uC Crash Course

Joshua Childs
Joshua Hartman

What is covered in this lecture

ESD
Choosing A Processor

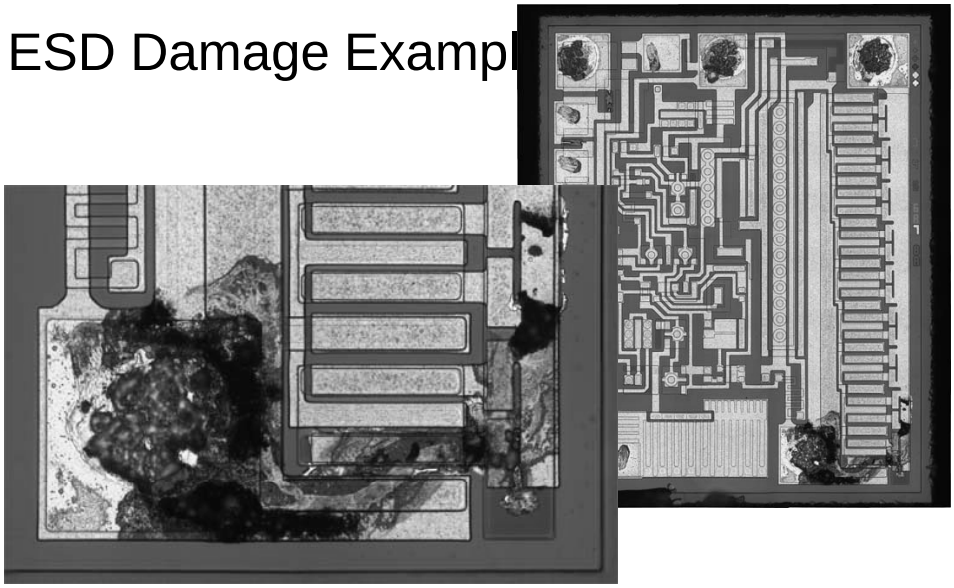
- GPIO
- USARTS
  - RS232
  - SPI
- Timers
  - Prescalers
  - OCR
  - ICR
  - PWM
- ADC
- Interrupts

ESD KILLS!

- Always be cautious of ESD warnings
- Some chips are more sensitive then others
- Use a ESD strap and mat
- If none are available touch a large grounded object

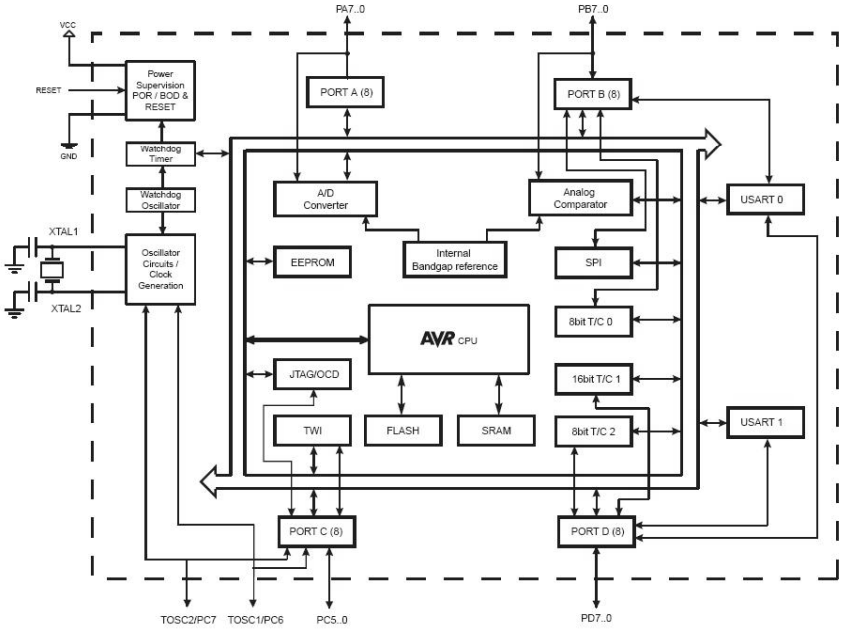
Pictures:
commons.wikimedia.org
www.ultrastatinc.com

