Lab1:POLIS hands-on laboratory session

NOTE TO READER: This lab is a text originally authored by unknown persons for the embedded systems class at UC Berkeley. This text is being used to illustrate the software designed at that institution.

The self-guided lab session will teach you how to specify, simulate and synthesize in the POLIS co-design environment. At this point we assume that, after today's lecture or reading the User's Manual and/or the POLIS book, you are familiar with the Polis design flow, the ESTEREL Programming Language and the CFSM model of computation.

The Lab is divided into three different parts:

1. An introduction and background to the POLIS co-design environment. (**You are required to read this before the lab**)
2. Esterel Introductory Tutorial.
   This will guide you through the operations to create and compile modules in the Esterel language. (**It is <u>recommended</u> you read this prior to the lab**)
3. Specification and Simulation of Seat Belt Alarm Controller.
   This is your first example of a Esterel/Polis/Ptolemy co-design application. This is more for familiarizing you with the software than actually using co-design principles.
4. Use the knowledge you gained in the previous sections to implement an elevator controller.

In case of troubles, ask the TA or you can visit the course website at http://www.cs.tamu.edu/course-info/cpsc489/rabi/.

1. An Introduction to the POLIS co-design environment

In the field of co-design/simulation several tools have arisen to facilitate faster and smoother development. One of which is the three-member team of Esterel, POLIS, and Ptolemy.

Esterel is a programming language that focuses on timing and signals as it main concern. It is similar to Verilog and VHDL, but is less hardware oriented. The reason that we are using Esterel is because currently, this is the only language supported by POLIS. More info can be found at http://www.esterel.org/

POLIS is a program that is used to translate specifications (in this case written in Esterel) into CFSMs(Co-design Finite State Machines). A CFSM, like a classical Finite State Machine, transforms a set of inputs into a set of outputs with only a finite amount of internal state. The difference between the two models is that the synchronous communication model of classical concurrent FSMs is replaced in the CFSM model by a finite, non-zero, unbounded reaction time. This allows for the program to break up the specification into a network of nodes, which are then in turn used to define the hardware and software elements of the system. More info can be found at
http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

Ptolemy is a graphical environment in which the different elements of a design can be edited and simulated. Within Ptolemy, you can take the various modules written in Esterel (called stars by Ptolemy) and specify their use within the system (galaxy) and whether they should be hardware or software when used in co-operation with POLIS. You can also simulate your design and view your results in a visual environment.
More info can be found at http://ptolemy.eecs.berkeley.edu/

2 Esterel: An Introductory Tutorial

This tutorial provides a quick introduction to the basic constructs of Esterel. For a complete description of the language, we recommend to read "The Esterel v5 Language Primer" by G. Berry. In this Section we first present a small example that shows how to run Esterel simulations, then we introduce the constructs that will be used use in the rest of the lab.

2.1 A small example (from "The Esterel Primer").

Specification:
"Emit output O as soon as input I has occurred n times.
Reset this behavior each time the input R occurs".

The Esterel file, to be created in esterel_ex/ex1.strl, is:

NOTE ON TEXT EDITORS: We assume that by this time in your academic career you have used a text editor in UNIX. This lab uses "pico" by default (which is the easiest to understand). You may, of course, use whatever you want (i.e. vi, emacs, etc.)

```
module Example:
constant n: integer;
input I, R;
output O;
loop
   await n I;
   emit O;
each R;
end module
```

To run the Esterel simulation, apply the following steps:

0. **cd esterel_ex**
1. **esterel –simul ex1.strl**
2. **pico ex1.h** (with a single statement, **#define n 10**)
3. **cc –c ex1.c**
4. **xes ex1 ex1**
5. **xes ex1** *(see Figure 1)*
6. Click on **tick** in the simulation main panel. In the panel displaying the esterel code *await* assumes color red and *each* is underlined. This mean they are being executed right now.
7. Click on **I**, **tick** in the simulation main panel. Repeat this action three times. Output **O** becomes red.
8. Click on **R** and **tick.**
9. Repeat point 6. and 7.

<u>When you turn in your report for this lab, include this Esterel file.</u>

Note that the value of the constant n is defined in the file ex1.h In this case, we have defined it to be three.

