Microcontroller Overview

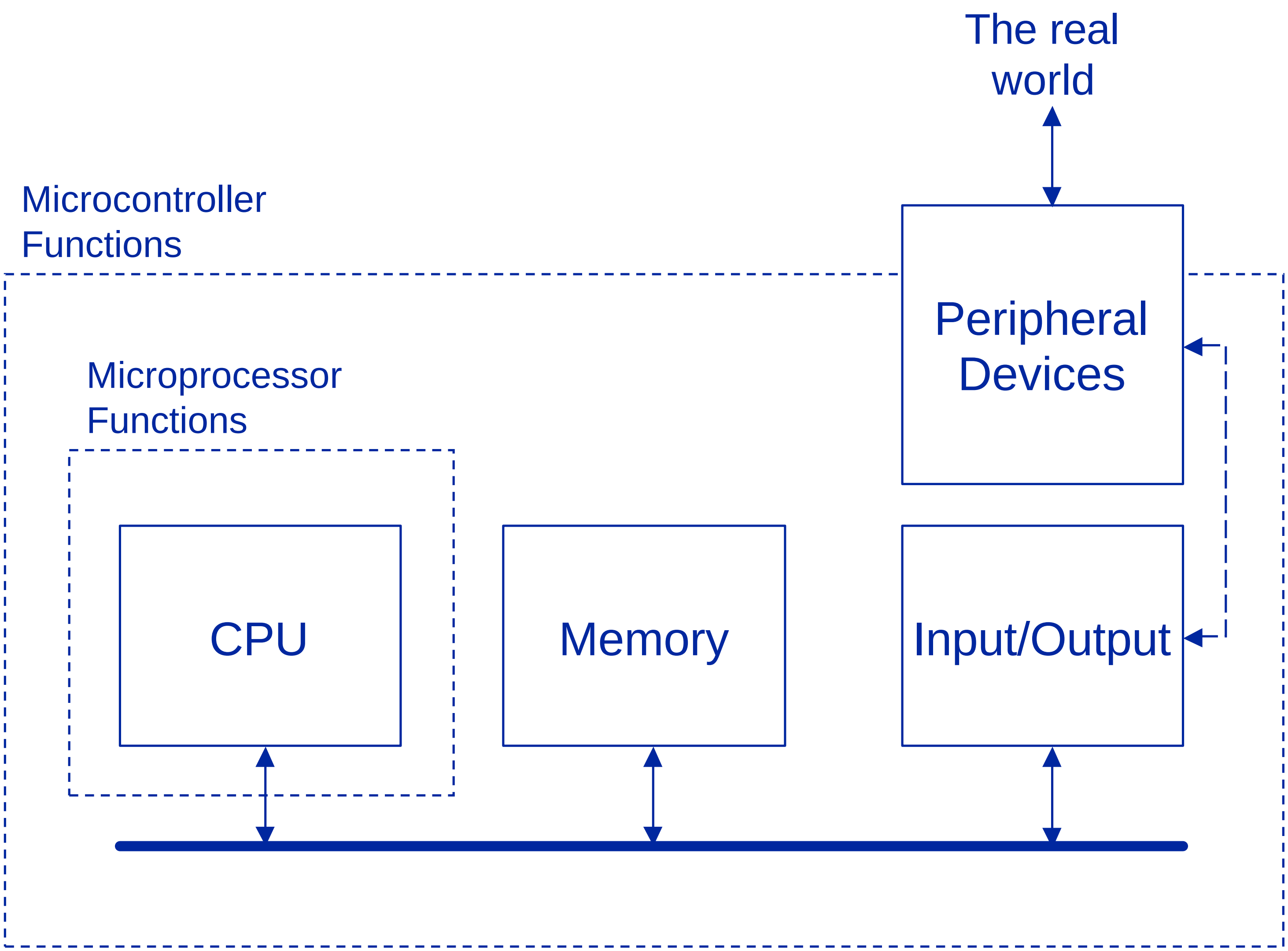
- Microprocessors/Microcontrollers/DSP
- Microcontroller components
  - Bus
  - Memory
  - CPU
  - Peripherals
- Programming

Microcontrollers vs. μproc. and DSP

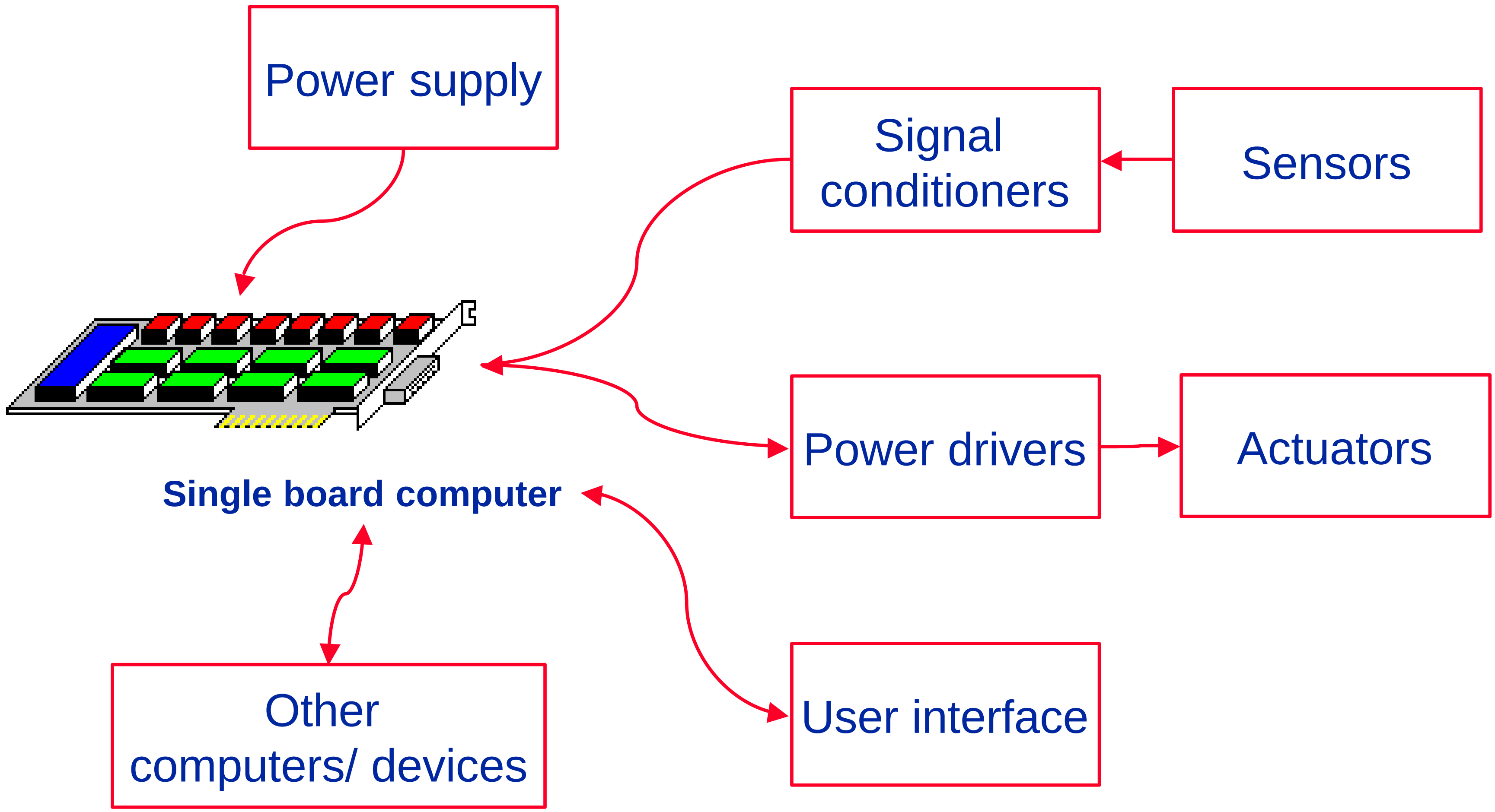
- **Microprocessors**
  - High-speed information processing
  - High-speed standard digital I/O and communication
  - Large memory space
  - Flexible architecture (e.g. DMA, PIC, IDE, etc.)
- **Microcontrollers**
  - General purpose parallel and serial I/O
  - Special functions (ADC, timers, drivers)
  - High-speed flexible interrupts
  - Small amount of RAM, ROM
  - Low power
  - Cheap!

