Chapter 6: Class Project

The purpose of this chapter is to tie together everything you have learned to create an IoT connected thermostat. It will use CAPSENSE™ buttons to control the desired (i.e. set) temperature and measure (or simulate) the local temperature. Then, it will set the mode of the thermostat based on the temperature values.

The display (either OLED or TFT) will be used to show information about the set temperature, measured temperature, mode, and Wi-Fi connection status.

Changes in the set temperature or measured temperature will be published to an AWS MQTT server. Changes on the cloud side to the set temperature will be sent to the device.

At the end of this chapter you will have created an IoT thermostat simulator.

Table of contents

6.1 Introduction 2
6.2 Details and hints 3
6.3 Example firmware architecture 5

Document conventions

Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

6.1 Introduction

Your project is to build an IoT thermostat. It will connect to the class AWS account and will publish updates to a *thing* which you will create. If you are using the CY8CKIT-028-SENSE shield, you will read the temperature from the pressure sensor. If you are using the CY8CKIT-028-TFT shield, you will simulate the temperature by reading the potentiometer on the CY8CKIT-062S2-43012 kit. You will use CAPSENSE™ buttons to adjust the "set temperature" for your thermostat. Based on the set temperature and measured temperatures, you will determine what mode the thermostat should be in (heating, cooling, or idle) and show this data on the display (OLED for the CY8CKIT-028-SENSE or TFT for the CY8CKIT-028-TFT).

The required functionality is:

- [ ] 1. Use I2C to read the temperature from the pressure sensor on the CY8CKIT-028-SENSE shield or use an ADC to read the voltage of the potentiometer on the CY8CKIT-062S2-43012 kit and use the value to simulate a range of temperatures between 50˚ F and 90˚ F.
- [ ] 2. Use the CAPSENSE™ buttons to adjust the set temperature for your thermometer.
- [ ] 3. Use the actual and set temperatures to determine what mode your thermostat needs to be in. Light the red user LED for heating and blue user LED for cooling.
- [ ] 4. Show the actual temperature, set temperature, mode (off, cool, or heat), and connection status on the display.
- [ ] 5. Any time the actual or set temperatures change, publish the new data to the cloud.
- [ ] 6. Subscribe to the current/state/desired topic of your *thing*'s shadow to determine when the set temperature has been updated remotely and update the local set value.
- [ ] 7. Advanced: Implement the IoT functionality using HTTPS instead of MQTT.

6.2 Details and hints

If you are using MQTT, it is probably best to start with the MQTT Client Example (see Chapter 4C).

You will edit the message so that it sends JSON messages to update the shadow instead of just alternately sending TURN ON and TURN OFF.

If you are using HTTP, the HTTP Bin example is a good starting point (see Chapter 4B). You will use POST requests to send your data to the server.

You will connect to the class AWS IoT endpoint:

amk6m51qrxr2u.iot.us-east-1.amazonaws.com

Note: If you are doing this class on your own, you should use your personal AWS account.

Your *thing* name will be "thermostat_<init>" where <init> will be your initials.

If you used the class AWS account for previous exercises, you may reuse a previously created *thing*.

Use I2C to read the temperature from the pressure sensor (for the CY8CKIT-028-SENSE) or use the ADC to measure the voltage on the potentiometer to simulate a valid temperature range (for the CY8CKIT-028-TFT).

Note: The ModusToolbox™ PSoC™ MCU class peripherals chapter includes an exercise that reads the temperature from the pressure sensor on the CY8CKIT-028-SENSE shield and displays the value to the OLED. You should read the values on a regular basis (e.g. every 500ms). It is recommended to have an RTOS task to handle reading the temperature value. Whenever the actual temperature changes, be sure to update the display and the cloud.

Use CAPSENSE™ buttons to control the set temperature

You will use the CAPSENSE™ buttons to change the set temperature. Look for a CAPSENSE™ code example. It is recommended to have an RTOS task to handle CAPSENSE™. Whenever the set temperature changes be sure to update the display and publish the new value to the cloud.

Show information on the display.

The display for your *thing* should look something like this:

- Actual Temperature: 72
- Set Temperature: 68
- Mode: Cooling
- Connected

If you are using the CY8CKIT-028-TFT, you can even display bitmaps with graphics to indicate the heating and cooling states. For example:

Heating:  Cooling: 

The starting (empty) shadow for your *thing* should look like the following. You will publish (MQTT) or post (HTTP) JSON messages to the *thing* shadow to provide updates.

Shadow State:

```
{
  "reported":{
      "actualTemperature": 0,
      "setTemperature": 0,
      "mode": "Idle",
      "IPAddress": "0.0.0.0"
  }
}
```

You can use the `snprintf` function to create the JSON messages.

