Project application note

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

1 State management

The management of the semaphore state is done via the `state` library. These include the following items:

- Definition of type `state_t`, which is an enumeration type which can assume four values: STATE_RG (red for cars, green for pedestrians), STATE_RFG (red for cars, flashing green for pedestrians), STATE_GR (green for cars, red for pedestrians) and STATE_YR (yellow for cars, red for pedestrians); a variable (`Current_Status`) of type `state_t` is also defined to keep trace of the 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_`StateX`); state functions for the four standard states are also grouped in an array of function pointers (Run_`State`) for easy reference (it is possible to call them via Run_`State[state]()`).

In particular, the six state functions perform actions described in the following tables.

`Run_State0`	
State	`Current_State` set to STATE_RG.
LEDs	Set to CAR_RED | PED_GREEN (0x60).
Timers	• The main timer is reset, set to match every 15 seconds (MR0) and started; • The blinking timer is reset, set to match every second (MR0) and started; • The maintenance timer is started.
Other actions	None.

`Run_State1`	
State	`Current_State` set to STATE_RFG.
LEDs	Set to CAR_RED | PED_GREEN (0x60). PED_GREEN is made flashing via the blinking timer.
Timers	• The main timer is reset, set to match every 5 seconds (MR1) and started; • The blinking timer is reset, set to match every 0.5 seconds (MR1) and started; • The play timer is started if the blind flag is set; • The maintenance timer is reset.
Other actions	The DAC output is initialized (via DAC_`Play()`) prior to starting the play timer if the blind flag is set.

`Run_State2`	
State	• `Current_State` set to STATE_GR; • `Blind_State` set to NO_BLIND.
LEDs	Set to CAR_GREEN | PED_RED (0x88).
Timers	• The main timer is reset; • The blinking timer is reset; • The play timer is reset.
Other actions	The DAC output is cleared.

Run_State3	
State	Current_State set to STATE_YR.
LEDs	Set to CAR_YELLOW | PED_RED (0x90).
Timers	The main timer is reset and started.
Other actions	None.

Run_Maint	
State	Maint_State set to MAINT.
LEDs	Set to CAR_YELLOW | PED_RED (0x90).
Timers	• The main timer is reset; • The blinking timer is reset and started; • The play timer is started.
Other actions	• The DAC output is initialized (via DAC_Play()) prior to starting the play timer; • Buttons interrupts are disabled (no pedestrian call is handled); • The ADC is started.

Run_NoMaint	
State	Maint_State set to NO_MAINT.
LEDs	No change.
Timers	The play timer is reset if the blind state is not set.
Other actions	• The ADC is stopped; • The DAC output is cleared if the blind state is not set; • Buttons interrupts are re-enabled; • The initial state function is called.

2 Buttons and RIT

All the buttons are debounced via the usage of the RIT, set to match every 50 milliseconds (TIME_50MS, 0x1312D0 ticks at 25 MHz). The RIT interrupt handler implements a multi-buttons debouncing technique, checking the state of the buttons for which debouncing was requested (via the RIT_Debounce() function) against three down variables, ensuring that the pedestrian request is executed only once per button pressure. 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 the current key (via the RIT_Debounce() function);
- 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;
- In STATE_GR, it starts the main timer;
- In STATE_YR, it does nothing.

3 Timers

All the four timers of the LPC1768 SoC are used for the purpose of the project, with the functions described in the following tables.

TIMER0	
Purpose	Main timer, it manages the transition among the different states.
Configuration	• MR0 set to 15 seconds (`TIME_15SEC`, 0x165A0BC0 ticks at 25 MHz) and used in `STATE_RG`; • MR1 set to 5 seconds (`TIME_5SEC`, 0x7735940 ticks at 25 MHz) and used in the other states.
Interrupt handler	It calls the state function relative to the next state is called by circular updating the `Current_State` variable.

TIMER1	
Purpose	Blinking timer, it manages the flashing of the LEDs and the start/stop of the speaker.
Configuration	• MR0 set to 1 second (`FREQ_1HZ`, 0x17D7840 ticks at 25 MHz) and used in `STATE_RG` (including maintenance mode); • MR1 set to 0.5 seconds (`FREQ_2HZ`, 0xBEBC20 ticks at 25 MHz) and used in `STATE_RFG`.
Interrupt handler	• If the system is in maintenance mode, the LEDs and the speaker are alternatively turned on and off every second by checking the current LEDs value; • If the system is in `STATE_RG` and the blind flag is set, the speaker is alternatively turned on and off every second by checking whether the play timer is running and the blind request button is not pressed (i.e. in interrupt mode); • 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 and the blind request button is not pressed.

TIMER2	
Purpose	Play timer, it manages the update of the DAC output going to the speaker.
Configuration	MR0 set to output 64 sine wave samples at a 440 hertz frequency (`FREQ_440HZ`, 0x377 ticks at 25 MHz).
Interrupt handler	It outputs the next sine wave sample to the DAC (via the `DAC_Play()` function).

TIMER3	
Purpose	Maintenance timer, it manages the polling of the joystick in `STATE_RG` and the reading of the ADC in maintenance mode.
Configuration	MR0 set to 50 milliseconds (`TIME_50MS`, 0x1312D0 ticks at 25 MHz).
Interrupt handler	It polls the value of the joystick (via `Joystick_Read()`), enters the maintenance mode if the conditions are satisfied (via `Run_Maint()`), reads the ADC value if in maintenance mode (via `ADC_Start()`) and exits the maintenance mode if the conditions are satisfied (via `Run_NoMaint()`).

4 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 the ADC 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 when the system is started.

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