  Usually this is a single-chip solution!
- **DSP - Digital signal processors**
  - Fast recursive signal processing
  - Fast multiply and accumulate
  - Some include floating point

Typical Bus-Oriented Microcomputer



Embedded Controllers

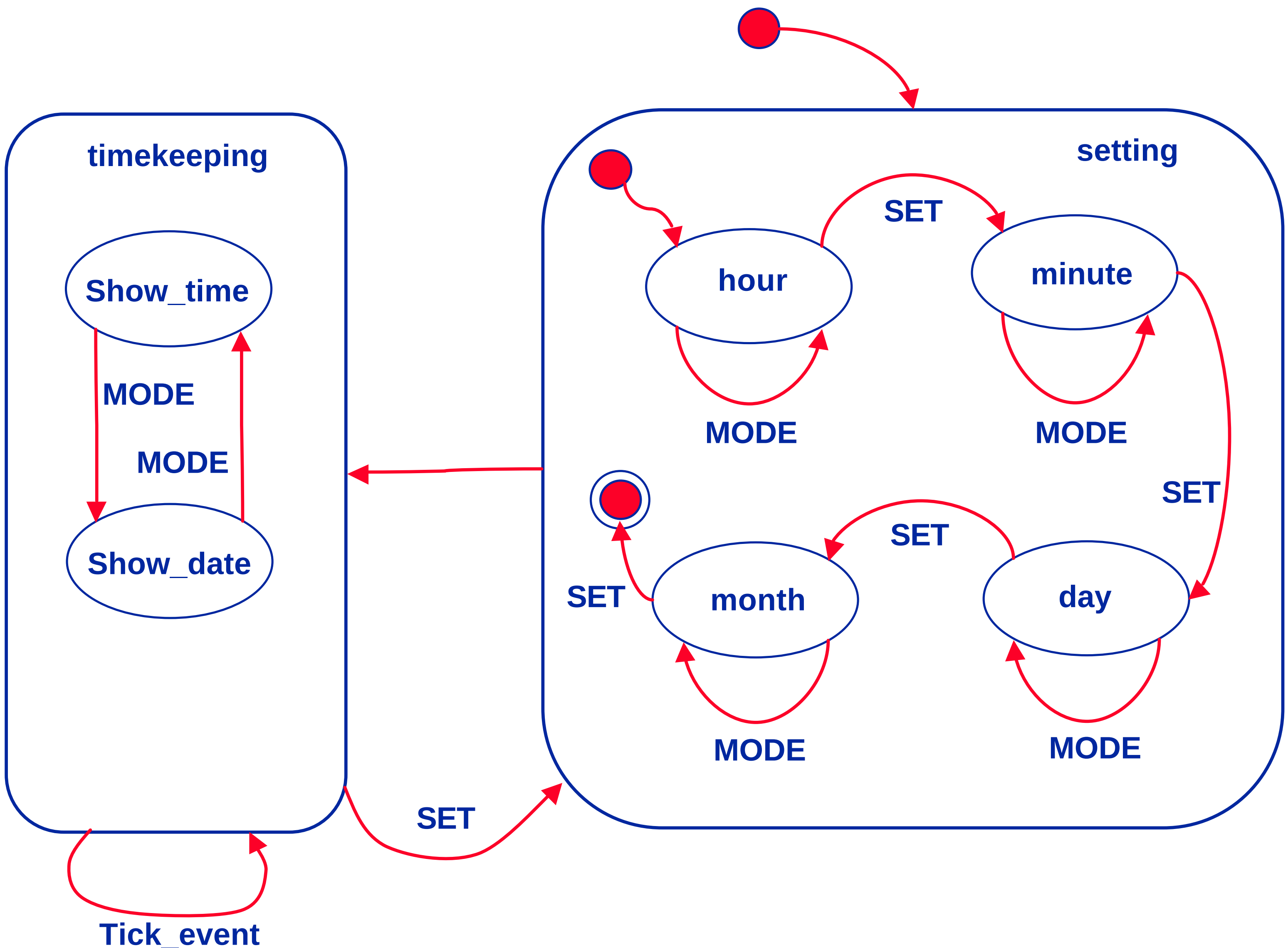


- **Choice of microprocessor or microcontroller**
  - **Driven by cost, power, reliability in application**
  - **Size becoming less of a factor in choice**

