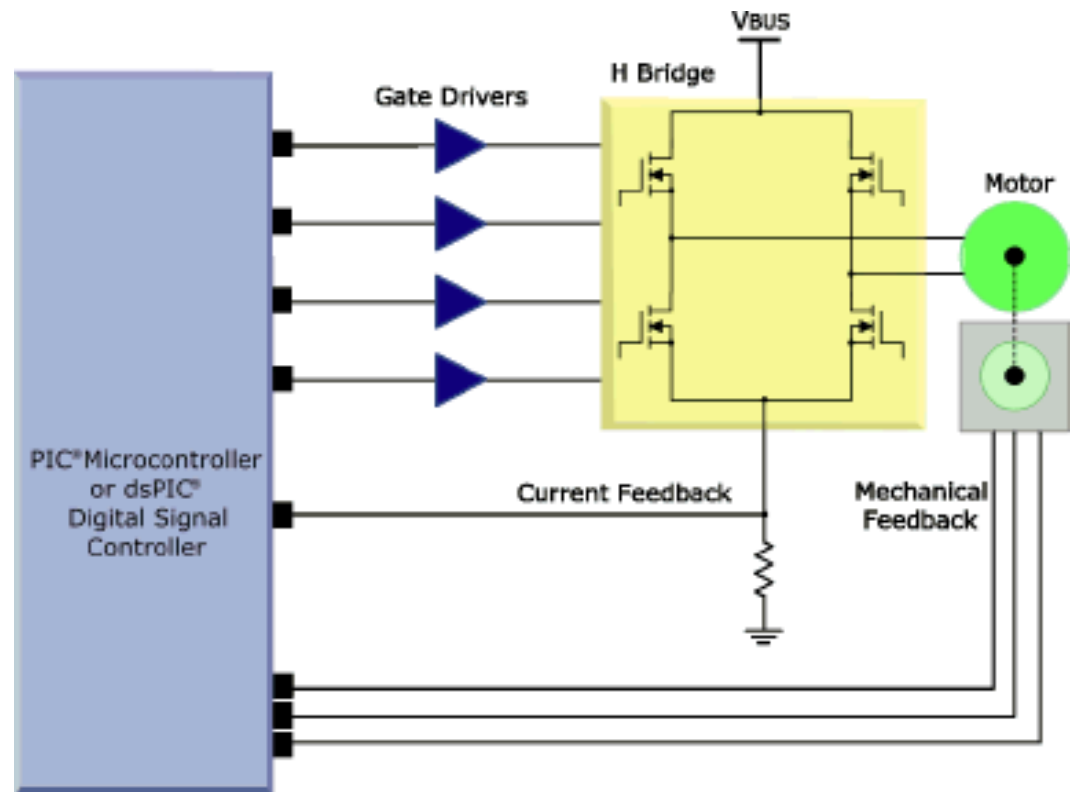


High Side Left	High Side Right	Lower Left	Lower Right	Action
On	Off	Off	On	Motor goes Clockwise
Off	On	On	Off	Motor goes Counter-clockwise
On	On	Off	Off	Motor "Brakes"
Off	Off	On	On	Motor "Brakes"
Off	Off	Off	Off	Motor "Coasts"

H-Bridge Configuration for DC Brushed Motor Drive

Desired PIC[®] MCU or dsPIC[®] DSC Features

- ECCP, MCPWM
- 10 Bit High Speed ADC
- Internal Comparator
- Change Notification
- Quadrature Encoder
- Internal TMRs



Choosing a PWM Frequency

■ Tradeoff between:

– Audible Noise

- Due to torque ripple (caused by the current ripple)
- Higher Frequency → lower torque ripple → lower audible noise

– Switching Losses

- Finite turn-on/turn-off times of MOSFET transistors → produces $V \cdot I$ at the same time
- Higher Frequency → higher switching power loss

– PWM Resolution

- 10-bit (1024 step) resolution @ 20kHz requires 50nS PIC instruction clock (20MIPs or higher)

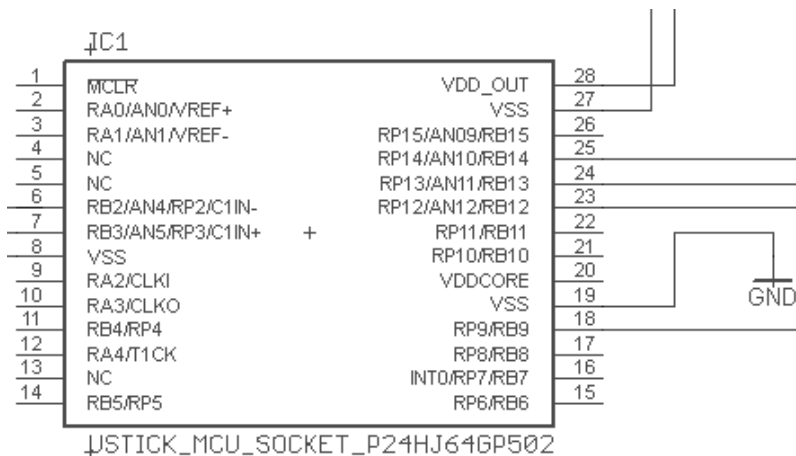
TB6612 Dual 1A Motor Driver

Key Specifications

- Integrated Dual H-Bridge – can drive 2 motors
- Logic power supply range:
 - 2.7–5.5 Vdc
- Maximum Motor Vbus Source Voltage:
 - 15 Vdc
- Maximum standby mode current:
 - 25 μ A
- Output Current capability:
 - $I_{out} = 1.2 \text{ A avg} / 3.2 \text{ A peak}$
- Output ON Resistance, $R_{ds(on)}$ (upper + lower FET)
 - 0.5 ohms (@ $V_M \geq 5 \text{ V}$)
- Built-in thermal shutdown and low-voltage protection
- Data sheet:
 - \Users Guides & Data Sheets\Motor\Driver\

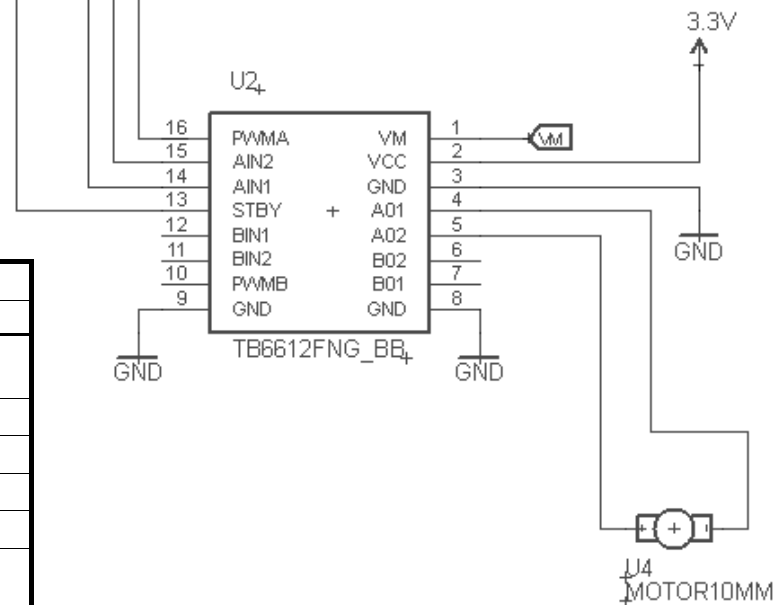


TB6612 Connection & Modes



Notes:
 RB10/RB11 are not avail. when using conn J3 on the uStk
 RB15 not avail. when using led LED1
 µStk board must be modified to provide 3.3V at VDD_OUT

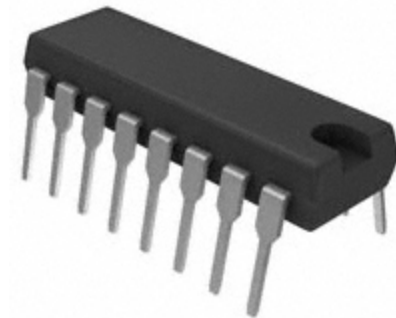
Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby



