UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS 249 **A. Sangiovanni-Vincentelli**
Fall 2012 **P. Nuzzo**

Homework 2
Due Tuesday, October 2, 2012

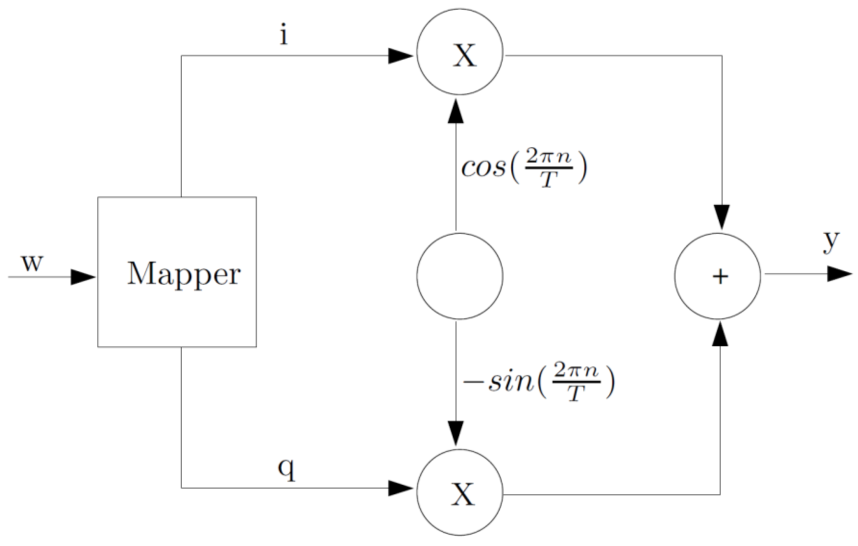
General remark: since we will mostly be dealing with design problems throughout this class, an answer to a homework problem may not necessarily be right or wrong. Grade is established mostly based on the reasoning that you follow while developing your solution. Therefore, make sure that you justify all your claims. You can use any kind of sources as long as you include references.

The goal of HW2 is to highlight differences and trade-offs between models of computation from a design viewpoint and apply some analysis algorithms for dataflow process networks and Petri nets.

1. **A digital modulator.** We assume that there is a source that generates encoded words $w \in V$ where V is a vocabulary of words. For instance, the vocabulary can be composed of strings of 5 bits in which case the $\#V = 32$. Each word is mapped to a pair of numbers by a mapper which is an injective function $f: V \rightarrow X \times Y$, where X, Y $\subset \mathbb{Z}$. Each pair $(x,y) \in X \times Y$ is a point in the Euclidean plane. The range of the function *Im(f)* is called a constellation. The two signals generated by the mapper are called the in-phase and quadrature components of the input signal and are denoted respectively by *i* and *q*. It is possible to send this complex signal on the air using a modulator. A modulator multiplies the in-phase component by a co-sinusoid, the quadrature component by a sinusoid, and then adds the two signals together. The functionality of this system can be written in the following way:

$$y[n] = f(w[n])|_x \cos\left(\frac{2\pi n}{T}\right) - f(w[n])|_y \sin\left(\frac{2\pi n}{T}\right)$$

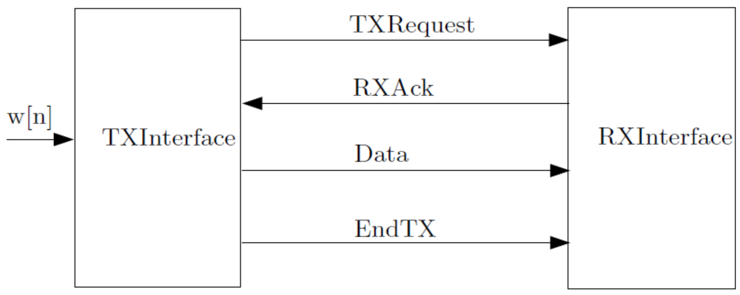
where $f|_x$ is the projection of f on the first component, $f|_y$ is the projection of f on the second component and T is a constant. An architecture used to implement this function is shown in the figure below.



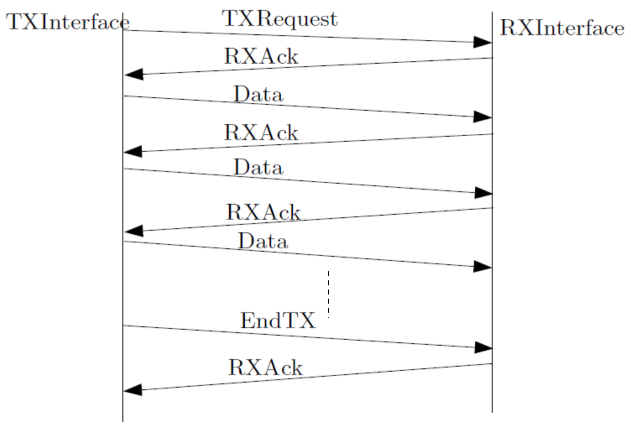
a) Describe the system using the Codesign Finite State Machines (CFSM) model of computation. In particular, write each block in the diagram as a CFSM. You may assume to have two functions f_x and f_y that can be called on transitions of the mapper CSFM.

b) Now describe the same system using the dataflow model of computation.

