Introduction to Embedded Systems

Lecture 1

These lecture notes created by Alex Dean, NCSU

Today

What Are Embedded Systems?

Why Are We ….?

Course Overview

Introduction to Microcontroller-based Circuit Design

Definition of an Embedded Computer

Computer purchased as part of some other piece of equipment

- Typically dedicated software (may be user-customizable)
- Often replaces previously electromechanical components
- Often no “real” keyboard
- Often limited display or no general-purpose display device

But, every system is unique -- there are always exceptions

A Customer View

Reduced Cost

Increased Functionality

Improved Performance

Increased Overall Dependability

Microcontroller and Starter Kit

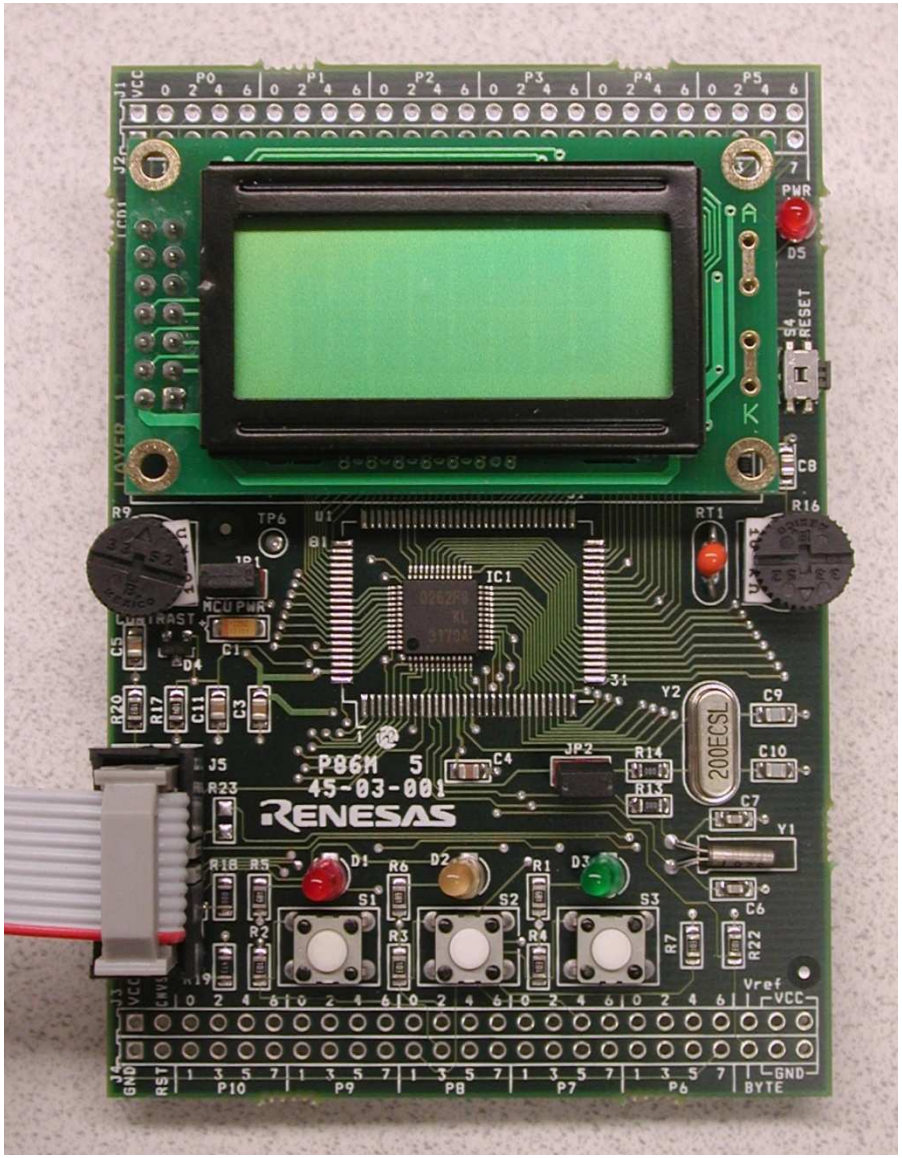
Mitsubishi Semiconductors is now “Renesas”

M16C/26 family of microcontrollers

- M30626
- 32K RAM, 384K Flash

SKP = starter kit

MDS = Microcontroller Data Sheet = M16C_Hardware_Manual_Rev0.9.pdf



M16C Demonstration

Configuring I/O ports

- Input: switches
- Output: LEDs

Echo switch state on LEDs

Human response time analysis

- How quickly can a person press a switch after seeing a light?

Processor response time analysis

- How quickly does the processor respond to the switch being pressed?

Processor speed evaluation

- How much work can the processor do in that time?