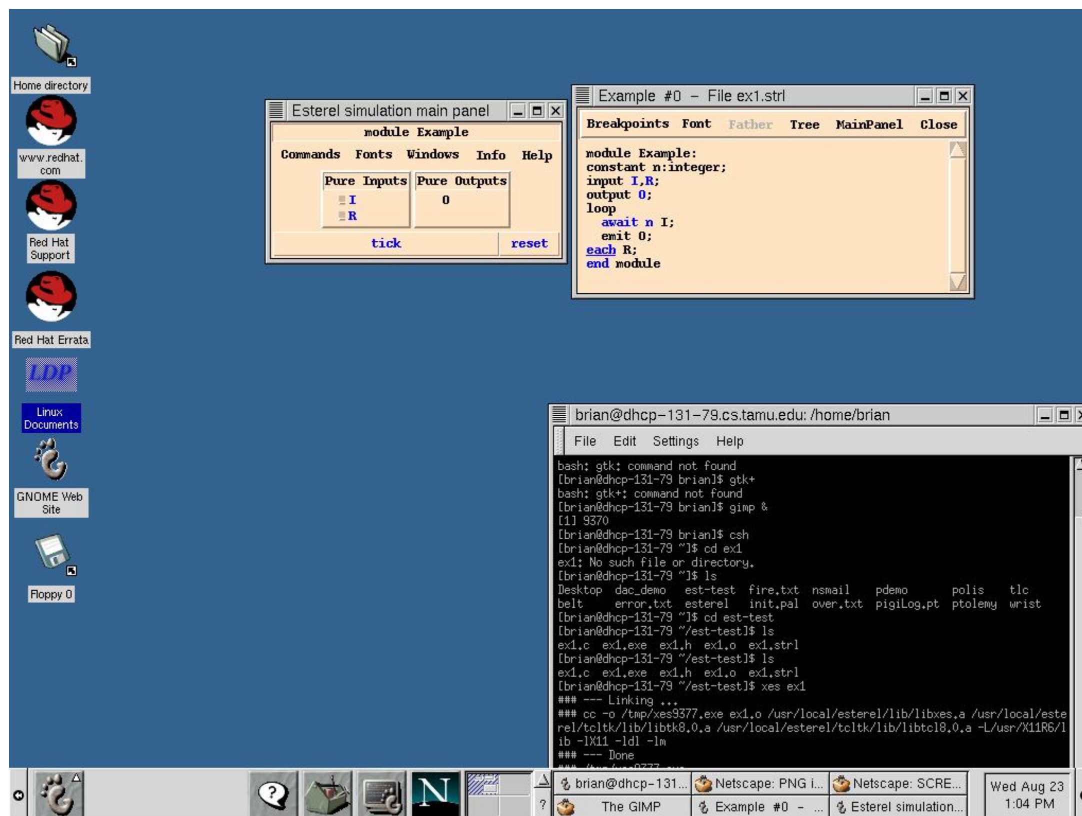


Fig 1. This is what you should see after executing the Esterel Simulator (xes)

Review of useful Esterel constructs

NOTE TO READER: These constructs are all that is needed to complete this lab. ***You may not use any other constructs beyond these and those shown in the previous example for this lab.*** *In future labs you may use any constructs you find (in books or on the web).*

1. Loop.

```
loop
  ...
end
```

The loop statement loops forever.

2. Await.

```
await I
```

Delay statement that terminates when I occurs.

```
await n I
```

It counts n occurrences of the event I and only then yields control to the next statement.

3. Emit.

```
emit O
```

Emission of a signal O is an instantaneous statement.

4. Abort.

```
abort
    p
when S
```

The abort .... when statement executes p until the signal S is detected and then is instantaneously killed. Using the *weak abort ... when* statement the body p is executed a last time at abortion time.

Example:

```
abort
   loop
      emit A;
      await B;
      emit C;
      await D;
   end;
when [ R1 or R2] ;
```

The internal loop is aborted as soon as any of the input signals R1 or R2 occurs.

