**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 a 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
  - piezoresistive pressure sensor: air/fluid pressure
  - microaccelerometers: vibration, Δ-velocity (car crash)
  - chemical sensors: O₂, CO₂, 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 high:

A/D Conversion Process

• Sampling rate
  - What is sampling? strobe light example
  - Nyquist criterion: minimum sample frequency ($f_s$) should be twice the highest frequency content ($f_{	ext{in}}$) 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

• Encoding
  - Provides unique binary code for every discrete voltage step between $V_{RH}$ and $V_{RL}$
    - $n = \# \text{steps} = 2^b$, $b=\#\text{bits}$

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 $V_{RL}$
  - $#\text{steps} = 2^b$
  - more levels (steps) $\rightarrow$ 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
  - $#\text{bits} = ?$
  - 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
  - $#\text{cycles} = #\text{bits resolution}$

- **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 (ADROH - 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

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

<table>
<thead>
<tr>
<th>SM1</th>
<th>SM0</th>
<th>Final Sample Time</th>
<th>Total Sample Conversion Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2 ATD clock periods</td>
<td>16 ATD clock periods</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4 ATD clock periods</td>
<td>20 ATD clock periods</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8 ATD clock periods</td>
<td>24 ATD clock periods</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>16 ATD clock periods</td>
<td>32 ATD clock periods</td>
</tr>
</tbody>
</table>

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?

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
**ATD Results Registers**

- **ADRxH**: memory address: $0070 - 007E eight bytes
  - after conversion, results placed in ADROH-7H
  - unsigned, weighted binary result
  - $1/2FS, $1/4FS, $1/8FS,...,$1/256FS, FS=full scale

- \( V_{DC} = V_{RL} + \left(\frac{\text{contents ADRxH}}{256}\right) \times (V_{RH} - V_{RL}) \)

### Example
If \( V_{RH} = 2.2 \), \( V_{RL} = 0.5 \), what analog value is represented by \( \text{ADR2H} = 00110110 \)?

### ATD Programming Example

**Structure chart:**

```
  +-------------------+     +-------------------+
  | ATD Converter     |     | ATD_INIT          |
  +-------------------+     +-------------------+
     | DELAY105         |     | CONVERT          |
     +-------------------+     +-------------------+
```

**Flow chart:**

```
  +-------------------+     +-------------------+
  | ATD_INIT          |     | DELAY105         |
  +-------------------+     +-------------------+
  | CONVERT           |     |                   |
  +-------------------+     +-------------------+
```

**Include:**
`#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 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.
```

```
  // 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<400; i++)
      asm("nop");
  }
```

```
  ATDCTL2 = 0x80; // power up the ADC and disable interrupts
  ATDCTL3 = 0x00; // select active background mode
  ATDCTL4 = 0x01; // select sample time = 2 ADC clks and set
                  // prescaler to 4 (2 MHz)
```

```
  printf("HELLO
");  // print the message to the LCD.
```

```
  ADC_convert(); // perform conversion and change to usable
                  // value
```

```
  while((ATDSTAT & 0x8000) != 0x8000) // Wait for conversion to finish
  {
    //printf("%x %x %x %x
", 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;
    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
", ones, tenths, hundreths);
  }
```

**Analog-to-Digital System.16**