COMS W4995-02
Languages for Embedded System Design
Homework 3

Prof. Stephen A. Edwards
Columbia University

Assigned October 28, 2002
Due November 18, 2002

You may submit the solutions either on paper or electronically, but not both. If you submit it electronically, send a single file that is text, PostScript, PDF, or Word. Don't send multiple files or an archive such as tar or zip. I just print out whatever you submit and read it; I don't try to run the programs.

Make sure your name appears at the beginning of the file you send.

I want the paper or electronic versions at the beginning of class (4:10 PM EDT) on the due date. This applies to both on-campus and CVN students.

1. **(40 points)** Esterel programming.

   I've installed the Esterel compiler on labdien.cc.columbia.edu, part of the cunix cluster (it core dumps on the others because it requires Solaris 2.7, not 2.5). The distribution is in my home directory, i.e., ~se2007/esterel. For those who prefer to run their own copy, the compiler can be downloaded from the Esterel Technologies web site[1]. It is available for Linux, Solaris, and Windows NT.

   Here's how to compile and run an Esterel program:

```
$ uname -r          # Check OS version, Solaris only
5.7                 # Can't be 5.5.1
$ cat > hello.strl
module Hello:
input A; output B;
await A; emit B
end module
^D
$ ~se2007/bin/esterel -simul hello.strl
$ cc -o hello hello.c ~se2007/lib/libcsimul.a
$ ./hello
Hello> ;            # Clock tick with no inputs
--- Output:
Hello> ;
--- Output:
Hello> A;           # Make input A present and tick
--- Output: B
Hello> ;
--- Output:
Hello>
```

[1] http://www.esterel-technologies.com/v2/download/download_free_software.html

(a) Write an Esterel module that implements a two-bit grey-code up counter whose counting sequence is 00–01–11–10–00. It should have two inputs, C, which increases the count, and R, which resets the counter; and two outputs B0 and B1 that should always indicate the current count (i.e., B0 should be present when the least significant bit of the count is 1). The new value should appear when C is present, and R should take precedence over C. Hint: Try to think concurrently. The best solution to this problem consists of a few processes running concurrently.

(b) Add a user interface to your little counter: Make it advance once every 5 arrivals of a "T" signal, and add an alarm signal "A" that is set off when the count reaches a given setting. Set the alarm with three environmental signals: S, S0, and S1. When S is present, S0 and S1 contain the new setting. Thus, your program should behave like this:

```
S S1;          # Set the alarm to 10
T; T; T; T; T; # Count is now 01
T; T; T; T; T; # Count is now 10 and A is present
```

For both programs, show me the source and an example run that tests it.

2. **(60 points)** More Esterel Programming

This is a simplified version of an industrial problem due to Bob Paddock[2].

Create an Esterel program that controls an overhead crane. Here's the interface:

```
module Crane:

input DISABLE;
input LOCKOUT;

input OFF, ON, START;

sensor JOY : integer;
input DR, BR;

output D1, D2, D3;
output B1, B2, B3;

output RUN, FAULT;

end module
```

This amounts to a controller for a very large electric motor driven through a bank of relays that either power it or connect it to resistors the size of hot water heaters that act as a brake. When braking, the motor becomes a generator and the resistors dissippate the kinetic energy of the crane, slowing and eventually stopping it. Such a design minimizes the number of (guaranteed-to-break) moving parts.

Outputs D1–D3 enable relays for driving and B1–B3 enable the relays for braking. Figure 1 show how the relays should be enabled in various modes.

Two LEDs controlled by the RUN and FAULT outputs indicate the state of the system to the user. When the unit is off, neither should be on. RUN should be

[2]Thanks Bob! http://www.unusualresearch.com/

state	B3	B2	B1	D3	D2	D1	Stick Value
Brake 7	X						0
Brake 6	X		X				1
Brake 5	X	X	X				2
Brake 4	X	X					3
Brake 3		X					4
Brake 2		X	X				5
Brake 1			X				6
Neutral							7 and 8
Drive 1						X	9
Drive 2					X	X	10
Drive 3					X		11
Drive 4				X	X		12
Drive 5				X	X	X	13
Drive 6				X		X	14
Drive 7				X			15

Figure 1: Relay states for the seven brake and drive states. An X indicates an output should be present in that state. The grey code encoding ensures that at most one relay changes at a time.

present whenever the unit is operating normally. If a fault occurs, the FAULT light should blink until the user switches the unit to OFF (see below).

The environment supplies two enabling inputs, DISABLE, and LOCKOUT, which should both be absent at all times for the unit to work. The unit should brake and flash the FAULT light if either of these is ever present.

The user's controls include of a car-like "ignition" switch that activates and deactivates the unit. The key can be removed when in the OFF position. To activate the unit, the user turns the switch briefly to START, then it returns to ON. The FAULT light should blink if the controls are not zeroed (i.e., JOY at 7 or 8, BR and DR off) when the unit is switched to start and continue to blink until the unit is returned to the OFF state.

In normal operation, the user operates an analog joystick whose values range from 0 (full brake) to 15 (full forward). 7 and 8 are neutral. Note that the joystick controls drive/brake, and *not* forward/reverse.

The BR/DR inputs come from switches on the joystick that become active below 7 and above 8 respectively. These inputs are redundant; they are meant to check for a fault in the joystick input.

Any sort of inconsistency on these inputs should be treated as a fault and cause the system to shut down. For example, BR and DR inputs being on simultaneously, BR being on when the joystick input reads above 8, etc. Furthermore, the system should shut down if the joystick number changes by more than one per clock cycle.

There are seven drive and seven brake states in addition to neutral. Figure 1 illustrates the relay activation rules for each of these.

(a) Write a module called Relays that continuously translates the JOY sensor value into the relay activation patterns in Figure 1.

(b) Write the topmost module. Start from the following template and flesh out the commented areas.

```
loop
  trap Fault in
    await START;
    abort
      % exit Fault if the controls are not safe
    when ON;
    abort
        sustain ON
      ||
        % Normal operation: run Relays
      ||
        % Check for faults and exit Fault if any appear
    when OFF

  handle Fault do
    abort
      % Blink the FAULT light
    when OFF
  end;
  pause
end
```

(c) Devise a simple test suite (i.e., a file that runs the simulator through some interesting states). Show me both this suite and the result of running it through the simulator.

The test suite will be some files that look like

```
OFF JOY=8;
OFF;
START;
ON;
ON;
ON JOY=9 DR;
ON JOY=10 DR;
ON JOY=12 BR;
```