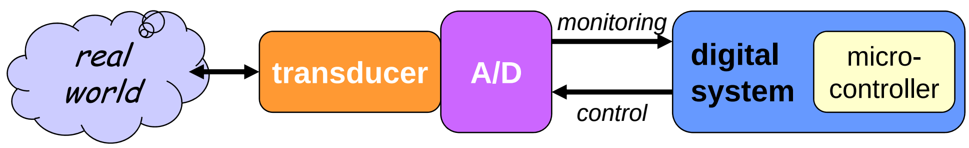
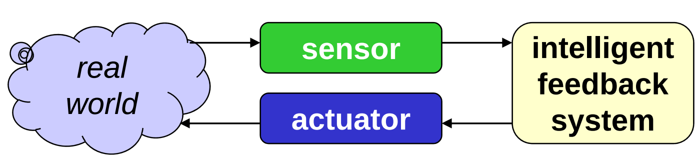
Analog-to-Digital Converters

- Terminology
  - analog-to-digital converter = ADC = A/D = AtoD
- Function
  - transform an analog signal into a digital signal for use (calculation, storage, decision making) in an digital system e.g., a microcontroller
- Motivation
  - the real world is analog
  - A/D needed to interface real world to digital systems
    - monitoring of real world events/phenomena
    - electronic intelligent feedback control of real world



Transducers

- Transducer
  - a device that converts a primary form of energy into a corresponding signal with a different energy form
    - Primary Energy Forms:
  - take form of a **sensor** or an **actuator**
- Sensor (e.g., thermometer)
  - a device that detects/measures a signal or stimulus
  - acquires information from the "real world"
- Actuator (e.g., heater)
  - a device that generates a signal or stimulus



Common Transducers

- Conventional Transducers

  large, but generally reliable; based on older technology
  - thermocouple: temperature difference
  - compass (magnetic): direction
- Microelectronic Sensors

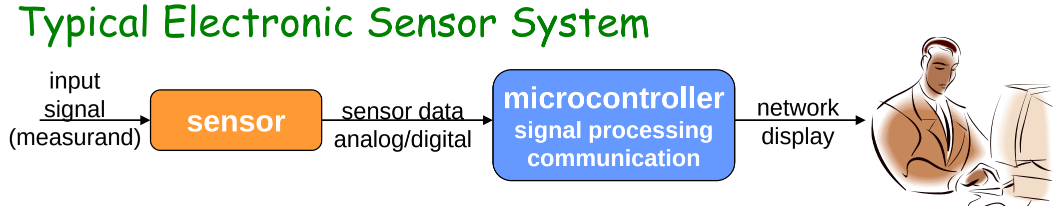
  millimeter sized; highly sensitive; less robust
  - photodiode/phototransistor: photon energy (light)
    - infrared detectors, proximity/intrusion alarms
  - piezoresisitve pressure sensor: air/fluid pressure
  - microaccelerometers: vibration, Δ-velocity (car crash)
  - chemical senors: O_2, CO_2, Cl, Nitrates (explosives)
  - DNA arrays: match DNA sequences

Sensor Systems

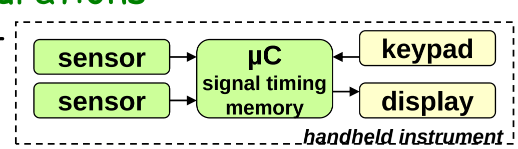
*Generally interested in **electronic sensor***

- convert desired parameter into electrically measurable signal

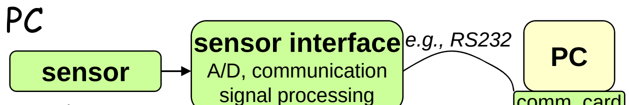
- Typical Electronic Sensor System



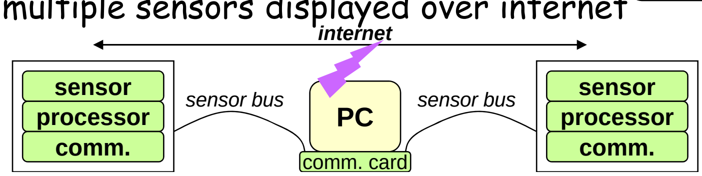
- Example Sensor System Configurations
  - digital sensor within an instrument
    - microcontroller
      - signal timing, data storage