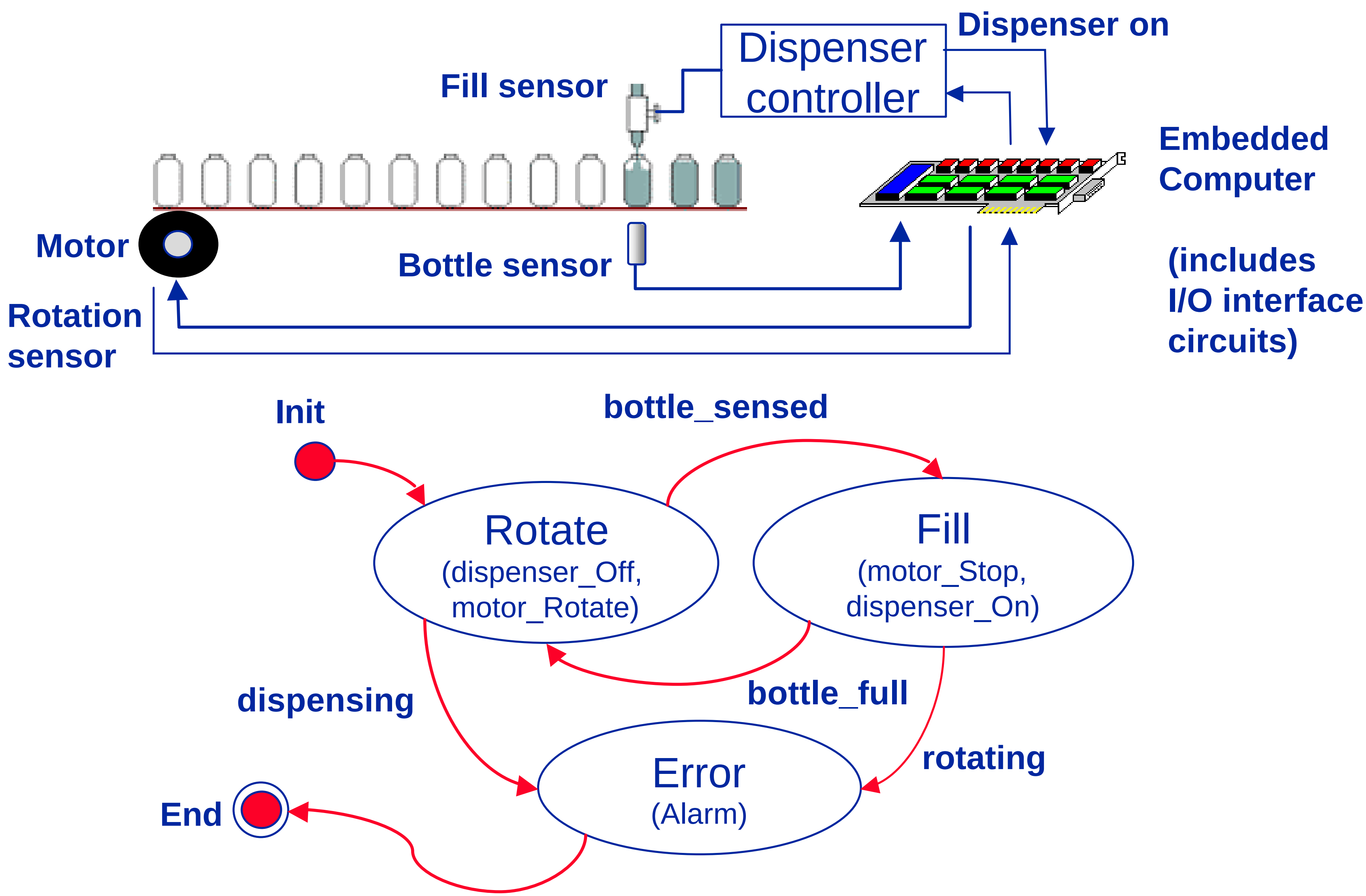
State Machines

- Useful abstraction for systems with discrete states
  - States are engaged based on events sensed in the world
- To set up a state machine:
  - Define the events
  - Define the states
  - Determine order of transition between states
  - Set the initial state of the state machine
- Example: let us consider a wristwatch

State Machine: Wristwatch



State Machines: Logic Control



State Machine Programming Example

```
enum State {Rotate, Fill,
            Error};
enum Event {bottle_sensed,
            bottle_full,
            dispensing,
            rotating};

void motor_Stop();
void motor_Rotate();
void dispenser_On();
void dispenser_Off();
void Alarm();

Static State s = Rotate;

                .

                .

                .
```

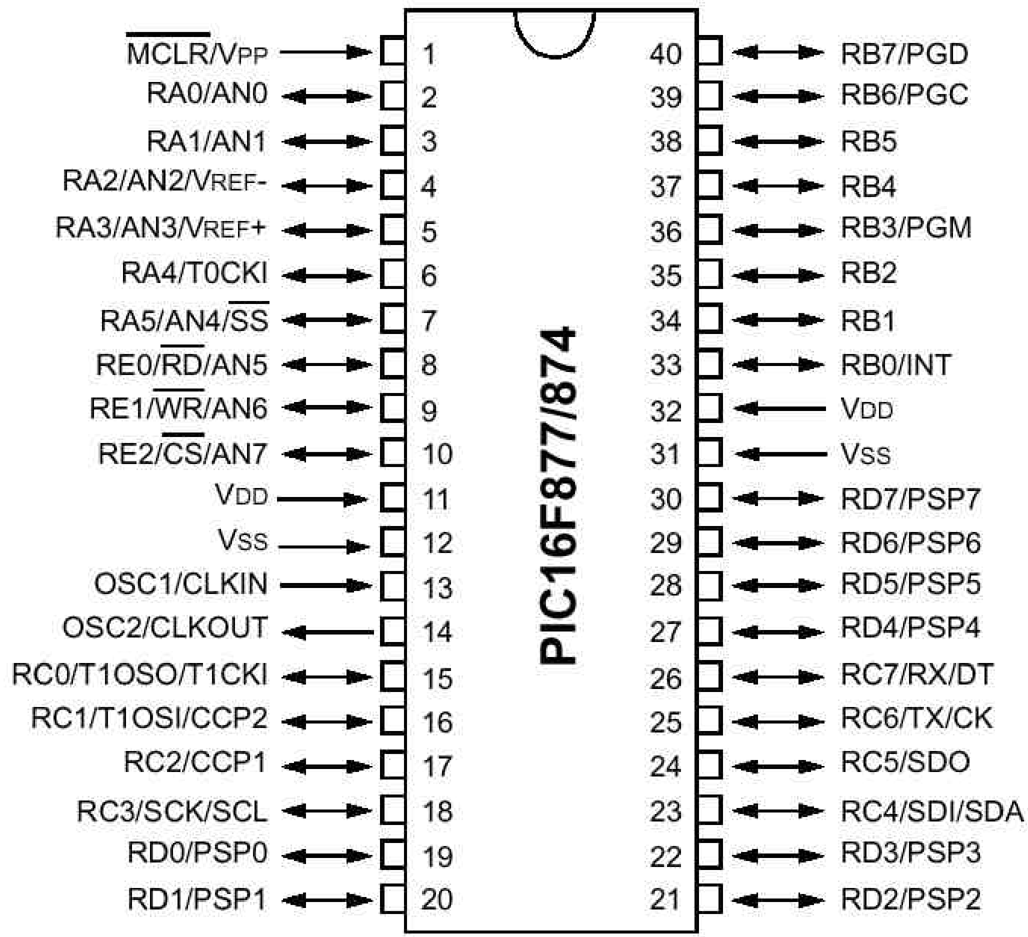
```
void Transition(Event e)
{ switch(s)
     {  case Rotate:
             switch(e)
             { case bottle_sensed:
                    s = Fill;
                    motor_Stop();
                    dispenser_On();
                    break;
               case dispensing:
                    s = Error;
                    Alarm();
                    break;
             } break;
        case Fill:
             switch(e)
             { case bottle_full:
                    s = Rotate;
                    dispenser_Off();
                    motor_Rotate();
                    break;
               case rotating:
                    s = Error;
                    Alarm();
                    break;
             } break;
        case Error:
               sleep();
               break;
     }
}
```

