Lab 8

RTOS Example

Objectives:

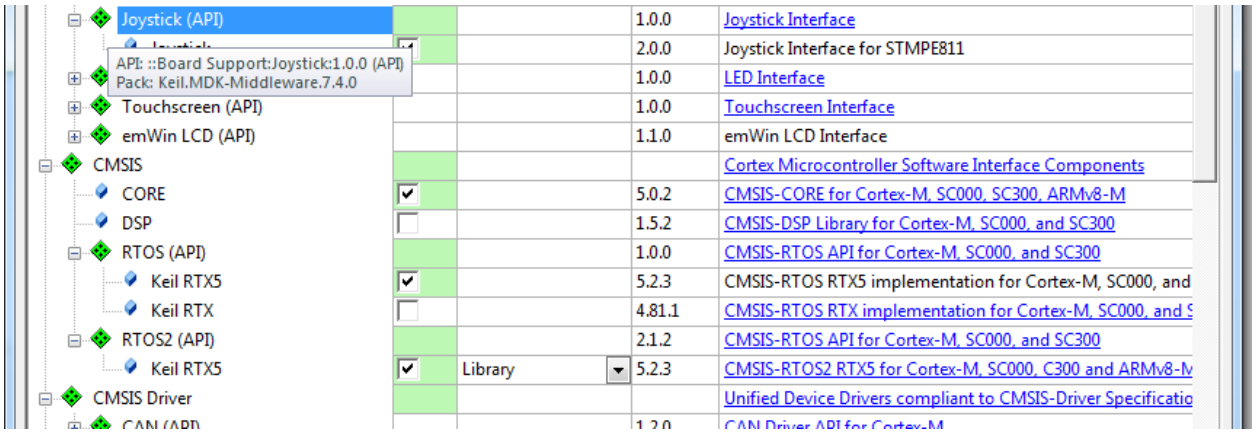
- Implement a basic FSM using an embedded RTOS.

Files Needed:

- Lab 8 spring 2019.zip.

Assignment:

1. Download the ZIP file above, unzip it, and open it in the Keil uVision5 development environment.
   Since an embedded system implemented with an RTOS often separates the project into one file per thread, the example for Lab 8 is presented as a complete project instead of a single code file. The process for creating it was essentially the same as Lab 7, with only a few small changes needed in the *Manage Run-Time Environment* window.



A portion of the window is shown above. The main checkbox is *CMSIS->RTOS2(API)->Kiel RTX5.* Once this box is checked when the project is being created, many unresolved items show in yellow, and clicking *Resolve* at the lower left adds most of the other checkboxes you see to get the Kiel RTOS to work.

The two other boxes that were explicitly selected were the GLCD, which was used in the last lab, and the joystick. The current joystick functions work with the RTOS, so the separate files for the joystick functions and the I2C from the last lab aren't needed. However, the serial port is still much easier to use with the separate files, so this project also uses them.

2. Build the program, enter debug mode, and run it to see what it does. You should see capital letters cycing in the upper left corner. Pressing the joystick up or down should cause a U or D respectively being writted to the right of the cycling letters. If a letter (or space) is received by the serial port at the standard 115200 baud, no parity, one stop bit, it is shown below the cycling letters.

   The purpose of this program is to show how the RTOS creates different threads, semaphores, and mutexes and makes basic function calls that use them. Spend time looking through the files to understand how the different pieces work together/independently. When in doubt, ask how something works. Extra time has been given to this lab to facilitate more questions.

3. Alter the program to implement the finite state machine below using proper RTOS concepts. The upshift input (*up*) to the FSM must come from receiving a 'U' or 'u' on the serial port, or detecting that the joystick was pressed downward relative to the screen. The downshift input (*dn*) to the FSM must come from receiving a 'D' or 'd' on the serial port, or detecting that the joystick was pressed upward relative to the screen. The output *gear* must write the corresponding character to the middle of the graphic LCD screen.

   In addition to the finite state machine above, the system must also send the following ASCII text messages to the PC using the serial port. First, it must send the following message only once before any other messages.

