Extra Points Project Part 2 – Application Note

Alessandro Iucci s262806

I implemented the second part of the project starting from the first part of it but adding some more components. Indeed, while for the first project I was just controlling the behaviour of TIMER0 and TIMER1, the buttons KEY1 and KEY2 and the leds, this time I used also TIMER2 (which I had to activate myself), the RIT, the speaker, the joystick and the potentiometer through the ADC library and I modified the behaviour of INT0.

I will analyse the files once at a time explaining the most important changes in each of them

sample.c

- SystemInit(): I just provided power supply to TIMER2 (that is the one I will use to make the speaker work)
- LED_init(): leds set accordingly to the reset state;
- Button_init(): interrupts enabled for KEY1, KEY2 and INT0 and GPIO ports set to EINT mode
- Initialization and enabling of RIT (used for joystick polling, ADC conversion and button debouncing)
- ADC_start_conversion(): I added this function to set up the volume of the loudspeaker at startup taking the value from the potentiometer. This function, from now on, will only be called if we are in state 4 (maintenance) and the RIT elapses
- Timers initialization and enabling:
  - TIMER0: Of 5 seconds, used to switch from one state to another
  - TIMER1: Of 0,5 seconds used to make the lights blink and the sound go on and off for states 0 and 1 and maintenance
  - TIMER2: used to make the speaker sound using a sinewave at the frequency of 349Hz

IRQ_button.c

Each button is debounced using 3 different shared variables (down for INT0, down1 for KEY1 and down2 for KEY2). When the button is pushed, the corresponding interrupt is disabled, the corresponding GPIO pin is unset as EINT and the corresponding variable is set to 1. The previous configuration will be restored once the button will be released and the RIT elapses. The three buttons have the same behaviour given by a switch-case based on the state we are in (if we are in state 1, the button will make the timers restart and the count variable used to count for 15 seconds to reset, if we are in state 2, we restart the timers and go back to state 1, if we are in state 2, we reset and enable TIMER0 to proceed after 5 seconds to state 3). The only difference between INT0 and KEY1 & KEY2 is that when we push INT0, the TIMER2, that is used to make the loudspeaker sound, will be enabled in any case, while for KEY 1 and KEY2 it will be enabled just for case 1 and case 2. The reason is that when we push INT0, we have to make a sound until the button is released, while the enable_timer(2) of KEY1 and KEY2 are related to the fact that we are coming back to state 1, so we will have an alternate sound on 1 s ON and 1 s OFF and I always start with the 1 s ON. This is also why every time I come back to state 1, the i variable is set to 1. More details in the IRQ_timer section.

IRQ_timer.c

First of all, the MCR of all the timers is set to 3, so when they elapse there will be an interruption, they will reset but they will not stop. Timer 0 is used to switch from one state to another (counting 5s or three times 5s to make 15s). Every time we switch state we have to reset and disable timers and set leds accordingly to the state we are entering. For all the timers, every time I should disable the TIMER2, I check if the INT0 button is still pushed, because if it is I must not disable the sound.

TIMER1 is used to make the alternating sound in state 0 (1s ON & 1s OFF), to make the flashing green light and sound in state 1 (0,5s ON & 0,5s OFF) and to make the flashing lights and sound in maintenance mode, state 4 (1s ON & 1s OFF). In case 0, I use a variable i that counts from 0 to 4 and then comes back to 0. Depending on the module of the division by 4, the timer will do different things. When the module is 0 or 1, the sound must be on, when the module is 2 or 3 the sound must be off. For case 1 we just check if LED corresponding to pedestrian green is on or off. If it is on we switch it off and disable the sound, if it is off we switch it on and turn on the sound. For case 4, maintenance, we just do the same as in case 0, but we also make the lights flash 1s On and 1s OFF.

TIMER2 is the one chosen to make the loudspeaker sound using a sinewave. I used a 5 bit shift otherwise the sound would have been too low. To regulate the volume through the potentiometer I used the global variable AD_current scaled to 0xFFF, which is the maximum value of the ADC so that when the potentiometer is all turned one side volume is set to 0, when it is all turned the other one is set to maximum.

IRQ_RIT.c

When the RIT elapses, first of all we check if we had some button pushed using variables down, down1 and down2 and if they are still pushed. If they are not pushed anymore we re-enable interrupts and set GPIO ports to the EINT mode. We do the same actions for the 3 buttons, except for INT. Indeed, when it is released, we also disable_timer(2).

Then we check the joystick only if we are in state 0 or in state 4 (maintenance). If we are in state 0 and push left, we change in state 4. If we are in state 4 and push right, we change in state 0. Even if we keep pushed the joystick in one of the two sides, something will happen only the first time the RIT elapses because after that none of the conditions of the if-else block will be verified. So there is no risk of multiple executions of the joystick functions.

For what concerns the potentiometer, if we are in state 4 and the RIT elapses, we make the conversion start.

IRQ_adc.c

When the conversion ends and I have to handle the interrupt, I will read the converted value of the potentiometer and make a bitwise NOT (because if the potentiometer is all turned right, we have the maximum value, while if it is all turned left, we have the minimum; but we want the opposite behaviour) and make a bitwise AND in order to have just the last 12 bits.