Overall system

- Embedded microcontroller
- Sensors
- Actuators
- External Environment (physical devices, surrounding environment, human user, etc.)
External environment as seen by an embedded microcontroller

- The physical components of the system outside the microcontroller. For example, in a robotic system, the physical components on the robot are part of the external environment as seen by the embedded microcontroller.

- The outside environment around the overall system is part of the external environment. For example, for a mobile robot, the external environment in which the robot is operating is part of the external environment.

- A human user of the system can also be part of the external environment. For example, a human user sending commands to a mobile robot.

- In a distributed system (wherein the overall system is comprised of multiple embedded systems, e.g., a multi-robot system), the other systems are also part of the external environment.
Typical Software Structure in Embedded Systems

- Multi-layer code architecture
- Layers such as:
  - processor start-up layer (e.g., initialization of stack and heap, initialization of interrupt vector table, etc.)
  - peripheral interface layer (e.g., code to interact with peripheral interfaces such as GPIO, etc.)
  - device driver layer (e.g., code to talk to specific sensor and actuator components)
  - ... other hardware-related layers depending on overall system structure
  - application-level code (e.g., sensor data processing and control algorithms, application logic, etc.)
- Using simulation methods, we want to test the embedded system code as much as possible (i.e., in as varied and as realistic conditions as possible) without needing to necessarily use the actual physical components of the external system.
Software-In-The-Loop (SITL) simulation: just extract the application-level code that would normally run on the embedded microcontroller and run it instead on a computer interfaced (in software) with simulation models of sensors, actuators, and external environment.
Hardware-In-The-Loop (HITL) simulation: interface to the actual embedded microcontroller system; instead of actual sensors, actuators, and external environment, use simulation models. Emulate the hardware interfaces from sensors and actuators to the microcontroller.

- **Embedded Microcontroller**
  - **Microcontroller hardware board with embedded code**
  - **Emulation software models of sensors and actuators**
  - **Emulations of hardware interfaces to microcontroller**

- **Simulation of external environment (or some parts of external environment)**

**Diagram Description:**
- **Embedded Microcontroller**
- **Sensor Emulations**
- **Actuator Emulations**
- **Simulation of external environment**
Simulation of embedded systems

- An approach in between SITL and HITL is to simulate the embedded microcontroller on a simulation computer.
  - Example: hardware emulator such as QEMU virtual machine emulator
- When simulating an embedded microcontroller, many levels of simulation are possible:
  - simulating the hardware architecture of the processor, simulating the instruction set, simulating the I/O interfaces and peripherals, etc.
HITL: Interfacing between the Simulation System (e.g., a computer) and the Embedded System

- Analog interfaces: use analog-to-digital converter (ADC) and digital-to-analog converter (DAC) components connected to simulation system to interface to embedded microcontroller.
  - Example: if an embedded microcontroller is to be interfaced to an analog sensor, then an emulation of the sensor interface could be implemented by using a DAC connected to the simulation computer. The numerical simulation on the computer writes to the DAC. The analog output of the DAC is read by the microcontroller just as the analog output of the actual sensor would have been read.
  - Example: if an embedded microcontroller is to be interfaced to an actuator that takes analog inputs, then an emulation of the actuator interface could be implemented by using an ADC connected to the simulation computer. When the microcontroller writes an analog command (that the microcontroller thinks is going to an actuator), the analog command is read by the ADC and fed (in software) to the numerical simulation on the computer.
Digital interfaces: As with analog interfaces, digital interfaces (e.g., GPIO, serial port, SPI, I2C, etc.) can be emulated by utilizing the appropriate components connected to the simulation computer.

Example: if an embedded microcontroller is to be interfaced to a sensor via a serial port interface, then an emulation of the sensor interface could be implemented by using a serial port interface connected to the simulation computer (e.g., a USB-to-serial converter connected to the simulation computer). The numerical simulation on the computer writes to the serial port. The serial port data is read by the microcontroller just as the output of the actual sensor would have been read.
HITL: Interfacing between the Simulation System (e.g., a computer) and the Embedded System

- Emulating the actual analog/digital interfaces of the sensors and computers is the ideal approach for HITL simulation since the microcontroller then sees an external environment during HITL simulation that is as close as possible to the actual external environment during actual system operation.

- However, if emulation of the specific analog/digital interfaces for some sensor or actuator involves more complexity than desired, then a small layer of stub code with *mock* interfaces can be utilized to mimic the data I/O of the actual interface with a separate interface mechanism while preserving the structure of the rest of the code.