PIC 16F877 Features

- PIC16F877 microcontroller
  - Up to 20 MHz operation
  - 8k x 14 words FLASH program memory
  - 368 bytes internal SRAM, 256 bytes data EEPROM
  - 33 TTL digital I/O lines total
  - 10-bit multiplexed analog input module (8 channels)
  - 2 Capture/Compare/PWM modules
  - 3 timers
  - 14 interrupt sources
  - Serial communications: MSSP/USART
  - Parallel communications: PSP

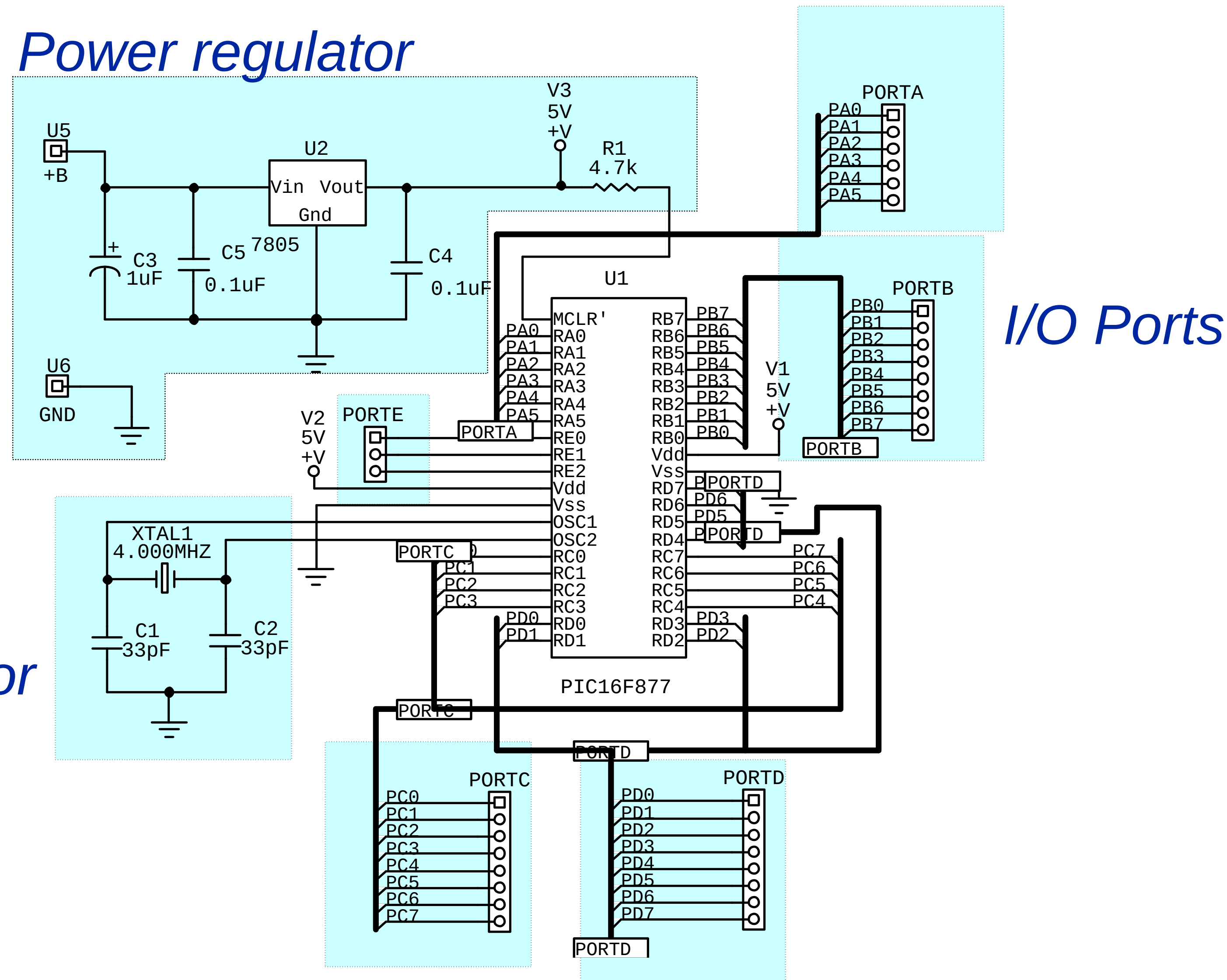
PIC 16F877

- **40 pin DIP**
- **Most port pins are multiplexed with alternate functional options**
  - **Timing inputs and outputs**
  - **Compare inputs and outputs**
  - **Interrupt inputs**
  - **Parallel I/O control**



