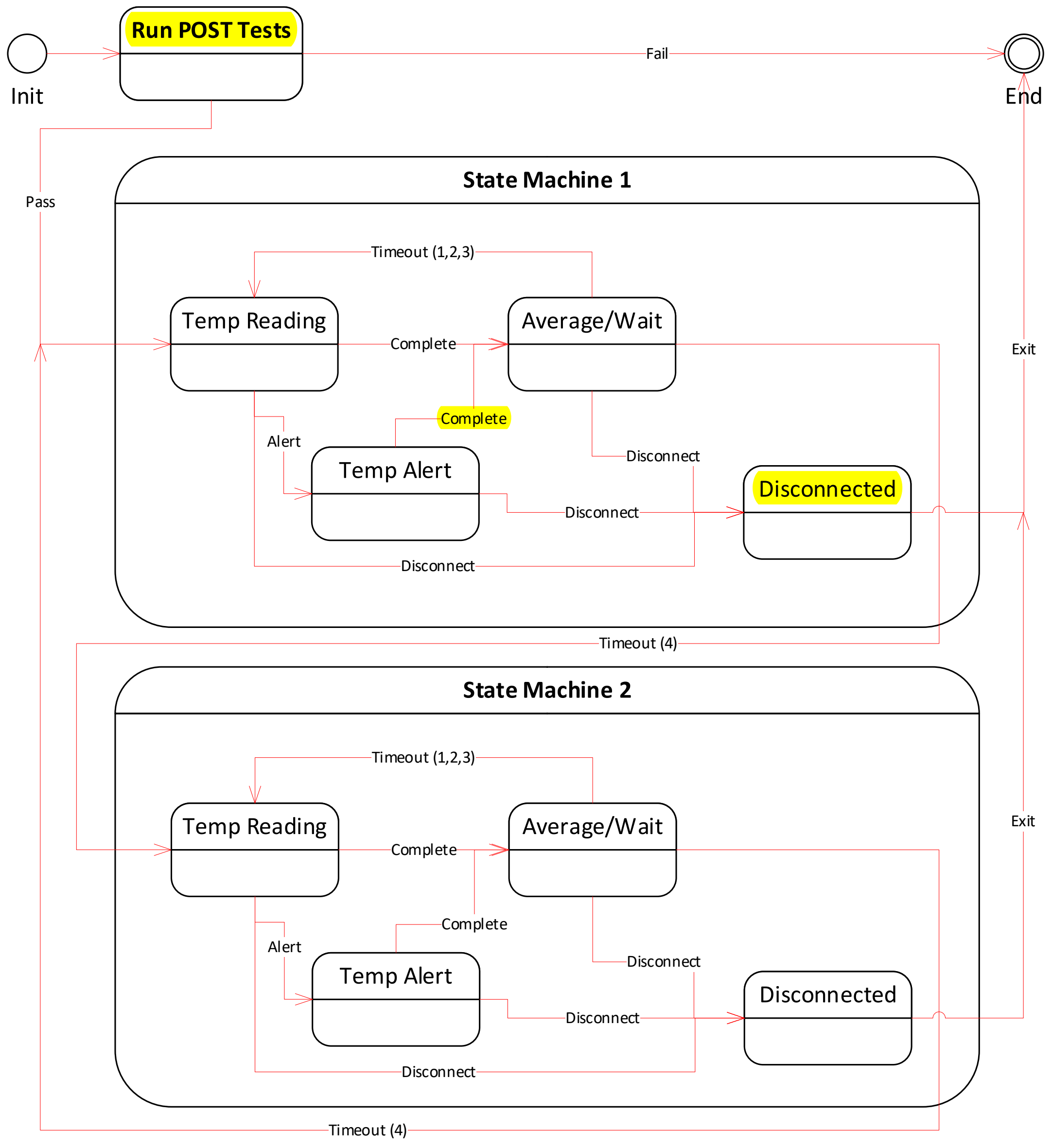
PES Project 4 – Sensor and State Machines - Total 100 Points

The fourth PES project will combine the KL25Z with an I2C-based temperature sensor, the TMP102. This setup will allow you to exercise more advanced firmware topics, including state machines, interrupts, timers, and I2C communications. This project is targeted to run on the KL25Z only – the program should be capable of running in three modes

1) In debug mode, with detailed debug messages logged to the UART.
2) In normal mode, with normal status messages logged to the UART.
3) In test mode, with detailed test messages logged to the UART.

