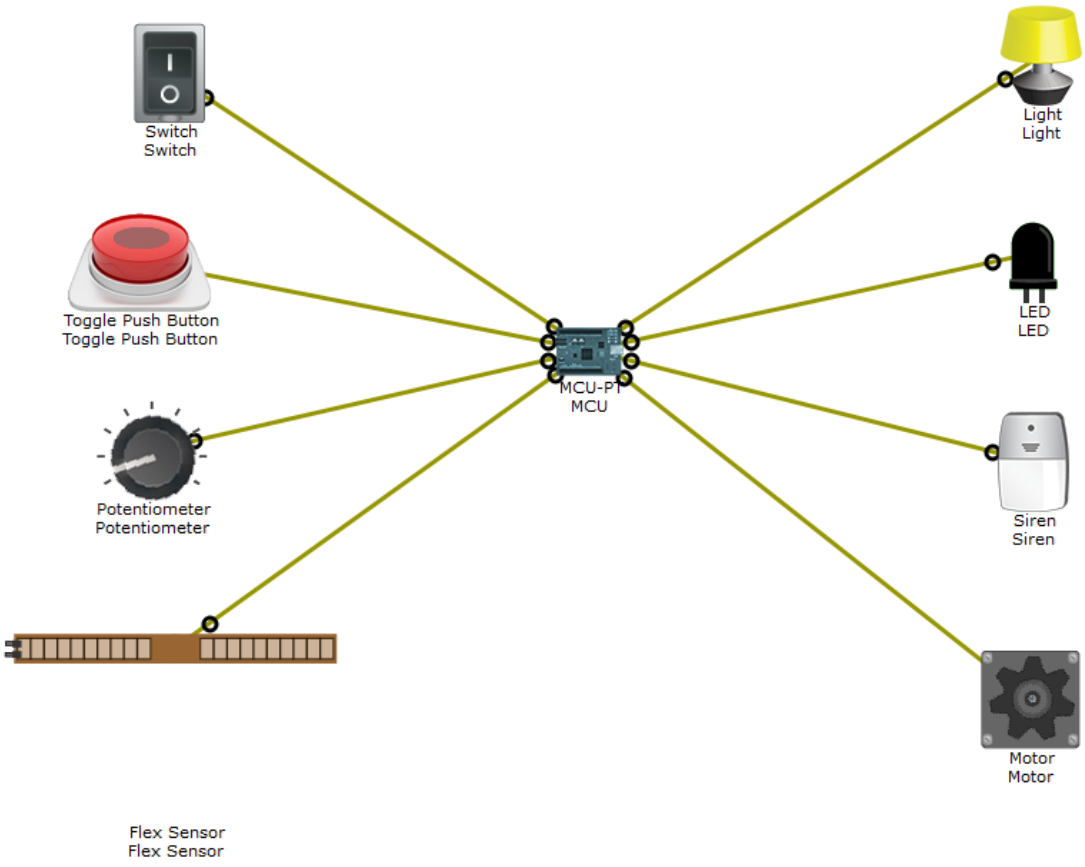
Packet Tracer - Sensors and the PT Microcontroller

Topology



Objectives

Part 1: Get familiar with sensors in Packet Tracer (PT) 7.1

Part 2: Get familiar with the PT Microcontroller (PT-MCU) in PT 7.1

Part 3: Get familiar with PT Microcontroller (PT-MCU) programming in PT 7.1

Background / Scenario

A microcontroller unit (MCU) is a small computer built on a System on a Chip (SoC). It contains a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications or applications that require few computer resources. Conversely, the microprocessors used in personal computers are commonly used to support other general purpose applications that require more computer resources.

Examples of applications that rely on microcontrollers are automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys, and other embedded systems. Mixed signal microcontrollers are also common, integrating analog components needed to control non-digital electronic systems.

Packet Tracer 7.1 brings support for an MCU emulator. The user can program the PT MCU to perform tasks similar to real-word MCUs. To simplify the process, the PT MCU can be programmed with Java and Python.

In this activity, you will launch MCUDEMO.PKT in Packet Tracer 7.1 and familiarize yourself with the PT MCU emulator and its programming aspect.

Required Resources

- 1 PC with Packet Tracer 7.1 or newer installed
- MCUDEMO.PKT file

Part 1: Getting Started

The PT MCU is a board that has a USB port, six digital I/O ports, and four analog I/O ports. The digital I/O ports on the PT MCU allow a user to connect digital sensors and actuators. The analog I/O ports allow a user to connect analog sensors and actuators. In this demo, there is a single PT MCU connected to four sensors and four actuators. The four sensors include a digital switch, a digital toggle push button, an analog potentiometer, and an analog flex sensor. The four actuators include a light, an LED, a siren, and an analog motor. The PT MCU is programmed in Python to continuously read the sensor values and then write to the actuators, governed by conditional logic. The relationship between the sensors and actuators can be summed up by:

- The Switch controls the Light
- The Push Button controls the LED
- The Potentiometer controls the Siren
- The Flex Sensor controls the Motor

Part 2: The Sensors and the PT MCU

a. Take a moment to analyze the topology. The **MCU** is placed in the center. The input devices (sensors and switches) are placed on the left and the output devices to the right.

b. Explore the MCU, sensors, switches and controlled devices to open their configuration windows. Notice that different devices have different tabs available.