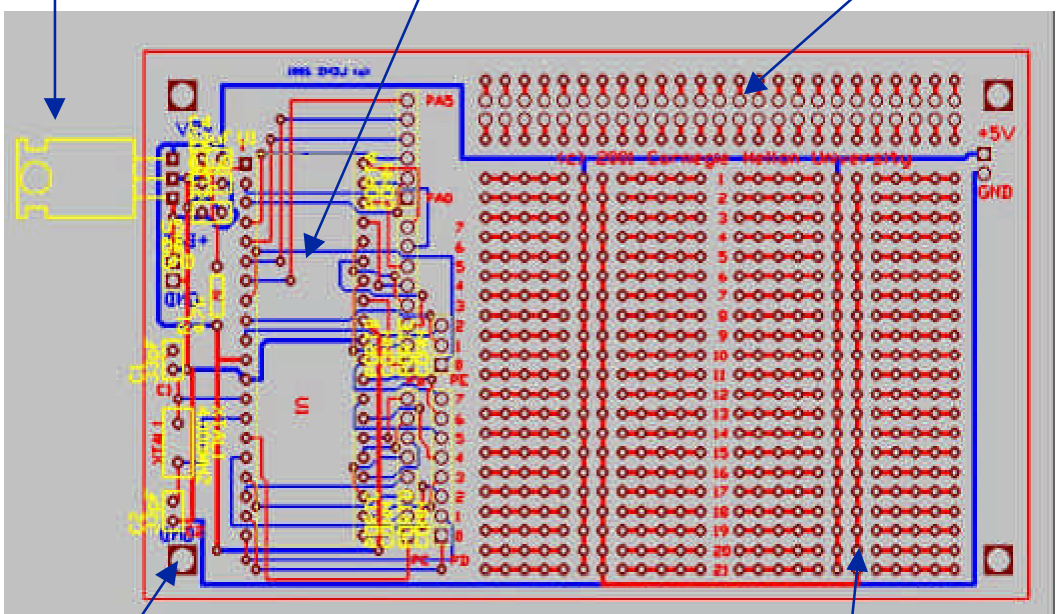
The PIC Prototype Board

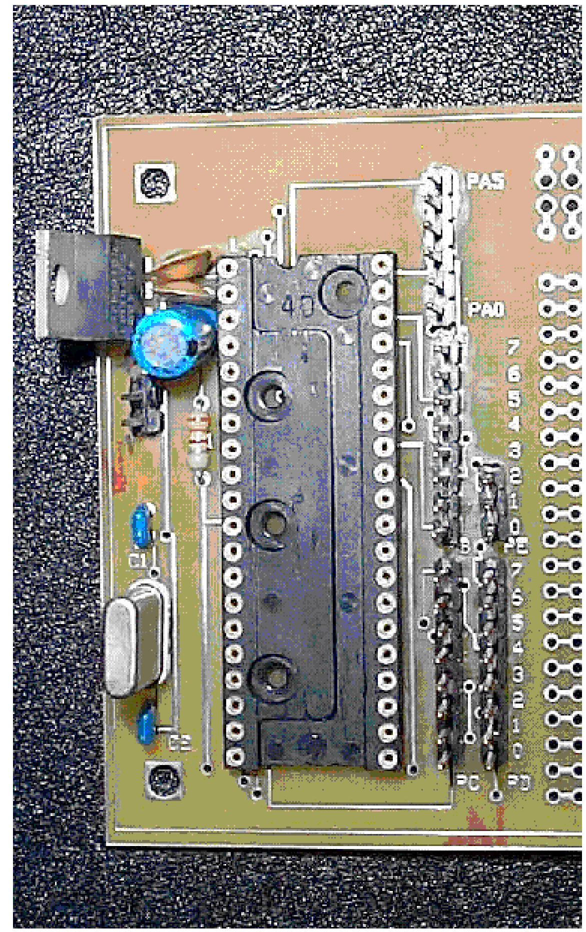
- **Runs at 4 MHz**
- **It also accepts the PIC16C74**



The PIC prototype board

- **Printed-circuit board size = 5.35” by 3.66”**





Electrical Characteristics

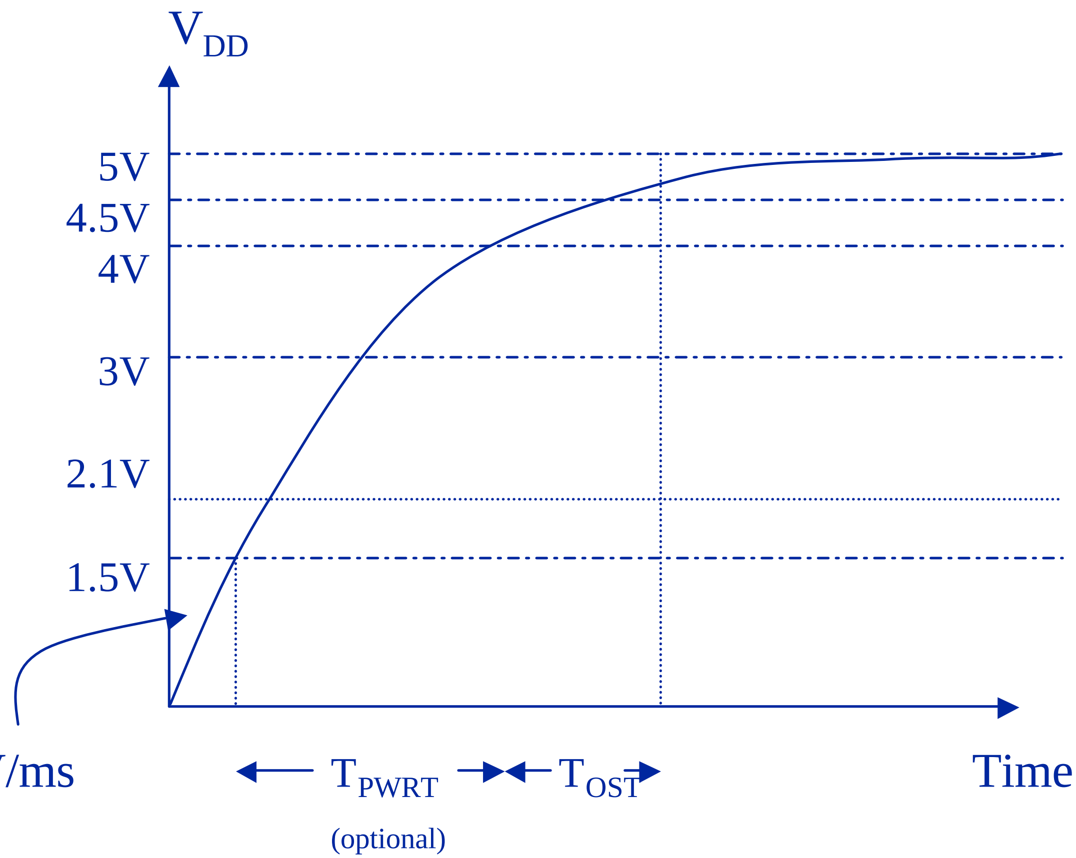
- Power
  - 5 V, 1 A regulator on board
  - < 2 mA typical (@ 4 MHz clock)
  - Microcontroller has a sleep mode
    - < 1 mA typical standby current
- Digital I/O current limit
  - High sink/source current: 25 mA
  - Don’t push it!

