Lab 6 State Machine and User Input

10/22/21

10/28/21

Jeremiah Webb & Troy Neubauer

Introduction:

Understanding and tracking a program's state allows a programmer to debug and understand the exact position of a program. Constructing a basic state machine gives a proof of concept of what a state machine can do when a user inputs specific data, in this case, joystick inputs. Furthermore, settings specific states when the program fails and thus making it visual to the programmer can help when debugging and benchmarking.

Code Snippet:

Task 2 Macros

```
#define LED_ON GPIO_PIN_RESET

#define LED_OFF GPIO_PIN_SET

#define LD_R_ON HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,
LED_ON)

#define LD_G_ON HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_G_Pin,
LED_ON)

#define LD_R_OFF HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,
LED_OFF)

#define LD_G_OFF HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin,
LED_OFF)

#define LD_R_TOGGLE HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin)

#define LD_G_TOGGLE HAL_GPIO_TogglePin(LD_G_GPIO_Port, LD_G_Pin)

#define JOY_C_IS_PRESSED HAL_GPIO_ReadPin(JOY_C_GPIO_Port,
JOY_C_Pin)

#define JOY_L_IS_PRESSED HAL_GPIO_ReadPin(JOY_L_GPIO_Port,
JOY_L_Pin)

#define JOY_R_IS_PRESSED HAL_GPIO_ReadPin(JOY_R_GPIO_Port,
JOY_R_Pin)
```

Task 3 State Machine

```
void USR_Task_RunLoop(void) {
switch (currentState) {
//State0
case State0:
if (stateChanged) {
     currentState = State1;
     stateChanged = true;
}
break;
//State 1
case State1:
     if (stateChanged) {
          LD_R_G_flash();
          stateChanged = false;
     }
     if (JOY_C_IS_PRESSED) {
          currentState = State2;
          stateChanged = true;
          HAL_Delay(bDelay);
          break;
     }
     LD_R_ON;
     LD_G_OFF;
     break;
//State 2
case State2:
```

```
     if (stateChanged) {
          LD_R_G_flash();
          stateChanged = false;
          LD_G_ON;
          LD_R_ON;
     }
     if (JOY_L_IS_PRESSED) {
     // provide your code to handle the left Joy key input.
          currentState = State1;
          stateChanged = true;
          HAL_Delay(bDelay);

          break;
     }
     if (JOY_R_IS_PRESSED) {
     // provide your code to handle the right Joy key input.
          currentState = State3;
          stateChanged = true;
          HAL_Delay(bDelay);
          break;
     }
     // provide your code for LED control
     LD_G_ON;
     LD_R_ON;
     break;
//State 3
case State3:
     if(stateChanged) {
```

```
            LD_R_G_flash();
            stateChanged = false;
            // provide your code
            // note that you need also provide your code so that
the
            // LEDs will blink alternately.
            LD_R_ON;
            LD_R_TOGGLE;
            LD_G_TOGGLE;
            HAL_Delay(bDelay);
      }
      if(JOY_L_IS_PRESSED) {
      // provide your code to handle the left Joy key input.
            currentState = State1;
            stateChanged = true;
            HAL_Delay(bDelay);
            break;
      }
            LD_R_TOGGLE;
            LD_G_TOGGLE;
            HAL_Delay(bDelay);
            break;
            default:
            LD_R_OFF;
            LD_G_OFF;
            break;
      }
}
```

Discussion:

Learning what macros are and how they function was extremely helpful. Macros significantly improved not only the legibility of the program but also the functionality. Allowing the programmers to not only be able to see exactly what the function does but easy program the HAL functions into the main program is extremely helpful. One thing to note is looking into previous labs helped with the implementation of such macros into actual functions that are physically shown, such as pressing the center of the joystick and the two LEDs turning on.

Result:

The lab overall was enjoyable, being able to see how state machines can be used to debug and implement programmer code is insightful into real-world applications. Understanding that state functions in addition to other debugging tools can be the ultimate “toolbox” for debugging a program and can be used in nearly any other code that contains the user’s inputs. Understandably, these state machines can be used to indicate a program’s ability to run code efficiently and can give an accurate time for benchmarking purposes.