EECS 461, Winter 2009, Problem Set 2[1]

issued: Wednesday, January 28, 2009 due: Wednesday, February 4, 2009.

1. In all sensor interfacing, it is necessary to minimize the response of the system to *noise* in the measurements. For example, in quadrature decoding, noise can cause spurious pulses. These can lead to erroneous updates of the counter, and thus to erroneous position measurements.

   There is a certain amount of noise immunity built into the MPC5553 eTPU, and thus into the QD function we use in lab. This is explained in Section 5.8.6 of the eTPU Reference Manual, available on the class website. Consider so-called "two sample mode." There is a digital filter that samples the input two times with a sample period that is a multiple of the system clock, which in our case is set to 40MHz. If the multiplier is equal to 2, then the sample frequency is 20MHz. If the samples are identical, then the input signal state is updated. A pulse that lasts only one sample is treated as noise. Hence we see that

   (i). Pulses of 2 CPU clocks or less in duration are rejected.

   (ii). Pulses of 4 CPU clocks or more in duration are passed.

   (iii). Pulses of 3 CPU clocks in duration may or may not be passed depending on how the edges of the pulse are phased with respect to the sample times.

   (a). What is the signal latency contributed by noise suppression to the QD function?

   (b). How does the latency change if the filter multiplier is set to 4?

   (c). Suppose the filter multiplier is set to 2, and a spurious pulse of finite duration greater than 4 CPU clocks appears on only one channel. Does this cause a permanent error in position count? Explain.

   (d). Suppose there are overlapping spurious pulses on both channels. Does this change your answer from question (c) above?

2. In this problem, we will use Matlab, Simulink, and Stateflow to construct a simplified model of a quadrature decoding algorithm. The purpose of the problem is both to learn more about quadrature decoding as well as to learn how to set up and run a Stateflow model. Much embedded control software consists of event driven state transitions, and is best modelled using finite state machines. In fact, Stateflow was developed so that engineers could model the interaction between such software and a model of a dynamical system. Stateflow uses Harel statecharts, which are also popular in the object oriented programming community.

   There are several Matlab files available on the class web site. The first of these, "run_quad_decode.m", is used to generate the two quadrature signals shown in Figure 1. These signals are then used as inputs to the Simulink model "quad_decode_fast.mdl", shown in Figures 2-3, written to model the "fast" mode of quadrature decoding as implemented on the MPC5553. Note several features of this model:

   (i). the secondary channel is ignored.

   (ii). by clicking on the Tools→Explore menu of the state chart, one can see that a transition is triggered only on the rising edges of the primary channel

   (iii). on the same menu, one sees that the counter is initialized to zero

   (iv). the counter is incremented in multiples of 4 (why?)

   Run the Matlab file "run_quad_decode.m", and inspect the plot of counter value vs time.

   (a) What value does the counter have at the end of the simulation?

   (b) Are the changes in counter value you see consistent with your understanding of the fast quadrature decode mode of the MPC5553?

   (c) Does the final value of the counter correctly represent the change in wheel position?