ESD Damage Exampl



bunniestudios.com/blog/images/ams1117_esd_lg.jpg

Microcontroller vs. Microprocessor



Choosing a Microcontroller

- External connections
  - GPIO
  - ADC
  - USARTS
- Processing Power
  - is 8 bits enough?
- Chip size
  - DIP or surface mount
- Electrical Power
  - This is based on the power source

Choosing a Microcontroller

- AVR - Atmel
  - Popular processor
  - Lots of support at UF
  - AVR Studio
  - Programmer ~$17 (University Program)
- Microchip - PIC
  - Comparable to Atmel
  - Can be programmed in Basic
  - Free Samples
  - Programmer ~$35
- TI -MSP 430
  - Low power
  - Steeper learning curve
  - Free Samples
  - Programmer ~$50 (University Program)

Choosing a Microprocessor

- Propeller
  - Multicore processor
  - Prebuilt functions for audio and video
- Freescale - Coldfire
  - 32 bit processor
  - Steep Learning Curve
- Freescale - ARM
  - 32 bit processor
  - Steep Learning Curve
- NIOS
  - Softcore processor
  - Contained in Altera FPGA
  - Custom Opcodes

Choosing a clock speed

Determine your processing power

- Special timing requirements
  - USART
    - Consult table in datasheet
  - Real Time clock
- Do you need a external crystal?
  - Atmel needs an external crystal for more then 8 MHz

Choosing a chip package

- Don't Fear surface mount components
  - See the TA's or Mike for soldering help
- DIP package
  - Can be used on protoboard
  - Excessively large package
- Surface Mount
  - Space efficient
  - requires PCB
- BGA
  - balls on bottom of package
  - harder to install

GPIO

- Ports are typically 8bits mapped to one memory address
- Atmel
  - Use DDRx to set data direction
    - 0 is input
    - 1 is output
    - PIC is reversed
  - Use PORTx to write a port
  - Use PORTx to enable pull up
  - Use PINx to read a port
- Low source/sink current
  - External device may be used for high current

Atmel Timer/Counters

•What is a timer/counter?
- It's a timer that counts based on the system clock
- Can be used to toggle a pin (you saw this in the homework)
- Can be used to perform some activity based on time delay
- Can generate a PWM (square wave) signal. PWM is great for driving motors.

•Atmega32 features:
- 8 bit timer/counter 0
- 16 bit timer/counter 1
- 8 bit timer/counter 2

Timer/Counter 0 Registers

- TCNT0 - The running clock
  - OCR0 - Output compare register (things usually happen when OCR0 = TCNT0). Also used for PWM

- TCCR0 - Used to "setup" the timer/counter.
  - Sets the timer speed
  - Sets up the PWM mode
  - Sets up Output compare

- TIMSK - The timer interrupt register. Enables overflow interrupts and output compare interrupts
- TIFR - Timer flag register. Bits will be set upon overflow or output compare

Setting the timer/counter speed

Bits CS02:C200 (clock select) inTCCR0

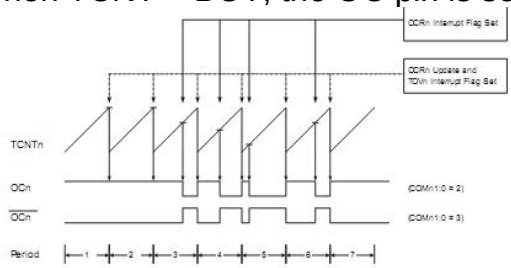
CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

PWM Modes

Normal Mode: Counter goes from 0 to TOP (255 for 8-bit timers). Not super useful, but could be used for exact timing, or long-term timing.

CTC Mode: Useful for generating very exact waveforms.

Fast PWM: Counts from 0 to TOP and then back to 0
When TCNT = OCR, the OC pin is cleared
When TCNT = BOT, the OC pin is set



Confusing to use!