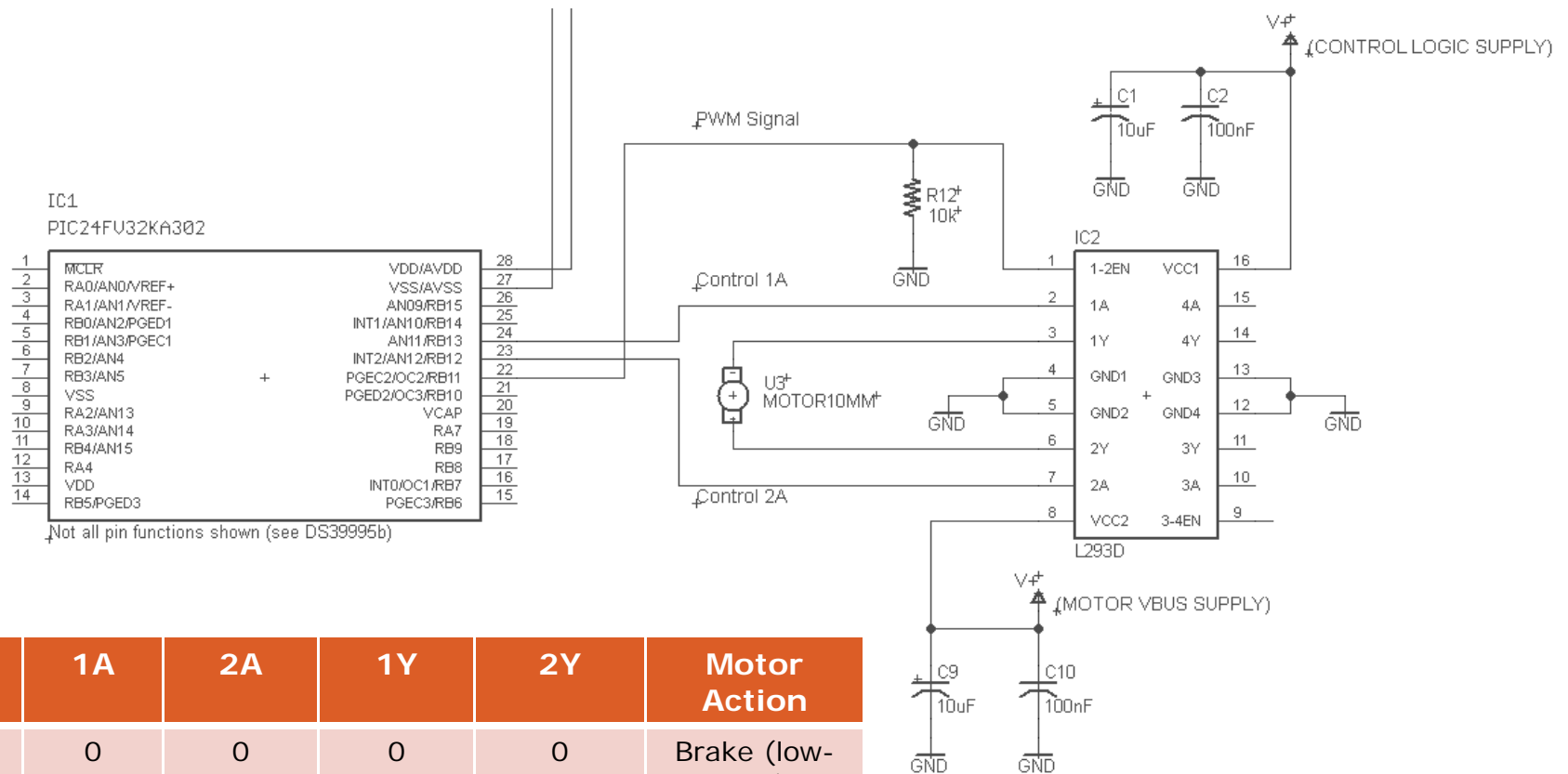
L293D Dual 600mA Motor Driver

Key Specifications

- Integrated Quad “½ H-Bridges” – can drive 2 motors
- Logic power supply range:
 - 4.5–7.0 Vdc
- Maximum Motor Vbus Source Voltage:
 - 4.5-36 Vdc
- Maximum standby mode current:
 - 8mA
- Output Current capability:
 - $I_{out} = 0.6 \text{ A avg} / 1.2 \text{ A peak}$
- Output ON Resistance, $R_{ds(on)}$ (upper + lower FET)
 - 0.5 ohms (@ $V_M \geq 5 \text{ V}$)
- Built-in diode protection
- Data sheet: \Users Guides & Data Sheets\Motor\Driver



L293D Connection & Modes



1-2EN	1A	2A	1Y	2Y	Motor Action
1	0	0	0	0	Brake (low-side)
1	0	1	0	1	Forward
1	1	0	1	0	Reverse
1	1	1	1	1	Brake (high-side)
0	x	x	High-Z	High-Z	Coast

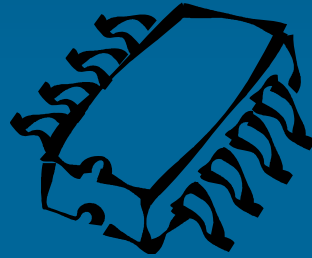
L293D vs. TB6612

Trade-offs

L293D	TB6612	Trade-off
16-PIN DIP	16-PIN SOIC	L293D preferred for solderless prototyping
8mA STDBY I _{dd}	25uA STDBY I _{dd}	TB6612 helps extend battery life
2.4 Volts dropped across switches (@0.6A)	0.6 Volts dropped across switches (@0.6A)	TB6612 allows more torque at same V _{bat} – extends Battery life
\$2.14	\$8.48 + headers**	L293D is much lower cost

**Cost of the TB6612 Prototyping Module – the IC itself is only \$1.69

PIC24 Timer and PWM Peripherals



Creating the PWM Signal

PIC24 Hardware Timer Features

- Five 16-bit General Purpose Timers / Counters
 - Similar functionality between all 5 timers
 - Asynchronous counter feature only in Timer1
- Period Registers for each
 - Interrupt generation on match
 - Reset on match
- Gated Timer operation on each
 - Interrupt on falling edge of gate
- Four of these timers (Timer 2+3 & 4+5) can make two 32-bit timers/counters

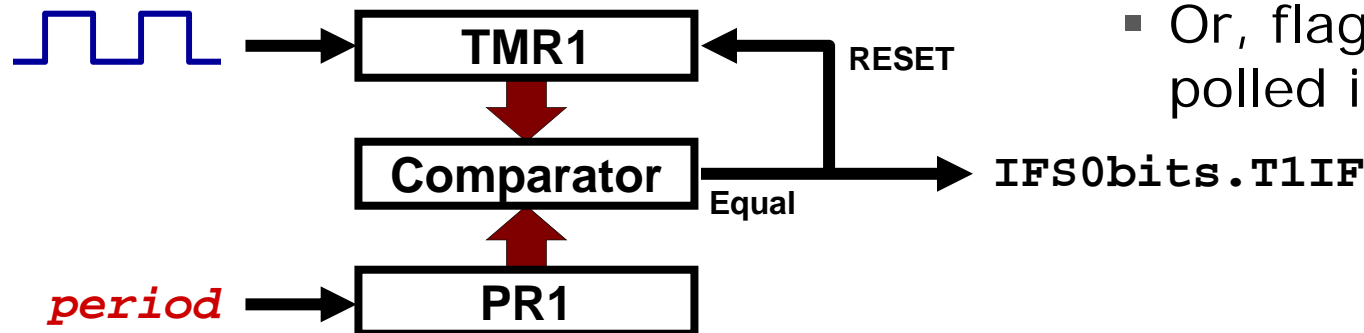
PIC24 Hardware Timer (Simplified)

Inputs

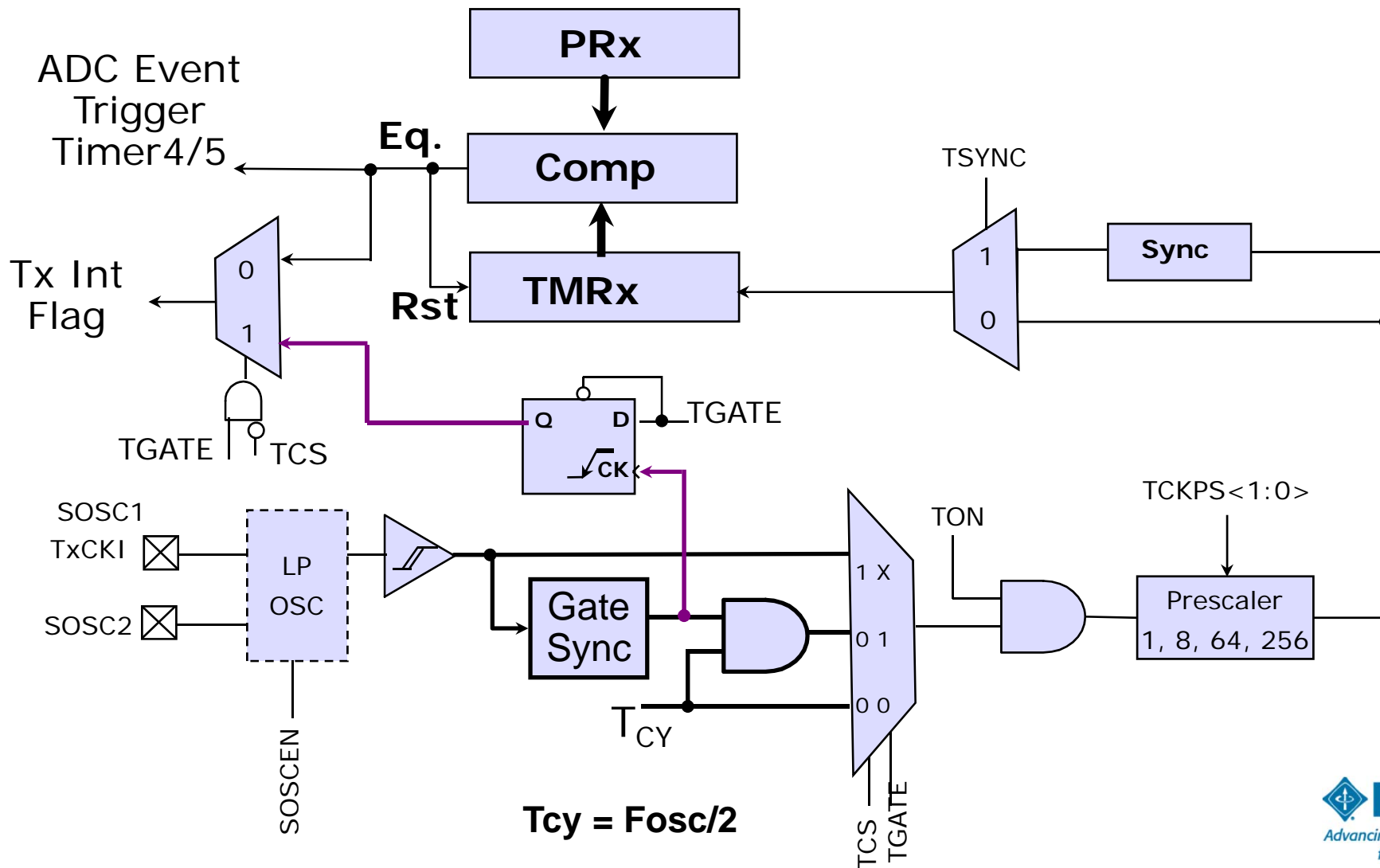
- Scaled clock (time-base)
- Period value

Output

- Timer Interrupt Flag set when $TMR=Period$
 - Eg. `IFS0bits.T1IF`
 - Flag can trigger a TimerX interrupt service routine
 - Or, flag can be polled in `main()`



Timer 1 (Detailed Overview)



Timer 1 SFRs & Bit-fields

REGISTER 11-1: T1CON: TIMER1 CONTROL REGISTER⁽¹⁾

R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
TON	—	TSIDL	—	—	—	—	—
bit 15							bit 8
U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
—	TGATE	TCKPS1	TCKPS0	—	TSYNC	TCS	—
bit 7							bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
 -n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

REGISTER 7-5: IFS0: INTERRUPT FLAG STATUS REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	AD1IF	U1TXIF	U1RXIF	SPI1IF	SPF1IF	T3IF
bit 15							bit 8
R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
T2IF	OC2IF	IC2IF	—	T1IF	OC1IF	IC1IF	INT0IF
bit 7							bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
 -n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown



Using Timer 1 for 100mS main() loop Delay ($F_{osc}/2 = 4 \text{ MHz}$)

■ Initializing the timer

```
void Initialize(void)
{
    // Set T1 Clk Source = Fosc/2
    T1CONbits.TCS = 0;
    // Set T1 Clk Pre-scale = 1:256
    // (TMR1 CLK = 15.625kHz)
    T1CONbits.TCKPS = 3;
    // Set period = ~100mS
    PR1 = 1563;
    // Clear T1 counter
    TMR1 = 0;
    // Clear T1 Int. Flag
    IFS0bits.T1IF = 0;
    // Start T1
    T1CONbits.TON = 1;
}
```

■ Using the delay

```
void main(void)
{
    Initialize();
    while(1)
    {
        Do_Something();

        // wait for T1 overflow
        while(!IFS0bits.T1IF);
        IFS0bits.T1IF = 0;
    }
}
```