[1] Revised January 22, 2009

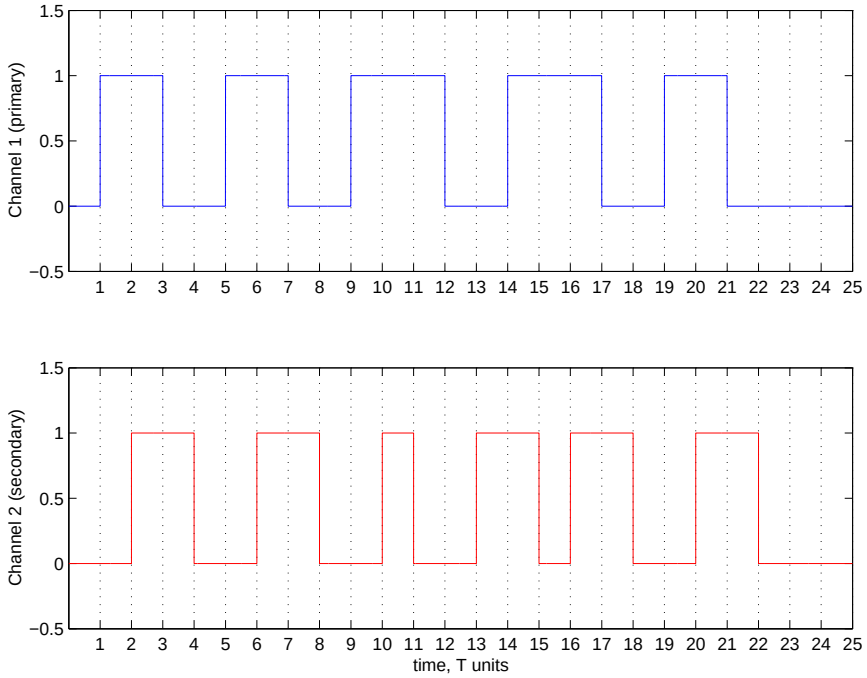

Figure 1: Quadrature Signals from Encoder

(d) Now inspect Figures 4-5, which are a partial implementation of the slow decode mode of the MPC5553. Change the Simulink model to have the form shown in Figure 4. To do so, open the "Mux" block and change the number of inputs to 2, remove the terminator from the secondary input, and enter it into the "Mux" block. With both primary and secondary inputs available, one can perform slow mode, or $4X$, quadrature decoding by using a state machine with 4 states, as shown in Figure 5. Complete Figure 5 by adding transitions between the four states. You will need to open the Tools→Explore menu of your Stateflow block, add the secondary input as an input event, and set the two events to trigger appropriately.

Your completed model of normal mode quadrature decoding should yield the plot of counter value shown in Figure 6. Hand in the corresponding plot from your program to show that it achieves the same result, as well as your Stateflow chart. (We reserve the right to test your program on a different set of quadrature signals, so you should perform such tests yourself to make sure that your code works in general, and not merely for the given examples.)

