PES Project 5 – UART Communications with PC - 100 Points (+10 bonus)

The fifth PES project will have you connect to the KL25Z USB port to send and receive data from a PC-based terminal program. This setup will allow you to exercise more advanced firmware topics, including UART communication device drivers, circular buffers, and more on interrupts and timers. You will optionally capture oscilloscope or logic analyzer output. You will also develop an FMEA document for the project. A bonus functionality will be to extend the circular buffers when they overflow.

This project is targeted to run on the KL25Z only – the program should (as in project 4) be capable of running in three modes:

1) In debug mode, with detailed debug messages.
2) In normal mode, with normal status messages.
3) In test mode, with detailed test messages.

You should also use a #DEFINE variable to select from blocking or non-blocking UART device driver use (see below), and a #DEFINE variable to select echo or application mode (see below).

Part 1. (90 points) UART-based Communications with PC Terminal Program

Circular Buffer Functions

Develop a set of circular buffer interface functions for characters. These functions should be capable of working with multiple circular buffers by referencing a provided pointer to a circular buffer structure. You will likely have individual circular buffers active for the transmit queue and the receive queue.

- Define structure and dynamic memory allocation method for circular buffers
  - Structure should include buffer pointer, head, tail, length, count
  - Buffer functions should update these values as required
  - You may use any memory allocation method you wish. You could use malloc() to allocate memory from the heap; you may also pre-allocate an array at initialization and use that block to provide sections of the array as buffer memory
  - Note that the maximum size of the buffer should equal the maximum number of elements – we will NOT use an empty byte as a wrap-around flag
- Define an enum for circular buffer errors (buffer functions should provide this as a return value)
- Function to add a new item to a buffer (should error if buffer is full)
- Function to remove oldest item from a buffer (should error if buffer is empty)
- Function to check is buffer full
- Function to check is buffer empty
- Function to verify buffer is initialized
- Function to verify the buffer pointer is valid
- Function to initialize a buffer
- Function to destroy a buffer

UART Device Driver 1 – Blocking/Polled

Develop a set of functions to implement polling of UART hardware. These functions should only use the predefined definitions in MKL25Z4.h (**do not use** higher-level functions provided by fsl*.h or other SDK includes). Note, for all UART transactions use the KL25Z UART0 to have access to the full UART function set.

- Function for UART hardware initialization including baud rate and serial message format (parity, start/stop bits)
- Function to check whether the transmitter is available to accept a new character for transmission
- Function to transmit a character assuming transmitter is available.
- Function using these two functions to wait (block) for the transmitter to be available and then once available transmit a character and return
- Function to check whether the receiver has a new character to receive
- Function to receive the character assuming receiver has data
- Function using these two functions to wait (block) for the receiver to receive a new character and then return that character
- An echo function that uses the above functions to echo received characters one at a time back to the PC-based sender

UART Device Driver 2 – Non-blocking/Interrupt

Develop a set of functions for using interrupts to detect the completion of a send or receive operation. You are encouraged to share/reuse code from the Blocking/Polled version above. These functions should only use the predefined definitions in MKL25Z4.h (**do not use** higher-level functions provided by fsl*.h or other SDK includes).

- Function for UART hardware initialization as above with addition of interrupt configuration and enable (configure the KL25Z so the UART generates an interrupt when ready to transmit a character, when a character has been received, or when there is an error)
- An interrupt service routine (ISR) function to handle the interrupts. This function should handle transmit and receive interrupts as well as errors.
- Function to check whether the transmitter is available to accept a new character for transmission
- Function to transmit a character assuming transmitter is available
- Function called by the ISR to supply the next character for transmit (i.e. on the IRQ, check the transmit interrupt is enabled and the transmit buffer is empty before placing the next character into the data register)
- Function called by the ISR for receiving a character (i.e. on the IRQ, verify a character was received and read it from the data register)
- An echo function that uses the above functions to echo received characters one at a time back to the PC-based sender

Remember, you should limit processing in the interrupt handler. The interrupt routine should be as short as possible. Each interrupt should clear associated flags when completed but only if they were set. Any significant processing should be done outside the interrupt context.

Echo Mode and Application Mode

If the #DEFINE for echo operation is in place, use the echo function in the device driver to send any received characters back to the PC terminal program.

If the #DEFINE for application operation is in place, run the following application:

Create an application that maintains a count of the unique characters that have been received by the UART device driver since the beginning of program execution. The report will show all characters received along with counts. Whenever the UART is able to transmit, send a formatted report to the PC terminal program. This report, and its associated data structure, should continue to accumulate characters and character counts for as long as a single execution is running.