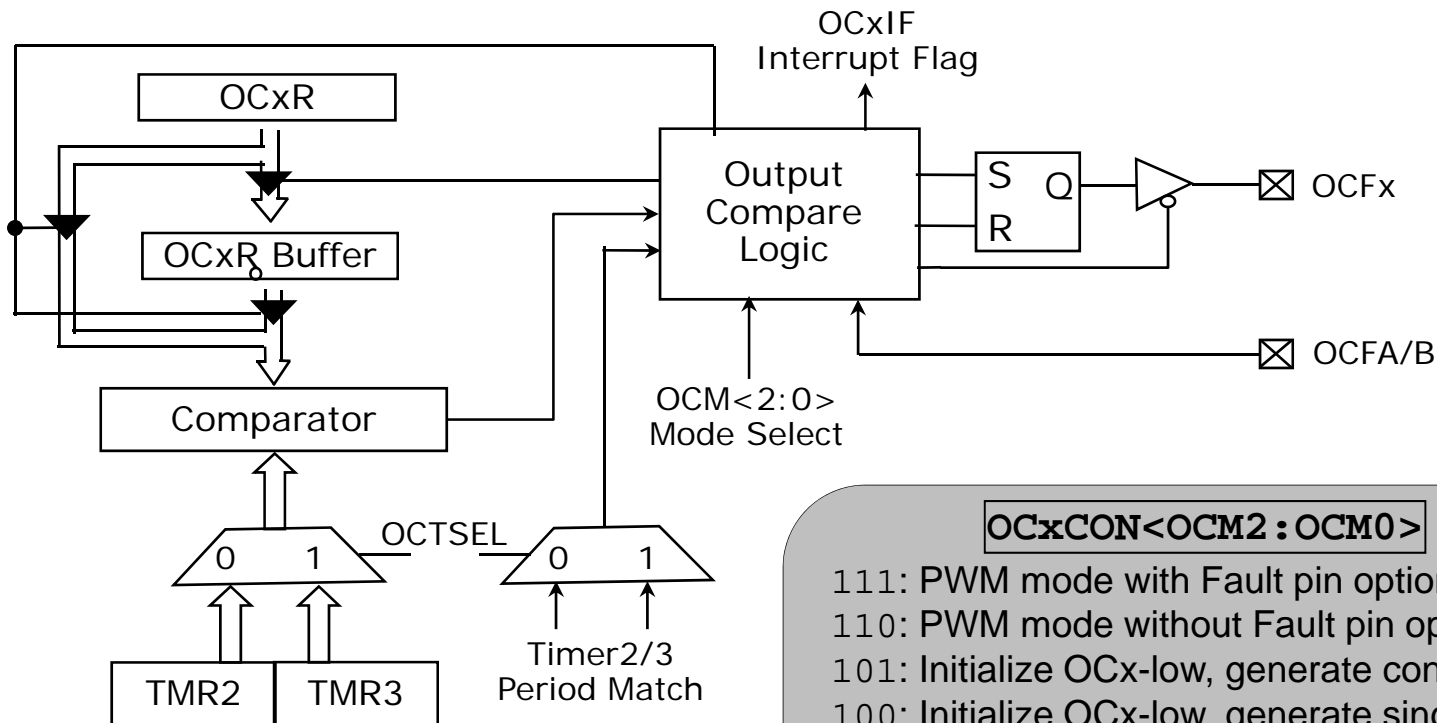
PIC24FV Output Compare/PWM

- Up to 3 Output Compare / PWM Channels
- Built-in Timer as time base
- 16-bit Compare
- Minimum $1T_{cy}$ pulse width allowed
- Several Compare Modes:
 - **Set, Reset or Toggle Pin**
 - **Single Pulse, Continuous pulse**
 - **PWM**
 - **Single Compare Match Mode (using OCxR)**
 - **Dual Compare Match Mode (using OCxR & OCxRS)**

PWM Mode

- PWM mode
 - 16-bit glitch-less (double buffered) PWM output
 - Full range of 0 to 100% duty cycle
 - Wide frequency range
 - Selectable PWM shutdown on fault detection
 - This is an *Asynchronous* shutdown

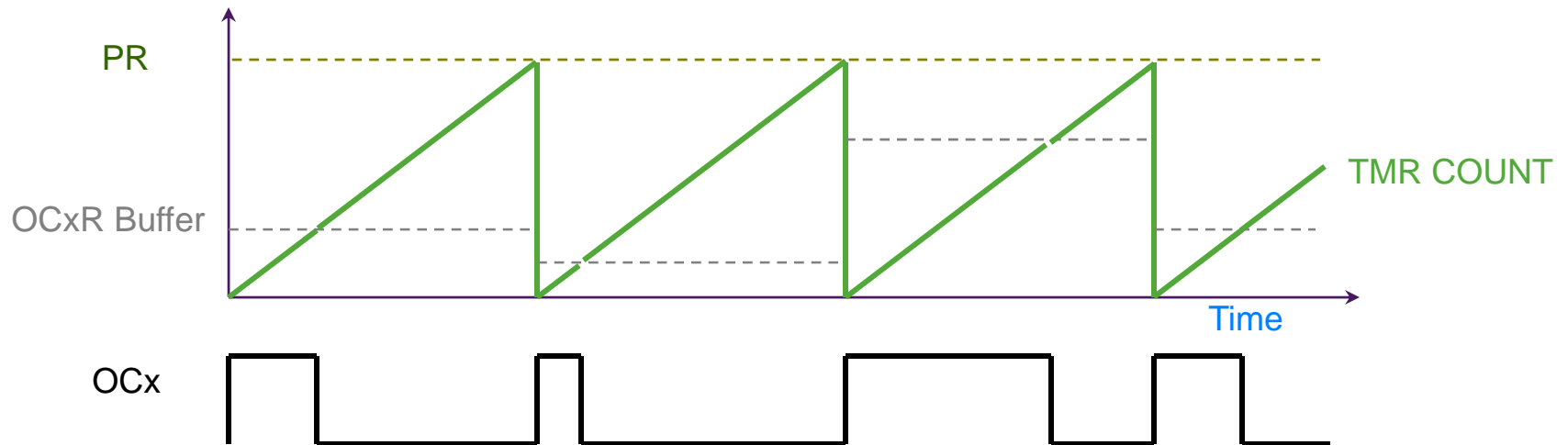
Output Compare: PWM Mode



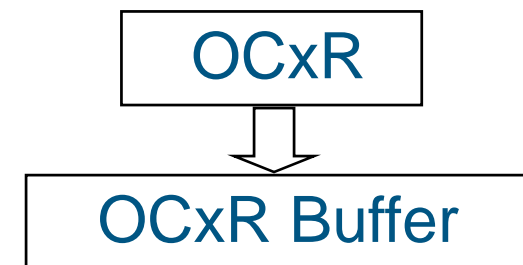
OCxCON<OCM2 : OCM0>

- 111: PWM mode with Fault pin option
- 110: PWM mode without Fault pin option
- 101: Initialize OCx-low, generate continuous signal
- 100: Initialize OCx-low, generate single pulse
- 011: Toggle OCx on every compare match
- 010: Initialize OCx-High, pull OCx low on compare match
- 001: Initialize OCx-low, pull OCx high on compare match
- 000: Compare module is turned off

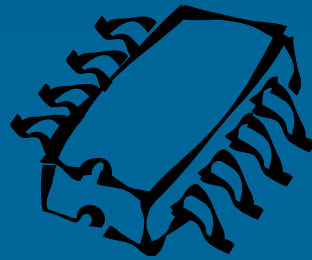
Output Compare: PWM Mode



- On timer period match
 - OCx set, TMR COUNT cleared
 - OCxR copied to OCxR Buffer
- On OCxR Buffer match
 - OCx cleared