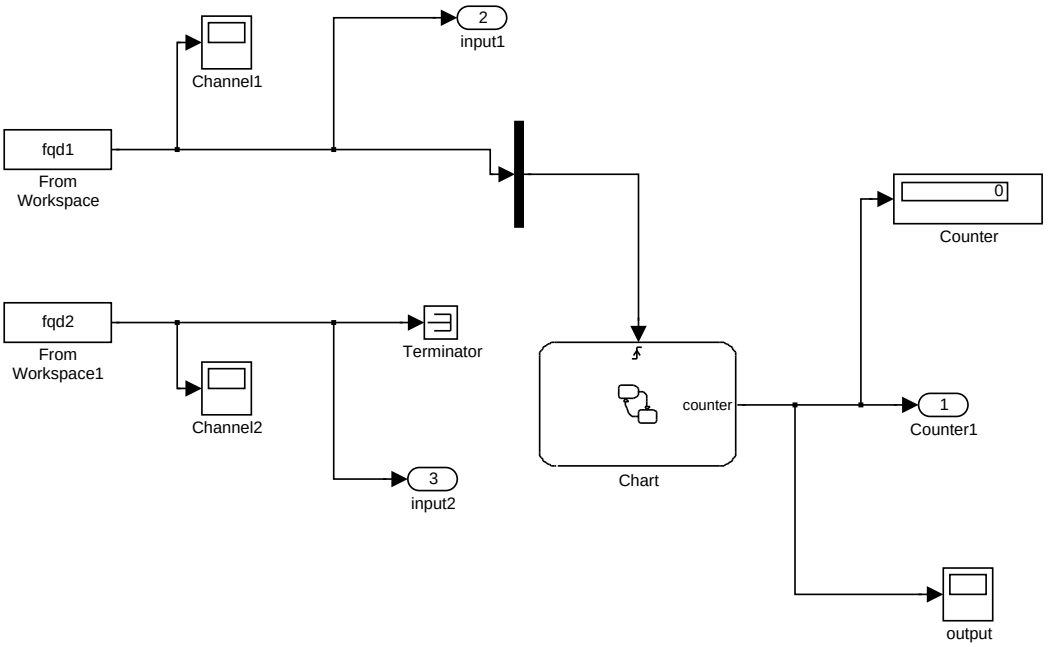

Figure 2: Simulink Model for Fast Mode Quadrature Decoding

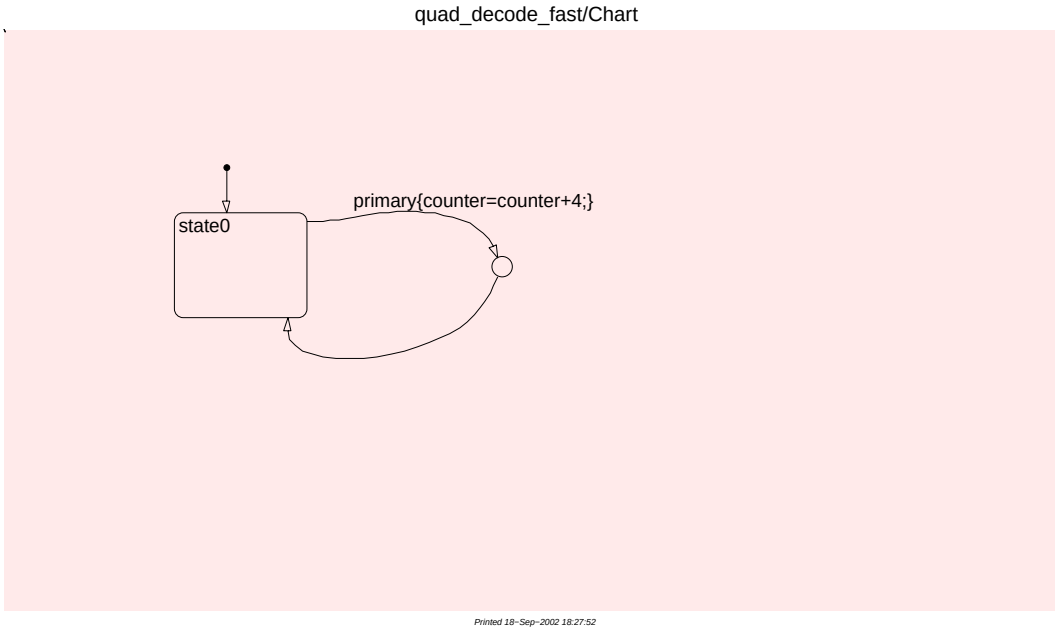

Figure 3: Stateflow Chart for Fast Mode Quadrature Decoding

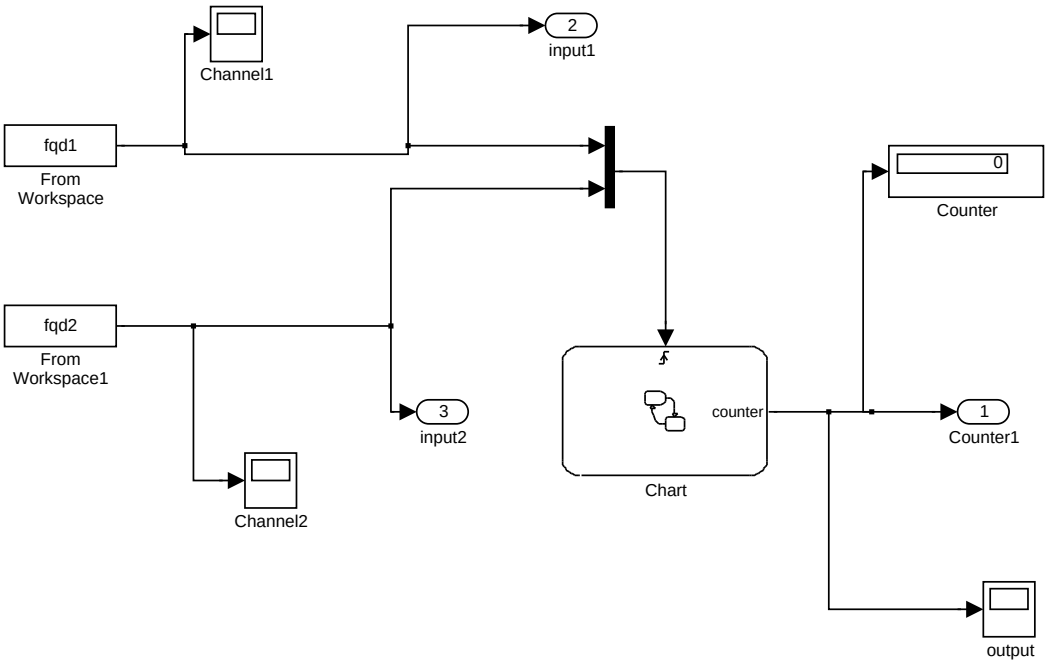