  - analog sensor analyzed by a PC

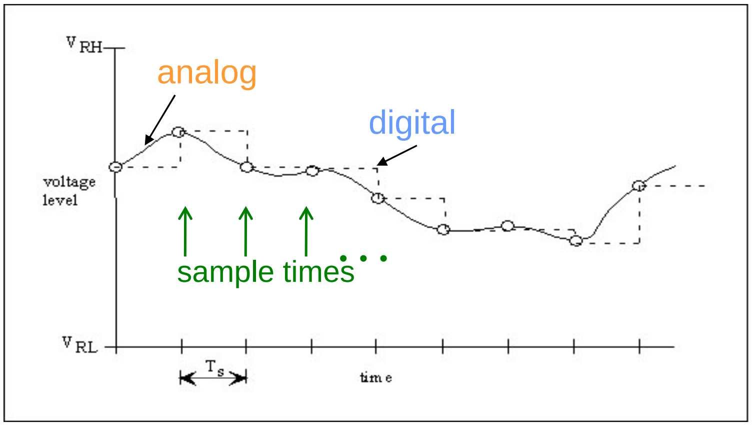


  - multiple sensors displayed over internet



A/D Basic Concepts

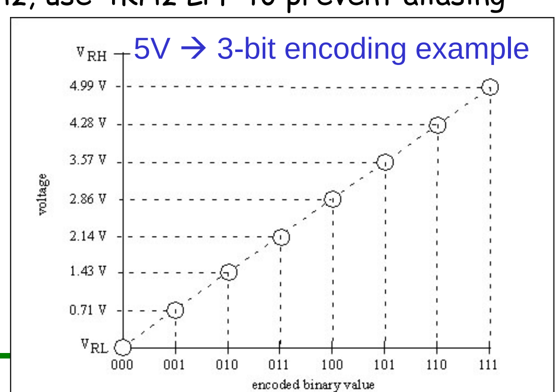
- Analog signal characteristics
  - maximum voltage
  - minimum voltage
  - voltage sensitivity
  - frequency components
- Digital sample feature
  - voltage reference <u>high</u>:



A/D Conversion Process

- <u>Sampling rate</u>
  - What is sampling? strobe light example
  - Nyquist criterion: minimum sample frequency (f_s) should be twice the highest frequency content (f_h) of the sampled signal
  - Time interval between samples:
  - Anti-aliasing filter: use LPF with
    - Example: sample human voice at 8KHz, use 4KHz LPF to prevent aliasing
- <u>Encoding</u>
  - Provides unique binary code for every discrete voltage step between V_{RH} and V_{RL}
  - $\boxed{n = \#steps = 2^b}$, b=#bits

EXAMPLE: What voltage is encoded by 101?



A/D Conversion Process

- Quantization: number of discrete levels the analog signal is divided into between V_{RH} and V_{RL}
  - #steps = 2^b
  - more levels (steps) → better representation of sampled signal
- Resolution: voltage per step
  - Resolution = $(V_{RH} - V_{RL})$/number of steps =

  EXAMPLE: V_{RH} = 5 V, V_{RL} = 0 V, quantization = 256 levels

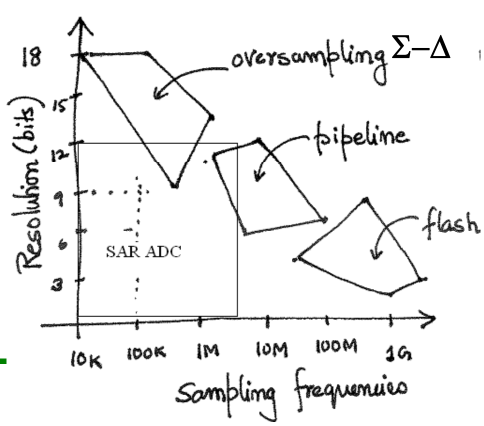
  # bits = ?

  Resolution = ?

