Final Course Project

Goals

The main goal of the final project is to let students practice the software design techniques learned in this course, which include:

1. Object-oriented design with C++
2. Model-driven design with statecharts and QP
3. Optimization and debugging techniques for embedded systems

The focus is on designing and representing high-level logic and behaviors, and translating them to executable code in a consistent manner.

Tasks

1. To complete the project, students will design and implement an embedded system of his/her choice using the course-provide baseline project which will be available in the *Finale Project* folder in Canvas.

   The compressed baseline project is named ***platform-stm32f401-nucleo_finalproject.tgz***

   The baseline project supports basic interfaces to certain course-provided hardware, including serial console, LED[*1] and button, WiFi module (via UART), accelorometer/gyroscope sensors (via I2C) and LCD (via SPI). These interfaces are introduced during the course. Note that the touchscreen and audio module are not currently supported.

   Note *1:

   The USER LED conflicts with the LCD display as they share the same GPIO pin. It is currently disabled in System.cpp. To enable it, uncomment the following code in System.cpp:

   ```
   // USER LED pin (PA.5) is shared with TFP display SPI clock pin.
   // It must not be enabled when the TFP is used (e.g. in LevelMeter).
   //evt = new UserLedStartReq(USER_LED, SYSTEM, GEN_SEQ());
   //me->GetHsm().SaveOutSeq(*evt);
   //Fw::Post(evt);
   ```

   In addition you would need to disable the **LevelMeter** active object in System.cpp since it uses the LCD. See the next point for details.

2. As an example, the baseline project implements a basic yet functioning level meter that acquires accelerometer data and displays the pitch (P) and roll (R) angles (in degree) on the LCD display. It demonstrates two algorithms in calculating pitch and roll from x, y and z accelerometer data.

   Since **LevelMeter** uses the acceleratometer (Sensor/Iks01a1) and LCD (Disp/Ili9341) hardware resources, it will prohibit you from using the same resources for your own projects. You can disable LevelMeter by uncommenting the follwing code in System.cpp:

```
// UW 2019 Uncomment this to bypass LEVEL_METER.
/*
evt = new Evt(DONE, GET_HSMN());
me->PostSync(evt);
return Q_HANDLED();
*/
```

3. At a minimum the serial console should be used to output system and state logs. In addition, it can be used to provide simulated input and output in a fasion similar to the washing machine and traffic light examples. You are encouraged to use other hardware supported by the baseline project for input and output, but it is not required.
4. You can use the sample active objects in **src/Template/SimpleAct** *or* **src/Template/CompositeAct** as starting points.

   SimpleAct is a simple active object *without* orthogonal regions while **CompositeAct** is a composite active object containing 4 orthogonal regions. You can choose *one* based on on the need/complexity of your project.

   A test command "`simp test`" is added to the console for SimpleAct. Similarly a test command "`comp test`" is added for CompositeAct. They provide hints for you to add your own commands.
5. For project ideas, you may check out the case studies (Chapters 19-23) in the course reference book:

   Gomaa, Hassan. Real-Time Software Design for Embedded Systems. 1st Edition. Cambridge University Press. 2016. (Online version available at UW library.)

   or use your imagination.

Submission

The due date is 6/15 Saturday 11:59pm. Please submit:

1. A video recording file (same format as in 2nd quarter) of about 5-10 minutes. This video file will be played back during the last class on 6/17.
2. A compressed file (tgz or zip) of the project source code. To reduce upload sizes, please "clean project" before compressing.
3. A capture file of the serial console output showing the log of a typical use case or scenario.