Embedded systems are product sub-systems controlled by a special-purpose computer. The computer provides software-control by using transducers called actuators and sensors.

- **Sensors** are transducers that convert physical energy to electrical energy. In general, sensors ***meter*** the environment by measuring some change in a physical parameter.
- **Actuators** are transducers that convert electrical energy to physical energy. In general, actuators ***change*** the environment by using electricity to change a physical parameter.
- **Microcontrollers** are single-chip computers that are the brain of ***most*** embedded systems.
- **Figure 1** is a basic model used to diagram embedded systems that use microcontrollers.

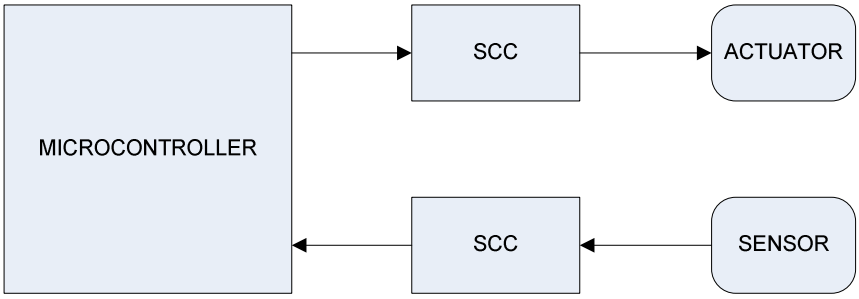

Figure 1: The Basic Embedded System Model

Analog sensors produce analog signals.

- Analog signals are continuous, real-values electrical signals.
- Example analog signals include temperature variation, speech, light variation, atmospheric communications signals, and bioelectric signals such as the ECG.
- Small analog signals may need to be signal conditioned and ranged to larger signals before presentation to the computer for conversion to binary numbers. Signal conditioning circuits are not covered in this embedded systems tech note.

Analog-to-digital converters (ADC) are common on-chip microcontroller I/O devices.

- Most ADCs have multiple input pins that can be converted one at a time.
- Some ADCs can convert multiple input pins simultaneously.
- Some ADCs provide built-in amplification and conversion of the differential voltage between two input pins.
- All ADCs convert the analog voltage to an n-bit binary result.
- Microcontrollers are available with 8-bit, 10-bit, 12-bit, and 16-bit converters. The bitwidth determines the granularity of voltage conversion.

Basic ADC facts can be stated to summarize the conversion process.

- **ADC power supply pins** are connected to power supply rails VDD and 0V to provide a full-scale reference voltage range.
- The **range** of the ADC is the set of binary numbers that can be produced by the ADC.

- The **resolution** of the ADC is the voltage differential that results in a new binary result.
- The resolution can be calculated by using the equation:

$$Resolution = \frac{Vdd - 0}{2^n}$$

- The range of the ADC output binary numbers is 0 to 2^n-1.
- The binary number that results for any real voltage conversion can be calculation using:

$$Result = \frac{realVoltage}{Resolution} = realVoltage * \frac{2^n}{Vdd}$$

- ADC results are integer whole numbers with fractional part truncated.
- Example conversions are shown in Table 1.

TABLE 1: EXAMPLE ADC CALCULATIONS (5V FULL-SCALE RANGE)

realVoltage	n	Resolution	Result in decimal
1.45 V	8	19.6mV	74
	10	4.88mV	296
	12	1.22mV	1187
	16	76.29uV	19005
3.68 V	8	19.6mV	188
	10	4.88mV	753
	12	1.22mV	3014
	16	76.29uV	48234
4.99 V	8	19.6mV	255
	10	4.88mV	1021
	12	1.22mV	4087
	16	76.29uV	65404
0V	8	19.6mV	0
	10	4.88mV	0
	12	1.22mV	0
	16	76.29uV	0
0.3V	8	19.6mV	15
	10	4.88mV	61
	12	1.22mV	245
	16	76.29uV	3932
1.92V	8	19.6mV	98
	10	4.88mV	393
	12	1.22mV	1572
	16	76.29uV	25165

The **ATmega32** has a very nice **on-chip ADC**.

- **Ten-bit** (10-bit) conversions are completed for all samples.
- **Left-adjusted results** are available for users that choose to only use the upper 8-bits of data.
- **Conversion accuracy** is related to ADC clock frequency. Atmel recommends 50KHz to 250KHz for best results.
- **ADC clock frequency** is programmable through control register bits.
- **Each conversion** takes 13 clock cycles of the ADC clock.
- **Single pins** can be sampled.
- **A differential voltage between two pins (with amplification)** can be sampled.
- **Interrupts** can be generated at conversion completion.
- **A conversion complete flag** is provided for polling when interrupts are not used.

The **ATmega32 ADC** is controlled by software through I/O control registers.

- **Configuration registers** ADMUX and ADCSRA select ADC behavior.
- **Result registers** ADCH and ADCL contain conversion results.

ATmega32 control register ADMUX is used for part of ADC configuration.

- **Select** the ADC power pins.
- **Choose** left-adjusted or full-result mode.
- **Select** the single pin or differential group for sampling.

ATmega32 control register ADCSRA is used for part of ADC configuration and completion monitoring.

- **Turn on** the ADC.
- **Start** a conversion.
- **Select** auto-triggering if used.
- **Enable** interrupts if desired.
- **Choose** an ADC clock frequency.
- **Monitor** ADSC for completion if not using interrupts. ADSC clears to 0 at the end of a conversion.

Example assembly language:

```
;** select power supply, left-adjust, and pin 5
;** ADMUX
;**  | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 |
;**  ---------------------------------------------------------------
;**
;** ADCSRA
;**  | ADEN  | ADSC  | ADATE | ADIF | ADIE | ADPS2| ADPS1| ADPS0|
;**  ---------------------------------------------------------------

;**   see Atmel reference guides for bit values
;**   assumes a system clock frequency of 16MHz
;**   16MHz / 128 = 125KHz = middle of recommended 50 – 250 KHz

           ldi   r16,0b01100101   ; AVCC,LAR,5
           out   admux,r16        ; update
           ldi   r16,0b11000111   ; ON,START,f=/128
           out   adcsra,r16       ; update
adcwait:   sbic  adcsra,6         ; if done, skip next line
           rjmp  adcwait          ; else keep waiting
adcdone:   in    r16,adch         ; get left-adjusted result
           …
```

Example C source code:

```
#include <avr\io.h>
#include <inttypes.h>

/* select power supply, left-adjust, and pin 5
 * ADMUX
 *  | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 |
 *  ---------------------------------------------------------------
 *
 * ADCSRA
 *  | ADEN  | ADSC  | ADATE | ADIF | ADIE | ADPS2| ADPS1| ADPS0|
 *  ---------------------------------------------------------------
 *
 *   see Atmel reference guides for bit values
 *   assumes a system clock frequency of 16MHz
 *   16MHz / 128 = 125KHz = middle of recommended 50 – 250 KHz
 */

int main(void)
{



     ADMUX = 0b01100101;         /* AVCC,LAR,5          */
     ADCSRA = 0b11000111;        /* ON,START, /128 */
     while (ADCSRA & 0b01000000)
     {
          /* do nothing while waiting */
          /* note that all bits clr'd */
          /* except 6.  ADC will clear*/
          /* it when done.            */
          /*                          */
          /* thus, AND returns true   */
          /* because it is non-zero   */
          /* until ADSC clears and    */
          /* then it will return 0    */
          /* because 0&0 = 0.         */
     }
     if (ADCH < 145)
     {
     …
     }
     else
     {
     …
     }
     return 0;
}
```