Phase Correct PWM

- The best PWM mode - great for motors
- Counts from BOT to TOP then back down to BOT
- On OC match, pin is either cleared or set
- For motors, you probably want a frequency of around 10kHz
- For 75% power, just set OCR to be .75*TOP

Initialization Code:

```
TCCR0 = _BV(WGM01) | _WGM(00); //sets up phase-correct PWM
TCCR0|= _BV(COM01);       //clear on up-count, set on down
TCCR0|= _BV(CS00);       //CLK = System clock/1
```

Using Code:

```
OCR0 = (uint8_t) (desiredSpeed * 0xFF);
```

ADC

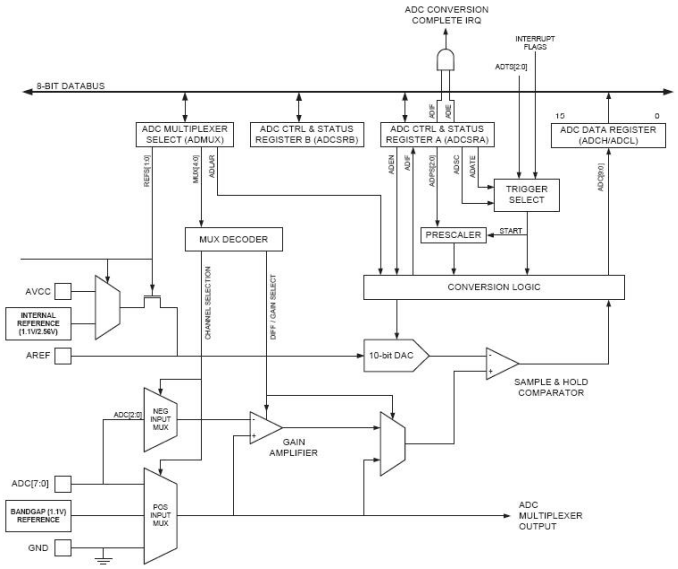
An analog to digital converter is used to sample and digitize analog signals.

Typical Applications:
- Audio Input
- Checking battery voltage
- Sensor- i.e.. IR range finder

These devices can be very simple to communicate with and are usually quicker to set up than a serial device

Converts a 0-5V voltage level into a 10-bit range (0 to 1023). You can divide things into ~.1% increments

ADC Diagram



ADC Registers

ADCSRA:

ADEN	Enables ADC
ADSC	Starts conversion(s)
ADIF	Conversion complete (an interrupt flag)
ADPS2:0	Divides the clock (like in the timer example)

ADMUX: A big multiplexer. Tells you which pin you want to do a conversion on. Also lets you select voltage reference.

ADC Gotchas

Frequency must be between 50kHz - 200kHz for full resolution
13 Clock cycles needed for a conversion (successive approximation)
First conversion actually needs 25 clock cycles. Good idea to throw away the first few.

USART

RS-232 Communication

- Async or Sync operation
- 5,6,7,8 or 9 data bits
- 1 or 2 stop bits
- Even, Odd or No Parity
- TTL Logic Levels
- Interrupts or polling of status registers

USART

SPI Communication

- Synchronous serial
- Requires 3 wires plus an enable
- Easy to connect multiple devices
- Master Operation
- 4 Modes of operation
- LSB or MSB
- High Speed

Interrupts

TA Rule of thumb: Don't use them if you can get away with it.

Interrupts make things happen "out of order". Usually you can get by with polling a device every so often and updating a value. I know polling seems like a "dumb way" of doing things. It's not.

If you really want to use them, keep the code as short as possible.
Useful for:

- Triggering that an ADC conversion is complete
- Having an event happen on intervals based on the clock
- External Interrupts: Do something when a pin changes. This is great if you have critical data to process
- Can tell you when tasks finish

I built a Segway and a robot. I used only one interrupt.

Debugging Tips

- Have a feedback system to verify program operation
  - LCD Screen
  - LED Bank
  - Serial Output to PC
- Use the Proper test equipment
  - DMM
    - Testing static signals and supply voltages
    - Will average AC signals
  - Scope
    - Verify signals
    - Look for noise
  - DMM
    - Verify logic levels of data
    - Multiple inputs

Useful Websites

AVR-LIBC: Detailed descriptions of library files. Really useful!
http://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/modules.html

AVRFREAKS: Smart robot guys with an amazing forum. Will solve all your problems
http://www.avrfreaks.net

The datasheet. Google for it. Read it. Then ask questions.