ENSE 481 Lab 3

Objective

The objective is to control a servomotor from our development board. We'll ultimately be using the MCBSTM32EXL board (board #2) since that board has the necessary wiring connections to the airflow valve servo. However most parts of the lab, except the actual control of the physical servo, can be easily adapted to the nucleo-f103rb (board #1) by generating the desired waveforms on a GPO pin, possibly connected to a LED, and viewing it on an oscilloscope. In fact, designing your code so it will adapt (probably in the build) to either target aligns with the spirit of this course.

All the instructions below relate to board #2. You would have to make certain changes in each phase to run on board #1.

Background

A servomotor is a motor with a feedback control system allowing rotational or translational position control. In our case we are using the HS-325HB BB: a long-discontinued small rotational servo designed for the RC aircraft hobby market. In the lab materials I have attached two pdf files with the general specifications:

- servo-motor-hs-325hb.pdf
- servo-motor-hs-325hb-extra.pdf

The shaft angle position is specified by sending the servo a coded signal: a sequence of pulses with a period of 20,000µs. As long as the pulses appear on the input line, the servo will maintain the angular position of the shaft even in the presence of perturbing torques. As the duty cycle of the pulse stream changes, the angular position of the shaft changes. The PWM (pulse-width modulated) signal transmits the desired angular position to the servo.

The servo position varies from 0° to 180° as the pulse width varies from 600µs to 2400µs. We'll use our board's timers to generate PWM rectangular waves which control the motor.

General Purpose Timers

The general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler. These timers may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare or PWM*). Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

PWM mode

As always, your primary reference to peripheral driver construction is the RM0008 reference manual from STMicro.

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

The OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. The OCx output is enabled by the CCxE bit in the TIMx_CCER register. Refer to the TIMx_CCERx register description for more details.
In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CNT ≤ TIMx_CCRx.

Procedure

PHASE 1 – PWM lights on the board using a sample program.

A sample project is provided in this handout. Download this code and test it on the target. Step through this code and understand how to control the PWM signals that are controlling the lights. Modify this code and ensure that you are able to control the speed of the lights. Use the reference manual and the board schematic for support. NOTE: this code is deliberately poorly written and can do with some improvement.

PHASE 2 – Modify the code to use PB7 as the output signal.

After completing PHASE 1 you should notice that the lights were attached to PB8 and PB9 using the alternate peripheral Timer 4 Channel 3 and 4 generating PWM. Look at our breakout board schematic and look for PB7 labeled TIMER4. This is in fact pin 137 (LQFP144 package) which has alternate functions I2C1_SCL/FSMC_NADV/TIM4_CH2. We'll configure it as TIM4_CH2 (Timer 4 channel 2) coming from the microcontroller, by enabling the clock for that device and *not* enabling the clocks for the other devices. Eventually we will be hooking up this line to our servo motor. Modify the code from phase 1 to control this line PB7. Consult the servo specification to determine registers PSC and ARR. **Check your PWM waveforms using an oscilloscope. If you have any doubts about the format, ask an instructor.**

Once you've verified your PWM is correct, you can connect the STM32 Breakout board to the Ping-pong interface board, which connects power and control to the valve servo motor. Ensure the servo works correctly.

PHASE3 – UI design

Your CLI should allow the user to input the desired position in a number of ways:

- the percentage the motor should turn. This will correspond to the PWM setting you will create. So for example 0% would be 600µs, or 0° on the servo. 100% would be 2400µs or 180°.
- the actual raw value to be written to the PWM to control the duty cycle. This allows you to see the actual raw values your system is using.

In addition your interface should bounds-check the user input, and provide a useful error message if the value is outside the legal range.

Submit your code as usual. The usual quality standards apply.