Reset Mechanisms

- The MCLR (master clear) pin
- Watchdog timer
- Power-on reset
- Brown-out reset

T_{PWRT} = 28 ms (min),
72 ms (typical)

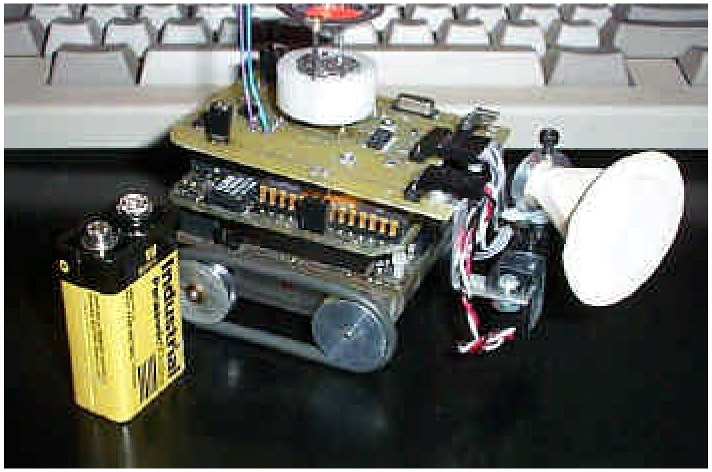
T_{OST} = 0 for RC oscillator
1024 T_{OSC} otherwise

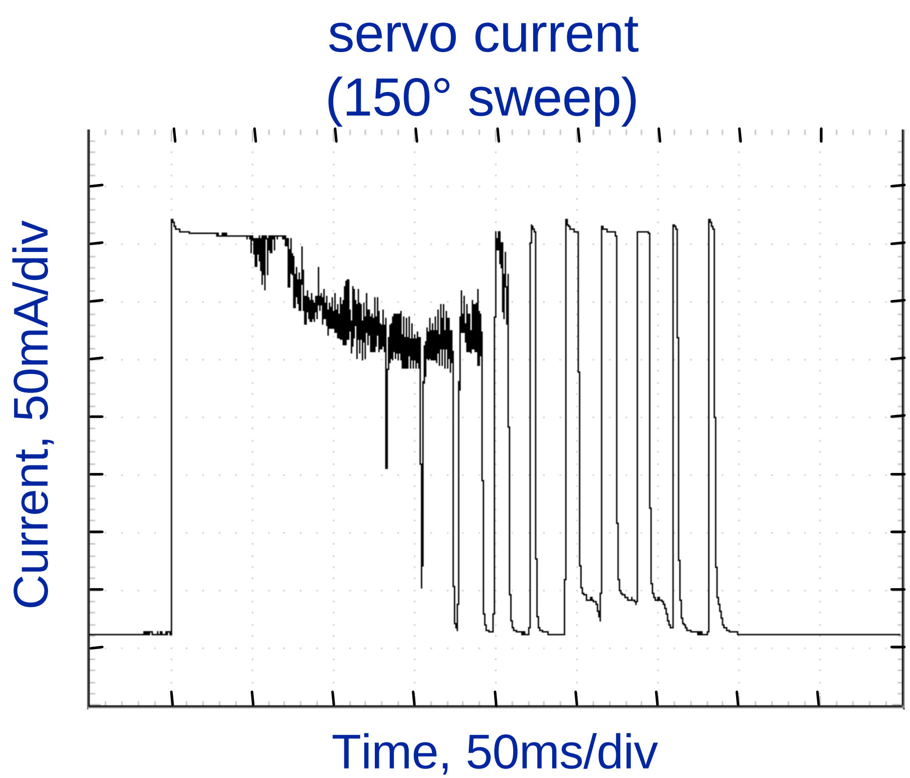


Initial rise time
must exceed 0.05V/ms

Beware of Current Transients!

- **Ensure that power suppyy has adequate current capability**
  - **Overcurrent will pull down voltage**
- **Use bypass capacitors across subsystems**





PIC Instruction Set

- RISC CPU
- Only 35 instructions to learn

- Anyway, we are going to use a C compiler!

C Compiler

- C Compiler / assembler from Custom Computer Services, Inc. (CCS) http://www.ccsinfo.com
  - Go from C program to .HEX file (Intel hex format), to be programmed into the microcontroller
- Integrated with MPLAB
  - Integrated Development Environment (IDE)
  - Execute and debug program
  - Step through program, set breakpoints, etc.
  - You can invoke PCM from MPLAB:
    - Create a new project
    - Select CCS as the LANGUAGE TOOL SUITE
    - Select .HEX file; click on NODE PROPERTIES; then select PCM as compiler
    - To compile, just BUILD the project

Development Cycle

- Design hardware architecture
- Define State Machine (or any other abstraction that represents the operation you want to accomplish)
- Iterative design:
  - Write code using C language
    - We use CCS's PCM C compiler
    - Don't need to use assembly language
  - Compile your code
  - Program firmware into the microcontroller
    - We use Microchip's MPLAB
  - Test your code in the prototype board
    - Include complete circuitry
  - Repeat as many times as required

PCM Compiler

- Subset of ANSI C
  - It has `typedef`, `float`, `struct`, etc.
  - `int` is 8 bits, `char` is 8 bits, `long` is 16 bits.
- Otherwise it looks exactly like C
- Some C I/O is supported
  - `printf, putc, getc, gets, puts`
- Built in Math Functions
  - `sqrt, sin, cos`
- Special functions for dealing with the microcontroller
  - `set_timer0`, `set_pwm1_duty`, etc.

