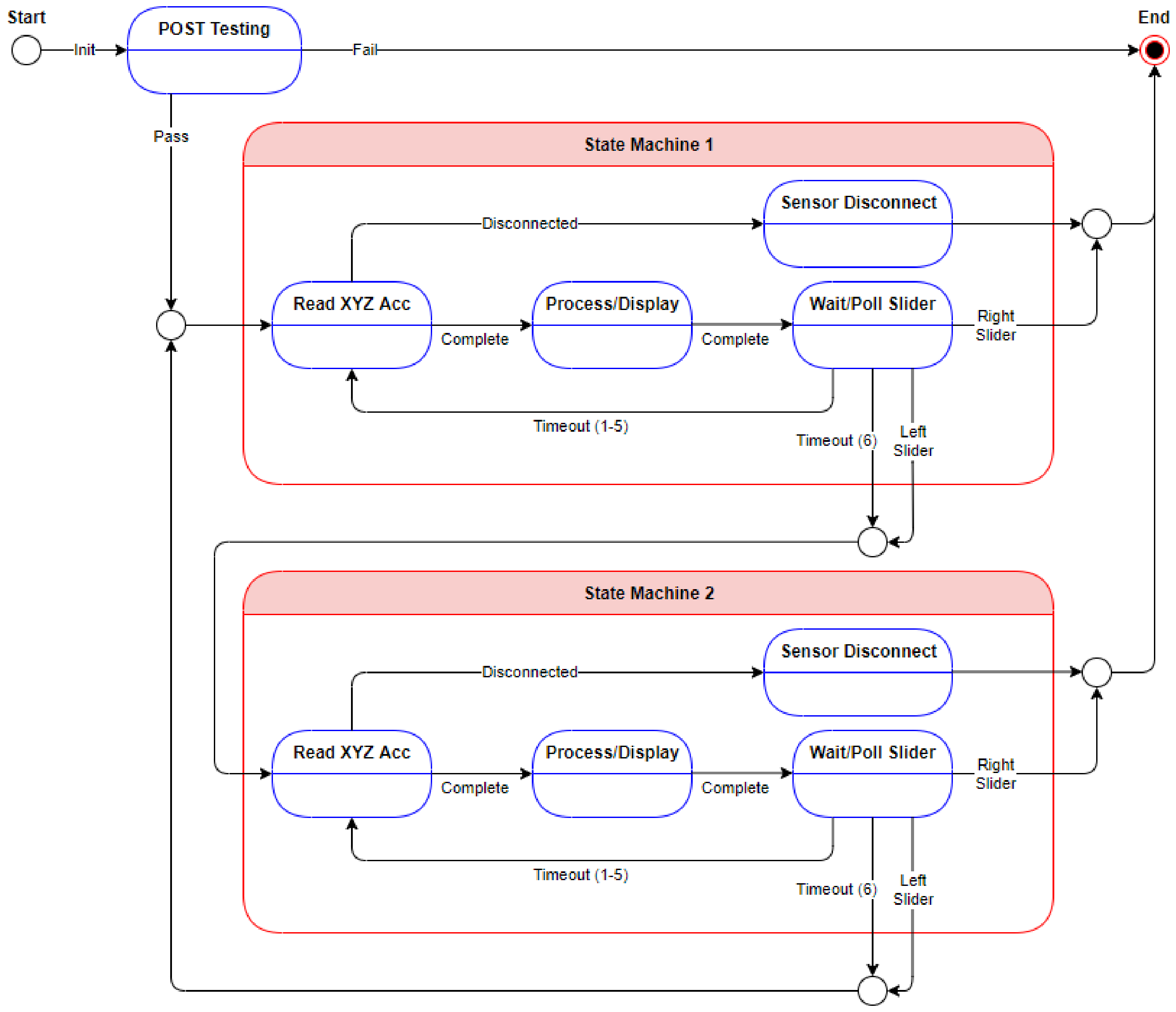
PES Project 4 – External Accelerometer and State Machines - Total 100 Points (+10 points extra credit)

The fourth PES project will combine the KL25Z with an I2C-based Accelerometer board, the LIS331. 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 Accelerometer (LIS331)

The main function of this KL25Z application will be managing communications with and monitoring of the LIS331 accelerometer 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 LIS331 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 Read XYZ Accel.
- In the Read XYZ Accel state you will access the LIS331 for its last X, Y, Z acceleration readings via I2C. There are two possible transitions from this state: Complete or Disconnect.
    - On Complete, the state will change to Process/Display.
    - On Disconnected, the state will change to Sensor Disconnect.
- In the Process/Display state you will use the current acceleration readings to display:
    - Last X, Y, Z readings and the state entry count
    - Average X, Y, Z readings
    - Low X, Y, Z readings
    - High X, Y, Z readings