2. Sender and Receiver. We want to design two protocol interfaces, as shown in the block diagram below.

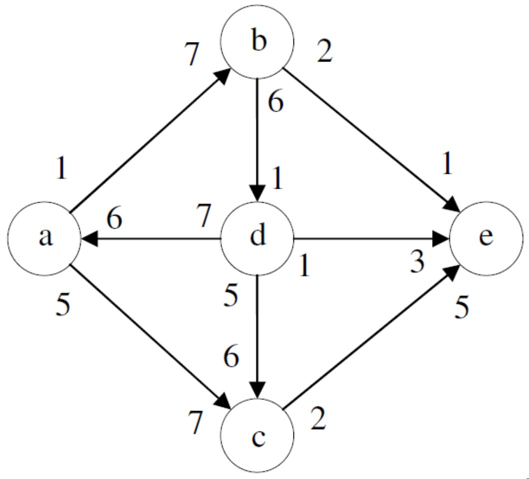


One scenario of this simple protocol is described by the activity diagram shown in the figure below. This scenario is triggered by an input of the TXInterface, which is an array of *n* words. When the interface receives the array, it stores the array in an internal state variable and starts sending it, word by word, to the destination interface. A transmission is initiated by the TXInterface with a TXRequest signal. The receiver answers with an "acknowledge". Then the TXInterface sends a series of data (depending on *n*) waiting for an "acknowledge" each time a data is sent. To close the session, the TXInterface sends an "end of transmission" signal and, before going to "idle" again, TXInterface waits for the last acknowledgment.



a) Describe the two interfaces using the CFSM model of computation.
b) Now describe the two interfaces using the dataflow model of computation. For this question, you may want to consider the two interfaces as two dataflow graphs made up of finer grain actors.

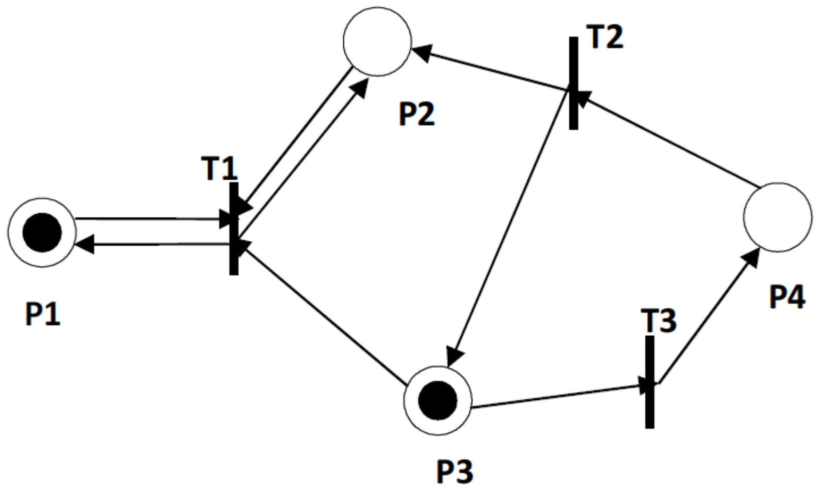
3. Analysis of Static Dataflow Networks. Consider the following static dataflow graph:



Determine:

a) the balance equations and a periodic firing vector;
b) a valid single appearance schedule, and add delays on edges (you can choose how) to make the schedule valid;
c) the buffer memory lower bound for a single appearance schedule as defined in the paper "Joint Minimization of Code and Data for Synchronous Data Flow Programs" by P. K. Murthy, *et al.*;
d) the lower bound on the amount of memory required by any schedule;
e) the buffer requirement of your schedule;
f) a schedule with lower memory requirements, by neglecting the single appearance assumption.

4. **Analysis of Petri Nets.** Consider the Petri net PN1 in the figure below:



a) Derive the coverability tree of PN1.
b) Find a Petri net PN2 such that:
    1. the coverability tree of PN2 is the same as that of PN1;
    2. in PN2, the marking M = (1, 1, 0, 0) is not reachable from the initial marking M_0 = (1, 0, 1, 0).