What Does a Program Look Like?

```
#include <16F877.H>                                 // include file

#device PIC16F877 *=16 ADC=10                       // set device type
#fuses HS,NOWDT,NOPROTECT                           // set fuses for programming
#use delay(clock=4000000)                          // 4 MHz clock speed
#use rs232(baud=9600, xmit=PIN_A3, rcv=PIN_A2)      // set serial I/O lines
#define INTR_PER_SECOND 15                          // (4000000/(4*256*256))

byte seconds;      // A running seconds counter
byte int_count;    // Number of interrupts left before a second has elapsed

#int_rtcc                                // Following function is
clock_isr() {                            // RTCC (timer0) overflow interrupt (255->0)
    if(--int_count==0) {
      ++seconds;
      int_count=INTR_PER_SECOND;         // Interrupts occur approx. 15 times/sec
    }
}

main() {

   byte start;

   int_count=INTS_PER_SECOND;
   set_rtcc(0);
   setup_counters(RTCC_INTERNAL, RTCC_DIV_256);
   enable_interrupts(RTCC_ZERO);
   enable_interrupts(GLOBAL);

   do {

      printf("Press any key to begin.\n\r");
      getc();
      start = seconds;
      printf("Press any key to stop.\n\r");
      getc();
      printf("%u seconds.\n\r",seconds-start);

   } while (TRUE);

}
```

include and other pre-compiler statements

Interrupt service routine

Main program body

Digital I/O Ports

- **16F877 – Ports A, B, C, D, E**
- **Accessed through 8-bit registers**
  - **Least significant bit = b0**
  - **Most significant bit = b7**

MSb LSb

b7	b6	b5	b4	b3	b2	b1	b0

Byte

- **Mnemonics for registers are listed in include files**
  - **E.g., Port B is called “PORTB”**
- **Or use #byte pre-compiler directive**
  - **#byte PORTB 6**

    // sets PORTB to the address 06h

Digital I/O Ports

- Setup the port's data direction register
  - TRISA, TRISB, TRISC, TRISD, TRISE
  - 1=input; 0=output for each bit
- For example:

```
#byte B_Port = 6
set_tris_b(0);
B_Port = 0;
```

- Some port pins have additional functionality
  - I/O may be overridden by other register bits

Bitwise Operations

- **Operators**
  - **& (AND)**
  - **| (OR)**
  - **^ (XOR)**
  - **~ (NOT)**
  - **<< (LEFT SHIFT)**
  - **>> (RIGHT SHIFT)**
  - **&=, |=, ^=, <<=, >>=**
- **To read a single bit, use AND mask**

```
if(PORTB & 04)
     puts("pin 2 high!\n");
```

- **To write a single bit, use OR mask**

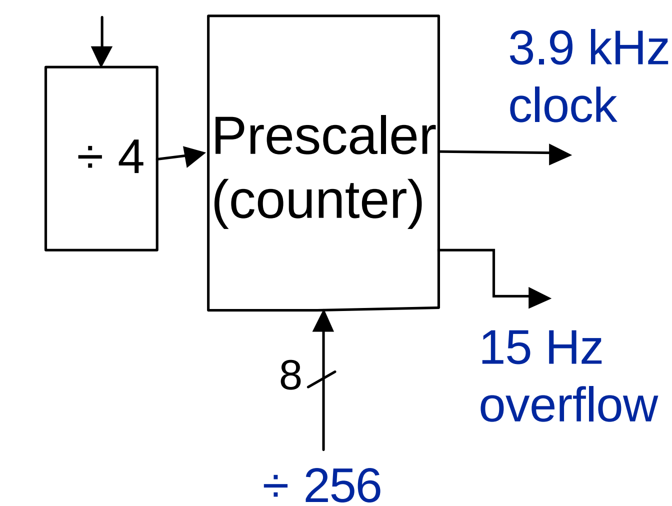
```
PORTB = PORTB | 1;
```

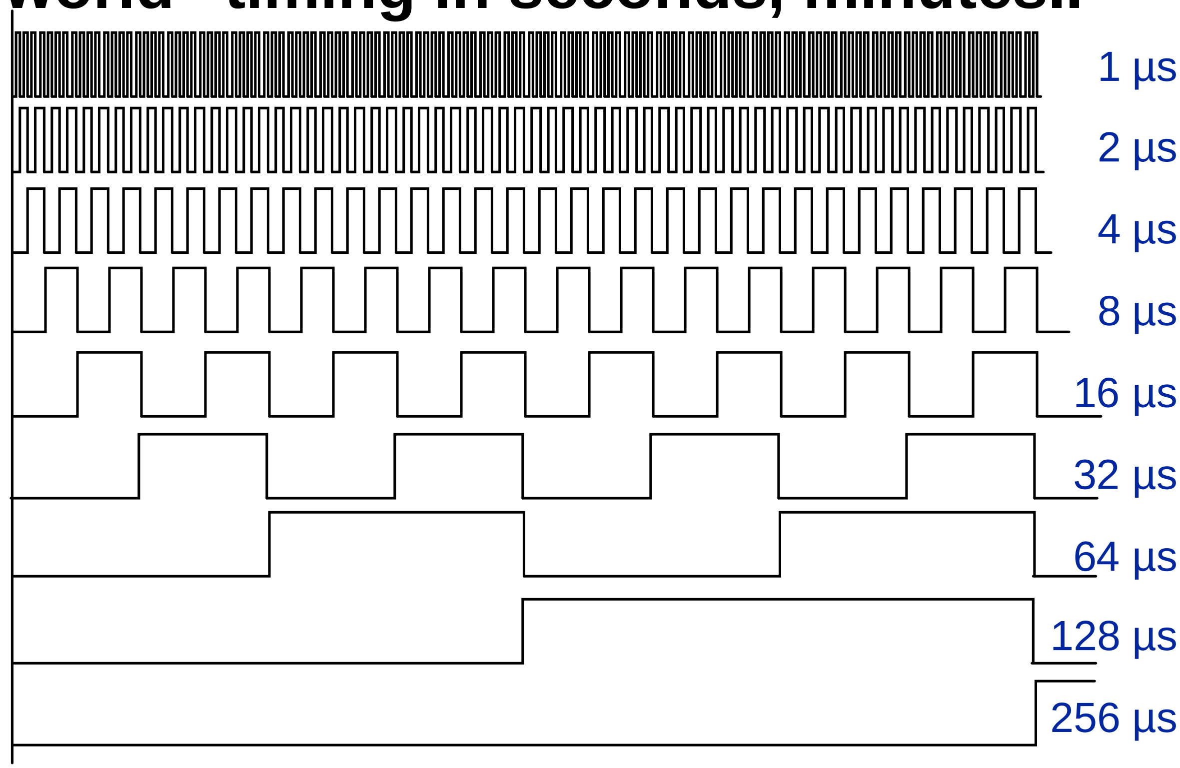
Timers

- 3 timers:
  - Timer0: 8-bit timer/counter with 8-bit prescaler
  - Timer1: 16-bit timer/counter with prescaler
  - Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Four basic functions:
  - timing input waveforms (input capture)
  - generating timed output waveforms (output compare)
  - generating PWM signals
  - counting pulses