  There is one possible transition from this state: Complete
    - On Complete, the state will change to Wait/Poll Slider.
- In the Wait/Poll Slider state, you will count the number of times you have entered this state and you will wait for a 3 second timeout. You MUST use the SysTick timer to create this specific 3 second delay. While waiting for the timer interrupt, you will poll the KL25Z Touch Slider. If you receive a left side touch event, you will go to the Left Slider transition. If you receive a right side touch event, you will go to the Right Slider transition. If the timer times out for state entries 1 through 5, you will go to the Timeout 1-5 transition, if the timer times out for state entry 6, clear the visit counter and go to the Timeout 6 transition. There are four transitions from this state: Left Slider, Right Slider, Timeout 1-5, and Timeout 6.
    - On Left Slider or Timeout 6, the state will change to Read XYZ Accel in the OTHER state machine
    - On Timeout 1-5, the state will change to Read XYZ Accel in this state machine
    - On Right Slider, the program will end
- The Disconnected state indicates the LIS331 is not responding for some reason. This state is reached via a Disconnect event, which should occur if the sensor is physically disconnected. There is one possible transition from this state, which is that the program will end.

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, <u>avoid duplicate code wherever possible</u>.
- At a minimum, the POST test must verify operation of the connected LIS331. 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 LIS331 sensor to your KL25Z. It is critical that you have a solid electrical connection for the sensor lines, but you'll also want to be able to disconnect the sensor to allow testing of the disconnected states in the program.
- You will need to create a LIS331.h file that contains macros for accessing registers to control and monitor the LIS331 board.
- You must control/monitor the LIS331 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. I2C communications basics have been reviewed in class. You will need to determine how to use I2C correctly for communications with the sensor.
- You must use BOTH interrupt-based I2C communication AND a polling approach for I2C communication with the LIS331. One method should be used for one state machine, one method should be used for the other. (I highly recommend you initially write a small test program to verify these I2C board communications before going into the full project structure.)
- You should use modular design for your code – I2C communication functions in one module, state machine code in another, etc.

Support

- Information on the LIS331 Sparkfun board, including the details of I2C communications can be found at: https://www.sparkfun.com/products/10345
- 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 (look for a code example 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.
- An example of polled/direct I2C communications is in the Dean book GitHub repo at: https://github.com/alexander-g-dean/ESF/tree/master/Code/Chapter_8/I2C-Demo
- 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 that can report to the logger. 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 POST Test state, green when in the normal operating states, and red if the LIS331 is disconnected or otherwise fails.

Slider Polling

- Reuse your LED control function from Project 2.
- Use slider value ranges that indicate as best you can a left side or right side touch event when actively polling. These ranges should be captured in #define constants.

Part 2 (20 points) – Capture Oscilloscope trace of I2C traffic between the KL25Z and the LIS331

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 a PDF submission.

If you have access to a logic analyzer, you could use that as an alternate for capturing the transaction waveforms. Please see the SAs for any assistance needed with the scopes or analyzers in the lab.

Extra Credit – SPI Communications to LIS331 – 10 points

The LIS331 can be communicated with using SPI as well as I2C. Create a third state machine structured as the two state machines above that uses SPI-based communications to talk to the LIS331. All three state machines must be demonstrated as operational in your demonstration to receive this extra credit. The SPI communications may be polling or interrupt based. See the lectures for sources of SPI examples.

Project Submission

The project is due on **Tuesday 3/31** (Tuesday after break) prior to class and will be submitted on Canvas. The project will also be demonstrated interactively with the class staff. Appointment slots will be posted for demonstrations. 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
- Description: description of the repo contents
- Observations: A description of any issues or difficulties you encountered on the project and how they were addressed, or any assumptions you made when met with incomplete specifications
- 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 scope captures
- Please include a Git tag on your final submission to allow the SAs to be clear about what was submitted for grading

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).

Grading

Points will be awarded as follows:

- 35 for the correctness of demonstrated code (execution of both cross-compiled code versions) **– code will be demonstrated with class staff after the due date**
- 35 for the construction of the code (including following style guide elements, required elements, and the quality of solution)
- 10 points for the README
- 20 points for the captured and annotated I2C scope images
- 10 points extra credit for an SPI communication protocol based third state machine.

Assignments will be accepted late for one week. There is no late penalty within 4 hours of the due date/time. In the next 24 hours, the penalty for a late submission is 5%. After that the late penalty increases to 15% of the grade. After the one week point, assignments will not be accepted. The team may submit the assignment using a late pass from each of the team members. Only one late pass can be submitted per project, and it will extend the due date/time 24 hours.