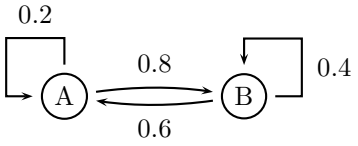
Prof. Bernd Finkbeiner
Dr. Calogero Zarba
Moritz Hahn

Problem Set 7
Due: June 26, 2007, 9:14 am

Embedded Systems

Problem 1 (Markov processes) [20 points]

Consider the following Markov process



(a) Assuming that initially the process starts in state A, what is the probability $s_2(A)$ that the process will still be in state A after 2 transitions?

(b) What is the limit probability $s_{\lim}(A) = \lim_{n\to\infty} s_n(A)$? What is the limit probability $s_{\lim}(B)$?

(c) Explain why you were able to answer the previous question without knowing the initial probabilities $i(A)$ and $i(B)$.

Problem 2 (A/D conversion) [20 points]

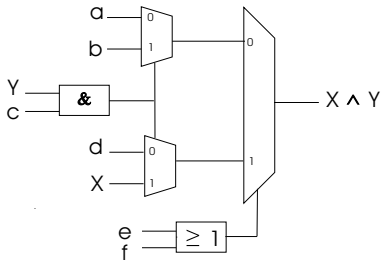
During the lecture we presented the successive approximations method as one possibility for converting analog signals into digital values. Using the successive approximations method, carry out the conversion of the input voltages $U_{\text{in}} = 2.25V$, $3.75V$, and $1.8V$ into the corresponding binary values. In each case, and for each step of the conversion, show:

- the arranged comparison voltage U_{ref};
- the binary value after each comparison.

The digital value should have a precision of 4 bits. Assume also that the working range of the A/D converter lies between $U_{\min} = 1V(0000_2)$ and $U_{\max} = 4.75V(1111_2)$.

Problem 3 (Digital circuits) [20 points]

(a) Consider the following CLB (configurable logic block) from a FPGA (field programmable gate array) of the company ACTEL:

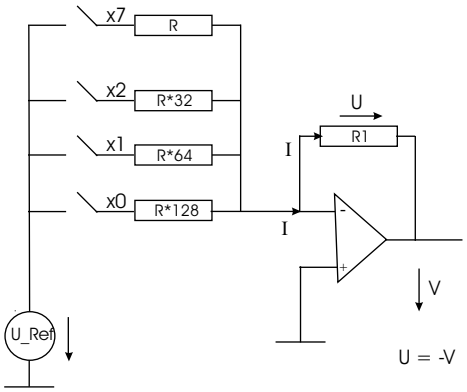


Implement the function $x \wedge y$ by giving an instantiation of the inputs a, b, c, d, e, and f.

(b) Implement the functions $\neg x$ and $x \vee y$ with the help of one multiplexer.

Problem 4 (D/A conversion) [20 points]

Consider the following A/D converter.



Assume that $U_{\text{ref}} = 5V$, $R = 1k\Omega$ and $n = 8$ bit.

(a) What should be the value of the resistance R_1, when the digital values map the voltage range between $U_{\min} = 0V$ and $U_{\max} = 10V$?

(b) The accuracy of the A/D converter strongly depends on the precision of the used resistances. With too much deviation from the desired values of the resistances, a *monotonicity error* could occur, i.e., the delivered current for a certain binary value d could be smaller than the delivered current for the binary value $d + 1$. The greatest impact of such monotonicity error holds for the binary values 01111111_2 and 10000000_2.

Compute the value of the delivered currents for the binary values 01111111_2 and 10000000_2.

(c) How much can the topmost resistance R deviate (upwards), without having a monotonicity error?

Problem 5 (SDL/Flexray) [20 points]

The distributed, fault-tolerant communication protocol FlexRay is based on the Lundelius-Lynch clock synchronization algorithm. The Lundelius-Lynch algorithm solves the problem of maintaining closely synchronized local times, assuming that processes' local times are closely synchronized initially.

As shown in class, the following pseudocode describes the behavior of a process `p` during one round of the Lundelius-Lynch algorithm. Remember:

- The function `getTime()` obtains the local time by adding the value of the variable `correction` to the value of the local read-only clock.

- The function `reduce()`, applied to an array, returns the multiset consisting of the elements of the array, with the f highest and f lowest elements removed. (That is, we assume that there are no more than f faulty processes.)
- The function `midpoint`, applied to a multiset of real numbers, returns the midpoint of the set of values in the multiset.
- `ALPHA` is the lenght of a single round.
- `BETA` is the length of time a process waits before adjusting the `correction` variable.
- `DELTA` is the expected delay of message delivery.

```
for(;;) {
  while (getTime() < alarm) {
    if (!buffer.empty()) {
      q = buffer.get();
      processes[q] = getTime();
    }
  }
  t = getTime();
  broadcast(p);
  alarm =  t + BETA;
  while (getTime() < alarm) {
    if (!buffer.empty()) {
      q = buffer.get();
      processes[q] = getTime();
    }
  }
  average = midpoint(reduce(processes));
  adjustment = t + DELTA - average;
  correction += adjustment;
  alarm = t + ALPHA;
}
```

Rewrite the above pseudocode using the Specification and Description Language (SDL). In particular, replace while-loops with the help of timers.