Computer Architectures - Extrapoint2 Report

Giovanni Garifo

17/01/2019

1 State transition with timers

I've developed my solution by following the finite state machine that represents the semaphore.

The core part of the project, the one which controls the transition between the different states, is implemented by using $TIMER0$. The state update depends on the current state of the semaphore, this information is encoded in a global variable: in the interrupt handler of the timer I just check in which status the semaphore is, and take the corresponding actions to move to the next state.

The flashing led and sound functionality is implemented by using $TIMER1$, this timer is enabled by the interrupt handler of timer0 when needed. Based on the current status and on the value of a local static variable, the timer handler will choose the action to perform: put on or off the leds, start or end the sinusoid feeding to the loudspeaker.

Regarding the sinusoid feeding to the loudspeaker, it's handled by the $TIMER2$, which when enabled feeds a value of the sinusoid corresponding to the LA note every 98 microseconds, given that I have 23 samples of a standard sinusoid of amplitude 1 and frequency 440Hz. The excel sheet used to calculate the sinusoid values is available within this report.

Wrapping it up: timer0 it's disabled only in the maintenance mode and when the car semaphore is green. It will enable timer1 if the led flashing or sound functionalities are required. Timer2 is only used to feed the loudspeaker with a quantized value of the sinusoid, and it's enabled by timer1 or by the button for the blind people. Timer2 required to be power on when initialized, because at reset it's in power down mode.

2 Debouncing, polling and ADC triggering with RIT

I've used the RIT to implement software debouncing of the buttons, to manage the polling of the joystick, and to trigger the A/D conversion.

The debouncing is implemented as usual, by disabling the interrupt mode for the buttons and checking if 50ms after the interrupt the button is still pressed by directly reading the corresponding GPIO pin, the only difference is that I've to take different actions based on the type button pressed. The button for blind people will setup the timer2 to start to reproduce the sound when pressed, until it will be released. To implement this behaviour I've choose to keep track of the kind of button pressed by using a global variable.

The polling of the joystick is very straightforward, given that it's hardware debounced. It allows to enter and exit the maintenance mode state. When doing this, it set up not only the correct semaphore status, but also switch on or off some functionalities: i.e. the ADC is turned off when exiting the maintenance mode, in order to have less the power consumption.

The RIT is also used to trigger repeteadly the AD conversion if the semaphore is in maintenace mode, so that the ADC can read the value of the potentiometer that will be used as a scaling factor to increase or decrease the amplitude of the sinusoid used to reproduce the sound.

I've decided to use the sinusoid of a LA note as reference because it's used by the cars clacsons and it's a note very well distinguishable by the human ear. The sinusoid values over a period have been quantized by 23 samples. The value of each sample of the reference sinusoid is multiplied by the scaling factor obtained by the AD conversion to obtain the values of the sinusoid with the amplitude correctly rescaled. The scaling factor will be always less or equal to 3.3V, to avoid overvolting the speaker.

When the sound will be switched on, the DAC will feed the values of the rescaled sinusoid to the loudspeaker, as written before, the feeding timing is handled by timer2.