  - For example, using a serial port to send data that mimics an analog interface if an ADC or DAC is not available to connect to the simulation computer.
  - Another example: using a debugger interface to directly change variables or processor registers. This can be automated and integrated into the overall HITL simulator.
Real-Time Simulation

- **Real-time:** simulation runs at 1:1 time vs. actual system, i.e., 1 second of elapsed time during simulation matches 1 second of actual time.
- Depending on complexity of the system models and for ease of testing, HITL simulation might be sometimes configured to be faster-than-real-time or slower-than-real-time.
- To ensure that the simulation computer and the embedded microcontroller operate on the same time base (e.g., if the simulation program on the computer is configured to run at 2x real-time, and if the microcontroller code is supposed to execute 50 iterations of a while (1) loop in 1 second of actual time, then, to match the time base of the simulation computer, the microcontroller should execute 100 iterations of the loop in 1 second of time during simulation).
  - Clock synchronization methods (e.g., sending a trigger signal) might be utilized to ensure synchronization of the time base on the simulation computer and the microcontroller.
Simulation enables safe testing under a wide range of scenarios.

- Testing of system behavior under various types of conditions.
- Scenarios can be set up and replicated quickly.
- Sensors, actuators
  - noise, malfunctions, faults, etc.
- Disturbances affecting physical system behavior.
- Various ambient environment characteristics.
Modeling of the external system

- Numerical simulation on, for example, a computer. This can be in C, C++, Matlab, etc. Also, specialized software simulation platforms such as v-rep (for robotic systems) can be used.

- In the past, analog simulators were commonly used (e.g., modeling dynamics of an aircraft through an electrical circuit designed to have similar dynamics)

- Various types of models
  - Continuous-time dynamic system models, e.g., a set of differential equations; for example, $M \ddot{x} = F$ for an object moving on a planar surface ($M$ is the mass, $x$ is the position, $F$ is the force)
  - Discrete-time system models, e.g., a set of difference equations
  - Discrete-event modeling – definitions of events and system behavior under events, e.g., a set of logical rules
  - Hybrid system models, e.g., combination of some continuous-time dynamic subsystems and some discrete event subsystems
Modeling of the external system

- Various characteristics of models:
  - Single-variable vs. multi-variable models
  - Linear vs. nonlinear models
  - Time-invariant vs. time varying (i.e., whether the model includes some explicit dependence on a time variable)

- Simplified models – exact modeling of a physical system is usually not possible. The model only needs to be of fidelity sufficient to enable reasonable testing of the overall system behavior.
Modeling of sensors and actuators

- Sensors measure signals from the external environment. Actuators output signals that influence (in some way) the external environment.

- Sensors and actuators can be modeled through a combination of two parts:
  - a nominal relationship between a physical signal and a measured/commanded signal
  - a model of noise and other non-idealities

- Nominal relationships between physical signals and measured/commanded signals
  - Example: Analog temperature sensor: \( v = aT + b \) where \( v \) is the analog output of the sensor, \( T \) is the ambient temperature, \( a \) and \( b \) are some constants.
  - Example: force actuator with digital input over serial port (with 12-bit command values); \( F = a\frac{x}{2^n} + b \) where \( F \) is the force output, \( x \) is the command sent over serial port, \( n = 12 \), \( a \) and \( b \) are some constants.

- Models of sensor and actuator characteristics: sampling rate, resolution, accuracy, noise characteristics, etc.
Modeling of human inputs

Human users can be modeled in multiple ways:

- **Human-in-the-loop**: actually use the human user to interact with the HITL simulator
- **Record a sequence of human user inputs and replay**.
- **Simulate a sequence of human user inputs.**
  - Timing characteristics should be chosen to reasonably model a human user.
  - However, to test system robustness to faulty user interface hardware (e.g., faulty wiring, faulty components, etc.), the testing can include simulated human user input sequences that are not very reasonable. For example, pressing of a button hundreds of times a second.
Modeling of embedded system program behavior

So far, we talked about modeling external components (outside the microcontroller). Now, we will look at ways to model the behavior of code on the embedded microcontroller.

But, why model behavior of *embedded programs*?