Source Code

```
#include "stdio.h"
#include "sfr626.h"
#include "SKP_LCD.h"

#define RED_LED (p8_0) /* from board data sheet */
#define YEL_LED (p7_4)
#define GRN_LED (p7_2)

#define LED_ON  (0)   /* 0 is ON for LEDs (active-low) */
#define LED_OFF (1)

#define DIR_IN  (0)
#define DIR_OUT (1)

#define SW1 (p8_3)
#define SW2 (p8_2)
#define SW3 (p8_1)

void init_switches() {
  pd8_1 = pd8_2 = pd8_3 = DIR_IN;
}

void init_LEDs() {
  pd8_0 = pd7_4 = pd7_2 = DIR_OUT;
  RED_LED = YEL_LED = GRN_LED = LED_ON;
  RED_LED = YEL_LED = GRN_LED = LED_OFF;
}

void test_switches(void) {
  while (1) {
    RED_LED = (!SW1)? LED_ON : LED_OFF;
    YEL_LED = (!SW2)? LED_ON : LED_OFF;
    GRN_LED = (!SW3)? LED_ON : LED_OFF;
  }
}
```

```
void main () {
  char buf[9];
  long int i, r=12345;

  init_switches();
  init_LEDs();
  InitDisplay();
#if (1)
  test_switches();
#endif
  DisplayString(LCD_LINE1, "Response");
  DisplayString(LCD_LINE2, " Timer  ");

  while(1) {
    for (i=0; i<200000+(r%50000); i++)
      ;
    i=0;
    RED_LED = YEL_LED = GRN_LED = LED_ON;
    while (SW1)
      i++;

#if (1)
    sprintf(buf, "%8ld", i);
    DisplayString(LCD_LINE1, buf);
    DisplayString(LCD_LINE2, "iters.  ");
#else
    sprintf(buf, "%8.3f", i*39.1/287674);
    DisplayString(LCD_LINE1, buf);
    DisplayString(LCD_LINE2, "millisec");
#endif
    RED_LED = YEL_LED = GRN_LED = LED_OFF;
    r=0;
    while (!SW1) /* wait for switch to come up */
      r++;
  }
}
```

Why Are We Using Such a Small Processor?

I've learned that 8 bits will never go away. Ever. Analysts and pundits have told me they see 8 and 16 bits disappearing over the next year or two, but developers disagree. I'm convinced we're on the brink of an explosion in embedded systems, with embedded processing filling every conceivable niche in our lives. Some of this will be the Internet appliances whose hype saturates all media channels. Much more will be tiny bits of processing, from smart tools to clever pens and intelligent wires. None of these needs a 32 bit monster.

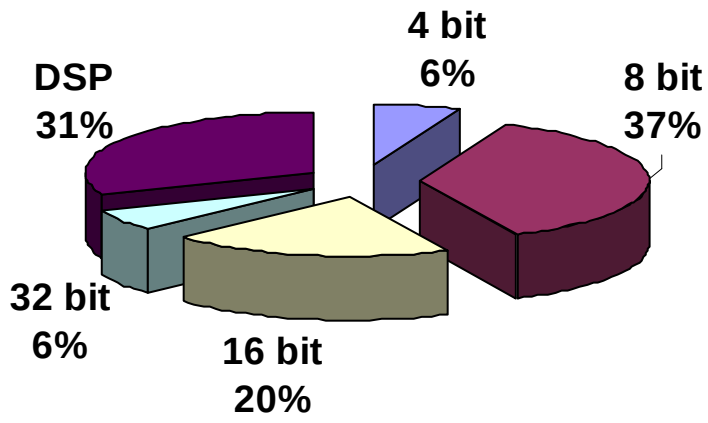
Jack Ganssle

Small Computers Rule The Marketplace

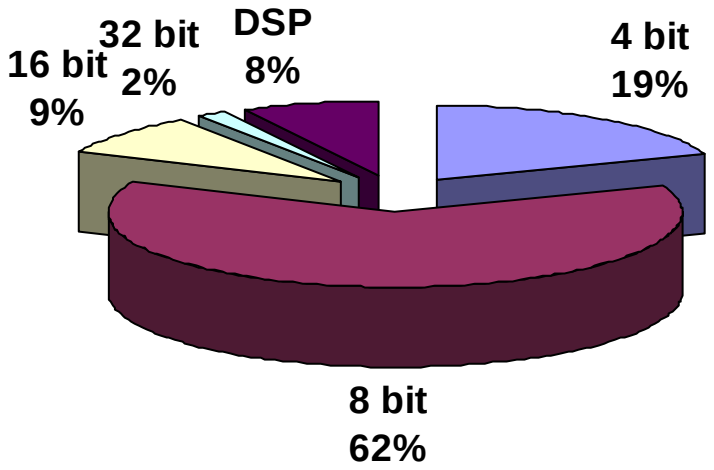
Embedded market growing (revenue up 17% - 30% per year)

PC market is saturated (US revenue 12/2000 down 30% from 12/1999)