Remember that spaces and carriage returns are not required in the JSON message. Also remember that quotation marks in the message must be escaped with a \ character in the code. For example, to create a JSON message to send the actual temperature from the integer variable actualTemp, you could do something like this:

```
char json[100];
snprintf(json, sizeof(json), "{\"state\":{\"reported\":{\"actualTemperature\":%d}}}",
actualTemp);
```

Note: *snprintf was used instead of sprintf so that sending more characters than the buffer size will not lead to a pointer overflow error, which can be very difficult to debug. However, you should still make sure the buffer is large enough for the message or else your formatted JSON message will not be correct.*

Note: *When doing initial testing, use the MQTT Test Client on the AWS site to examine the messages that you are sending. Note that this works even if you are using HTTP to post data since the update to the shadow still causes a notification to any MQTT subscribers. For example, to see all shadow messages for the thing named: thermostat_key, you would subscribe to:*

$aws/things/thermostat_key/shadow/#

Messages that show up on the topic $aws/things/ thermostat_key /shadow/update are the messages that you are sending. You will also see messages that tell you whether the broker accepted or rejected your update on …/update/accepted or …/update/rejected

Once you see that the broker is accepting your updates, go to your *thing* and click on the Shadow. You will then see the data that is published by your *thing* in real time.

Getting a new set temperature from the Cloud

In addition to sending the actual and set temperatures to the cloud, you should also be able to change the set temperature from the Cloud and have it update on the thermostat.

If you are using MQTT, use the subscriber project as a reference. Many functions are common between the publisher and subscriber so you will not need to duplicate those.

If you are using HTTP, you will use a GET request to get the data required. In this case, you will need to poll occasionally for data since there is no concept of subscription in HTTP.

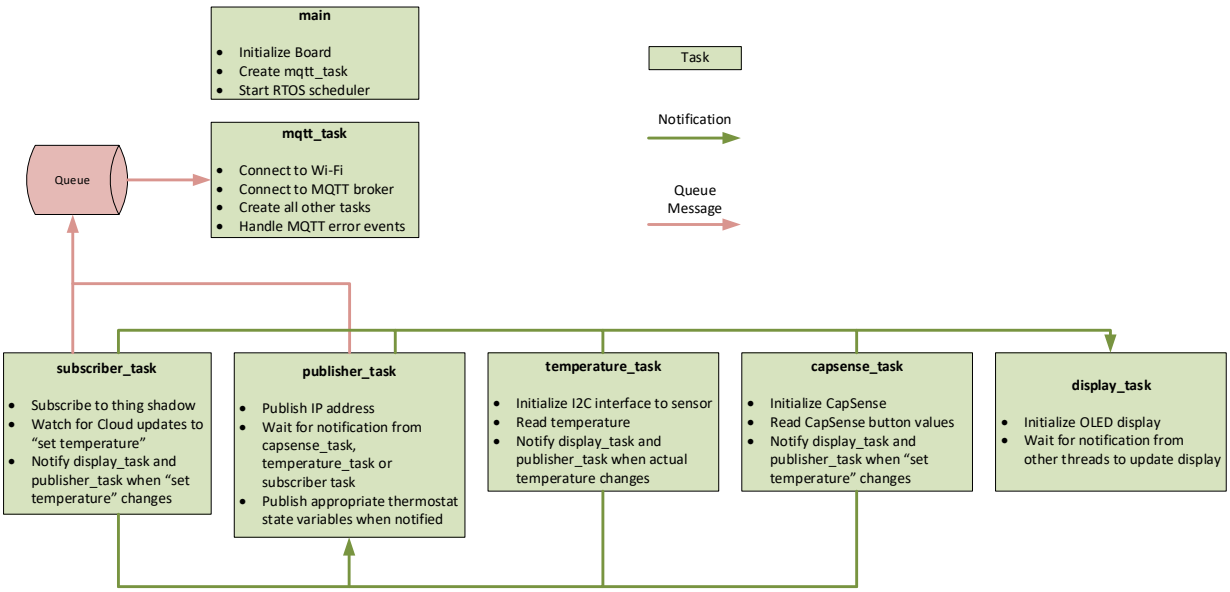
6.3 Example firmware architecture

For those having trouble getting started, we have prepared an example architecture of one way to approach this problem. To maintain modularity and reduce complexity it is HIGHLY RECOMMENED that you add additional functionality in new RTOS tasks. For example, you may want to separate the functionality as follows:

1. main: get everything up and going:
    a. Initialize the board and peripherals
    b. Create a main IoT task, semaphores, queues, mutexes, timers
    c. Start the IoT task
2. mqtt_task:
    a. Connect to Wi-Fi
    b. Connect to the broker (MQTT)
    c. Create and start all other tasks
    d. Handle IoT error events
3. subscriber_task: Handle MQTT subscriptions and messages
4. publisher_task: Handle MQTT publishing
5. temperature_task: Read temperature from the pressure sensor using I2C or simulate the temperature by measuring the potentiometer voltage using an ADC
6. capsense_task: Monitor CAPSENSE™ buttons
7. display_task: Update the display (either OLED or TFT)

Interaction between the tasks can (and should!) be controlled using notifications, semaphores, queues, and mutexes.

A pictorial representation of the architecture described above is shown here:



Note: *The figure shows the case for the CY8CKIT-028-SENSE shield. For the CY8CKIT-028-TFT shield, the main function, temperature task and display task will change, but the others should be identical.*

Note: *A solution is provided for both versions of the project using the architecture described above.*

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by

Infineon Technologies AG

81726 Munich, Germany

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.