ESS: Lab 4

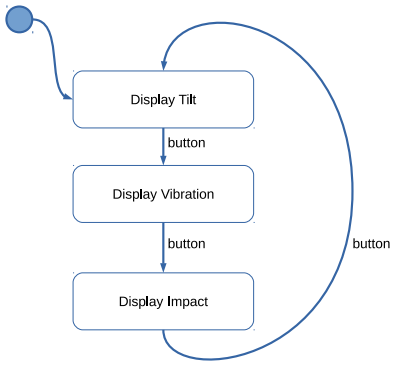
Reactive Operation

Task 1:

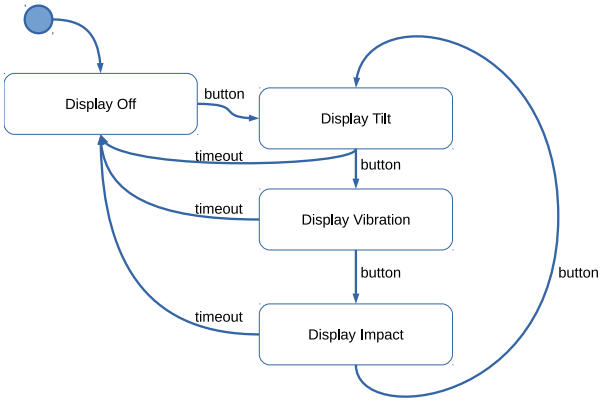
Finite State Machine

A finite state machine is a good fit for embedded systems, as it combines states and events, resulting in transitions to new states if certain conditions/actions occur.

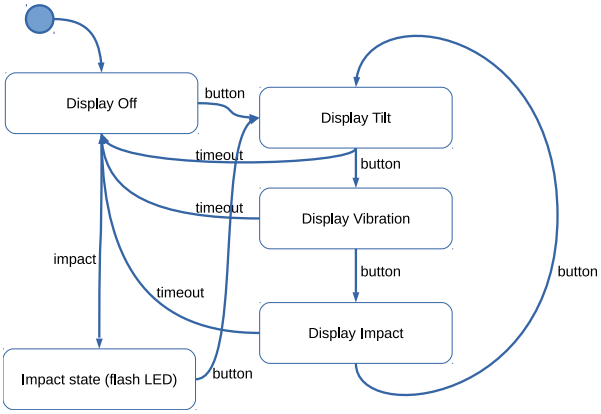
(a) Implement a state machine to cycle through the three displays of accelerometer readings (tilt, vibration, impact) each time the button is pressed.



(b) The LED display is very power consuming however, so it would be better to keep it off, and only turn it on when a button is pressed. To protect against inadvertent operation, after not receiving any button press in any display state within 15 secs, it should revert to off mode, as the FSM below shows. Implement a system that accomplishes this.



(c) As part of the system requirements, it was desirable to indicate visually if a pallet had been damaged (e.g. subject to excessive shock/impact). If the impact has been detected, it should flash the red LED briefly every few seconds to draw attention to this fact. The FSM is shown below. Implement a system which accomplishes this.



(d) There is a lot of duplication in this finite state machine. Refactor into a hierarchical/nested state machine and compare implementation clarity.

Task 2:

⋆Real Time Operating System

STM32CubeIDE supplies a real time operating system called FreeRTOS. A sample project of two LEDs blinking will be provided. The RTOS sample project is located at "RTOS_Sample" folder. Same to the "ess_skeleton", double click the ".project" file to import the project into STM32CubeIDE. Use this to implement a simple program that cycles through the display (tilt, impact, vibration) every time the button is pressed. The button itself should run in its own task. As an example of a simple two task system, note that we use osDelay(ms) here so that the scheduler can switch to another thread while the current thread waiting:

```
#include "cmsis_os.h"

/* Definitions for blink01 */
osThreadId_t blink01Handle;
const osThreadAttr_t blink01_attributes = {
  .name = "blink01",
  .priority = (osPriority_t) osPriorityNormal,
  .stack_size = 128 * 4
};
/* Definitions for blink02 */
osThreadId_t blink02Handle;
const osThreadAttr_t blink02_attributes = {
  .name = "blink02",
  .priority = (osPriority_t) osPriorityBelowNormal,
  .stack_size = 128 * 4
};
void StartBlink01(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
    osDelay(500);
  }
}
void StartBlink02(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
    osDelay(600);
  }
}
```

```
void main (void)  {
    /* Init scheduler */
    osKernelInitialize();
    /* Create the thread(s) */
    /* creation of blink01 */
    blink01Handle = osThreadNew(StartBlink01, NULL, &blink01_attributes);
    /* creation of blink02 */
    blink02Handle = osThreadNew(StartBlink02, NULL, &blink02_attributes);
    /* Start scheduler */
    osKernelStart();
}
```

Note that you have to start with a RTOS sample, and copy the files and codes you need from your previous project into the RTOS project. Because the RTOS feature is not enabled in the previous project skeleton, if you try to migrate RTOS code into your previous project, it would probably fail.