Global Microcontroller Market Share by Revenue (1999)



Global Microcontroller Market Share by Volume (1999)



Options for Implementing Embedded Systems

	Implementation	Design Cost	Unit Cost	Upgrades & Bug Fixes	Size	Weight	Power	System Speed
Dedicated Hardware	Discrete Logic	low	mid	hard	large	high	?	very fast
	ASIC	high ($500K/ mask set)	very low	hard	tiny - 1 die	very low	low	obscenely fast
	Programmable logic – FPGA, PLD	low	mid	easy	small	low	medium to high	very fast
Software Running on Generic Hardware	Microprocessor + memory + peripherals	low to mid	mid	easy	small to med.	low to moderate	medium	moderate
	Microcontroller (int. memory & peripherals)	low	mid to low	easy	small	low	medium	slow to moderate
	Embedded PC	low	high	easy	medium	moderate to high	medium to high	moderate

Course Overview

Introduction to Embedded Systems

M30626 Processor

- M16C Instruction Set Architecture
- Circuit Design

Programming

- Assembly Language Programming
- C Programming Review
- C and the Compiler

Software Development

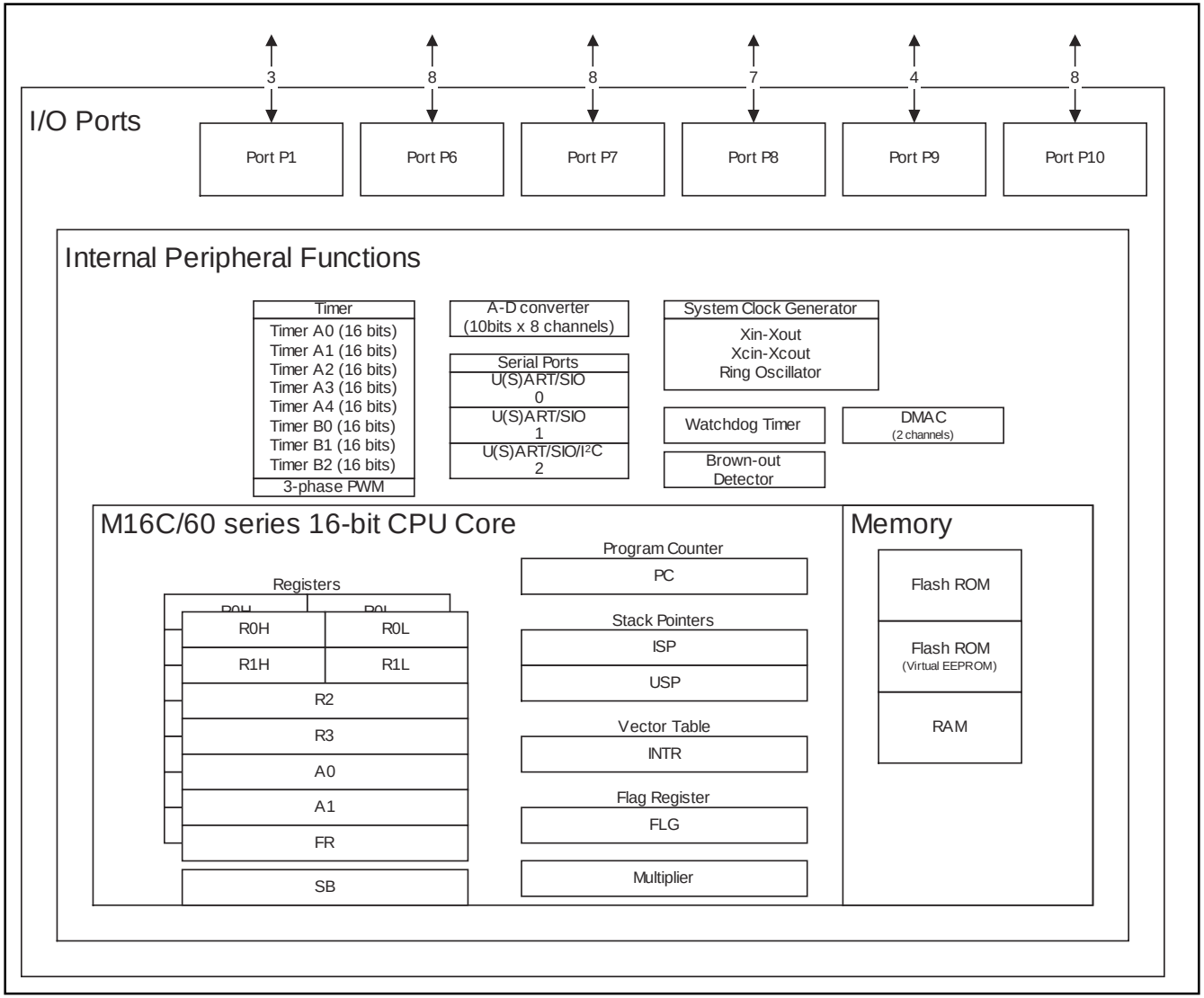
- Debugging
- Simulation Design and Debugging

