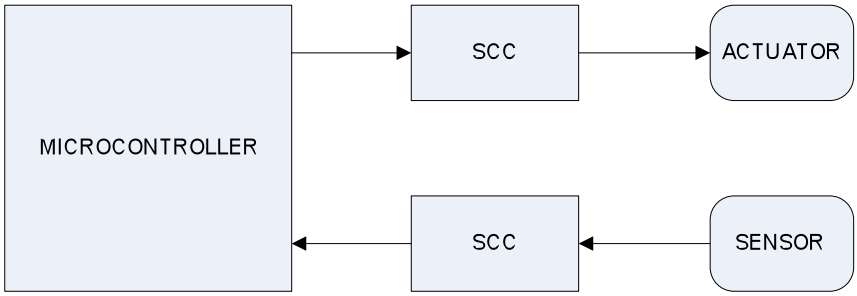
Embedded systems are product sub-systems controlled by a special-purpose computer.

- Embedded systems control one system task or a small number of system tasks.
- Embedded systems can be categorized into application domains such as transportation, appliances, building systems, audiovisual, gaming, and medical.
- Embedded systems have design constraints that are different from personal computers. For example, embedded systems must often fit in a small footprint, tolerate high vibrations, hit a low cost-point, and be a low-power device.

Real-time embedded systems must respond to event as they occur.

- Hard real-time systems ***must*** meet system response time constraints.
- Soft real-time systems ***should*** meet system response time constraints.
- Not all embedded systems have real-time constraints.
- Interrupts and timers are hardware features used to implement real-time responses.

Basic embedded system modeling describes the system inputs, outputs, and design constraints.



- A basic model can be used to start the design for all embedded systems.
- The model names the key parts of the embedded system.
- Model refinement occurs as the design constraints are solved.

Transducers convert energy from one form to another.

- Embedded systems use electromechanical transducers.
- **Sensors** are transducers that convert physical energy to electrical energy. For example, the microphone converts sound to voltage. Similarly, a photo-receiver converts photons to voltage or current. Other examples are accelerometers, thermometers, buttons, switches, flowmeters, and load cells for pressure.
- **Actuators** are transducers that convert electrical energy to physical energy. For example, the LED converts current to photons. Similarly, motors convert voltage to rotations motion. Other examples are solenoids, speakers, valves, and relays.

Signal conditioning circuits protect the computer from inappropriate signal voltage or current levels.

- **Isolation** prevents unwanted electrical energy from reaching the embedded computer. For example, unwanted electrical frequencies can be filtered out by signal-conditioning circuits. Similarly, large spikes in voltage and current can be leveled out by signal-conditioning circuits.
- **Ranging** ensures that sensor and actuator voltages fall within the correct mathematical range. For example, analog signals are often very weak with only millivolts of energy. This energy is not sufficient for computer sampling. Thus, signal conditioning circuits strengthen, or amplify, the energy in the signal into the volts range.

Microcontrollers and **microprocessors** can both serve as the embedded system "brain."

- Microprocessors are single-chip processors. Memory chips and I/O devices must be added on the motherboard of a microprocessor-based embedded system. This increases the space and cost constraints. However, microprocessors are often used when speed or advanced numeric calculations are required.
- Microcontrollers are single-chip computers. Memory and I/O devices are implemented on chip. This reduces the space and cost constraints. Microcontrollers are preferred if high-speed and advanced numeric calculations are not required.

Microcontroller I/O devices facilitate interfacing to sensors and actuators.

- **Software-controlled port pins** provide attachment points for sensor and actuator signals.
- **Analog-to-digital converters** sample analog sensor signals into binary bytes for software use.
- **Communications devices** send data using SCI, SPI, I^2C, USB, CAN, PS2, Ethernet or ZigBee.
- **Timers** provide high-precision timing of real-time constrained input and output waveforms.
- **Interrupt controllers** provide immediate response to real-time system events.
- **EEPROM memories** provide non-volatile storage for event or data logging.