Timers

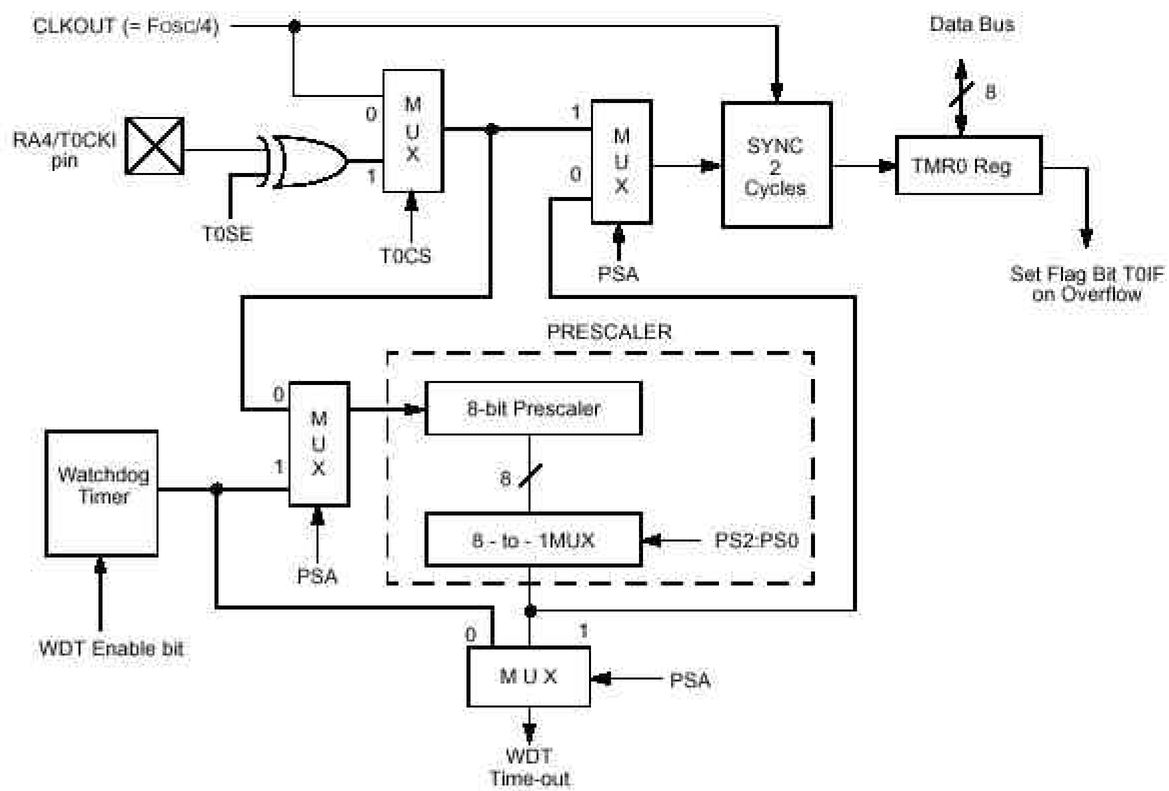
- **Master clock set by external crystal**
  - **20 MHz max, we typically set to 4 MHz**
- **Programmable timer prescaler**
  - **Divides clock down by binary fraction**
  - **Useful for “real world” timing in seconds, minutes..**
  - **Overflow at 256 counts**





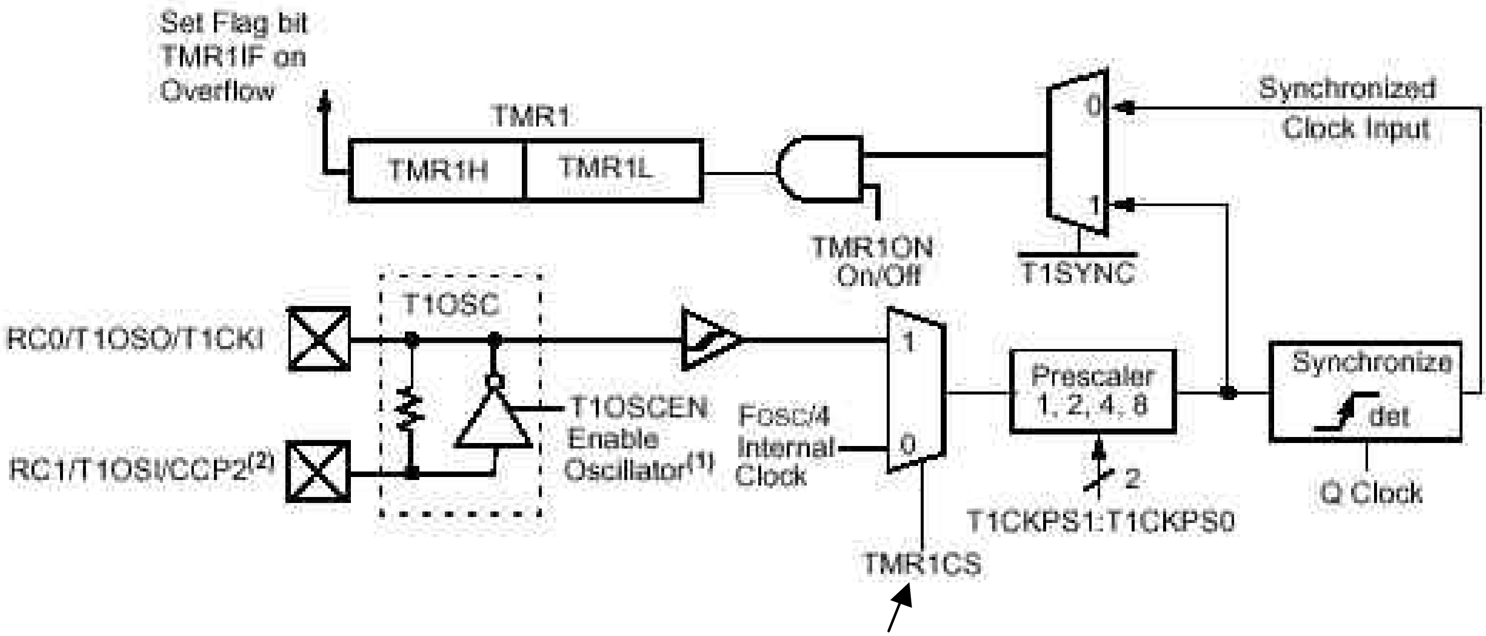
Timer 0

- Timer or counter
- Control register is OPTION_REG
- 8-bit prescaler
  - Divide by 2 up to 256



Timer 1

- 16 bit timer
- Acts as a timer or a counter



0 = timer
1 = counter

Timer 2

- 8 bit timer
- PWM clock
- Prescale options 1/1, ¼, 1/16
- PR2, “period” register
- Postscaler options 1/1 to 1/16