  - Analog value:
- Data rate: # of A/D result bits per second, $d = f_s \times b$

Common A/D Architectures/Structures

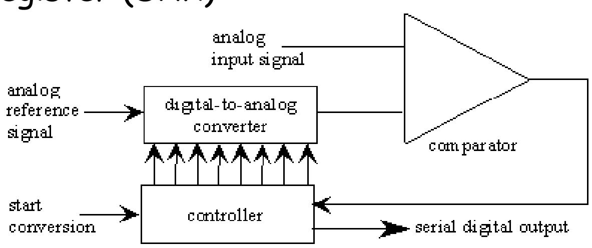
- Flash: bank of comparators sample input signal in parallel
  - very fast (GHz+), size limits resolution: number of comparators needed, 2^N - 1, doubles with each additional bit
- Integrating: measures the time it takes to integrate a reference signal to match input value
  - common in digital voltmeters because of linearity and flexibility
- Sigma-Delta: oversamples signal by a large factor and filters the desired signal band
  - slow but high resolution
- Successive-approximation: constantly compares input voltage to reference value; reference value moves closer to input value each cycle
  - #cycles = #bits resolution
- Pipelined: combines successive approximation and flash
  - fast, high resolution, small die (chip) size

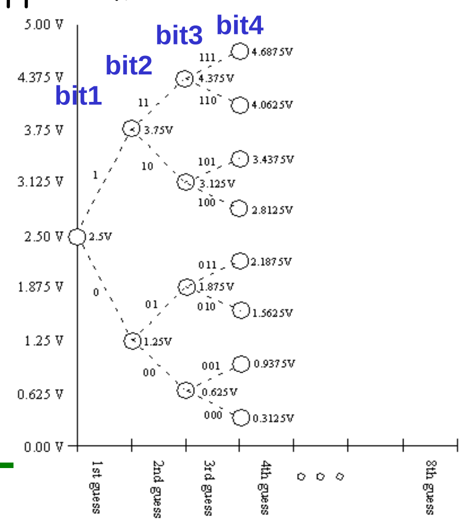


Example: Best choice for:
High speed?
High resolution?

Successive Approximation A/D

- Constantly compares input voltage to reference value set by an internal DAC
- DAC value cuts voltage range in half each cycle to approach input value
  - each cycle → one more bit of resolution
- Final binary value stored in a successive approximation register (SAR)





Example: What is 3b ADC result for 1.75V?

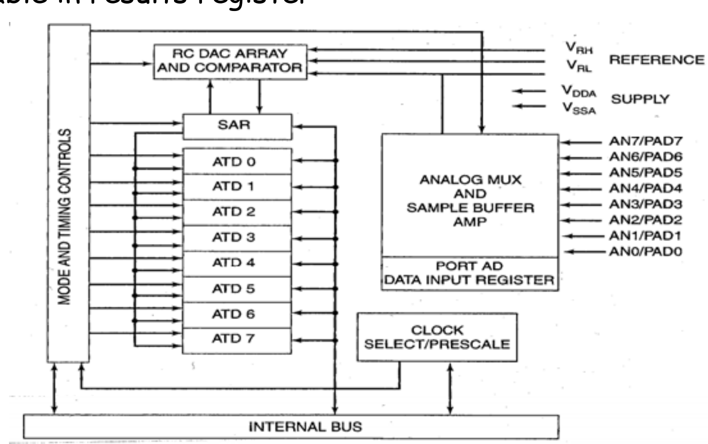
bit 1?

bit 2?

bit 3?

68HC12 ATD System

- Eight ATD **analog** inputs on
- Inputs fed to analog multiplexer
- Single signal fed to successive approximation converter
- Initiate conversion by writing to control register
- Upon conversion complete, flags set in status registers
- Results available in results register



HC12 ATD Registers

- Control registers - configures ATD for specific operation (ATDCTL0 - ATDCTL5)
  - used to tailor an ATD conversion sequence
- Status registers - two-byte register containing ATD status flags (ATDSTAT)
- Result registers - contains binary weighted result after conversion (ADR0H - ADR7H)
- Test registers - used in special modes

ATD Control Registers

- **ATDCTL2**: memory address: $0062
  - ADPU: "on/off" switch
    - 0: off, 1: on (0 after processor reset)
  - must wait 100 us after "on" prior to using ATD
  - AFFC: ATD Fast Flag Clear
    - 0: normal clearing - write to ATDCTL5
    - 1: fast clearing - cleared when first result register read

Register: Analog-to-Digital Converter Control Register 2 (ATDCTL2) Address: $0062