Microcontrollers dominate the embedded systems market.

- The 4-bit microcontroller is widely used in lower-cost products. Some examples are watches, razors, toothbrushes, and toys.
- The 8-bit microcontroller is the most commonly used microcontroller in the embedded systems market with billions of them powering automotive systems, elevators, appliances, and many other products!
- The 16-bit and 32-bit microcontrollers are used in products such as PC peripherals, digital cameras, advanced instrumentation panels, and military systems.
- **Remember** that ***any*** product can be designed around ***any*** microcontroller!

ATMEGA32 EEPROM programming requires the use of three I/O control registers.

- The 16-bit **EEPROM address register** is used to specify one of the available 1024 EEPROM locations. The address register is organized as two distinct 8-bit registers called **EEARH** and **EEARL,** where EEARH corresponds to the high byte of the address and EEARL corresponds to the low byte of the address. **EEARH** and **EEARL** allow byte-size data movement using the **ldi** and **out** instructions.
- The 8-bit **EEPROM data register** is used to write data into the EEPROM after specifying the address during an EEPROM write-cycle. It also contains the read data from the specified address during an EEPROM read-cycle.
- The 8-bit **EEPROM control register** is used to coordinate read and write cycle behavior. The **EECR** has only four control signals in the lower nibble and no control signals in the upper nibble. Users can write 0 over the upper nibble or use **sbi** and **cbi** assembly language instructions or **and-or** mask instructions in C on the lower nibble bits. **Refer** to the Atmega32 reference guide for a description of the four control bits.

ATMEGA32 EEPROM write-cycle control requires seven steps:

- **Step 1: Assign** the EEPROM address to EEARH and EEARL. **Assembly language programmers** should use **ldi-out** instructions to move each byte separately. **C language programmers** can set all 16-bits in one assignment by assigning a 16-bit pointer value directly to the EEAR register variable.
- **Step 2: Assign** the EEPROM data to EEDR.
- **Step 3: Disable interrupts** – interrupting an EEPROM cycle can cancel it.
- **Step 4: Poll** the EEWE bit in the control register EECR until it is zero. This guarantees that no EEPROM write is in process when starting a new write.
- **Step 5: Set** the EEMWE bit in the control register. This bit is the "master write enable" bit. It will be automatically cleared when the write-cycle completes.
- **Step 6: Set** the EEWE bit in the control register. This bit is the "write enable" or "write-eeprom" bit. **If** EEMWE has also been set **then** an EEPROM write will occur. **Note** that this bit actually causes the write-cycle to EEPROM memory but depends on the previous step. This is a protection mechanism to prevent accidental EEPROM overwrite. This bit will be automatically cleared when the write-cycle completes. **Poll** this bit (step 4) before writing another byte to the EEPROM.
- **Step 7**: **Re-enable interrupts** – interrupts can be re-enabled after the write completes.

ATMEGA32 EEPROM read-cycle control requires five steps:

- **Step 1: Assign** the EEPROM address to EEARH and EEARL. **Assembly language programmers** should use **ldi-out** instructions to move each byte separately. **C language programmers** can set all 16-bits in one assignment by assigning a 16-bit pointer value directly to the EEAR register variable.
- **Step 2: Disable interrupts** – interrupting an EEPROM cycle can cancel it.

- **Step 3: Poll** the EEWE bit in the control register EECR until it is zero. This guarantees that no EEPROM write is in process when starting a new read. Only one EEPROM operation can be in progress at any time.
- **Step 4: Set** the EERE bit in the control register. This bit is the "read enable" bit. It will be automatically cleared when the write-cycle completes. **Note** that this bit actually causes the EEPROM read-cycle. This bit will be automatically cleared when the read completes and thus can be polled but there is no need because EEPROM read-cycles complete in one instruction and thus the EEDR can be read immediately.
- **Step 5: Re-enable interrupts** – interrupts can re-enabled after the read-cycle completes.