Assignment 3

GPIO_OUT (USER_LED) HSM

1 Goal

The goal of this assignment is to implement an LED pattern generator using HSM (Hierarchical State Machine). This approach allows the source code (implementation) to be directly traceable to the statechart (design) which can in turn be traced back to the requirements (specifications).

See **Session 2.1.** *Model-Based Software Development of the following publication* and **Figure 1** of:

Automatic code generation for instrument flight software. Kiri L. Wagstaff, Edward Benowitz, Dj Byrne, Ken Peters, Garth Watney. Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2007.

(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.548&rep=rep1&type=pdf)

2 Setup

1. Download the compressed project file (platform-stm32l475-disco-assignment3.zip).
2. Backup your existing project folder. Decompress the downloaded project file to a local folder.
3. Launch STM32CubeIDE. Hit F5 to refresh the project content.

   Or you can right-click on the project in Project Explorer and click "Refresh".
4. Clean and rebuild the project. Download it to the board and make sure it runs.

   In the UART console, type the command "`gpio on 0`" and verify that the following text is output on the console and the USER LED does NOT blink:

   ```
   pattern = 0, repeat = 1
   ```
5. The statechart was created with the free tools named **UMLet**. You can download the latest version (14.3) with this link:

   https://www.umlet.com/download/umlet_14_3/umlet-standalone-14.3.0.zip

   (or on this page: http://umlet.com/changes.htm. Be careful NOT to click on those advertisement ‘Download’ links!)

   The stand-alone version is more stable (vs plugins). It runs on Java, so you would need to install Java on your PC. Decompress the downloaded file and double click *umlet.jar* to launch it.

3 Tasks

1. Imagine the following requirements are handed to you about an LED pattern feature:

   (a) Upon a pattern request event, the system shall show the specified LED pattern.

   (b) If the pattern request has the "isRepeat" flag enabled, the system shall show the specified LED pattern repeatedly.

   (c) Upon an *off* request event, the system shall turn off the LED.

2. Think about the above requirements. Could you find any gaps in the specifications?

3. Even for a single LED, specifying its behaviors is not as simple as it appears. Now review the provided statechart in **Src/app/GpioOutAct/GpioOut/GpioOut.uxf**. Does it tell us more about how the system should behave?

4. Replace the two files **GpioPattern.h** and **GpioPattern.cpp** under **src/GpioOutAct/GpioOut/** with the ones you modified in the previous assignment.

5. Implement the statechart in **Src/app/GpioOutAct/GpioOut/GpioOut.cpp**.

   All required changes are marked with "`// Assignment 3`"

   The header file already contains declarations for all the state functions, internal events, timers, member variables required by the statechart design.

6. Test your code with the console command:

   gpio on <pattern index from 0 to 1 > <0 for non-repeating; 1 for repeating>

   (e.g. "gpio on 1 1" to show pattern 1 repeatedly.)

   gpio off

   See how you can start a new pattern or stop it at any time. For example you can stop a pattern in the middle of a cycle. You can change the pattern before a cycle has finished. The system is reactive and responsive *by design*, which is the essence of real-time systems.