	7	6	5	4	3	2	1	0
	ADPU	AFFC	AWAI	0	0	0	ASCIE	ASCIF
Reset:	0	0	0	0	0	0	0	0

Register: Analog-to-Digital Converter Control Register 4 (ATDCTL4) Address: $0064

	7	6	5	4	3	2	1	0
	0	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Reset:	0	0	0	0	0	0	0	0

- **ATDCTL4**: memory address: $0064
  - controls sample timing for conversion sequence

SMP1 SMP2	Final Sample Time	Total 8-bit Conversion Time
0 0	2 ATD clock periods	18 ATD clock periods
0 1	4 ATD clock periods	20 ATD clock periods
1 0	8 ATD clock periods	24 ATD clock periods
1 1	16 ATD clock periods	32 ATD clock periods

Example: What is the max ATD sampling frequency?

ATD Control Registers

- **ATDCTL5**: memory address: $0065
  - configure conversion mode for ATD
    - S8CM: select 8 channel mode
      - 0: four, 1: eight conversions
    - SCAN: enable continuous scan
      - 0: single 1: continuous conversion
    - MULT: enable multiple channel conversion
      - 0: single channel, 1: multiple channels
    - CD,CC,CB,CA: set channels for conversion

Example: How many different input channels can be selected?

Register: Analog-to-Digital Converter Control Register 5 (ATDCTL5) Address: $0065

	7	6	5	4	3	2	1	0
	0	S8CM	SCAN	MULT	CD	CC	CB	CA
Reset:	0	0	0	0	0	0	0	0

S8CM	CD	CC	CB	CA	Channel Signal	Result in ADRx if MULT = 1
0	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
0	0	1	0	0	AN4	ADR0
			0	1	AN5	ADR1
			1	0	AN6	ADR2
			1	1	AN7	ADR3
0	1	0	0	0	Reserved	ADR0
			0	1	Reserved	ADR1
			1	0	Reserved	ADR2
			1	1	Reserved	ADR3
0	1	1	0	0	V_{RH}	ADR0
			0	1	V_{RL}	ADR1
			1	0	$(V_{RH} + V_{RL})/2$	ADR2
			1	1	Test/reserved	ADR3
1	0	0	0	0	AN0	ADR0
		0	0	1	AN1	ADR1
		0	1	0	AN2	ADR2
		0	1	1	AN3	ADR3
		1	0	0	AN4	ADR4
		1	0	1	AN5	ADR5
		1	1	0	AN6	ADR6
		1	1	1	AN7	ADR7
1	1	0	0	0	Reserved	ADR0
		0	0	1	Reserved	ADR1
		0	1	0	Reserved	ADR2
		0	1	1	Reserved	ADR3
		1	0	0	V_{RH}	ADR4
		1	0	1	V_{RL}	ADR5
		1	1	0	$(V_{RH} + V_{RL})/2$	ADR6
		1	1	1	Test/reserved	ADR7

Shaded bits are "don't care" if MULT = 1 and the entire block of four or eight channels makes up a conversion sequence. When MULT = 0, all four bits (CD, CC, CB, and CA) must be specified and a conversion sequence consists of four or eight consecutive conversions of the single specified channel.

ATD Status Registers

- **ATDSTAT**: memory address: $0066,0067 two bytes
  - SCF: Sequence Complete Flag
    - indicates specified conversion sequence is complete
  - CCx:
    - indicates channel currently undergoing conversion
  - CCFx: Conversion Complete Flag for each result register

Register: Analog-to-Digital Converter Status Register (ATDSTAT) Address: $0066

	7	6	5	4	3	2	1	0
	SCF	0	0	0	0	CC2	CC1	CC0
Reset:	0	0	0	0	0	0	0	0

Register: Analog-to-Digital Converter Status Register (ATDSTAT) Address: $0067

	7	6	5	4	3	2	1	0
	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
Reset:	0	0	0	0	0	0	0	0

ATD Results Registers

- **ADRxH**: memory address: $0070 – 007E eight bytes
  - after conversion, results placed in ADR0H-7H
  - unsigned, weighted binary result

    1/2FS,1/4FS,1/8FS,…1/256FS., FS=full scale
- $V_{DC} = V_{RL} + ([\text{contents ADRxH}]/256) \times (V_{RH} - V_{RL})$

