Antonio Martí Campoy
amarti@disca.upv.es

Introduction:

In following exercises, you will configure and use the Timer A to create PWM signals.
Create a new project to develop the new functions. You can call this project my_games. Don't forget to change the clock speed to 24MHz in system_msp432p401r.c

Exercise 18.

I'm sure you agree with me that the brightness of the RGB led is too high. In this exercise you will use Timer A (in fact you will use two Timers A) to control the brightness of the RGB led.

First of all, you have to find the outputs of the timers where the RGB led is connected.

Now, create a module called timer (timer.c and timer.h). In this module, create a function called *void config_blue_pwm(void)* to configure the digital pin and the timer. Pay attention, the configuration of the pin is different from the configuration you did in previous exercises. Use SMCLK (SMCLK is 24MHz) divided by 4 as clock source for the timer, set timer in mode UP and compute the value of CCR0 to have a PWM period should be around 800Hz. Configure the duty cycle for 25% and set output mode in 7.

In function main create an infinite loop, and before this infinite loop call *config_blue_pwm();*

If the blue led bright is low, pass to next exercise. Otherwise, ask the teacher.

Exercise 19.

Create a function called *uint8_t pwm_blue (uint8_t duty).*

This function will receive the duty cycle for the blue led in percentage, in the range 0% to 100%.

If the duty cycle is lower than 5%, you must off the led, changing the output mode to 0 and setting the bit OUT to 0.

If the duty cycle is greater than 95%, you must switch on the led, changing the output mode to 0 and setting the bit OUT to 1.

If the duty cycle is between 5% and 95%, you have to compute the number of counts to write in TAxCCRn to get the requested percentage of duty cycle. Remember to change the output mode to 7.

If the duty cycle is out of the range [0%, 100%] switch off the led and return 1. Otherwise return 0.

In function main try this function increasing and decreasing the duty cycle using the Booster Pack upper and bottom buttons. You can copy to this project your module digital and use interrupts to detect the activation of button.

Warning: Show your code to the teacher.

Exercise 20.

Create a function *config_red_green_pwm()* to configure the red and green leds to be controlled by PWM. Use the same parameters than led blue.

Create two functions, *uint8_t pwm_red (uint8_t duty)* and *uint8_t pwm_green (uint8_t duty)* to set the duty cycle of each led. Remember to check the value of the parameter and return 0 or 1.

In order to check the functions use the joystick to control red and green led. You can copy your module *analog* to this project.

Warning: Show your code to the teacher.

Exercise 21.

In the previous code there is a possible source of hazard when you change the duty cycle. For example, when changing the duty cycle of blue led, you write the new value in register TIMER_A2->CCR[1]. There are two scenarios:

1. Guess the new value for register CCR[1] is 3000 and current count in the timer is 2000. In this case, the timer will continue to count until to 3000 and then reset the output. This way you get immediately the new power in the output. Everything is OK.
2. Guess the new value for register CCR[1] Is 1000 and current count in the timer is 2000. In this case, the timer will continue to count until the end of the period (in the case of the led until 7500) keeping the output in high level. During a period you are delivering 100% power when in fact you are asked to reduce the power.

Scenario 2 is not a problem when working with leds, but when you are working with motors, radiotherapy machines or other kind of devices, these power strokes may produce malfunction, damage to the device or even injuries to people.

The solution is to synchronise the update of the register CCR[1] when the PWM re-start, that is, when the timer overflows to zero and re-start counting.

You have to enable the overflow interrupt, TAIFG flag, and in the ISR (`TA2_N_IRQHandler`) update the value of CCTL[n]. Don't forget to access TAxIV register and create some variables to know when you must to update the different registers CCRn. You will also need some static, module variables, to keep the new value of counts until you need it.

Warning: Show your code to the teacher.

Exercise 22.

You can add led control to the game. First, reduce the brightness of the led when selecting the car colour. Then, include in the game loop a function to set the colour and/or brightness of the led as function of the time the player is driving, the speed or simply changing colours.

Exercise 23.

In buttons.c you have created three functions uint8_t check_and_clear_XX_flag (void) to avoid race condition when checking the flag of the buttons. Move the three function into just one, uint8_t check_and_clear_any_button_flag (uint8_t button) where you pass as parameter the number of the button you want to check its FLAG. Create constants in the way # `#define B_UP  0` (and so on) to use names instead of numbers. Comment old uint8_t check_and_clear_XX_flag (void) functions and replace by the proper call to uint8_t check_and_clear_any_button_flag (uint8_t button).

Exercise 24.

Try with the buzzer.

To produce the sound configure a timer in mode UP and output mode 0. The period of the timer is the half the period of tone of the sound (in human audible range). Use interrupt (I suggest you to using CCIFG of CCTL[0]). In the ISR toggle the output of the pin where the buzzer is connected. Remember to configure the buzzer pin as digital output (not timer output).

Also, you have to stop playing the sound after "duration" miliseconds, so you have to count inside the ISR the number of interrupts to stop the playing sound.

Think if you create one function (configuring the timer and start to play the sound) or two functions (one to configure the timer letting it stopped and the other one to start to play the sound). Anyway, create at least a function that receives the tone and the duration of the sound.

Exercise 25.

Add the buzzer to the game. You can use it to make some beep when the user press a button, make the sound of the car engine (related to speed, of course) or produce the crash sound.