For example, if the PC sent the characters in the word "Character", the resulting report would look like (with data in ASCII value order):

C – 1; a – 2; c – 1; e – 1; h – 1; r – 2; s – 1; t – 1

Note that this report may be interrupted by incoming data from the PC, but once transmit is clear, the report should restart and attempt to send the report data to the PC again.

It is up to you to decide what triggers the printing of the report; perhaps, when some fixed number of characters has been processed, or perhaps if a particular character is processed. Your approach to this functionality needs to be documented appropriately in your software via comments and in the README observations or execution report.

Unit Testing

Using uCUnit, create a test suite to test the circular buffer code thoroughly. Examples:

- Create a buffer
- Test data access
- Verify wrap remove (test that your buffer tail point can wrap around the edge boundary when a remove occurs at the boundary)
- Verify wrap add (test that your buffer can wrap around the edge boundary and add to the front)
- Overfill (test that your buffer fails when too many items are added)
- Over empty (test that your buffer fails to remove an item when empty)
- Destroy the buffer
- Etc.

This test suite should be run anytime the program is executed in test mode.

Identify Critical Sections (operations that should not be interrupted)

Define two macro functions to disable and enable interrupts in any elements of the code you believe should be atomic, that is, should run without being interrupted.

```
#define START_CRITICAL()
#define END_CRITICAL()
```

Logging – add timestamps

Use the logging code implemented in Project 4 with the following extension:

- Modify enums as needed to track new modules/functions
- Add a timestamp to all logged messages
    - Timestamps will be character strings in the form: "HH:MM:SS.n" where H = Hours, M = Minutes, S = Seconds, n = tenths of a second
    - The timestamp will be based on time elapsed since program start
- Use the SysTick timer set to 10 Hz to provide a .1 second counter as a time reference; this timer should be enabled at the start of any program execution
- Once your UART drivers are working, use the UART drivers for logging instead of API functions. (This can be a good method to debug and verify UART driver 1.)

LEDs

- You may reuse your LED control function from any earlier projects
- The LED should be set as follows
    - initialize to blue
    - set to blue when the program moves to receiving or waiting to receive
    - set to green when the program moves to transmitting or waiting to transmit
    - set to red if there is any error condition detected

General

- For this project, you will develop using the MCUXpresso IDE environment. 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.
- You should use modular design for your code – UART drivers in one module, application code in another, etc.

Extra Credit (10 points) – Extending Circular Buffers

- Whether you're using circular buffers to hold the report out data counts or transmit and receive data, you can have situations where the buffer overflows. In the code above, we simply throw an error when this occurs.
- For 10 points extra credit
    - Create function(s) to add (or reallocate) memory to resize a circular buffer
    - Use this function in a situation (such as the report storage) when an overfill error occurs
    - Include a test of this function in the uCUnit test case suite

[Optional] Capture Oscilloscope trace of UART traffic between KL25Z and the PC

If you have access to an oscilloscope or a logic analyzer, capture a character transaction while in Echo mode showing the receive and transmit operations between the PC and the KL25Z.

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, 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.

Part 2 (10 points) – FMEA (Failure Modes and Effects Analysis) document for the project

Create an Excel (or other) spreadsheet with the following columns:

- Potential Failure Mode (how could the failure occur)
- Potential Failure Effect (what is the impact of the failure)
- Severity (1-10) (how severe is the issue, 10=most severe)
- Potential Cause (what could make this happen)
- Occurrence (1-10) (how frequently would this happen, 10=most frequent)
- Controls (What is in place to prevent either the cause or the failure)
- Detection (1-10) (how hard is this error to detect, 10 = hardest)
- Risk Priority Number = Severity * Occurrence * Detection

Pick a minimum of six error modes you can think of in the interactions between the PC and the KL25Z. Fill out the columns with your observations on the error mode and determine the Risk Priority Number for the risk. Sort the six error modes by the RPN. Include the spreadsheet in your project repo submission as a PDF.

Project Submission

The project is due on **Tuesday 4/14** prior to class and will be submitted on Canvas. The project must be demonstrated to class staff (either in a recorded video or by web conferences). We will be setting up appointment slots to allow a 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 5 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 UART transaction scope captures as discussed above

- The repo should also contain a PDF of your FMEA spreadsheet
- Please include a Git tag called Final 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:

- 40 for the correctness of demonstrated code (execution of the three build targets – test, debug, normal, for blocking/non-blocking UART communication, and for Echo/Application modes) **– code will be demonstrated with SAs on or after 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)
- 10 points for the README
- 10 points for the FMEA DF
- 10 points possible extra credit for development, use, and test of resizing existing circular buffers.

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.