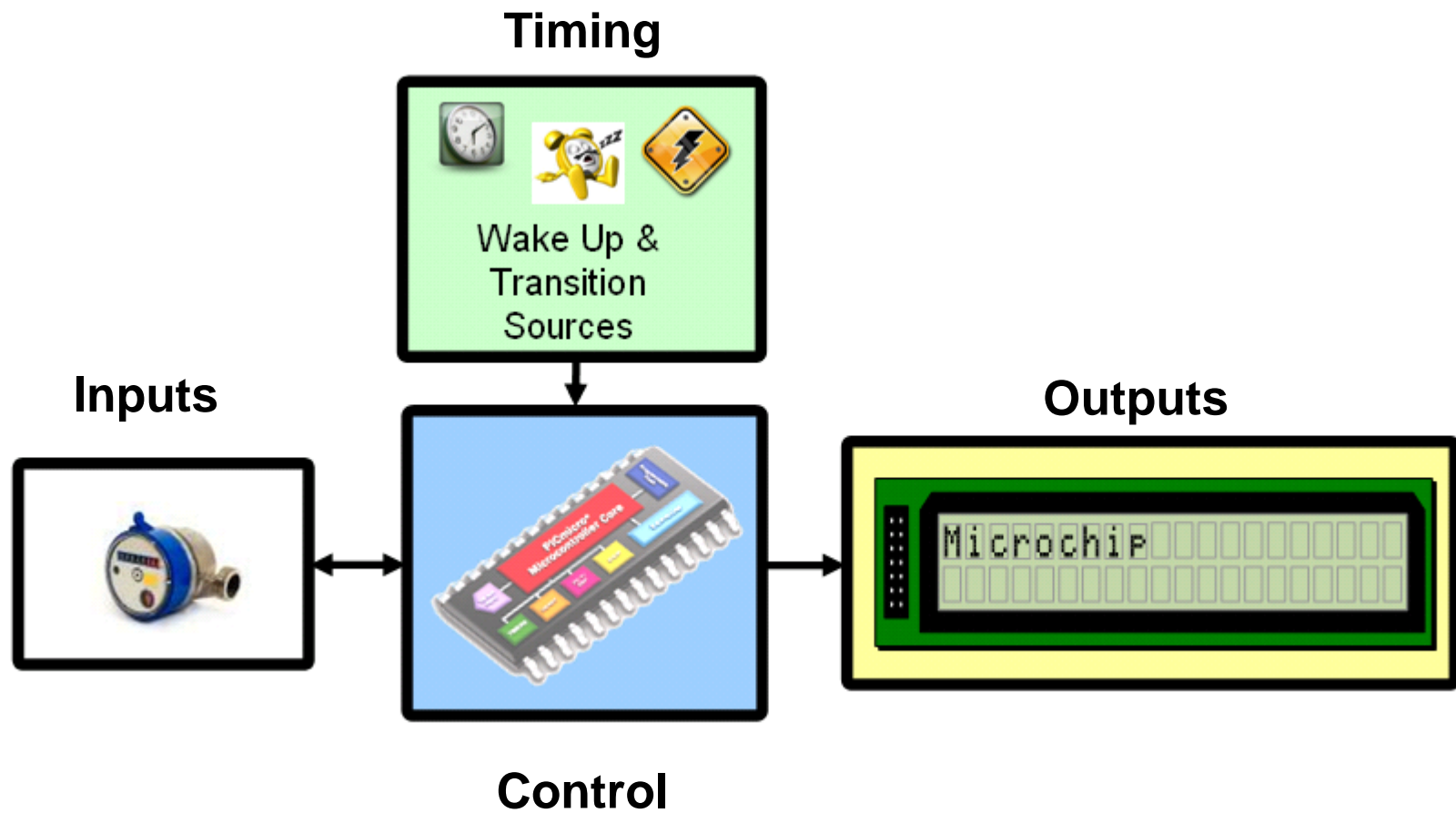


The System Control Loop

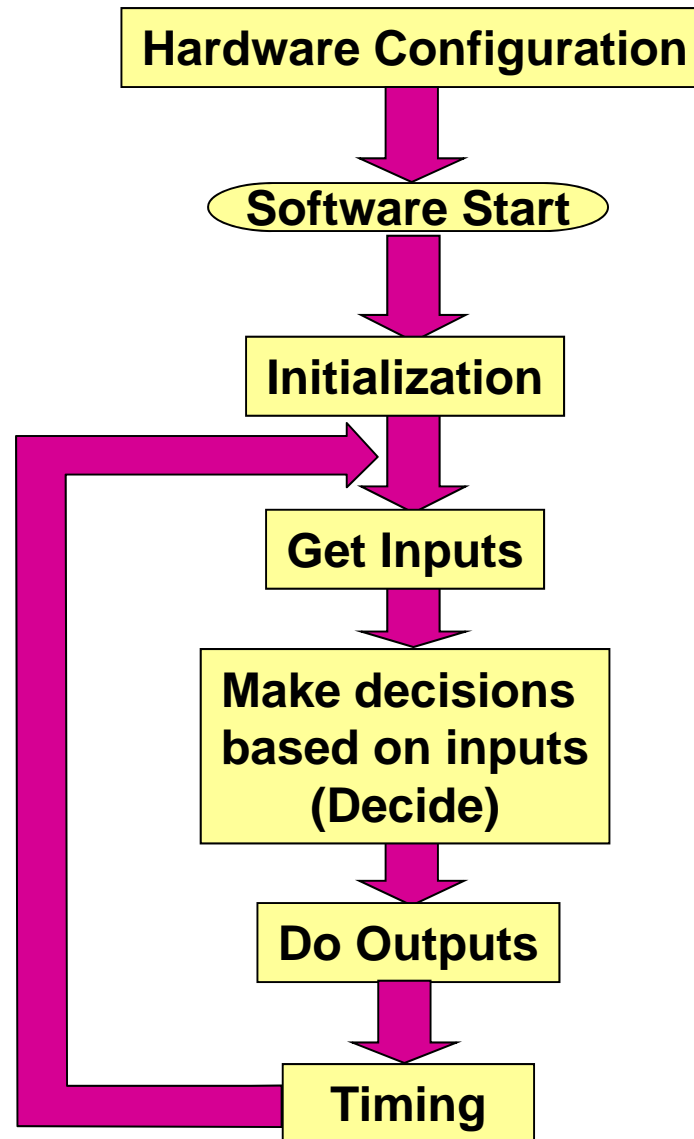


Adding Structure to our Firmware

Embedded System Overview



The System Control Loop



Basic Control Loop Code

Example C-Code:

```
main()
{
  Initialize();
  //infinite loop
  while(1)
  {
    Get_Inputs();
    Decide();
    Do_Outputs();
    Timing();
  }
}
```

Hardware Configuration

Software Start

Initialization

Get Inputs

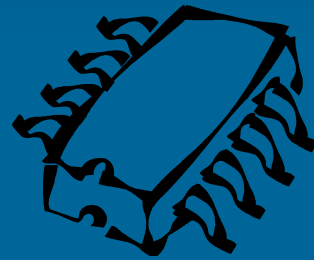
Make decisions
based on inputs
(Decide)

Do Outputs

Timing

Lab 3

Motor Control Using the PWM Peripheral



Lab 3

Motor Control Using The PWM Peripheral



Purpose

The purpose of this lab is to experiment with driving the GM8 motor in both CW and CCW modes using the provided firmware.



Procedure

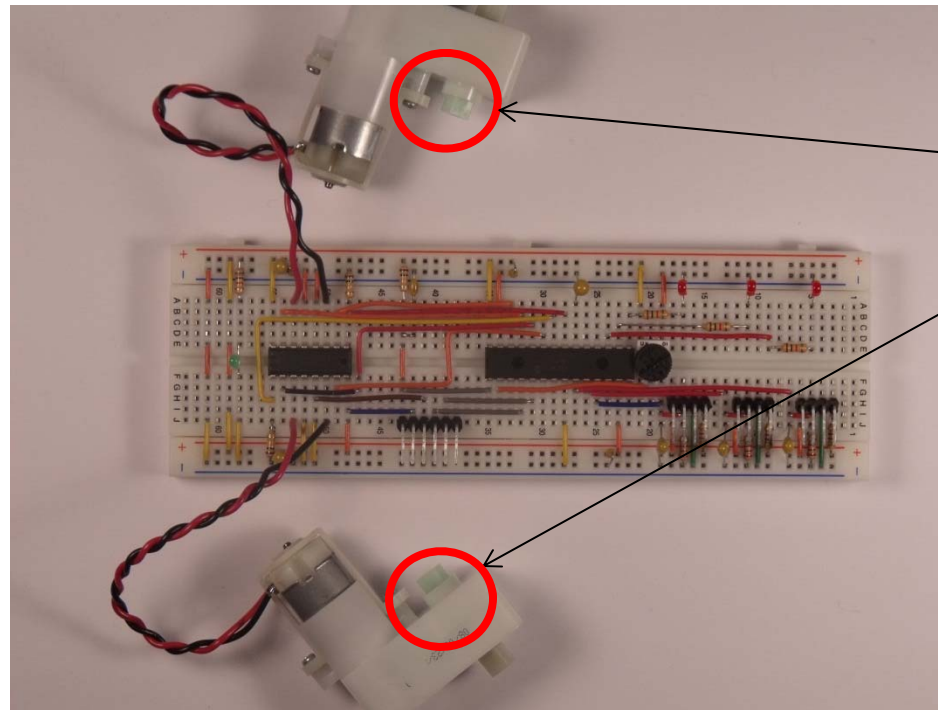
Follow the instructions in the lab manual

Lab 3

Motor Control Using The PWM Peripheral



Results



GM8 spins!

Lab 3

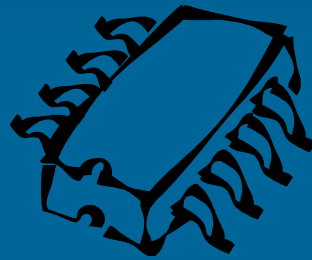
Motor Control Using The PWM Peripheral



Conclusions

- Driving Miss Motor:
 - A DC Motor is simultaneously a DC generator
 - Torque is directly proportional to V_a
 - Torque is inversely proportional to speed
 - In open-loop, final speed based on load and V_a
 - The 4 modes of a motor
 - PWM is the most efficient way to create varying DC voltage for V_a
 - PIC24 hardware timer and PWM peripherals can generate the PWM for you (freeing your code to do other things)

Robot Inputs



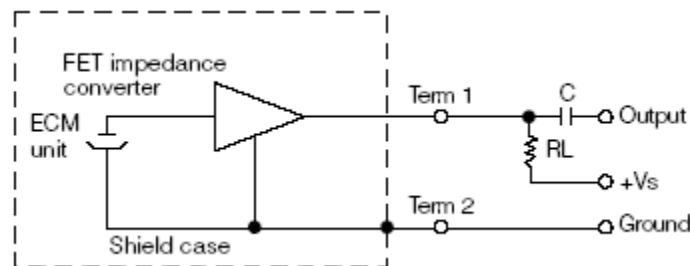
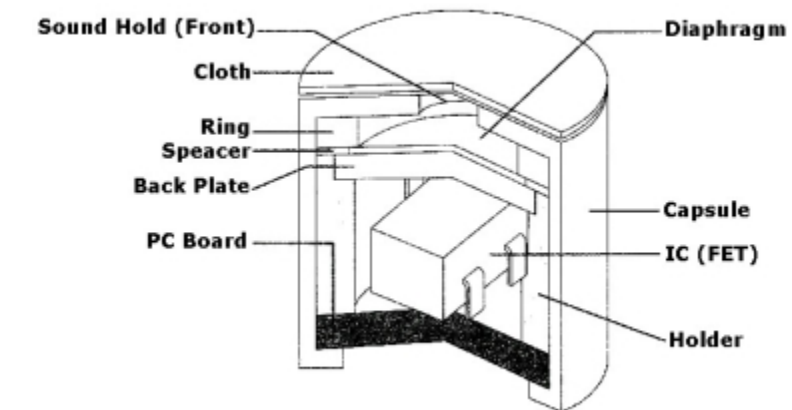
- **What is a Sensor ?**
- **Optical Sensing used in Line Following Robots**
- **Analog-to-Digital Converter Peripheral**

What is a Sensor?

- A component that converts a physical measurement variable to an electrical signal (voltage or current)
 - Mechanical (acceleration, velocity, force)
 - Thermal (temperature)
 - Optical (light intensity, spectrum)
- Requires a sensing element, which absorbs energy
- Requires a transduction element to convert the absorbed energy to an electrical output
- Optionally, includes electronic circuitry for signal conditioning
 - “Amplify”, “Filter”, and “Bias” for Analog-to-Digital conversion.

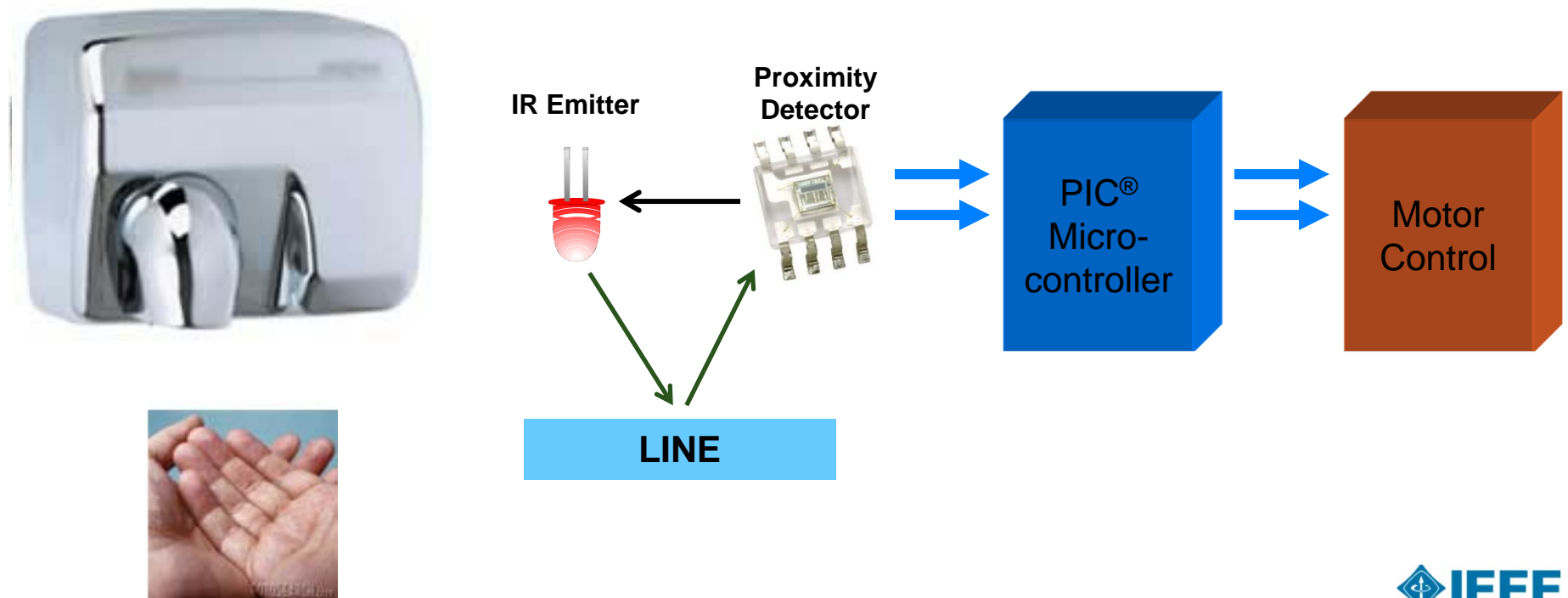
Example: Electret Condenser Microphone

- A condenser microphone is essentially a capacitor, with one plate of the capacitor moving in response to sound waves (the 'sensing element').
- The movement changes the capacitance of the capacitor, varying the voltage across it (the 'transduction element')
- ...and these changes are buffered (using a FET) to create a measurable signal at the output.



