Project application note

Matteo Corain S256654
Computer Architectures – A.Y. 2018-19

1 State management

The management of the state of the semaphore is done via the `state` library (`state.c` and `state.h`). These include the following items:

- Definition of type `state_t`, which is an enumeration type which can assume four values: STATE_RG (red semaphore for cars, green for pedestrians), STATE_RFG (red semaphore for cars, flashing green for pedestrians), STATE_GR (green semaphore for cars, red for pedestrians) and STATE_YR (yellow semaphore for cars, red for pedestrians); a variable of type `state_t` is also defined to keep trace of the current status, called `Current_Status`;
- Definition of types `blind_t` and `maint_t`, which are essentially Boolean enumeration types (`NO_BLIND` and `BLIND` states for `blind_t`, `NO_MAINT` and `MAINT` states for `maint_t`); two variables of types `blind_t` (`Blind_Status`) and `maint_t` (`Maint_Status`) are also defined to keep trace of the current status;
- Definition of state functions to be run at the beginning of each state (`Run_State0`, `Run_State1`, `Run_State2`, `Run_State3`, `Run_Maint`, `Run_NoMaint`).

2 LEDs

For the sake of the application, LEDs are used in the way described in the project description. At the beginning of the program, before entering the first state, LEDs are initialized to all zeros via the `LED_Init()` function; then, they are set in the state functions in the following way:

- STATE_RG sets the LEDs to CAR_RED | PED_GREEN (0x60);
- STATE_RFG sets the LEDs to CAR_RED (0x20), switching the PED_GREEN LED alternatively on and off via the blinking timer;
- STATE_GR sets the LEDs to CAR_GREEN | PED_RED (0x88);
- STATE_YR sets the LEDs to CAR_YELLOW | PED_RED (0x90).

3 Buttons and RIT

Buttons are configured in the way described by the project description. They are initialized via the `Button_Init()` function before entering the first state and set in edge-sensitive interrupt mode; all the buttons are debounced via the usage of the RIT, which is set to match on 50 milliseconds (`TIME_50MS`, equivalent to 0x1312D0 ticks at 25 MHz). When a button is pressed, the following actions are taken:

- If the pressed button is `INT0`, the blind state is set and the speaker is started;
- The button is configured in GPIO mode and the RIT is enabled for debouncing;
- If, after 50 milliseconds debouncing, the button is still pressed, the pedestrian request is handled (via the `Handle_Req()` function);
- When the button is released, the button is restored to interrupt mode (if the button is `INT0`, the speaker is also stopped); if no other buttons are pressed, the RIT is stopped.

The `Handle_Req()` function behaves differently based on the state the system is currently in:

- In STATE_RG, it resets the main timer;

- In STATE_RFG, it executes the state function of STATE_RG to return to the previous state;
- In STATE_GR, it starts the main timer;
- In STATE_YR, it does nothing.

4 Timers and joystick

All the four timers of the LPC1768 SoC are used for the purpose of the project, with different purposes:

- TIMER0 is used as the main timer, managing the transition among the different states; it has two match registers set, one to 15 seconds (TIME_15SEC, 0x165A0BC0 ticks at 25 MHz) and one to 5 seconds (TIME_5SEC, 0x7735940 ticks at 25 MHz), which are used respectively by STATE_RG and by all the others; when the interrupt handler is called, the state function relative to the next state is called (circular update of Current_State);
- TIMER1 is used as the blinking timer, managing the flashing of the LEDs and the start/stop of the speaker when needed; it has two match registers set, one to 1 second (FREQ_1HZ, 0x17D7840 ticks at 25 MHz) and one to 0.5 seconds (FREQ_2HZ, 0xBEBC20 ticks at 25 MHz), which are used respectively by STATE_RG (including maintenance mode) and by STATE_RFG; when the interrupt handler is called, the following actions are taken:
    - If the system is in maintenance mode, the LEDs and the speaker are alternatively turned on and off every second;
    - If the system is in STATE_RG and the blind flag is set, the speaker is alternatively turned on and off every second;
    - If the system is in STATE_RFG, the LEDs are alternatively turned on and off every half a second, eventually turning on and off accordingly also the speaker if the blind flag is set.
- TIMER2 is used as the play timer, managing the update frequency of the DAC output going to the speaker; it has a single match register set to output one of the 64 sine wave samples at a 440 hertz frequency (FREQ_440HZ, 0x377 ticks at 25 MHz); when the interrupt handler is called, the DAC is used to output the next sample (via the DAC_Play() function).
- TIMER3 is used as the maintenance timer, managing the polling of the joystick and the reading of the ADC in maintenance mode; it has a single match register set to 50 milliseconds (TIME_50MS, 0x1312D0 ticks at 25 MHz); when the interrupt handler is called, the value of the joystick is read (via the Joystick_Read() function), the maintenance mode is entered if the conditions are satisfied, the ADC value is read if in maintenance mode (via the ADC_Start() function) and the maintenance mode is exited if the conditions are satisfied.

5 ADC and DAC

The ADC module is used only in maintenance mode to set the volume of the speaker. Its value is read from the potentiometer in its interrupt handler (after starting the conversion in the maintenance timer) and 0xFFF minus the read value is copied to the DAC_Volume variable (requirements state that high voltages on the potentiometer mean low volumes). The value of the ADC is also read at the beginning of the program, to preemptively set the volume of the system when it is started.

The DAC module is used to output sine wave samples (64 samples were used) to the speaker. It is generally managed via the play timer, whose handler calls the DAC_Play() function, selecting the next sample and writing it to the appropriate register, after having it scaled via the DAC_Volume variable.