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.

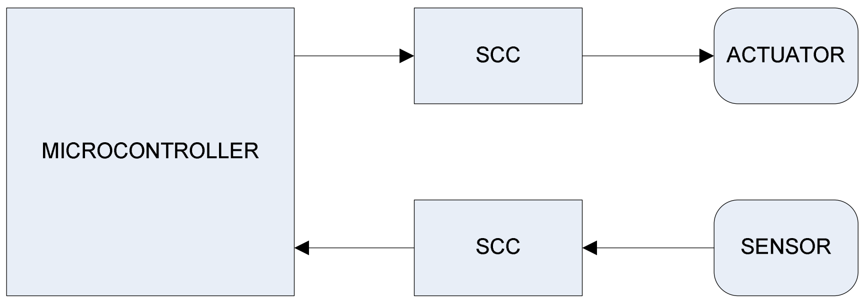

Figure 1: The Basic Embedded System Model

Pins interface sensors and actuators to the microcontroller.

- **Ports** are named collections of pins for easy programmatic reference.
- **Input-only** port pins are only used for sensor signals.
- **Output-only** port pins are only used for actuator signals.
- **Bidirectional** port pins can be used for either sensor or actuator signals.
- **Pins** can be shared by a number of I/O devices to keep the size of the chip small.

I/O control registers hold binary numbers that determine port pin behavior. Each program will use different types of control registers based on the system functionality and the I/O devices used.

- **Configuration control registers** configure I/O devices that use port pins. Examples include clock timing, interrupt masks, and pin edge response.
- **Direction control registers** control the direction of bidirectional port pins.
- **Input registers** sample sensor signals on command.
- **Output registers** control the voltage output to actuator signals.
- **Parameter registers** hold parameters needed by I/O device functions. Example include time values for timer functions and voltage values for analog comparators,
- **Result registers** hold results produced by some I/O devices.
- **Status registers** flag events for software notice.

 © Dr. Russ Meier, Milwaukee School of Engineering

The **Atmel ATmega32** 8-bit microcontroller has a pin interface summarized in Table 1.

- **Discrete digital I/O** is available on all port pins.
- **On-chip I/O devices** share the port pins.

TABLE 1: ATmega32 PORT PINS		
PORT NAME	**PINS AND DIRECTIONS**	**SHARED BY THESE I/O DEVICES**
PORTA	8 bidirectional pins	Analog-to-digital converter
PORTB	8 bidirectional pins	SPI serial, analog comparator, timer, external interrupts
PORTC	8 bidirectional pins	Timer, JTAG in-system programming, two-wire serial
PORTD	8 bidirectional pins	Timer, external interrupts, UART serial

ATmega32 discrete digital I/O is controlled using direction, input, and output registers.

- **Direction** is controlled using direction registers DDRA, DDRB, DDRC, and DDRD. A zero in a bit position selects input electronics for the corresponding pin while a one in a bit position selects output electronics.
- **Output** voltages are controlled using output registers PORTA, PORTB, PORTC, and PORTD. A zero in a bit position drives 0V onto the port pin. A one in a bit position drives +5V onto the port pin.
- **Input** voltages are sensed using input registers PINA, PINB, PINC, and PIND. A zero represents sensing a 0V signal. A one represents sensing a +5V signal.
- **Remember**: in with the P***IN*** register, out with the PORT register.

Some sensors have **open-collector** or **open-drain** signals.

- This type of signal will not drive the wire to +5V. Instead, it open-circuits when not at 0V.
- The solution connects a resistor between the signal and +5V. The resistor provides the "pull-up" or "pull-high" function when the sensor open-circuits.
- **Ohm's law** explains the pull-up function. **Figure 2** diagrams a pull-up resistor on a sensor signal connected to a microcontroller port pin. Modern CMOS microcontroller input pins have no current flow. Therefore, when the sensor releases the signal from 0V, the open-circuit at both ends of the signal wire means no current can flow through R. The voltage loss across resistor R is 5-0*R = 5V. Thus, the resistor has "pulled-up" the signal to +5V.

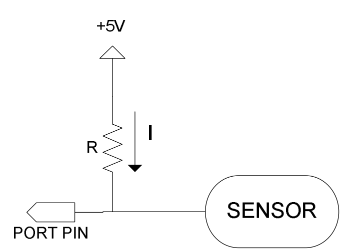

Figure 2: Pull-up resistors

 © Dr. Russ Meier, Milwaukee School of Engineering

Built-in pull-up resistors are provided at every port pin of the ATmega32.

- Pull-up resistors can be turned on or off by software control.
- Pull-up resistors only need to be used if the sensor signal does not drive to +5V.
- Pull-up resistors can be turned on for all unconnected port pins resulting in +5V bit reads.

ATmega32 digital I/O port configuration is summarized in Table 2.

TABLE 2: ATmega32 PORT PIN CONFIGURATION			
CONTROL REGISTER BITS		PORT PIN BEHAVIOR	
DDR bit	**PORT bit**	**Direction**	**Pull-up on?**
0	0	Input	No
0	1	Input	Yes
1	0	Output	No
1	1	Output	No

ATmega32 assembly language examples:

- AVR assembly provides many ways to set and clear bits in control registers.
- These examples illustrate using a do-no-damage approach to control one bit at a time.
- These examples will be slower and longer than code overwriting all bits simultaneously.

Configuring output and driving voltage

```
;** configure portb3 as output pin
;** do no damage to other ddr bits

     in    r16,ddrb
     ori   r16,0b00001000
     out   ddrb,r16

;** write +5V onto port pin PB3
;** do no damage to other pins

     in    r16,portb
     ori   r16,0b00001000
     out   portb,r16

;** write +0V onto port pin PB3
;** do no damage to other pins

     in    r16,portb
     andi  r16,0b11110111
     out   portb,r16
```

Configuring input and reading voltage

```
;** configure portb3 as input pin
;** turn on pull-up

     in    r16,ddrb
     andi  r16,0b11110111
     out   ddrb,r16
     in    r16,portb
     ori   r16,0b00001000
     out   portb,r16

;** read pin voltages

     in    r16,pinb

;** was PB3 zero?
;** if PB3 = 0 then L1 else L2
     sbrc  r16,3
     rjmp  L2
L1:
```

 © Dr. Russ Meier, Milwaukee School of Engineering

C programming language examples:

- C provides many ways to set and clear control register bits.
- These examples illustrate a do-no-damage approach using bitwise AND and OR logical operators.
- These examples will be longer and slower than code overwriting all bits simultaneously.
- These examples are written using the WinAVR gcc installation.

Configuring output and driving voltage

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

int main(void)
{

	/* configure PB3 as output */
	DDRB = DDRB | 0b00001000;

	/* write PB3=1 then PB3=0 */
	PORTB = PORTB | 0b00001000;
	PORTB = PORTB & 0b11110111;

	/* do other work */
	…


	return 0;
}
```

Configuring input and reading voltage

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

int main(void)
{
	uint8_t pv;

	/* configure PB3 as input */
	/* pull-ups on            */
	DDRB = DDRB & 0b11110111;
	PORTB = PORTB | 0b00001000;

	/* read pin voltages */
	pv = PINB;

	/* if-then-else using PB3 */
	/* if PB3 = 1 then, else */
	if (pv & 0b00001000)
	{
		/* do true part */
	}
	else
	{
		/* do false part */
	}

	/* do other work */
	…

	return 0;
}
```