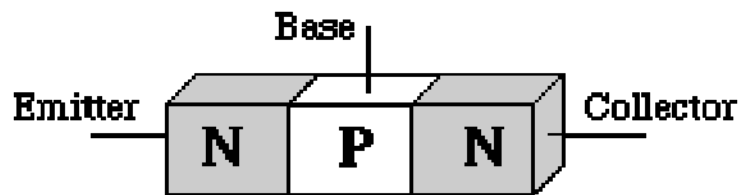
Optical Sensing Used for Line-Following Robot "Reflective – Proximity"

Sensor Detects Reflected IR radiation from Black Line



Phototransistors

- Bipolar transistor with large, exposed base
- Light generates base current which is then amplified by h_{fe} (typically 500 to 1500)

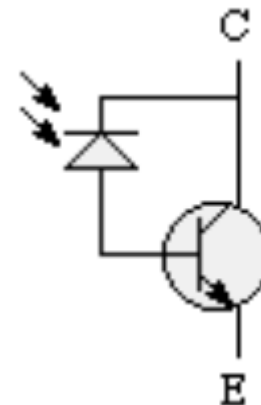


$$I_C = (h_{fe} + 1) I_\lambda$$

I_C is the collector current

h_{fe} is the current gain

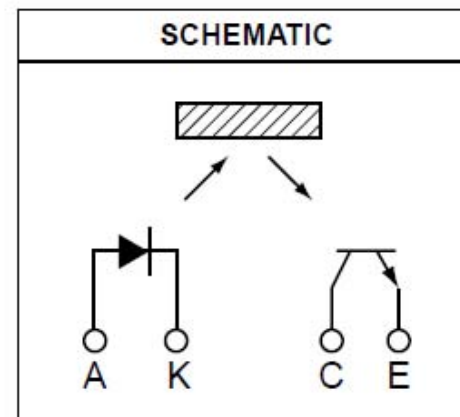
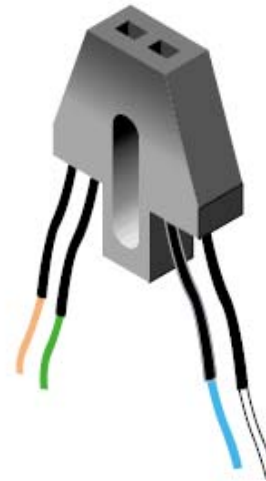
I_λ is the photo-induced base current



Equivalent Circuit

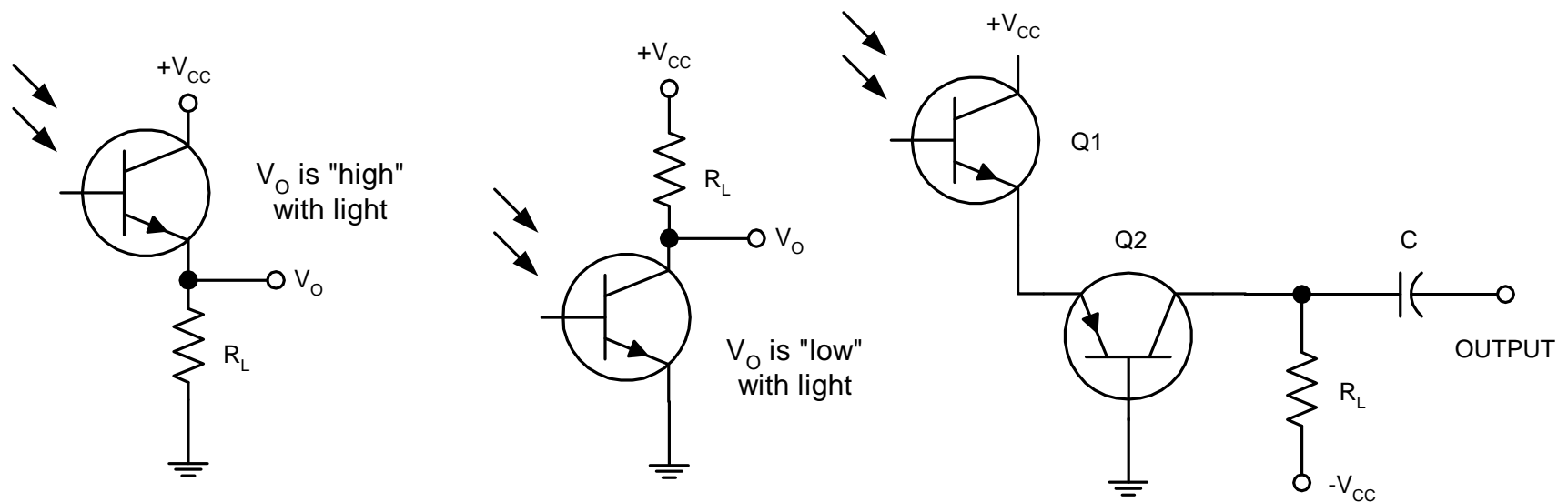
QRB1134 Phototransistor Reflective Object Sensor

- Emitting diode
- NPN silicon phototransistor
- Response proportional to reflected light



Using Phototransistors

Basic Phototransistor Circuits



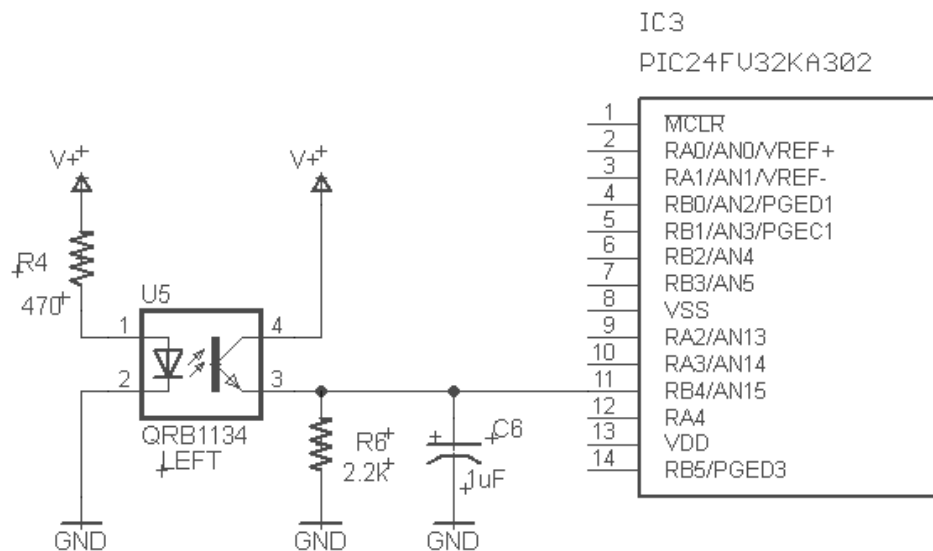
High-Going Switch

Low-Going Switch

Improved Speed Configuration

Interfacing QRB1134

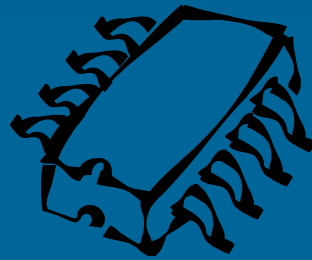
Line Detection



There are 3 sensors used for line-detection – only 1 circuit is shown here

- When light is sensed by the phototransistor, capacitor C6 is charged and this signal is routed to the PIC24 A/D input.
- Black line absorbs IR light and results in lower A/D count.
- Requires calibration (experimentation) to establish a threshold count value between reflection and absorption

Processing Analog Signals on a Computer



Analog (or “Real-World”) Signals

- Analog signals are patterns of information represented by physical quantities
- The pattern maintains an exact physical “analog” of information
 - Examples:
 - Sound wave intensity at your ear maintains an exact physical analog (amplitude/frequency) to a vibrating body in the room
 - Bumps on a record maintain an exact physical analog to the amplitude/frequency of the originating sound wave amplitude
 - Magnetism on a cassette tape
 - Height of a fluid in a thermometer
 - Voltages & currents in an electrical circuit connected to the output of a microphone
- “Real-word” signals are limited by the finite energy expended in their production, and will satisfy physical requirements of
 - Finite signal value (value does not diverge, no singularity)
 - Finite signal energy & power
 - Finite signal bandwidth (No step changes!)
 - Analog signals are continuous in both time and amplitude
 - Car cannot instantaneously change velocity
 - Room temperature cannot instantaneously step change by n

Digital Signals

- Digital signals are patterns of information represented by a finite set of digits per sample
 - discrete values, defined at specific time instants

- Characteristics
 - Numbers are represented by symbols (“digits”) using positional notation
 - Decimal Symbols → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Binary Symbols → 0, 1
 - Positional Notation → $45310 = (4 \times 10^2) + (5 \times 10^1) + (3 \times 10^0)$

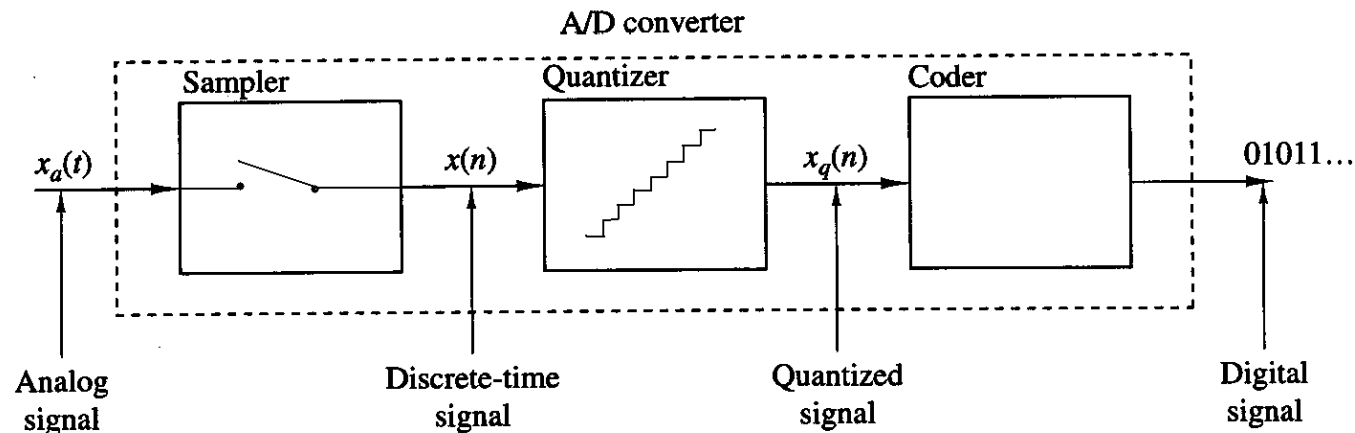
- Inside a computer, all information is represented by numbers

What is an A/D?