5. Loop ... each

Temporal loop: loop + strong abortion statement.

```
loop
  p
each d
```

is equivalent to:

```
loop
    abort
       p; halt
    when d
end
```

6. **Every do**

Temporal loop:

```
every d do
    p
end
```

is equivalent to:

```
await d
loop
   p
each d
```

3 Specification and Simulation of Seat Belt Alarm Controller

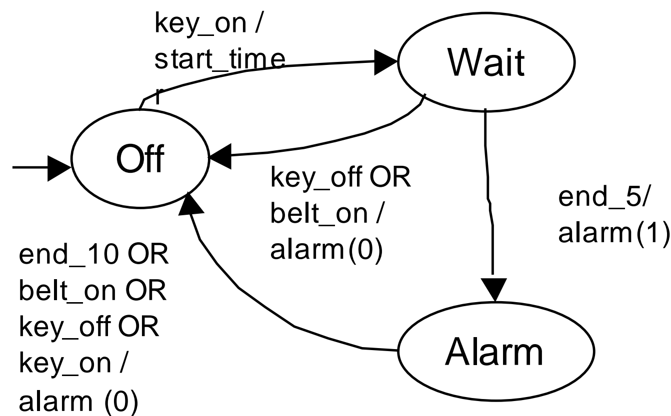
In this part of the tutorial you will

1. Write an Esterel file for the controller.
2. Simulate controller with Esterel simulator for correctness.
3. Write an Esterel file for the timer.
4. Simulate timer with Esterel simulator for correctness.
5. Compile Esterel description into Ptolemy stars and create testbed for simulation in Ptolemy.
6. Do functional Simulation using Ptolemy Co-simulation framework
7. Do performance simulation using Ptolemy Co-simulation framework.

Suppose that we want to specify a simple safety function of Automobile: a seat belt alarm. A typical specification given to a designer would be:

"Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for ten seconds or until the key is turned off."

We will use two interacting modules to realize this specification: a controller, and a timer. The specification of the controller can be represented in a reactive finite state form as follows:



3.1 Create Esterel file for controller

First, starting from your home directory:

mkdir ~/belt
cd ~/belt

Create a file with the name **belt_control.strl** that has the following characteristic:

a. name of module is: **belt_control**
b. inputs are all pure events: **reset**, **key_on**, **key_off**, **belt_on**, **end_5**, and **end_10**.
c. outputs are: **start_timer** (a pure event), and **alarm** (an event with data part typed Boolean).
d. has the behavior specified above.

You may use some or all of the following construct in the body of the module

loop *Statement* end
abort *Statement* when *DelayExpression*
emit *SignalIdentifier* (*Expression)*
;
every *DelayExpression* do *Statement* end
or
emit *SignalIdentifier*
await *DelayExpression*

Recall that in Esterel, *DelayExpression* can be defined by the occurrence of some signal. Also be careful about () and []. () is used to denote data while [] is used to group statements.

Convince yourself that this Esterel file indeed has the same functionality as the FSM above.

3.2 Esterel simulation of the controller

Compile the Esterel file for simulation with the following two commands:

esterel –simul belt_control.strl
cc –c belt_control.c
xes belt_control belt_control

and simulate it:

xes belt_control

At this point, you should get the x-window simulation interface of Esterel. you should

a. click on **tick** to output any initial events.
b. click on **key_on**, then **tick**, and see that **start_timer** is emitted
c. click on **end_5**, then **tick**, and see alarm output **alarm(true)**
d. click on **belt_on**, then **tick**, and see alarm output **alarm(false)**

Note the color change on and around the keyword and statement of the esterel code as it is being executed.

3.3 Create Esterel file for timer

Create the file **timer.strl** that corresponds to the functionality of timer. The timer should have the following characteristics:

a. module name: **timer**
b. constant declaration for **count_5** and **count_10**, both integers.
c. input pure events: **msec**, **start_timer**
d. output pure events: **end_5**, **end_10**
e. The functionality of the timer is: *Every time a start_timer occurs, wait for count_5 msec and output end_5, wait for another count_10 msec and output end_10. If these operation is not completed when the new start_timer comes in, start over and discard the current computation.*

Some of the statements you may want to use are:

every *DelayExpression* do *Statement* end
await *Constant SignalIdentifier*
emit *SignalIdentifier*

Convince yourself that the Esterel file you wrote does satisfy the requirements.

3.4 Esterel Simulation of timer

Create the file **timer.h** containing the following two lines:

#define count_5 5
#define count_10 10

This will serve as constant definition for Esterel simulation. To be correct, it should be 5000 and 10000 respectively, but for the purpose of Esterel simulation, we will scale it down to 5 and 10 for now.