Part 1. (80 points) State-based I2C Communications with Temperature Sensor (TMP102)

The main function of this KL25Z application will be managing communications with and monitoring of the TMP102 I2C temperature sensor using a POST (Power-On Startup Test) and a pair of state machines.



Process Steps (as shown in UML State Diagram above)

- Run a POST test to verify the TMP102 is working. If it is not responding, end the program, else start State Machine 1.
- Upon entering State Machine 1, the active state will be Temp Reading.
- In the Temp Reading state you will access the TMP102 for a temperature reading via I2C. There are three possible transitions from this state: Complete, Alert, or Disconnect.
    - On Complete, the state will change to Average/Wait.
    - On Alert, the state will change to Temp Alert.
    - On Disconnect, the state will change to Disconnected.
- In the Average/Wait state, you will take use the last temperature reading to update an average temperature value. Normal status messages should print the last reading and the current average. You will then wait 15 seconds for a timeout. Transitions from this state are: Timeout (1,2,3), Timeout (4), or Disconnect. Note that Alert transitions (interrupts) are disabled while in this state.
    - On Timeout, you will use a local counter to increment the number of timeouts that have occurred. If you are on timeout 1, 2, or 3, you will return to state Temp Reading. If you are on timeout 4, the counter should be reset to 0, and you should move to state Temp Reading on the other State Machine.
    - On a Disconnect event, the state will change to Disconnected.
- The Temp Alert state will be reached whenever the sensor detects a negative value for temperature. Alert transitions (interrupts) will be disabled while in this state. The temperature will be read as normal. Transitions from this state will be Complete or Disconnect.
    - On Complete, the state will change to Average/Wait.
    - On Disconnect, the state will change to Disconnected.
- The Disconnected state indicates the TMP102 is not responding for some reason. This state is reached via Disconnect events, which should occur if the sensor is physically disconnected. In this case, the transition if for the application to be ended.

Other Application Requirements

- The two state machines will be implemented differently. State machine 1 will be a state-oriented state machine as described in lecture. State machine 2 will be a table-driven state machine. Carefully consider communications and support functions that can easily be used by both state machine implementations, avoid duplicate code wherever possible.
- At a minimum, the POST test must verify operation of the connected TMP102. You may also wish to cycle the LED or perform other startup validation.
- In test mode, your program should execute a set of µcUnit test cases to verify expected system behavior of your choice. A minimum of ten test statements should be run, with logging of results. After the test cycle, the system should exit.
- You will need to connect the TMP102 sensor to your KL25Z. You'll want to be able to easily switch off or disconnect the sensor to allow testing of the disconnected states in the program.
- You will need to create a TMP102.h file that contains macros for accessing registers to control and monitor the TMP102 sensor board.

- You must control/monitor the TMP102 via I2C. Your code should use direct access of the I2C control registers found in MKL25Z4.h. You may not directly use the pre-provided I2C functions found in the SDK examples (FSL_I2C.h) but you may wish to review their operation to create your I2C functionality. The I2C communications basics will be reviewed in class. You will need to determine how to use I2C correctly for communications with the TMP102. (The SAs have experience working with the TMP102 boards.)
- You may use either interrupt-based communication OR a polling approach for I2C communication with the TMP102.
- You should use the Alert function of the TMP102 to set the alert level and to cause the alert event that drives the code to enter the Temp Alert state.
- You can use an alert temperature level of your choosing for testing. For demonstrations, your temperature alert should fire whenever the temperature is less than 0 degrees C. We have spray frost cans to cause the sensors to read negative temperatures.
- You should use modular design for your code – I2C communication functions in one module, state machine code in another, etc.