- Modeling can be used as part of design and defining requirement specifications for a system.
- Using modeling and/or simulation to refine/optimize system design, tune system parameters, etc.
  - For example, design a model of the embedded program behavior, implement a prototype in a higher-level language, and test in a pure software simulation *before actually writing code for the embedded microcontroller*.
  - Modeling can help in reducing development time and cost; modeling also helps in quick design iterations
  - Modeling helps in creating reusable blocks of functionality (both hardware and software)
  - There are also code generation systems that can use a model in specific formats (e.g., state machine models) to automatically generate code for an embedded microcontroller.
Modeling of embedded system program behavior using Finite State Machines (FSMs)

- Modeling of a system as being in one of a finite number of states; transitions between states occur upon *events*.
  - Next state is a function of the current input and current state.
  - Edge labels have the notation `event[condition]/reaction`
    - The notation `event[condition]/reaction` indicates that a transition happens when the *event* occurs if the *condition* holds and that the *reaction* also occurs (e.g., setting of some variables to some values) when the transition occurs. If there is no specific *condition*, then the notation `event/reaction` is used.
- Mealy and Moore automata (state machines). In Moore automata, the output is a function of the current state. In Mealy machines, the output is a function of the current input and the current state.
Modeling of embedded system program behavior using Finite state machines (FSMs)

Pure FSMs may not be a practical approach. Number of “states” may be too high.

- Example: a counter (an integer that is incremented every time a loop runs)
- Concurrent “threads” (in the context of modeling, interrupt service routines are like concurrent “threads”)
  - If two state machines are concurrently operating with the first having 10 possible states and the second having 5 possible states, the combined state machine would have 10*5 = 50 possible states.
- Solution is to generalize the concept of state machine and concept of state
  - A very useful tool to simplify state machine models is hierarchy: each state is itself a state machine or, more generally, a process.
State machine models can be simplified using hierarchy and concurrency.

- In a hierarchical state machine structure, the state machine can have *superstates* which have embedded state machines within them. When the state machine is in a particular superstate, then the corresponding embedded state machine within the superstate is considered to be running.

- In a concurrent state machine structure, multiple state machines (which can communicate or share variables) are considered to be running at the same time (i.e., concurrently). In general, a state machine model can be written using both hierarchy and concurrency concepts to simplify the state machine structure.
Modeling of embedded system program behavior using hierarchy and concurrency

- Hierarchy and concurrency can be thought of in terms of AND states and OR states.
  - A hierarchical state machine structure with each superstate corresponding to one of multiple possible “substates” (such as substates \( \text{num\_items}=0, \text{num\_items}=1, \text{etc.} \), within a superstate) corresponds to OR states (since a superstate is in the first possible substate OR second possible substate OR third possible substate, etc.).
  - A concurrent state machine structure wherein multiple state machines are considered to be running simultaneously corresponds to AND states since the overall system is in a state corresponding to the state of the first state machine within the system AND the state corresponding to the second state machine within the system, etc.
The concept of state can be generalized to a concept of a computation unit (an *actor*) that runs a specified computation (e.g., some lines of code).

A program/system is a directed graph (i.e., nodes with arrows between them) with nodes being *actors* (computations) and edges (arcs) being sequences (*streams*) of events (*tokens*).

Hierarchical graph: an *actor* in the graph might itself be a directed graph.

The actors execute on a sequence of *firings* (which can be thought of as indivisible quanta of computation). Each firing can consume and produce tokens.
Example of modeling embedded system program behavior: A simple calculator

- Consider an embedded microcontroller that is used to implement a very simple calculator, which provides only “increment by 1” and “decrement by 1” functionalities. The calculator has a Power button which switches it ON or OFF. Initially, the calculator is off.

- When the user presses the Power button, the calculator switches on and initializes a counter to 0. Every time the user presses the Increment button, the counter value increases by 1 and every time the user presses the Decrement button, the counter value decreases by 1. At any point, the user can press the Power button to switch off the calculator. Also, if the user does not press any of the buttons for 60 seconds, the calculator should automatically switch off.

- Events: TimerEvent, IncrementPressed, DecrementPressed, and PowerPressed
Example of modeling embedded system program behavior: A simple calculator

State machine model for the given system. The notation `event[condition]/reaction` indicates that a transition happens when the `event` occurs if the `condition` holds and that the `reaction` also occurs (e.g., setting of some variables to some values) when the transition occurs. If there is no specific `condition`, then the notation `event/reaction` is used. Here, it is assumed that the `TimerEvent` resets on any button press (i.e., the `TimerEvent` occurs if the user does not press any button for 60 seconds).
Example of modeling embedded system program behavior: A simple calculator

Embedded model within ON:
- IncrementPressed/inc_dec = +1
- DecrementPressed/inc_dec = -1

A hierarchical model for the given system with a generalization of the concept of state (as actors). The ON state includes an embedded model.
Timing and Data Communication models

Various timing models:
- Continuous-time
- Discrete-time
- Synchronous, Asynchronous
- Discrete-event

*Time* typically has a central role in embedded systems.
- *Real-time* requirements
- Combination of multiple timing structures ... multi-rate synchronous blocks, interrupt handlers, etc.

Various data communication models:
- Unsynchronized
- Synchronized
- FIFO