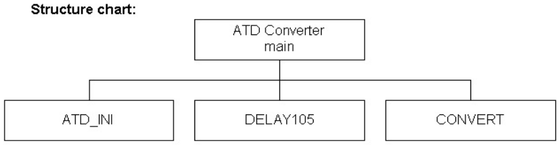
Register: Analog-to-Digital Converter Result Register 0 (ADR0H)	Address: $0070
Register: Analog-to-Digital Converter Result Register 1 (ADR1H)	Address: $0072
Register: Analog-to-Digital Converter Result Register 2 (ADR2H)	Address: $0074
Register: Analog-to-Digital Converter Result Register 3 (ADR3H)	Address: $0076
Register: Analog-to-Digital Converter Result Register 4 (ADR4H)	Address: $0078
Register: Analog-to-Digital Converter Result Register 5 (ADR5H)	Address: $007A
Register: Analog-to-Digital Converter Result Register 6 (ADR6H)	Address: $007C
Register: Analog-to-Digital Converter Result Register 7 (ADR7H)	Address: $007E

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset:	0	0	0	0	0	0	0	0

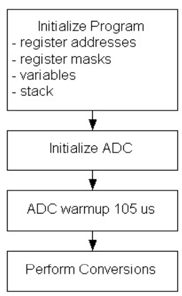
Example: If V_{RH} = 2.2, V_{RL} = 0.5, what analog value is represented by ADR2H = 00110110?

ATD Programming Example

Structure chart:



Flow chart:




```
/* File Name: voltmeter.c
 * File Created: 04-14-02
 * File Modified:
 * Author(s): Abbie Wells
 * Carrie Hernandez, LCD Portions
 */

/* This program will create a simple voltmeter using the onboard
 * analog to digital converter in the HC12.  It will perform one
 * conversion and then the user will have to manually restart the
 * program to convert another voltage.  The voltage will then be
 * displayed to the LCD.
 */

 #include <hc12.h>
 #include <stdio.h>

 #define DECIMAL 0x2E  // define macro for a decimal point in ASCII
 #define V 0x56          // define macro for a "V" in ASCII

 void delay_100us(void);
 void ADC_convert(void);
 void delay_5ms(void);
void main(void)
 {
 printf("HELLO\n");
 ATDCTL2 = 0x80;        // power up the ADC and disable interrupts
 printf("ADC2\n");
 delay_5ms();          // wait for ADC to warm up
 printf("warmed up\n");
 ATDCTL3 = 0x00;       // select active background mode
 ATDCTL4 = 0x01;       // select sample time = 2 ADC clks and set
                       // prescaler to 4 (2 MHz)
 printf("ready\n");
 ADC_convert();        // perform conversion and change to usable
                       // value
 }
```

```
/* void ADC_convert(void): function to perform a single conversion */

void ADC_convert()
{
unsigned int sumadr;
unsigned int avg_bin_voltage;
unsigned int int_voltage;
unsigned int ones_int;
unsigned int tenths_int;
unsigned int hundreths_int;
char ones;
char tenths;
char hundreths;


ATDCTL5 = 0x03;          // sets up ADC to perform a single conversion,
                         // 4 conversions on a single channel,
                         // and store the results ADR0H - ADR3H.


while((ATDSTAT & 0x8000) != 0x8000)          // Wait for conversion to finish
{
 ;
}

//printf("%x %x %x %x\n", ADR0H, ADR1H, ADR2H, ADR3H);
sumadr = ADR0H + ADR1H + ADR2H + ADR3H;
avg_bin_voltage = sumadr/4;
int_voltage = (100*avg_bin_voltage/256)*5;
ones_int = int_voltage/100;
ones = (char)(ones_int + 48);
tenths_int = (int_voltage - ones_int*100)/10;
tenths = (char)(tenths_int + 48);
hundreths_int = (int_voltage - ones_int*100 - tenths_int*10)/1;
hundreths = (char)(hundreths_int + 48);
printf("%c.%c%cV\n", ones, tenths, hundreths);
}
```

```
/*************************************************************/
/*100us delay based on an 8MHz clock                          */
/*************************************************************/
void delay_100us(void)
{
int i;
for (i=0; i<400; i++)
  {
  asm("nop");
  }
}
/************************************************************/
/*5 ms delay based on an 8MHz clock                          */
/************************************************************/
void delay_5ms(void)
{
int i;
for (i=0; i<800; i++)
  {
  delay_100us();
  }
}
```