   Time Gear

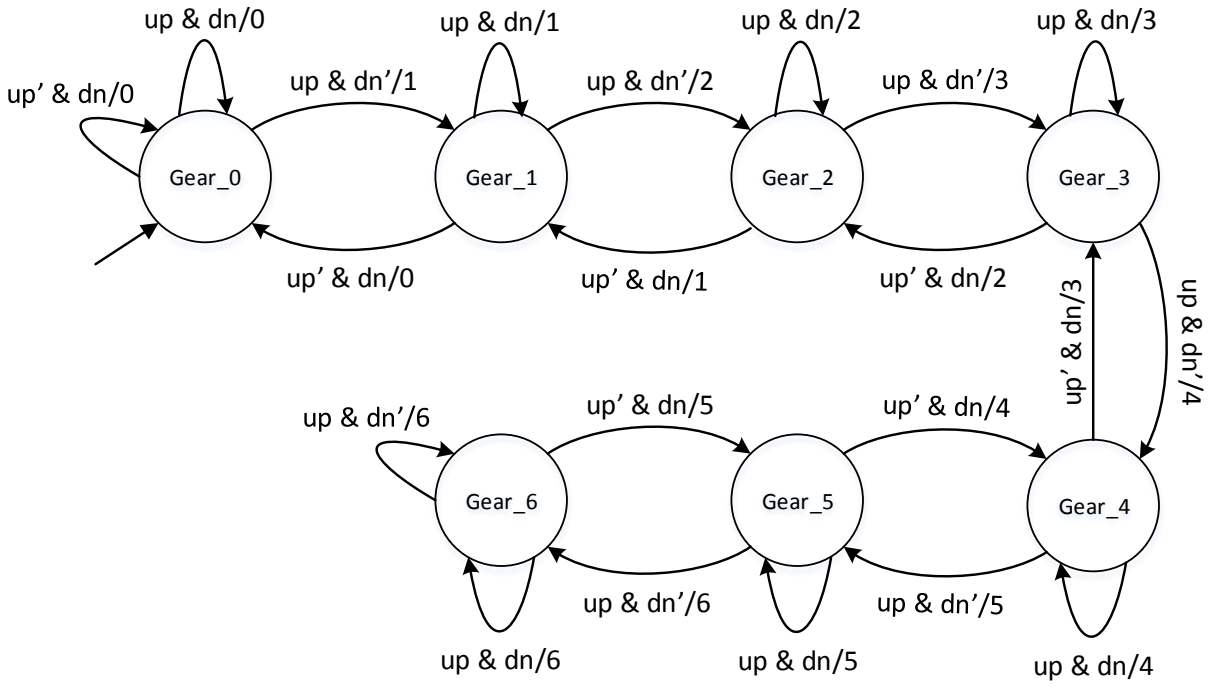
   After that message, it must send the following message according to the description below.

   sss.hh g

   In this message, *sss.hh* is the time since the system began executing in which *sss* is the three-digit number of seconds and *hh* is the two-digit number of hundredths of a second, and *g* is the current gear. This message must be transmitted every *N* hundredths of a second or whenever the gear is changed. Note that this is really an extended finite state machine that is being implemented concurrently with the first FSM. Assume that the time *sss.hh* does not overflow (you are free to address this if you want to.) Also, make N a single, easily modifiable constant in your project that determines how often the message

periodically transmits, and changing the constant should require recompiling. A good starting point is 50, and you can call it something more intelligent than N.

Inputs: up, dn : pure
Output: gear : character



Deliverables:

Upload an electronic copy of the deliverables to Canvas per the lab submission guidelines. Please note: **ALL** deliverables must be present in the single PDF report.

- Zip file of the project. To minimize the file size, please select Project->Clean Targets before sipping up the file.
- PDF report of all source code.
- Properly comment the code.

Scoring:

- 10 points – Compliance with Submission Guidelines
- 40 points – Correct functionality
- 30 points – Proper use of RTOS concepts (i.e. semaphores, mutexes)
- 10 points – Commenting