- Interfacing
  - Using and Programming Interrupts
  - Digital I/O Peripherals: General Purpose, T/C and PWM
  - Analog I/O Peripherals
  - Serial Communications and Peripherals
- Optimizations
  - Performance Analysis
  - Power Analysis
- Multithreaded Systems
  - Threads, Tasks and Simple Scheduling
  - Real-Time Operating Systems
  - Threaded Program Design

Microcontroller vs. Microprocessor

Microcontroller has peripherals for embedded interfacing and control

- Analog
- Non-logic level signals
- Timing
- Communications
  - point to point
  - network
- Reliability and safety



Designing a Microcontroller into a System

Power supply

Clock signal generator

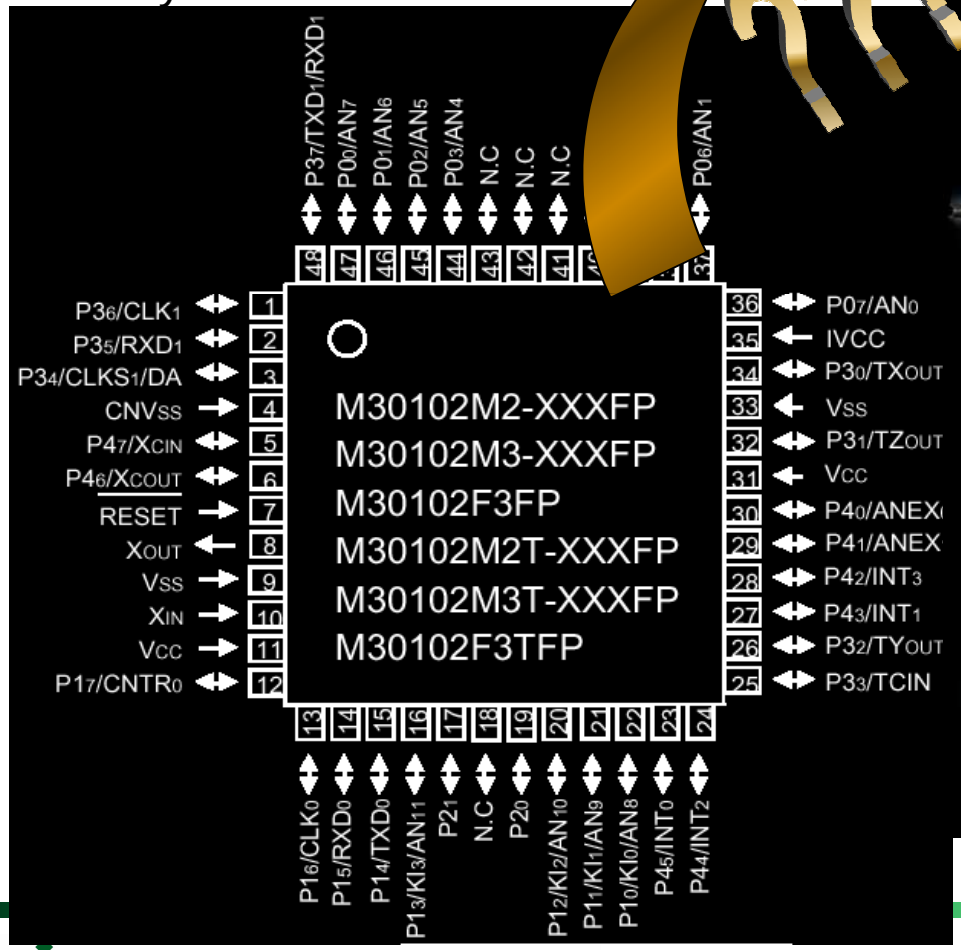
Reset controller

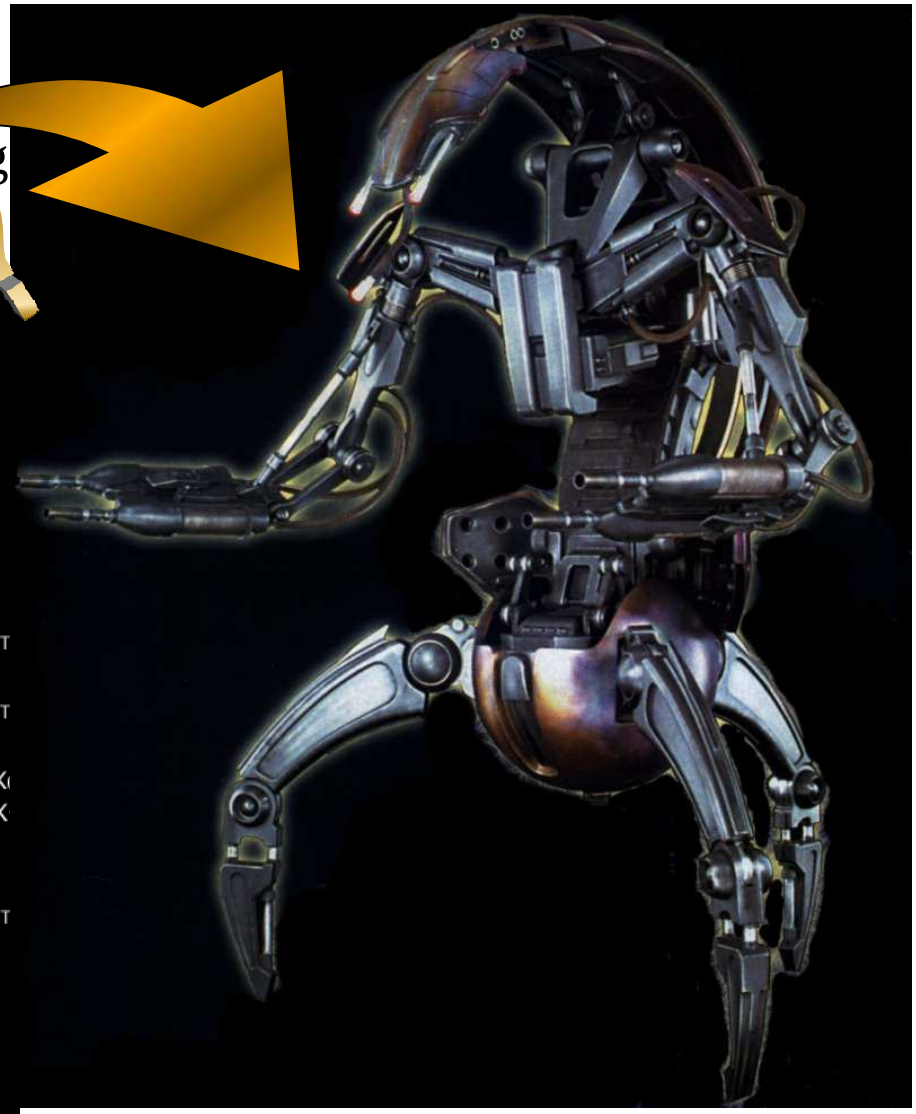
Memory

Digital interfacing

Analog interfacing

Communications





Power Supply

What do we need (voltage and current)? Look at datasheet for voltage and current for microcontroller and other circuits

- Table 1.18.2 Recommended Operating Conditions (p. 171-MDS)
  - Supply voltage: 2.7 V <= V_{CC} <= 5.5 V
- Table 1.18.5 Electrical Characteristics (p. 173 - MDS)
  - Supply current typically 28.0 mA, max 38.0 mA (V_{CC} = 5.0 V, $T_{Ambient}$ = 25 C, f(X_{IN}) = 20 MHz)
- Don't confuse with Table 1.18.1 **Absolute Maximum Ratings**

Where do we get the power? Plug into wall or use batteries

- Wall - Need to drop 120 VAC to 5 VDC, use big power supply
  - Transformer-based
  - Switching

Batteries

Battery ≡ >1 cell

Cell can be modeled as ideal voltage source with a series resistance

- Series resistance induces a voltage drop as current rises

How long will it last?

- Cells can be modeled as having a constant capacity (1 amp-hour = 3600 coulombs = 3600 amp-seconds) *(less accurate)*
  - Battery life (hours) = capacity (amp-hours)/current (amps)
- Can also predict life based on discharge plot *(more accurate)*

What if voltage or current isn't right?

- Can put cells in series (add voltages) or parallel (add currents)
- Can use a voltage regulator (linear or switch-mode)

Battery Power

A 800 mAhr battery will power a device that draws 200mA for how long?

800 mAhr / 200mA = 4 hr

Practice: 720 mAhr cell phone battery will power a phone that draws 4 mA average for how long?