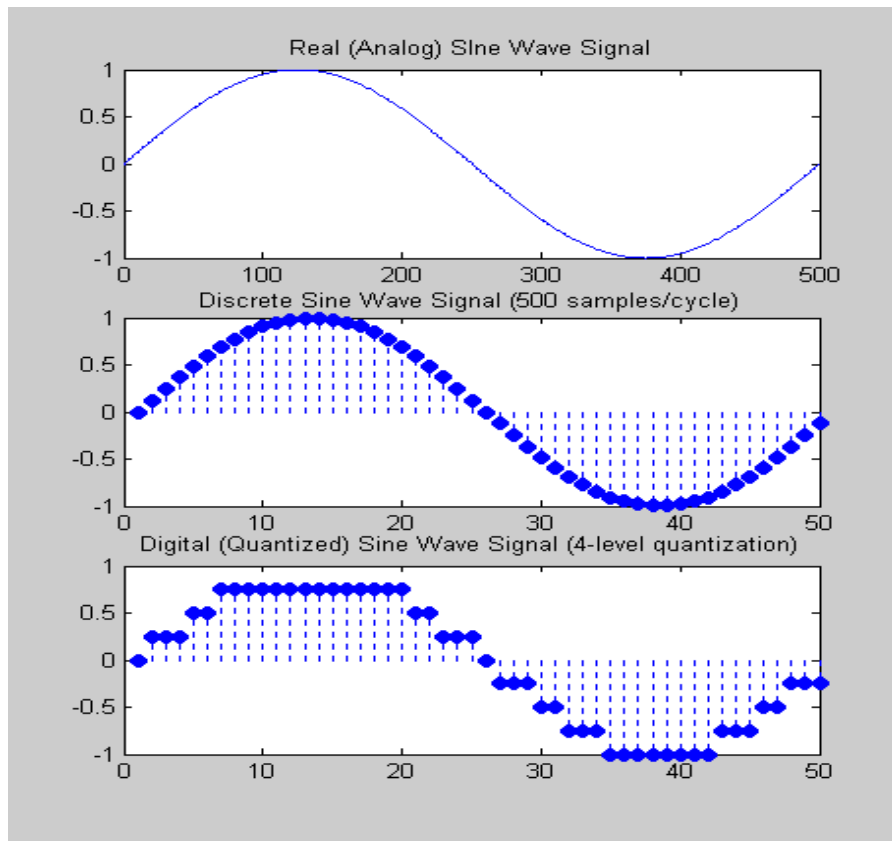
- Analog-to-Digital converter (ADC)
- Circuit that maps a range of analog voltages to a range of binary integers
 - Analog signal = infinite # levels
 - Digital signal = finite # levels
 - # levels = 2^N where N = number of conversion bits
 - Eg. 10-bit ADC produces 1024 numbers (0-1023) representing voltage range 0 to V_{dd}
 - Mapping is linear (1:1)

Basic Analog-to-digital Conversion Blocks

- Sampler
 - Samples the signal at discrete time intervals
- Quantizer
 - Approximates the sampled voltage with a level from a fixed set of 2^n possible voltage levels via ROUNDING or TRUNCATION
- Encoder
 - Encodes the measurement in a convenient format for communication or processing



Analog, Discrete, Digital Signals



Analog Signal

Discrete time signal

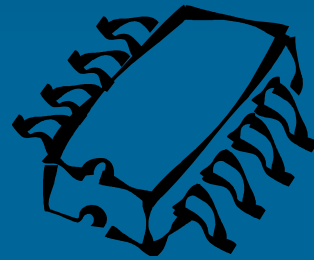
Quantized signal

0, 255, 255, 255, 512, 512, 1023, 1023, ...

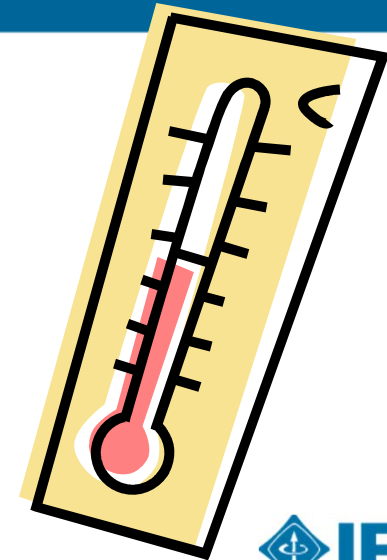
Digital signal (decimal representation)



PIC24 Analog to Digital Converter



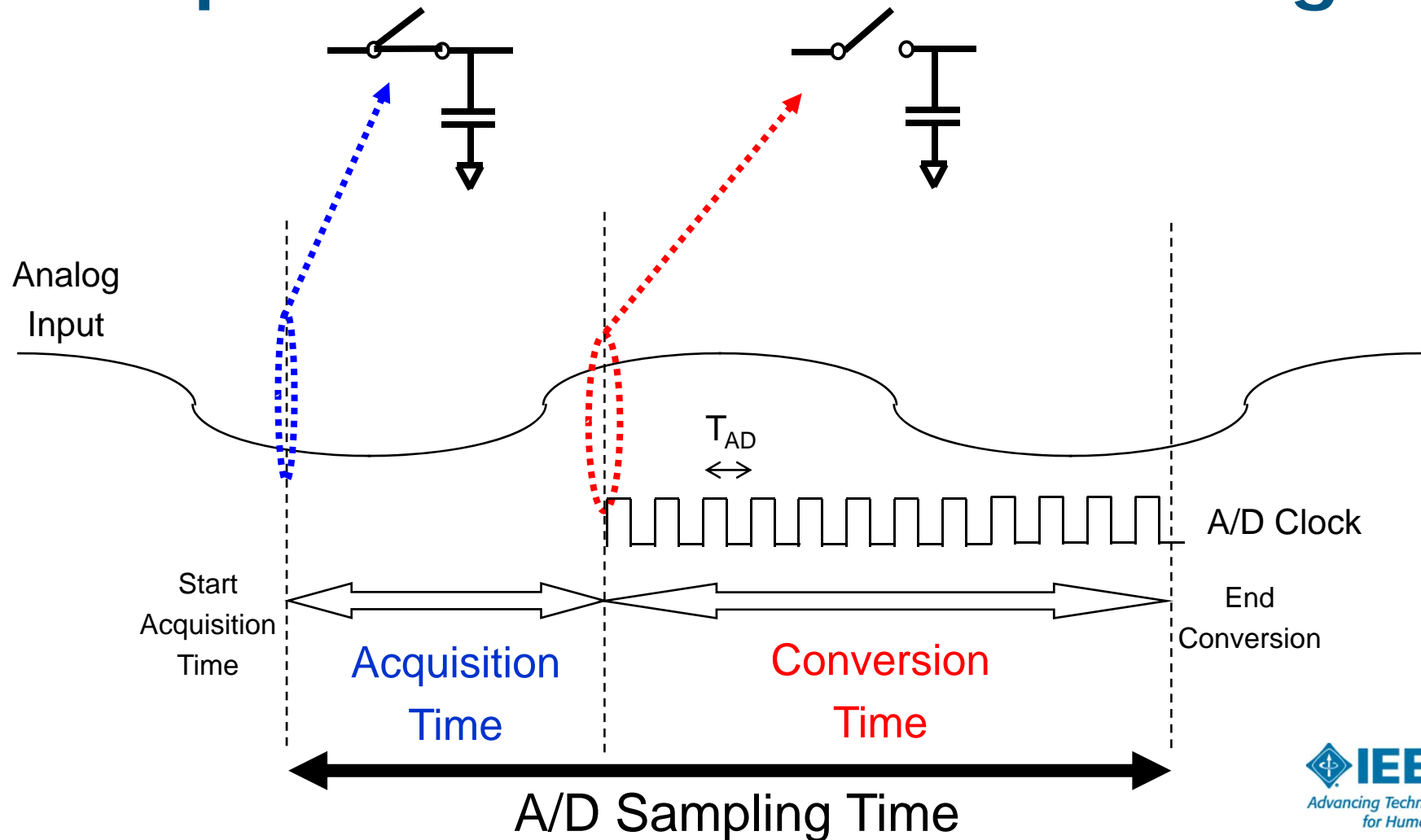
Digitizing the sensors



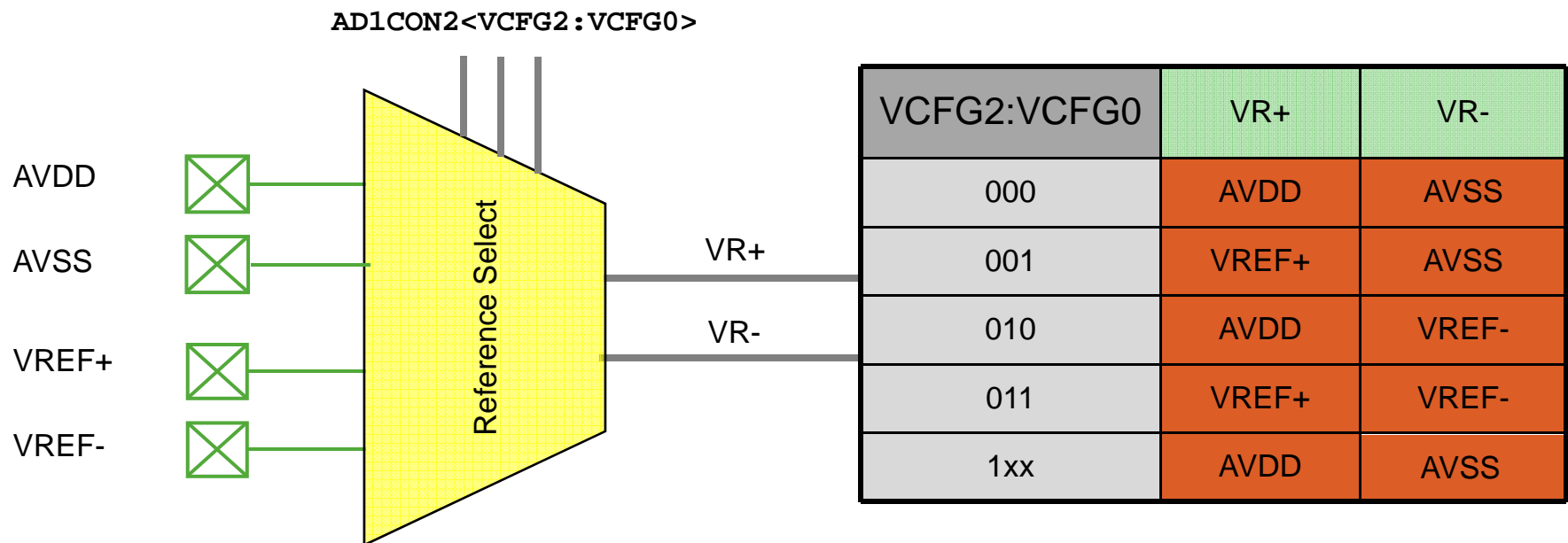
10/12-bit ADC Features (PIC24FV)