Support

- Information on the TMP102 Sparkfun board, including Arduino connectivity and typical higher level functions: https://learn.sparkfun.com/tutorials/tmp102-digital-temperature-sensor-hookup-guide
- Datasheet on the TMP102 chip from TI: https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf
- KL25 Reference Manual – includes I2C Communications in Chapter 38: https://www.nxp.com/docs/en/reference-manual/KL25P80M48SF0RM.pdf
- Example I2C Interrupt-based Communication: https://community.nxp.com/thread/319111 (code is in I2C_Example.zip) – Using the example would require some mapping to SDK provided .h files; the NVIC_ICPR is the equivalent of NVIC->ICPR in CMSIS/core_cm0plus.h, for instance.
- KL25Z I2C SDK examples in MCUXpresso (fsl_i2c.h/.c)

Logger Extensions

- Reuse your Logger code from Project 3, adding the following extensions.
- Create an enum for tracing your code by function name. Create an enum value for each function in your program. When that function logs a message, it provide its enum value to the logger function to identify itself. The logger will translate the enum to an appropriate string for logger displays.
- A new function, log_level, should be added. Another enum should be used for three log level settings – Test, Debug, Status. In Test mode, all messages will be printed. In Debug, only Debug and Status messages will be printed. In Status mode, only Status status messages will be printed.
- Logger functions will need an added argument to indicate whether the logging message is considered a Test, Debug, or Normal message. This should be also be indicated in the output string for the logging print.
- Example Messages (contains level, function name, message):
    - Test: log_level: log level set to Test
    - Debug: led_control: LED set to Blue

LED Control

- Reuse your LED control function from Project 3.
- The LED should be set to blue whenever the program is in the Temp Alert state, green when in the Temp Reading or Wait state, and red if the TMP102 is disconnected or otherwise fails.

Part 2 (20 points) – Capture Oscilloscope trace of I2C traffic between the KL25Z and the TMP102

You will need to capture both I2C read and write transactions between the TMP102 and the KL25Z using the SDA and SCL lines between the two elements.

Any clear image of the two transactions is fine. You should be able to annotate the image of the transaction to show key fields (start, address, data, stop, etc.) being transferred in your PDF submission.

If you have access to a logic analyzer, you could use that as an alternate for capturing the transaction waveforms.

Project Submission

The project is due on **Tuesday 11/5** prior to class and will be submitted on Canvas. The project will also be demonstrated to class staff (and when possible by remote students in video or web conferences). We will be setting up lab demo slots to allow a more detailed review of the submission with the student teams.

The Canvas submission will consist of two parts:

Part 1 is a single GitHub repo URL which will be accessed by graders to review code and documentation. This will consist of any C code or include files, captured output files, and a README Markdown document that includes:

- A title (PES Project 4 Readme)
- Names of your team
- A description of the repo contents
- Observations: A description of any issues or difficulties you encountered on the project and how they were addressed.
- Installation/execution notes: for others who may use the code – this should include compilation instructions for the SAs to more easily grade
- The repo should also contain the PDF with your annotated I2C transaction scope captures as discussed above
- Please include a Git tag called Final on your final submission to allow the SAs to be clear about what was submitted for grading
- Please note – code being pushed to Git should not be zipped – we should be able to pull project folders directly from your Git repos.

Part 2 will be a PDF containing all C code and README documentation – the PDF is used specifically for plagiarism checks: your code should be your team's alone, developed by your team. You should provide a URL for any significant code taken from a third party source, and you should not take code artifacts from other student teams. However, you may consult with other teams, the SAs, and the instructor in reviewing concepts and crafting solutions to problems (and may wish to credit them in documentation).

Development environment for the project

For this project, you will develop using the MCUXpresso IDE environment as shared in class demonstration. See the SAs for any assistance you need in your development environment.

Your code should follow the ESE C Style Guide as closely as possible.

When compiling use -Wall and -Werror compiler flags. Your code should have no compilation errors or warnings.

Grading

Points will be awarded as follows:

- 35 for the correctness of demonstrated code (execution of the three build targets – test, debug, normal) **– code will be demonstrated in class on the project due date, we will work with remote students for demos.**
- 40 for the construction of the code (including following style guide elements and the quality of solution)
- 5 points for the README
- 20 points for the captured and annotated I2C transactions
- No extra credit for this assignment

Project 4 is due on Tuesday 11/5 at 3:30 PM. Assignments will be accepted late for one week, at a penalty of 15% of the grade. After that point, assignments will not be accepted.