Model-based Software Design Assignment Report

Gabriele Cuni
277957

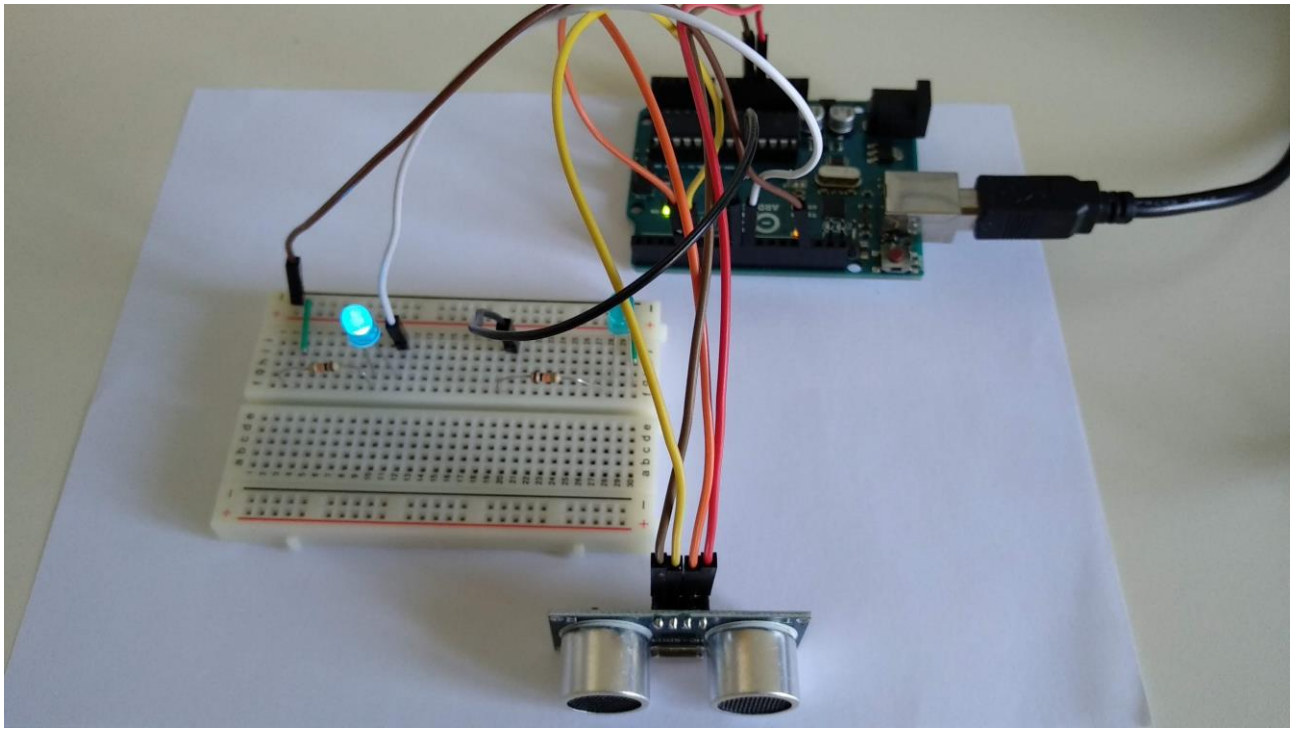
1. Simulink project work-logic

The Simulink controller model I made is based on two parallel states: one performs the processing logic which tackles the trigger signal generation and the echo duration count, the other does the monitoring logic that leads to the LED blinking.
When the trigger signal generated by the controller goes down the echo duration count logic kicks in; the echo_up state measures the echo duration, converts it into centimeters and stores the measurement in a four-cell array by means a sliding average is done.
Thanks to the echo_up_check variable the above logic is aware if the sensor echo response is missing, in this case the distance measurement is set to zero.
The monitoring state is split into three sub-states which deal with in range, out of range and disconnected sensor cases. Any sub-state sets the LED blinking frequency and continuously switches the LED on and off according to its own policy. It is interesting to notice that the disconnected sensor state is achieved when the measurement distance is set to zero by the processing logic.
The integration step is 10 microseconds long, which means my controller cannot be more accurate than 2 millimeters. This choice allows to simulate the full sensor accuracy that is 3 millimeters in the real world; on the other hand, a higher feasible step size would result in lower accuracy. The trigger duration generated by the controller is 20 microseconds so that the sensor model cannot miss any of the signals.

2. Google test

Google Tests are fed with the Simulink test input and output vectors which are loaded into the central memory by means of two functions able to instantiate dynamic arrays. These arrays are chosen so that the Simulink Test achieves 100% coverage.
The Google tests achieve 96% statement and 84.8% branch coverage; The maximum coverage is not met because two counter saturations and one round operation cannot be executed.
The first one is the round instruction that is never run with a negative distance measurement, the second one is a cast to uint16 protection, which means to guard overbound, the last one is a cast to uint8 maximum range protection. All these branches cannot be executed, because they are against the controller function logic or sensor performance.

3. Arduino project



The Arduino project I made includes two LEDs: the left one performs the blinking logic and the right one turns on if Arduino overruns. This is possible thanks to the Simulink detection system which sets on the PIN 8 in case of overrun.
The model is made with the ultrasonic sensor block from the support package for Arduino. This choice forces me to get rid of the controller processing logic since it is done by the ultrasonic block itself. Nevertheless, new considerations on the step size must be done. Since the ultrasonic block performs accurate distance measurements no matter the step size, the model integration step is chosen to manage the blinking logic only. The smallest time quantity the model must tackle is 1 millisecond which comes from the blinking logic when the measurement distance is 99 centimeters as the blinking duty cycle.
Given the above reasoning the model step size is set to 1 millisecond, then against the documentation, the ultrasonic block sample time is set to 0.1 seconds, since smaller values turn the overrun LED on. I made such a decision because empirical evidence shows that it works even though the two steps should be the same.
It is interesting to notice that if the sensor power source is shut down, Arduino goes into the overrun scenario and the LED blinks with a higher frequency than requested. Nevertheless, when the power supply is restored, so is the blinking logic, but the overrun LED remains on.