   In the **MCU** window, the **Programming** tab holds the running Python code. The Python code defines the device's behavior.

c. ALT + click (hold the ALT key while clicking a device) to interact with a device.

   ALT + click the **switch** to turn on/off the **light**.

   ALT + click the **push button** to turn on/off the **LED**.

   ALT + click, hold and drag the **potentiometer** to control the volume of the **siren**.

   ALT + click, hold and drag the **flex sensor** to control the speed of the **motor**.

d. Take note of what **MCU** ports the sensors, switches, and devices are connected to.

Part 3: Programming the MCU

Note: Python used in PT is an open source Python to JavaScript interpreter that is not updating to Python 3.0. For this reason there may be slight differences in the syntax between the code observed in PT and that in devices using Python 3.

a. Open the **MCU** located in the center of the topology.

b. Select the **Programming** tab to gain access to the Python code running on the **MCU**.

c. Take a look at the code and try to understand it. Below is a summary of the tasks it performs:

In lines 1 and 2, all classes in the **time** and **gpio** libraries are imported into the program. This is important to give access to time and general purpose input/output (GPIO) functions.

Four global variables are then declared and initialized in lines 4, 5, 6 and 7. These variables are going to be used to hold sensor values.

A function called **readFromSensors()** is then created (lines 9 through 18). First, the function prepares the sensor variables (lines 10 through 13). The **readFromSensors()** function then calls two other functions **digitalRead()** and **analogRead()** to capture the status of the sensors and store it in the appropriate variables (lines 15 through 18). Notice that **digitalRead()** and **analogRead()** take a pin number as a parameter. By connecting specific sensors to specific pins, the program is able to capture specific sensor status.

Another function is then created in lines 20 through 39: **writeToActuators()** is used to change the status of the actuators based on the status of the sensors. In line 21 through 24, the program tests the contents of the **switchValue** variable. Since **switchValue** stores the status of pin 0 (see line 15), the program can decide if it should turn on the light; if the value stored in **switchValue** is equal to HIGH (voltage is applied or switch is ON), then the program turns on the light by writing the value 2 to the actuator 2. Conversely, if **switchValue** equals to LOW (no voltage or switch is OFF), the program turns off the light by writing 0 to the actuator 2.

Similarly, lines 26 through 39 test and modify other actuators based on their respective controlling sensors.

Lines 41 through 54 define the **main()** function. As stated by its name, the **main()** function is automatically executed when the MCU is first turned on. Lines 42 through 46 initialize the pins; pins 0 and 1 are setup as INPUT (lines 42 and 43) while pins 2, 3, and 4 are setup as OUTPUT. This is important because INPUT pins receive voltage while OUTPUT pins emit voltage generated by the MCU itself.

An infinite **while** loop is created in lines 48 through 51. Because the condition in the loop simply states **true**, the MCU will be executing lines 49, 50, and 51 forever. This infinite **while** loop forces the MCU to:

1. Run the **readFromSensors()** function in line 49.
2. Run **writeToActuators()** function in line 50.
3. Wait 1 second in line 51 (1000 ms = 1 second).
4. Restart the loop from the top by going back to line 49 and running **readFromSensors()** function again.

Notice that while infinite loops are usually undesired, it is useful in this program; the infinite loop here ensures the **MCU** is always checking the status of the sensors and switches (by running **readFromSensors()** every second) and always taking appropriate action towards the actuators (by running **writeToActuators()**) based on the status of the sensors.

d. Currently, the light is controlled by the switch and the LED is controlled by the push button. Modify the code to make the switch control the LED and the push button control the light.

Part 4: Reflection

The introduction of the programmable PT MCU in Packet Tracer 7.1 allows for a powerful IoT simulation environment. The use of Python as programming language also contributes for a robust platform.

a. CHALLENGE 1: Port SparkFun Starter's Kit circuit 1, "Blinking an LED" into Packet Tracer 7.1, using the PT MCU as the microcontroller.

   Tips: You will have to port the code presented on SIK to Python.

   You will also have to modify the pins used on SIK to adapt to PT MCU's pin numbering system.

b. CHALLENGE 2: Using the concepts presented on **SparkFun Starter's Kit** circuit 1 **Blinking and LED**, circuit 4 **Multiple LEDs** and circuit 5 **Push Button**, use Packet Tracer 7.1 or newer to create a circuit that illuminates one of eight LEDs in sequence, every time the push button is pressed.

Requirements:

You must use 8 LEDs and they must be lined up.

Every time the **Push Button** is pressed, the currently lit LED goes dark and the next one illuminates.

There must be only one LED lit up at any given time.

Tip: Use the PT MCU as your microcontroller.