- Successive Approximation (SAR) conversion
- Conversion speeds of up to 500 ksps
- Up to 16 analog input pins
- Automatic Channel Scan mode
- Selectable conversion trigger source
- 16 word conversion result buffer

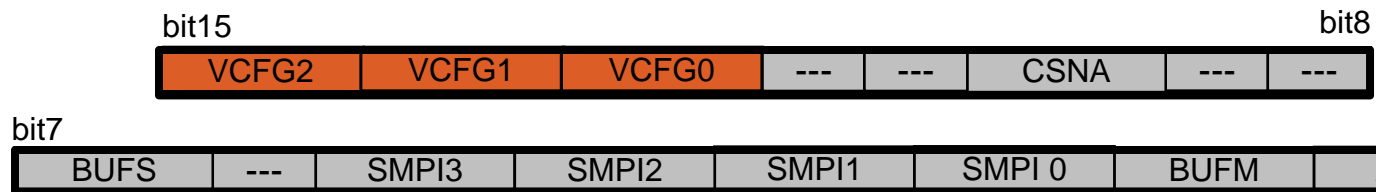
10-bit ADC Acquisition & Conversion Timing



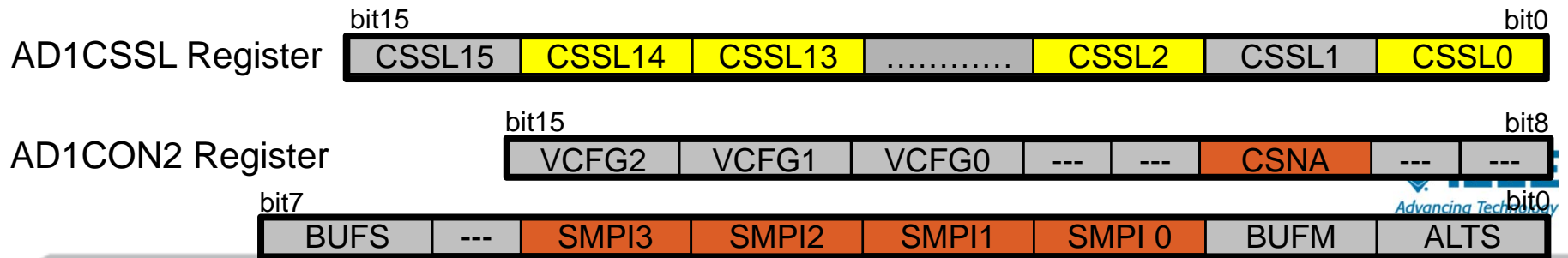
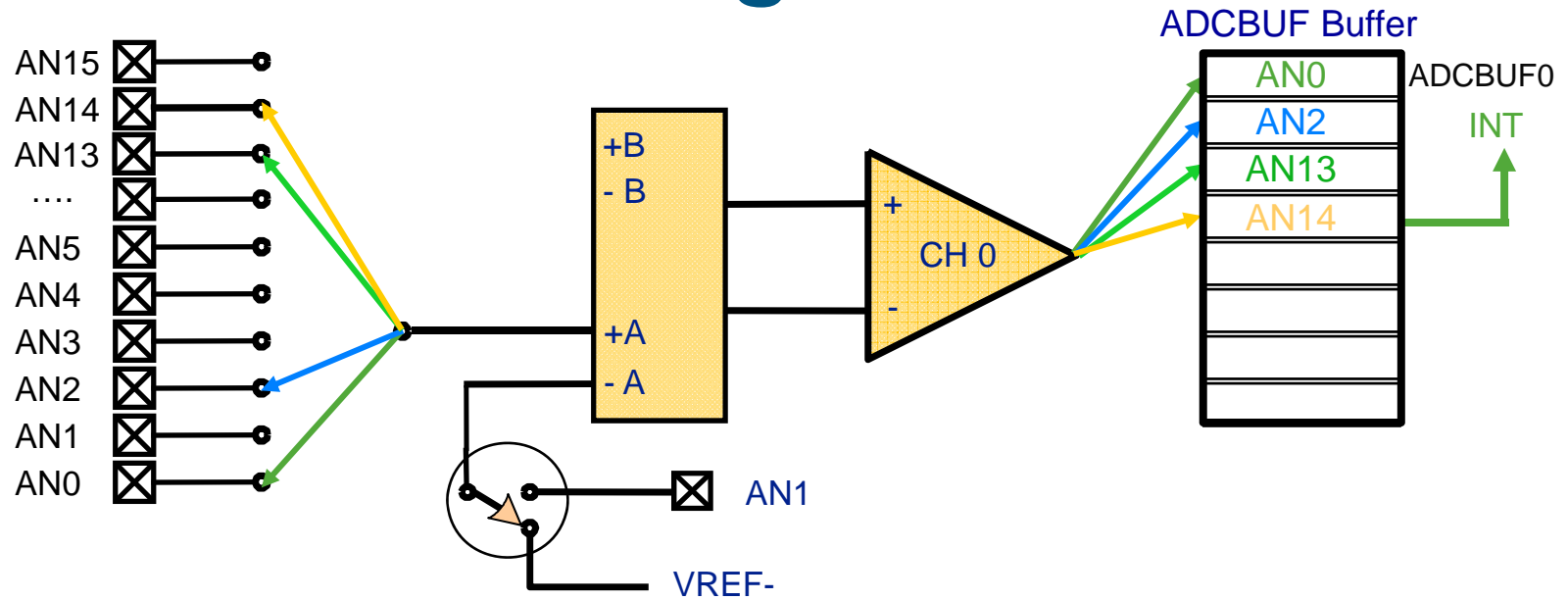
10-bit ADC Voltage Reference



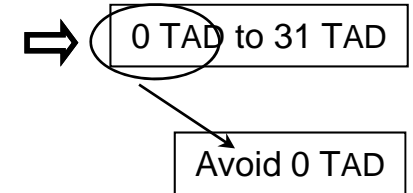
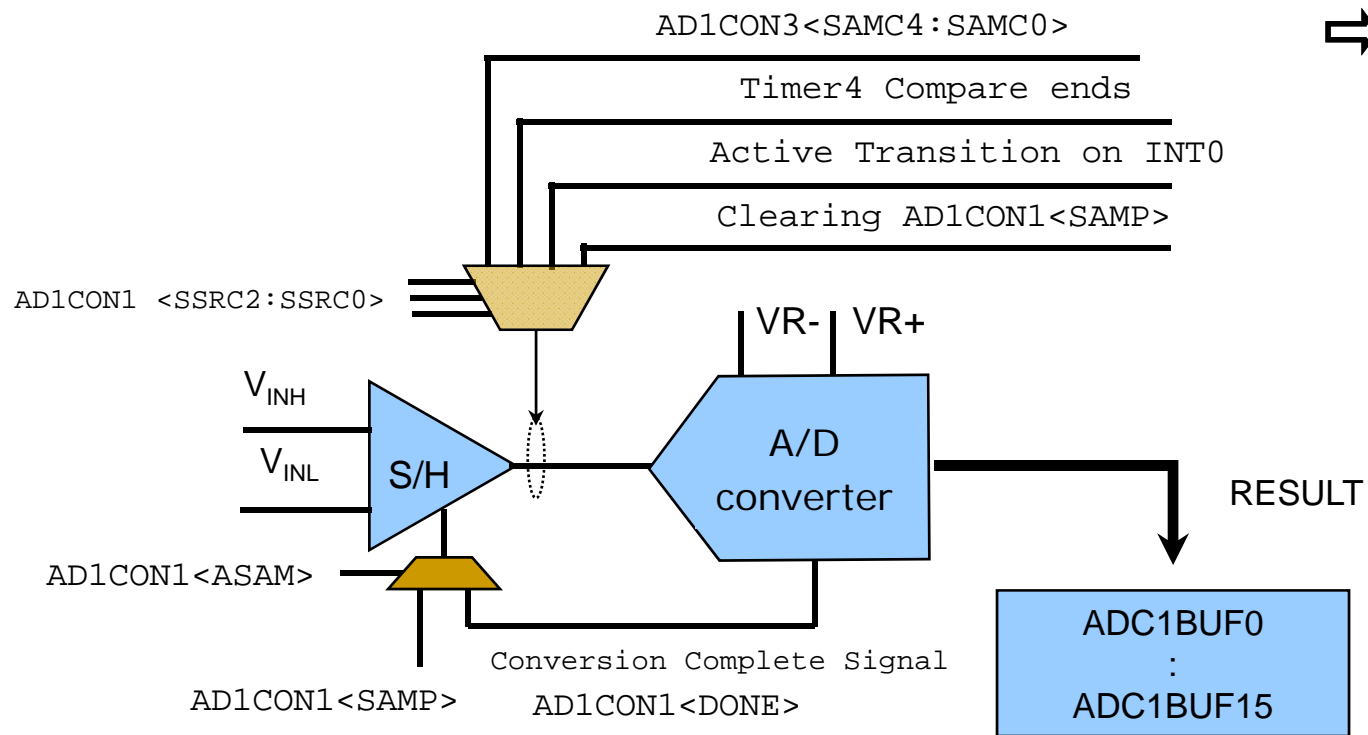
AD1CON2 Register