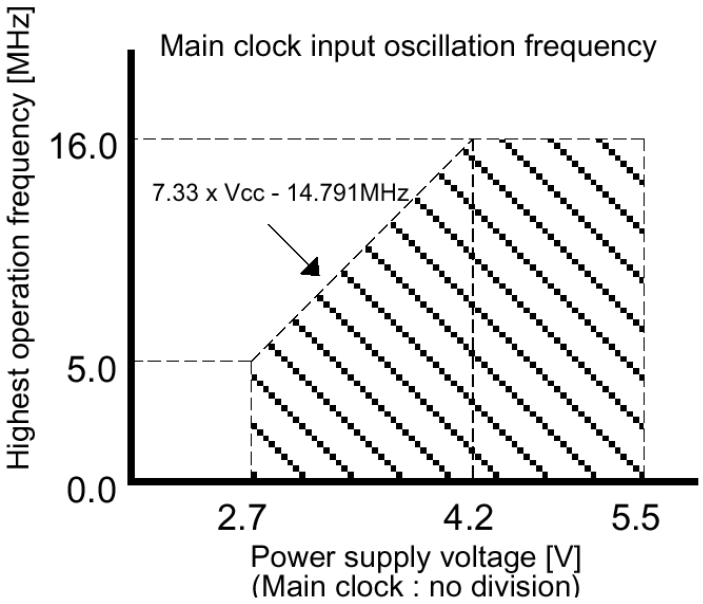
Clock Signal Generator

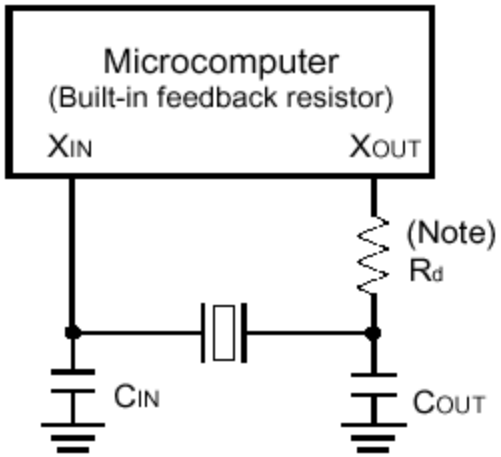
Why? To make the logic run!

Max. frequency? See MDS p. 171 (similar graph)

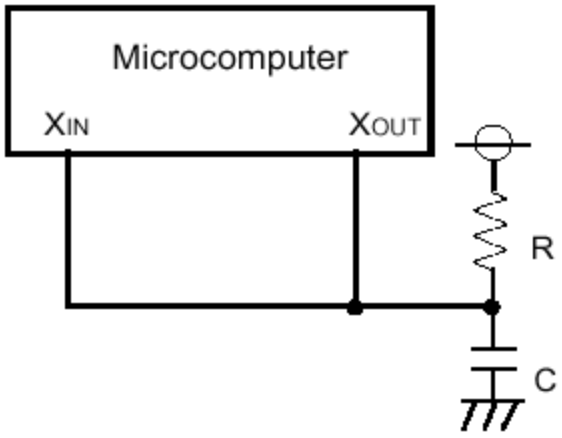
Sources

- External crystal or ceramic oscillator
- External RC oscillator
- External clock ($)

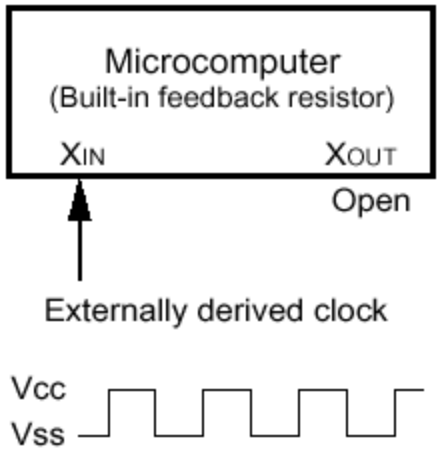




External ceramic oscillator



External RC oscillator

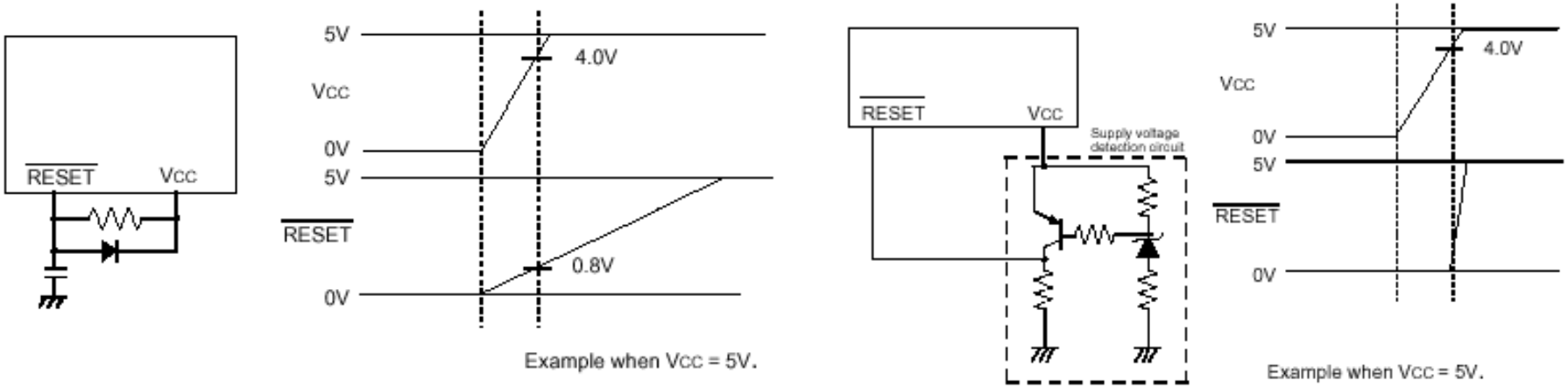


External clock input

Reset Controller

Why? So the processor starts off in a predictable state (e.g. program start address, operating modes…)