Compile the Esterel file for simulation with the following two command:

esterel –simul timer.strl
cc –c timer.c
xes timer timer

and then to start the simulation:

xes timer

At this point, you should get the x-window simulation interface of Esterel. you should

a. in main panel, select **command-keep inputs**
b. click on: **tick**, **start_timer**, **tick**,**start_timer msec**,**tick**,**tick**,**tick**,**tick**
a. now you should see **end_5** event gets emitted
b. do “**tick**” 10 more times, and see **end_10** gets emitted
c. you may want to throw in a **start_timer** before the counting is complete and see the count get restarted.

3.5 Convert Esterel description into Ptolemy stars and create simulation testbed

We will first need to convert the esterel file to a form that is simulatable in Ptolemy with Polis semantics. Create a file **Makefile.src** containing the following:

TOP = belt
STRL = belt_control.strl timer.strl
PTLUC = -a 68hc11 –a 68332

belt will be the top level design name, and **STRL** define the esterel files involved, while **PTLUC** define the processor that is being considered. They will come into play when one performs co-simulation in the next section.

Generate a **Makefile** from this source by executing

makemakefile

We should now compile esterel into delay augmented C-file for simulation and also generate all necessary wrapper for ptolemy simulation. This can be done as follows:

mkdir ptolemy
make ptl

Look at the screen printouts to see what is actually done for you by the Makefile. We are now ready to start building up simulation block in Ptolemy. Go to the **ptolemy** directory and start the Ptolemy simulator:

cd ptolemy
pigi belt

You should wait for the picture of Ptolemy, then click **OK.**

A summary of useful Ptolemy commands is shown below. You can access most of these command also by menu, The **Schematic** menu (generic netlist commands) is activated by the **CTL-middle mouse button** and the

pigiRPC menu (simulation-specific commands) is activated by the **SHIFT-middle mouse button**.

Create Star	*****
Open palette	**O**
Create/Copy	**c**
Full Screen	**f**
Pan	**p**
Zoom-in	**z**
Zoom-out	**Z**
Close-window	**CTL-D**
Select	**s**
Unselect	**u**
Undo	**U**
Move	**m**
Delete	**D**
Create-icon	**@**
Open-facet	**F**
Look-inside	**i**
Edit Parameter	**e**
Run	**R**
Get info	**,**
Save window	**S**

The **delete** key can be used to erase a point, box or line segment if it has been incorrectly entered, while **CTL-U** erases the complete list of points, boxes and lines.
Saving netlists (galaxies) periodically, by typing

S

on top of their window is generally a good idea.

To create a Ptolemy star from an Esterel file, type

*

in the **belt** window.
The name of the star is **belt_control**, Domain is **DE** for discrete event. **Star src directory** will be the directory where the ptolemy wrapper is created, in this case, it should be **~/belt/ptolemy.** The pathname of the Palette should remain **./user.pal** That is where you can find the star you just created.

Do the same thing for **timer**

After we have created the computation nodes, we will create a design that contains these two interacting CFSMs. Open the user.pal palette to access the stars that we just created:

O

Scroll down with **middle mouse button** to get to **./user.pal** and select it, then click **OK** Go back to the **belt** window, and make a point by clicking the **left mouse button**. In **user.pal** window, put the cursor on top of **controller**, type:

c

This should create a copy of the **belt_control**.
In order to get the full screen view, type:

f

If you need to pan around the screen, the center of the screen will move to the cursor point if you type:

p

Pan until the **belt_control** icon is at the top center edge of the screen. Create a point well below it by clicking the left mouse button, and copy **timer** over from the **user.pal**

If you ever need to zoom in at an area, drag it a box around it with the **left mouse button**, then type:

z

You can now close **user.pal** by typing

CTL-D

on top of the window.

Never close any window with methods provided by the window manager, since this will terminate the Ptolemy application. Always use **CTL-D**, **Cancel**, **Close**, **OK** supplied by Ptolemy.