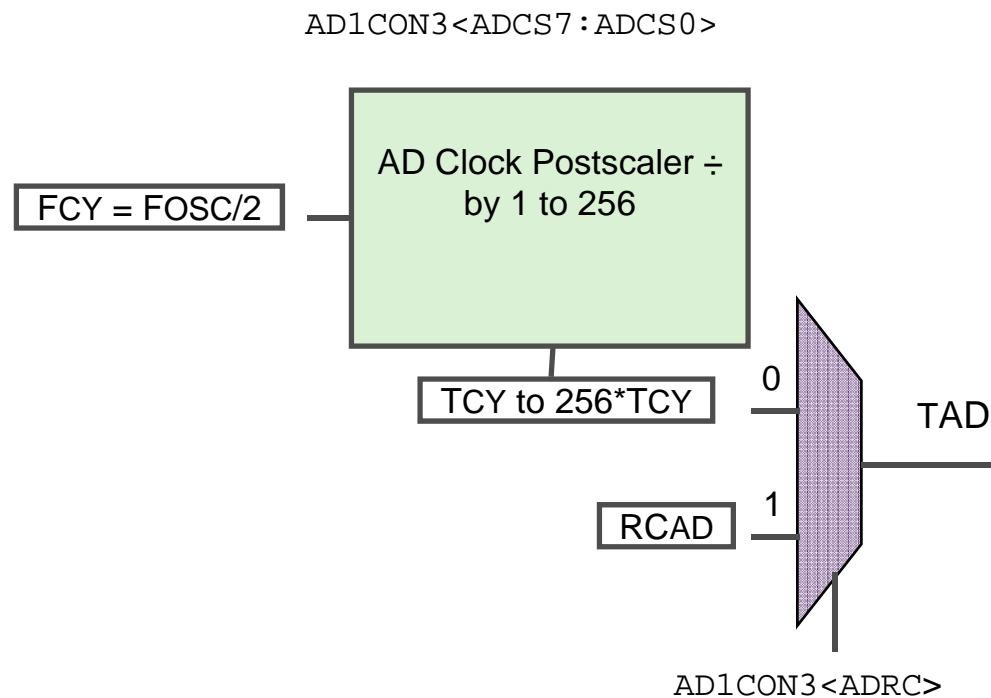
10-bit ADC Channel Scanning



10-bit ADC Sample Control



10-bit ADC Clock Selection



10-bit ADC Format Control

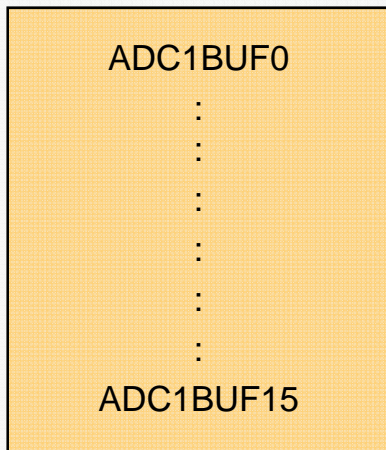
RESULT
FORMAT

```

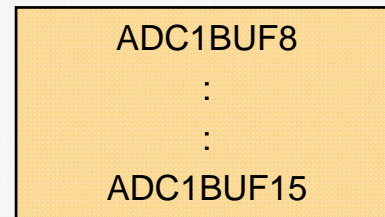
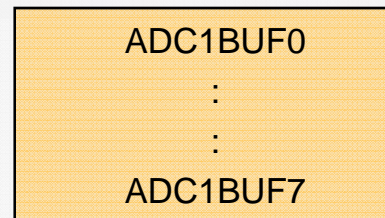
0000 00dd dddd dddd
ssss sssd dddd dddd
dddd dddd dd00 0000
sddd dddd dd00 0000
    
```

UNSIGNED INTEGER
SIGNED INTEGER
UNSIGNED FRACTIONAL
SIGNED FRACTIONAL

AD1CON1<FORM1:FORM0>



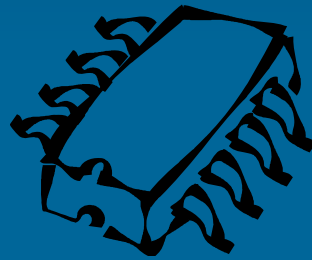
AD1CON2<BUF0M> = '0'



AD1CON2<BUF0M> = '1'

Lab 4

Line Detection Using the ADC Peripheral



Lab 4

Line Detection Using The ADC Peripheral



Purpose

The purpose of this lab is to experiment with the Analog-to-Digital Converter (ADC) peripheral, calibrating the given firmware to enable the opto-sensors to reliably detect the presence of the line.



Procedure

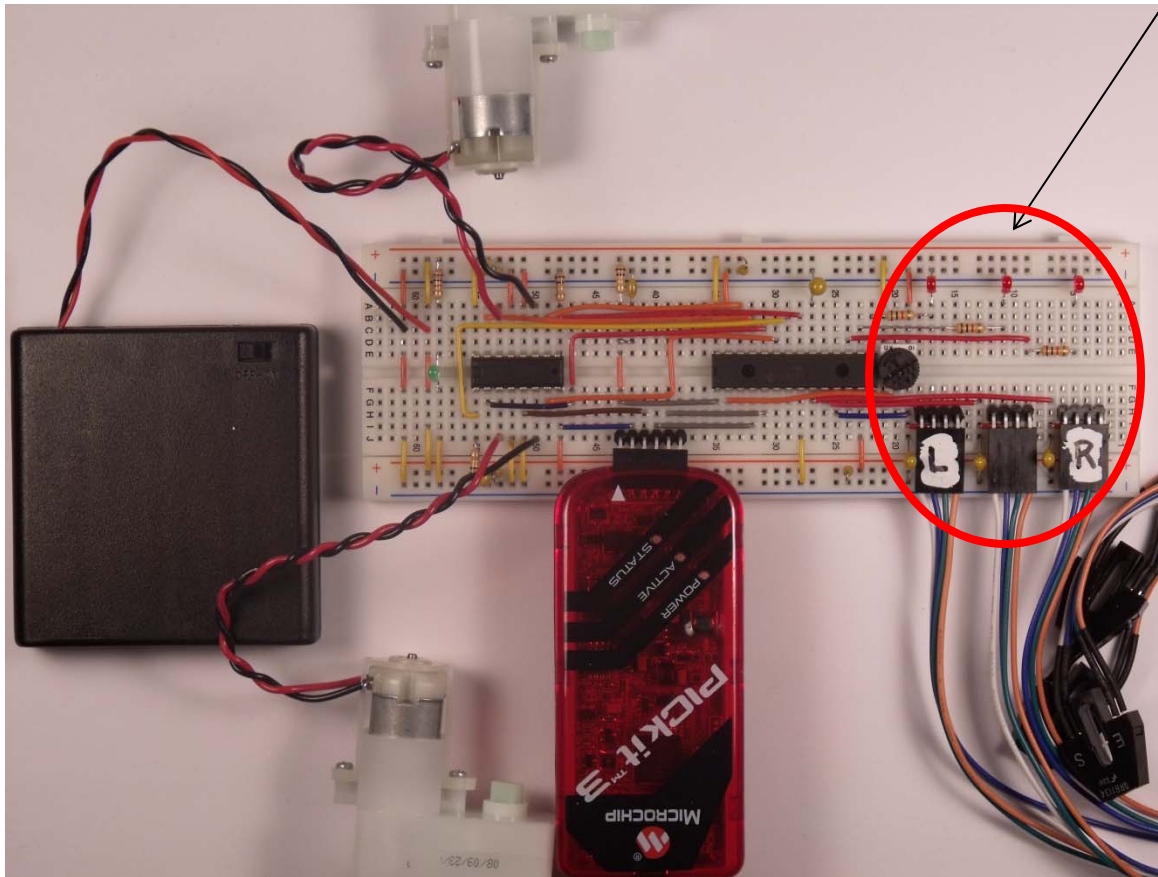
Connect the Opto-Sensors to the prototype
Determine the ideal trip voltage (difference between black & white)
Modify/Build/run the firmware. Test: Does it detect the line??
Follow the instructions in the lab manual

Lab 4

Line Detection Using The ADC Peripheral



Results



LEDs light when corresponding sensor is hovered over Black line.

Motors still spin per POT setting.

Lab 4

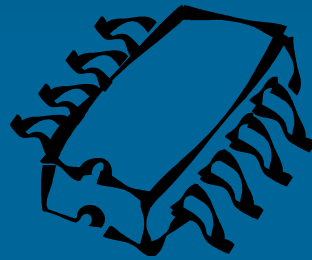
Line Detection Using The ADC Peripheral



Conclusions

- QRB1134 is an infrared opto-sensor which senses received IR and converts to an analog voltage output
- Analog-to-digital Converter on the PIC24 converts this signal to a number, scaled from 0-1023
- Calibration must be performed for simple line detection to compensate for varying lighting conditions.
- Implemented simple line-detection mechanism using the PIC24FV ADC.

Class Summary



Class Summary

- Learned the PIC24 Architecture and Programmer's Model and how to create C applications using the MPLAB PIC24 C-compiler
- Assembled the electronic control subsystem for the robot
- Learned how to drive DC motor using the PWM peripheral
- Learned how to use the ADC peripheral to measure the opto-sensor inputs and detect the line

References

Books – PIC24



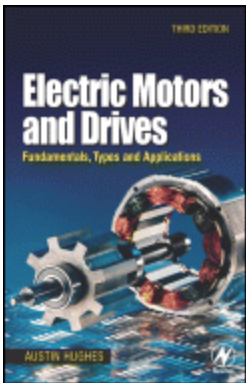
- Programming 16-Bit Microcontrollers in C
1st Edition (March, 2007)
Lucio Di Jasio
ISBN-10: 0-7506-8292-2
ISBN-13: 978-0-7506-8292-3



- Microcontrollers: From Assembly Language to C Using the PIC24
1st Edition (December, 2008)
Bryan Jones and Robert Reese
ISBN-10: 1584505834
ISBN-13: 978-1584-505838
www.reesemicro.com

References

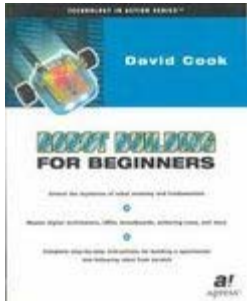
Books - Electric Motors



- Electric Motors and Drives
3rd Edition (December 2005)
Austin Hughes
ISBN-10: 0-7506-4718-2
ISBN-13: 978-0-7506-4718-2

References

Books – Basic Electronics & Robot Building



- Robot Building for Beginners
2nd Edition (2010)
David Cook
ISBN-10: 1-43022-748-6
ISBN-13: 978-1-43022-748-9
www.robotroom.com



- Intermediate Robot Building
2nd Edition (2009)
David Cook
ISBN-10: 1-59059-373-1
ISBN-13: 978-1-59059-373-8
www.robotroom.com

Other References

- PIC24FV32KA302 Family Data Sheet DS39995B
- MPLAB® C for PIC24 and dsPIC help files
- PIC24FV Family Reference Manual Sections
 - In “\Users Guides & Data Sheets\MCU”

**Thank
You!**