- Reset processor whenever
  1. Power supply voltage drops below a threshold
  2. Something catastrophic happens
- M30626: Hold ~RESET low for >= 20 cycles of ring osc.
- Solutions (MDS p.12)
  - Can also use reset controller IC



Result of Reset

System control registers are initialized to predefined values

Listed in full on MDS p.19-22

Address	Register Name	Acronym	Value after Reset
0000_{16}			
0001_{16}			
0002_{16}			
0003_{16}			
0004_{16}	Processor mode register 0	PM0	0 0 0 0 0 0 0 0
0005_{16}	Processor mode register 1	PM1	0 0 0 0 1 0 0 0
0006_{16}	System clock control register 0	CM0	0 1 0 0 1 0 0 0
0007_{16}	System clock control register 1	CM1	0 0 1 0 0 0 0 0
0008_{16}			
0009_{16}	Address match interrupt enable register	AIER	0 0
$000A_{16}$	Protect register	PRCR	0 0 0 0 0 0
$000B_{16}$			
$000C_{16}$	Oscillation stop detection register	CM2	0 0 0 0 0 0 0 0
$000D_{16}$			

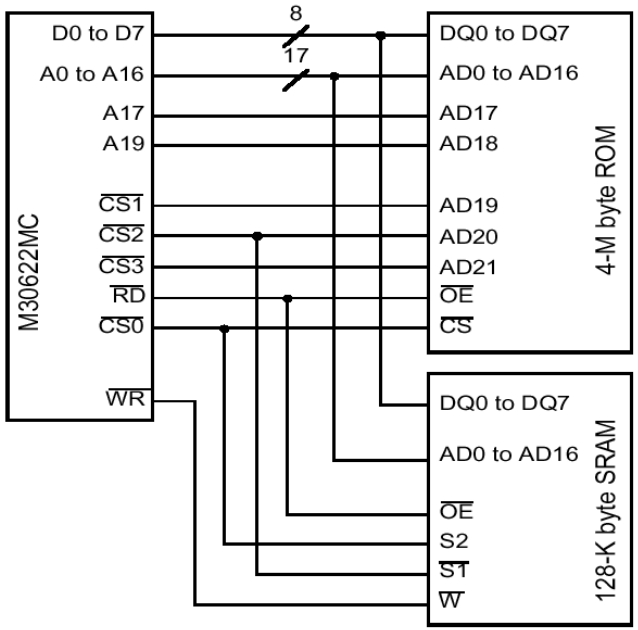
et cetera

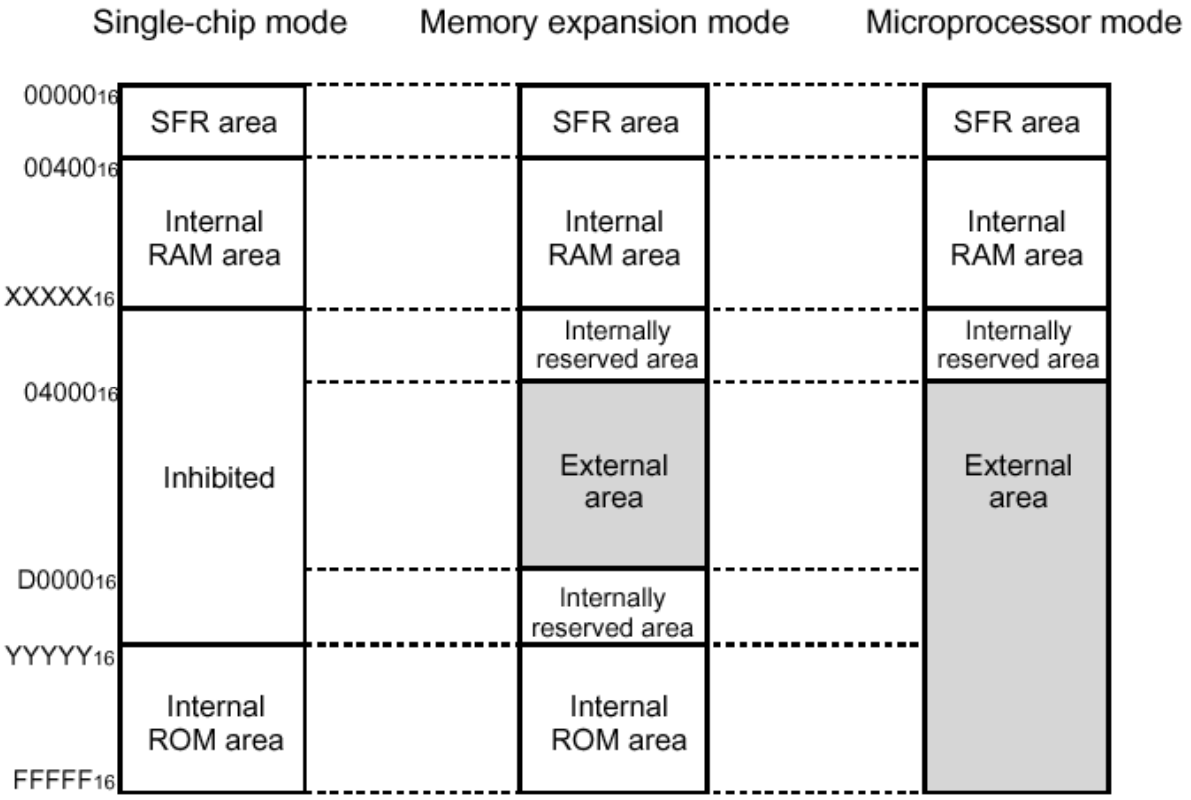
Memory

Most MCUs have on-chip memory for code + data

Might need more: expand it

- Expansion mode
- Microprocessor mode
- Uses many pins
- Limits I/O





Modes for M16C/62 MCU (not ours)

Digital Interfacing

Why? To communicate with simple sensors (switches), actuators (LEDs, motors) and other digital logic (real-time clock)

Problem: Logic level outputs are often not what sensors provide or actuators need (see MDS p.171, table 1.18.2)

- Voltage may be out of range for inputs
  - Logic Low: 0 to $0.2*V_{CC}$
  - Logic High: $0.8*V_{CC}$ toV_{CC}
  - Undefined: between low and high.
- Current may be inadequate to drive output
  - Maximum I_{OH} = -10 mA peak, -5 mA average
  - Maximum I_{OL} = 10 mA peak, 5 mA average

Solution covered in **General Purpose Digital I/O** class

Analog Interfacing

Why? To communicate with *analog* sensors and actuators

- Many devices use analog signals, not digital (microphone, thermometer, speaker, video camera…)

Input: Analog to Digital Converter (ADC)

- Produces multibit binary number AD = $(2^{Nbits}-1) * V_{In}/V_{Ref}$
- Nbits = 8 or 10 for our chip (user-selectable)
- Takes a finite amount of time (conversion speed), typically microseconds or milliseconds (3.3 μs for our chip)

Output: Digital to Analog Converter (DAC)

- Converts 8-bit binary number to equivalent voltage $V_{Out} = V_{Ref} * n/255$
- Typically need to buffer this signal to increase drive current