You may want to move the icons around. To do so, first you need to select the icon by typing:

s

on top of the icon. Alternative, you can first drag out an area for selection with the **left mouse button**. Unselect is done by

u

while general undo is done by

U

After an object is selected, you can move its outline by means of the **right mouse button**, and then type**:**

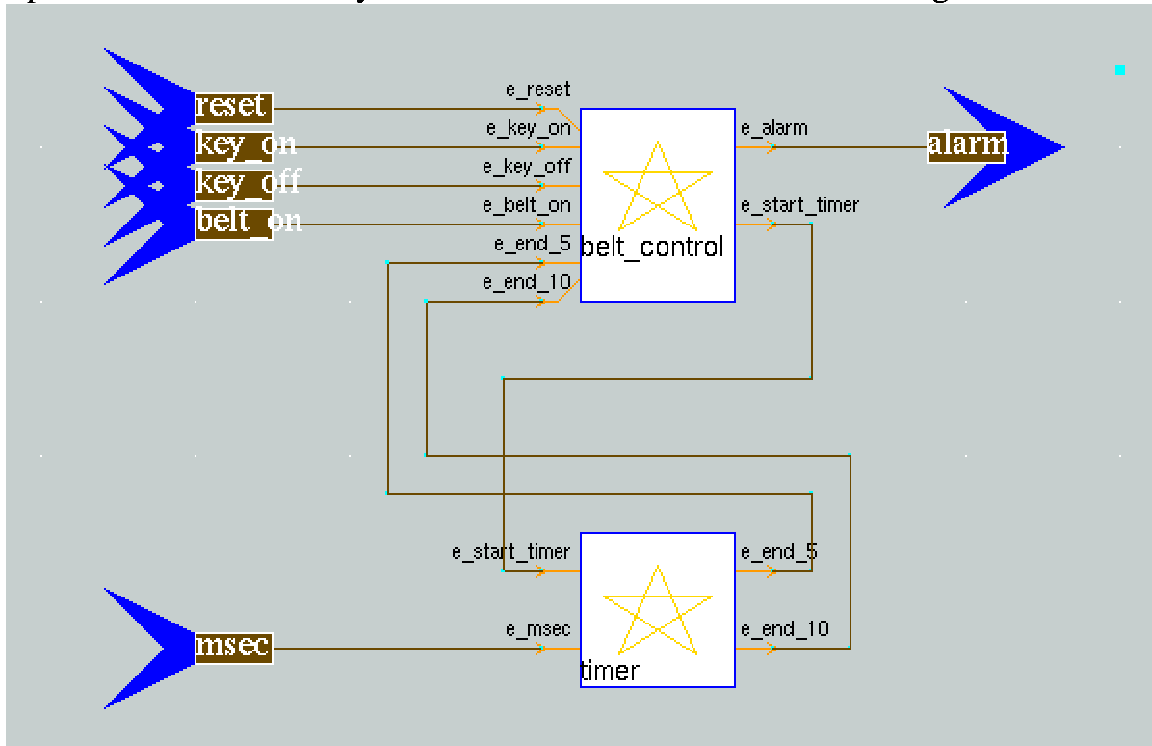
m

Remember to unselect with **u** an object after you have moved it.
To delete an object, select it first, and type

D

Now we need to create output and input ports. Open the palette called **system** and copy over the **input** and **output** icon. You will need 5 input icon and 1 output icon.

Space the icons so that you will be able to create the following screen:



To create a connection between the port and **e_reset** signal pin, put the cursor at the edge of the port, and make a point with **left mouse button**, click on that point again and drag with **left mouse button** till destination point, let go, and type:

c

Never use boxes to create connections. If they appear because of incorrect clicking, remove them with **delete**.
To create jags in the line, let go the **left mouse button** before dragging it in a different direction. Complete all the connection according to the figure.

To name the port, put the cursor on top of the top most input port, and type:

"reset"c

You can see what you typed in the Ptolemy command window. Continue till all ports are named. Edit the target of the galaxy by typing:

T

and choose **<parents>,** choose **OK**

Put the cursor on top of timer and edit its parameter by typing**:**

e

enter **5000** for both **end_5** and **end_10**. Click **OK**. If the simulation is too fast, just increase those values and run it again.