Figure 4: Simulink Model for Normal (4X) Mode Quadrature Decoding

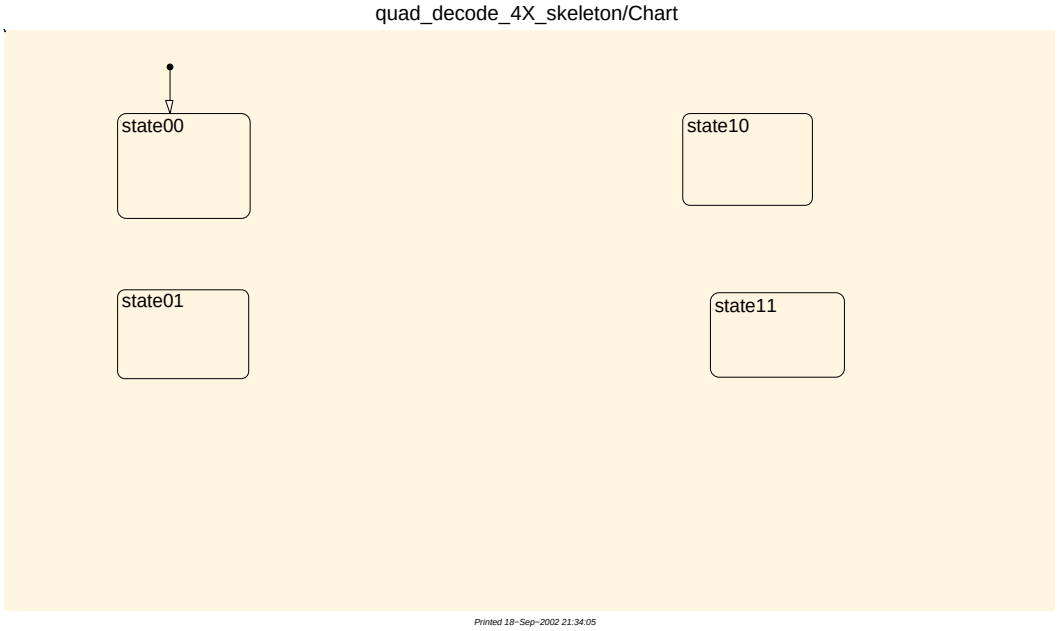

Figure 5: Incomplete Stateflow Chart for Normal (4X) Mode Quadrature Decoding

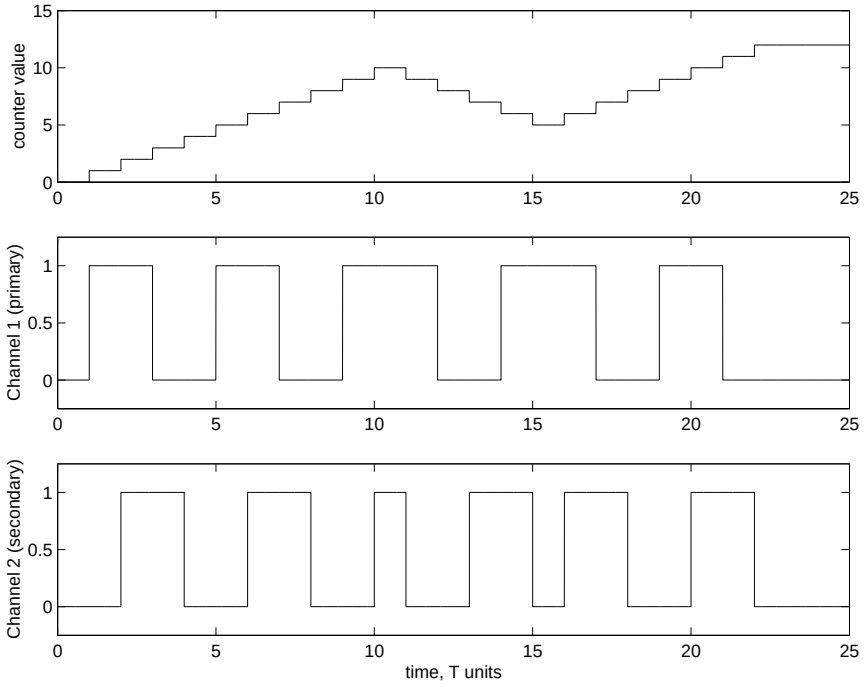

Figure 6: Counter Value vs Time for Normal Mode Decoding