INFO-F-410 Embedded Systems Design
Lustre

Gilles Geeraerts

1 Introduction

The aim of this exercise session it to get yourself familiar with the `Lustre` language. To this aim, we will rely on the tutorial which is distributed along with the Lustre V4 software suite. You should have received a copy of this document during the exercise session.

2 Setting up Lustre

Before you can use Lustre, you have to download and install the tools on your Linux account:

1. Download the tarball file from:

   ```
   http://www-verimag.imag.fr/DIST-TOOLS/SYNCHRONE/lustre-v4/
   distrib/linux64/lustre-v4-III-c-linux64.tgz
   ```

2. Uncompress it with:

   ```
   tar xvzf lustre-v4-III-c-linux64.tgz
   ```

3. For some obscure reasons, the permissions are not always set correctly in the `bin/` directory, so, do:

   ```
   cd lustre-v4-III-c-linux64/bin
   chmod u+x *
   cd ../..
   ```

4. Now, you must modify your `.bashrc` file to have access to Lustre. Edit this file (it is a hidden file located in your home directory), and append the following lines (replacing `aturing` by your actual login):

   ```
   export LUSTRE_INSTALL=/home/aturing/lustre-v4-III-c-linux64
   source $LUSTRE_INSTALL/setenv.sh
   ```

5. Restart your terminal, and type the `lustre` command, you should get an output looking like:

   ```
   ggeeraer@Ubuntu-VB:~$ lustre
   usage: lustre <lustre file> <main node> [<options>]
   recognized options :
   -default : display the default options
   ```

```
-o <file> : define the output file
-v : set verbose mode
...
```

3 Exercises

3.1 Exercise 1: the raising edge

We'll start with the example reviewed during the lecture: the raising edge, which is described in section 1.1 of the tutorial.

1. Create a `raising.lus` file describing the raising edge node.
2. Simulate the node using the `luciole` tool, as described in 1.1.2. Check that the nodes behaves as expected.
3. Translate the node into a C source file using the `lus2c` tool (note that the `lux` tool described in the tutorial is not available as it is part of another package). Observe the generated C code. It contains two files: `raising.h` and `raising.c`. The `raising.c` file defines a structure `raising_ctx` structure that contains all the internal data of the node (including its input `X` and output `Y`). In particular, the structure contains a **`void`** pointer **`void`** `* client_data` that can be used to output data to the external world. The important functions are:
   - `raising_ctx* raising_new_ctx(`**`void`**`* cdata)` that creates and initialises a new `raising_ctx`, returns a pointer to it and initialises its `client_data` field.
   - **`void`** `raising_I_X(raising_ctx* ctx, _boolean V)` that sets the input (field `X`) of the structure `ctx` to the Boolean value `V`.
   - **`void`** `raising_step(raising_ctx* ctx)` that executes one tick of the node and computes the values of the output. These values are not stored into the respective fields of the structure, but, rather, the corresponding output functions are called—in this case **`void`** `raising_O_Y(`**`void`**`* cdata, _boolean b)`.
   - **`void`** `raising_O_Y(`**`void`**`* cdata, _boolean b)`: this function is called by **`void`** `raising_step(raising_ctx* ctx)` once the output values are computed. You must define it (by default, it is defined as **`extern`**).
4. Write a simulator for the raising edge node by adding a `main.c` file. This should contain a **`while`**`(1)` loop that:
   - Reads a Boolean value from standard input
   - Writes this Boolean value in the input of the raising edge node using **`void`** `raising_I_X(raising_ctx* ctx, _boolean V)`.
   - Computes the output of the node with **`void`** `raising_step(raising_ctx* ctx)` and prints it.

   To have access to the input, you can declare a Boolean variable `b_out` in the `main()`, and let the **`void`**`* client_data` pointer of the context structure point to it. Then, modify the **`void`** `raising_O_Y(`**`void`**`* cdata, _boolean b)` function in such a way that the `b` value computed by the node is stored in the `b_out` variable.

5. Add a monitor checking that the output of the raising edge node is never true in two consecutive steps. Prove this property using the `lesar` tool.

3.2 Exercise 2: Numerical values

Lustre specifications can also manipulate numerical values. As an example, implement the `COUNTER` node described in section 1.2.1 of the tutorial and simulate it.

3.3 Exercise 3: Arrays

Implement the binary adder described in section 3.2 of the tutorial to learn how to use arrays in Lustre.

3.4 Exercise 4: More verification

Implement the train example of section 4.3 and use the `lesar` tool as described.

3.5 Exercise 5: Simulating a PID controller with Lustre

Let us first briefly recall the equations that allow to model a closed loop system with a PID controller. The variables are:

1. r, the reference value. The output should always be as close as possible to r.
2. u_t, the input of the environment that will be connected to the output of the controller.
3. y_t, the output of the system (at time t), i.e. the output of the environment.
4. e_t, the error (at time t), equal to $r - y_t$. It is the input of the controller.
5. w_t, the perturbation (at time t). It is an input to the environment.

The control law of a PID controller is (for $t \geq 2$):

$$u_t \quad = \quad P \times e_{t-1} + D \times (e_{t-1} - e_{t-2}) + I \times \left(\sum_{i=1}^{t-1} e_i \right)$$

Intuitively, the derivative term $D \times (e_{t-1} - e_{t-2})$ allows to keep the oscillations as low as possible, while the integrator term $I \times \left(\sum_{i=1}^{t-1} e_i \right)$ allows to enforce convergence to the reference value.

Let us now exploit the example of the course, a system to regulate the speed of a car. The equation that describes the car's speed is as follows (where y_t is the speed at time t) at all time $t \geq 1$:

$$y_t \quad = \quad 0,7 \times y_{t-1} + 0,5 \times u_t - w_t$$

- Build a lustre node with inputs u and w, output y that models this equation. Fix the initial speed to 20.
- Build a lsutre node for the PID controller (P, I, and D parameters can be constants).
- Combine these two nodes to build a model of the whole closed loop system.

Now, we have to determine proper values for parameters P, I and D. This is called ***PID Tuning***... To start, we fix the disturbance w to 0.

- Assign the value 0 to parameters I and D, and observe, in the `luciole` simulator, the behaviour of the system for different values of P (by letting $r = 50$, and initial speed $= 20$). Try at least the following values for P: -3, 0.5, 2 and 5.
- Determine a value for P that seems adequate, i.e., yields a fair accuracy (less than 20% error) without letting the system diverge.
- Then, adapt D to lower oscillations, and further adapt I to let the system converge. Remind that $D \leq 0$ and $I \geq 0$.
- How does the system react to disturbances ? Try $w = 5$ and $w = -5$.

An empirical method to tune P, I and D has been introduced by ZIEGLER and NICHOLS in 1942. Apply it and compare to your previous findings:

- First let $I = D = 0$.
- Then, gradually increase P up to the point where the system oscillates and diverges. This value of P is called *ultimate gain*, and is denoted P_u. The oscillation period is called *ultimate period* and is denoted T_u.
- Then, let $P = 0.6 \times P_u$, $I = 0.5 \times T_u$ and $D = 0.125 \times T_u$.
- How does the system react to disturbances ? Try $w = 5$ and $w = -5$.

To finish, modify the controller to take into account saturation of the u_t variable. The possible values for u_t are in $[0, 45]$. Observe how the response of the system is changed by this.