The ptolemy galaxy that represents the seat belt alarm controller is now complete. Create an icon in user.pal by typing:

@

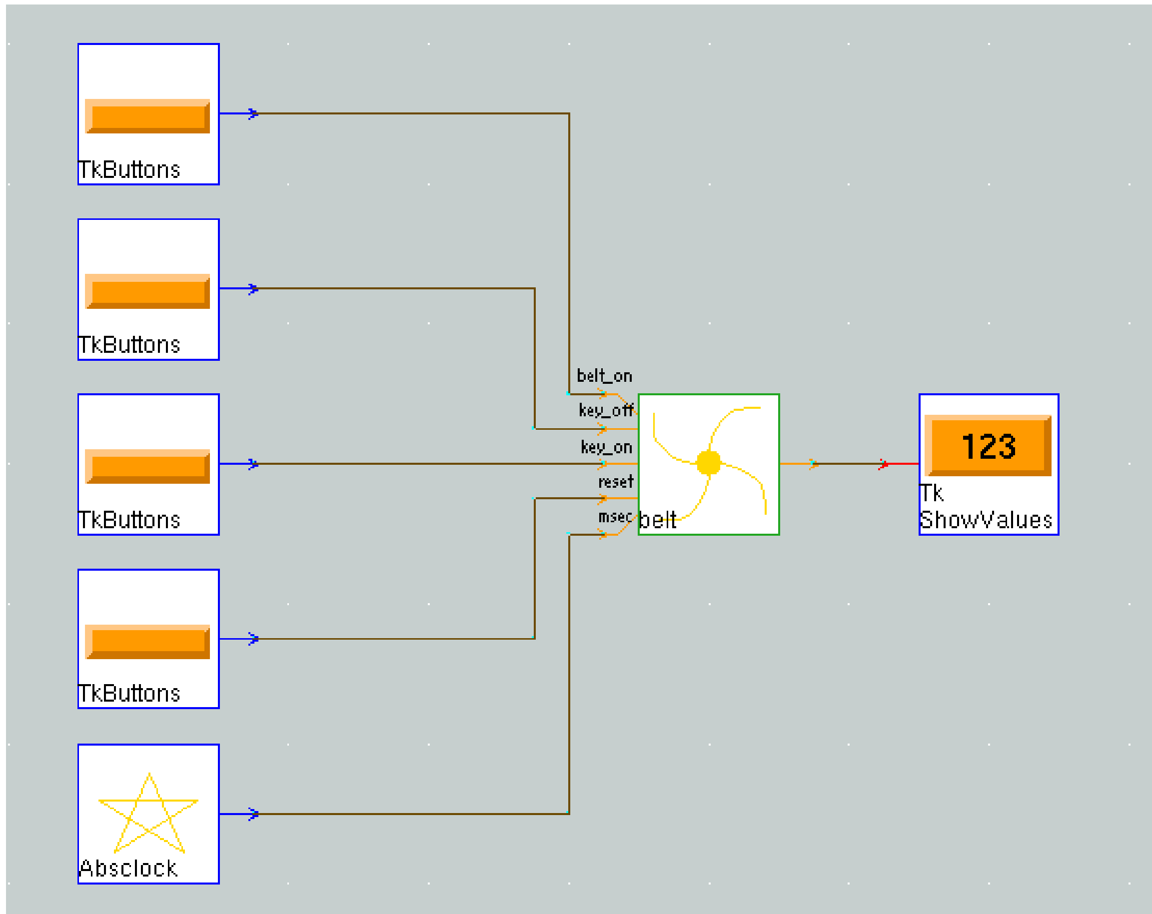
click on **OK** to put the galaxy in the **user.pal**

Next, we will build the testbed for the simulation. Open it by typing:

F

Choose the **test_belt** galaxy, then click **OK**

You will need to create the following schematic for testbed**:**



belt is found in your **user.pal**. In order to find **Absclock** put the cursor on top of **Polis** in **user.pal** and type:

i

to look inside. Likewise, look inside **Utility**, look inside **Sources**, and copy over **Absclock.**

TK Button and **TK Show Values** are in the palette called **de.pal** (accessed via O). Look inside **Signal Sources** and **Signal Sinks** respectively**.**

Position Cursor on an empty portion of the design and type:

e

to get the parameter list. Fill in

- **CPU 68hc11**,
- **SCHEDULER RoundRobin**
- **Firingfile fire.txt**
- **Overflowfile over.txt**

(both file names are relative to your home directory).
Add a new parameter with name **Clock_freq**, type **INT** and value **1.**
Add another new parameter with name **implem**, type **STRING** and value **HW**.