Solution covered in detail in **Analog I/O Peripherals**

More Analog Interfacing

Can use a comparator to detect when a voltage exceeds a given threshold

Some microcontrollers have built-in comparators (not ours)

Communications

Why? To communicate with smart components on networks and other processors

How? Use dedicated protocol controller chips which translate bytes of data into streams of bits with extra features for

- Error detection and/or correction
- Addressing
- Requesting data
- Message content, format and priority

Solutions covered in more detail in **Serial Communications** class

Miscellaneous

Leftover Pins - f*rom Table 1.17.1, MDS p.169*

- Port pins: either
  - Configure for input and connect directly to V_{SS}
  - Configure for output and leave disconnected
- $P8_5$ (^NMI/^SD): after setting to input, pull up to V_{CC}
- X_{OUT} (if using external clock on X_{IN}): Leave disconnected
- AV_{CC} : Connect to V_{CC}
- AV_{SS} and V_{REF} – ADC reference voltage: Connect to V_{SS}

Connect a bypass capacitor (>= 0.1 µF) between V_{CC} and V_{SS} pins close to the MCU for noise and latch-up prevention

Why Are We…?

Using C instead of Java?

- C is the de facto standard for embedded systems because of
  - Precise control over what the processor is doing.
  - Predictable behavior, no OS (e.g. Garbage Collection) preemption
  - Modest requirements for ROM, RAM, and MIPS, so much cheaper system

Learning assembly language?

- The compiler translates C into assembly language. To understand whether the compiler is doing a reasonable job, you need to understand what it has produced.
- Sometimes we may need to improve performance by writing assembly versions of functions.

Required to buy microcontroller boards?

- The best way to learn is hands-on.
- You will keep these boards after the semester ends for use in other projects (e.g. Senior Design, Digital Systems Interfacing, etc.)

Why Are We Supposed to Read So Much?

- Because you do your best learning outside of class
  - You can learn at your own pace
- So that we can cover more interesting things in class
  - Provide an overview and framework for what you will be reading
  - Perform design experiments with prototypes to introduce concepts
- So we can get out of class early
- Corollary: ***You won't do well in this class if you don't understand the material in the assigned readings***

Why So Much Work?

Learn by doing