Edit the parameter of the input sources and output sink to give them a meaningful name. Modify the **identifiers** of **TkButtons** to **reset**, **key_on**, **key_off**, **belt_on**, and the label of **TKShowValue** to **alarm**

Edit the target of the top **test_belt** window using **T** .
Change any schedulers already selected to "NO" and the Resource Contention Scheduler to "YES"

3.6 Simulation of seat belt alarm controller

We are finally ready to run the simulation. Go to the **test_belt** window and type:

R

and enter **1000000000** for the time stamp to stop. Click on **key_on** and watch the **alarm** go briefly on then off again. Feel free to look inside any star, galaxy or universe (**i**) and change any parameters (**e**), For information on any star, galaxy or university, type

,

on top of the object. Note that the **Absclock** star acts as a real time clock and always emits a token every *x* msec, where *x* is defined by the **Interval** parameter. **Clock_freq** denotes the hardware and software clock frequency in MHz. If you click on the **debug** flag in the run control window, the number that you see has a period of (**interval***Clock_freq**) μsec. In this case with both **interval** and **Clock_freq** set to 1, the period is 1 μsec.

Change the implementation of **timer** to **HW** and that of **belt_control** to **SW**, with the **e** command over their icons in the **belt** galaxy. Save the **belt** and **test_belt** galaxies by typing

S

when the cursor is in their windows.
Now exit Ptolemy by typing **CTL-D** on all the windows, observe the **over.txt** file in your directory. There are no overflows even for **68hc11** at 1Mhz, so **over.txt** should be empty.

Now you can generate a mixed hardware/software implementation, as specified in Ptolemy, by typing:

make hwsw

The hardware partition (**timer**, interfaces and address decoders) is contained in the file **belt_part.blif**.
The software partition (**belt_control** and the OS) is in **belt_part_sg**. directory. Analyze both of them to understand how the interfacing works.

3.7 What to turn in:

Turn in both Esterel files and a screenshot of your Ptolemy work (the belt window, the test belt window, and the simulation)

NOTE ON SCREENSHOTS: There is a tutorial of how to do a screen shot in Linux on the TA's website.

4 The Elevator Controller

You have now gone through a full example of how to use the POLIS co-design environment. You will now use that knowledge to implement the following design:

"If the elevator is stationary and the floor requested is equal to the current floor, then the elevator remains idle.

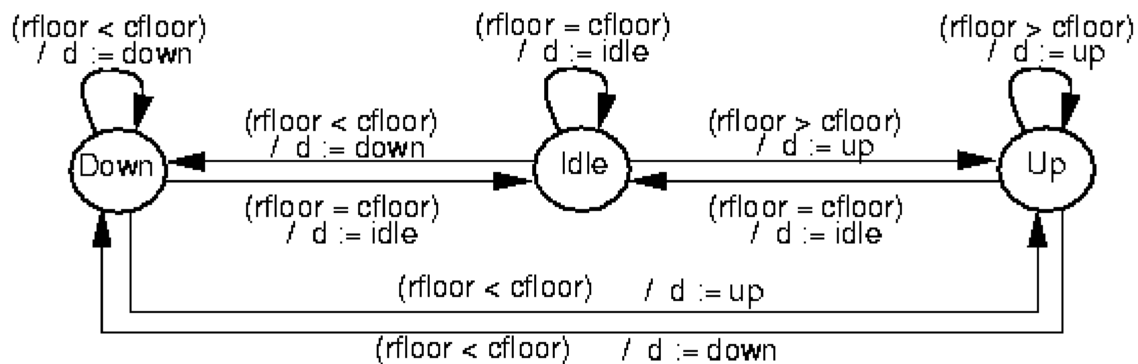
If the elevator is stationary and the floor requested is less than the current floor, then lower the elevator to the requested floor.

If the elevator is stationary and the floor requested is greater than the current floor, then raise the elevator to the requested floor."

(a)

```
loop
    if (rfloor = cfloor) then
      d := idle;
    elsif (rfloor < cfloor) then
      d := down;
    elsif  (rfloor > cfloor) then
      d := up;
    end if;
end loop;
```

(b)



(c)

Turn in all Esterel files and screenshots of the Ptolemy work, as well of an Explanation of how your design works.