COMPSYS 701 – Lab 4

Introduction

Welcome to the fourth lab of COMPSYS 701. The purpose of this lab is to introduce you to the SystemJ programming language. To begin with, we'll look at a SystemJ example, run it on the computer, and then run it on the board. We'll then run the control program for a real world mechatronics device using an external testing board.

Additional materials:

- SystemJ Programming Manual

Prerequisites:

- Quartus II installed
- Cygwin installed
- Java installed

Part One – Introducing SystemJ

SystemJ is a programming language that uses the Globally Asynchronous, Locally Synchronous Model of Computation. It is able to be compiled to Java, and we'll demonstrate that here. First things first, a quick tiki-tour.

Below in Figure 1 is a sort of "Hello World" program for Synchronous-Reactive programs, introduced in [1] by Berry. The system has three input signals, A, B, R, and two outputs, O and clearO.

This program waits for A and B signals, and then emits O. The system will then wait for a reset signal (after which it emits clearO). This behavior is represented in the FSM in Figure 2.

For more information, consult Chapter 2 of the SystemJ Programming Manual. For now though, let's run this program on the desktop.

```
ABROCD(
    input signal A, B, R;
    output signal O;
    output signal clearO;
)->{
    while(true){
        // Set up the preemptive abort to reset when R is receieved
        abort(R){
            // Await for A and B synchronously.
            {await (A); System.out.println("Got signal A");} || {await (B); System.out.println("Got signal B");}
            // Emit O, now that A and B have been receieved.
            emit O;
            System.out.println("Emitted signal O");
            // Wait indefinitely
            while(true){
                pause;
            }
        }
        emit clearO;
        System.out.println("Emitted signal clearO");
        pause;
    }
    pause;
}
```

Figure 1: The Classic ABRO example in SystemJ

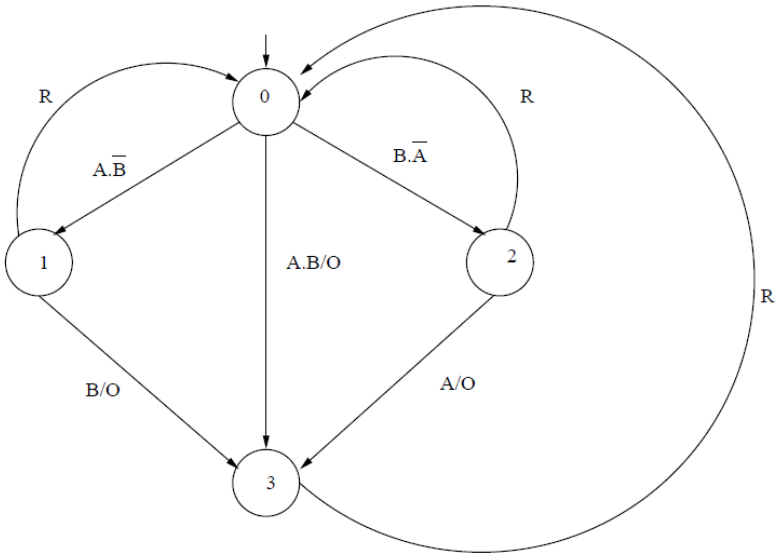

Figure 2: ABRO as an FSM from [2]

First, download and un-zip *SystemJ-Tools.zip* from CECIL. Open up a new Cygwin terminal, and navigate to *SystemJ-Tools\sysj-examples\run-on-desktop\ABRO*. Execute runABRO with:

```
./runABRO
```

A control panel will appear! Try clicking the buttons and observing the outputs in the Cygwin terminal.

SystemJ is composed of clock domains, within which programs execute in a synchronous fashion (you'll learn, or will have learnt, about this in COMPSYS 723). Clock domains run in parallel in an asynchronous fashion. This is the underlying notion of the GALS model of computation.

At this stage you don't need to know too much about the specifics of the language. Let's try another example.

The next program simulates a pump controller that monitors both water and methane levels in different parts of a workplace in a mine. When the system detects the water level above a certain limit it emits an output signal, which triggers a water pump, in order to lower the water level in the workplace. At the same time, the system also continuously checks the methane level and triggers an alarm when the gas level rises higher than the allowable limit.

The SystemJ system consists of two clock-domains; sensor and controller that captures external conditions and controls the alarm/pump respectively. Assume that the environment for this example provides a valued signal with an integer number indicating the water level, and pure signals for the status of methane level. In order to simulate the example program, navigate to SystemJ-Tools\sysj-examples\run-on-desktop\PumpController and execute:

```
./runPumpController
```

Open up pumpcontroller.sysj and have a look at the code. Play around, and see if you can understand what's going on.

Part Two – SystemJ can run (almost) anywhere!

You may have noticed in the source code that SystemJ features some very Java-like statements, or that there were Java files in the src directories. This is because SystemJ is built around Java. Furthermore, SystemJ can be compiled to Java code, and that is what was running in Part One. SystemJ can be run anywhere where Java can be run, which means desktop and embedded platforms that have Java Virtual Machines. The power in SystemJ is that the same code can be executed on any platform without needing to rewrite! So let's see what happens when we run SystemJ code on our JOP! Make sure you have *CS701JOP* (from CECIL, which you used in Lab 1) handy.

Let's make ABRO work on the FPGA board, where the A, B, and R signals can be switches SW0, SW1, and SW2 on the board.

Copy `ABRO.sysj` and `ABRO.xml` from *SystemJ-Tools\sysj-examples* into *SystemJ-Tools\compiler-jop\bin*. Navigate to this directory in Cygwin. Invoke the compiler by running:

```
./compile ABRO.sysj
```

And then run:

```
./makemain ABRO.xml
```

You should notice that a couple of files have been generated: ABRO.java in the ABRO directory and ABROCD.java in the current directory. Open both these files, and add, at the top, the line:

```
package ABRO;
```

Move ABROCD.java into the ABRO directory, and then move the entire ABRO folder to:

```
java\target\src\sysj
```

Now we're set to download this to the board! cd to the root of *CS701JOP* and then execute

```
make -e P1=sysj P2=ABRO P3=ABRO
```

Once it's all downloaded, you should be able to use the switches. Observe the console output and see if this matches up with the behaviour when the program was running on your PC.

Now try running the pump controller example on the JOP. You'll need to do the same steps as before, including specifying a package name. Try to compare ABRO.xml and pumpcontroller.xml to see how input and output signals are mapped to switches and LEDs on the DE2-115 board.

Instead of re-synthesising the hardware, you can use:

```
make japp -e P1=***** P2=******* P3=*******
```

This will compile the java code, reprogram the board, and download the code.

Part Three – The Work Piece Loader

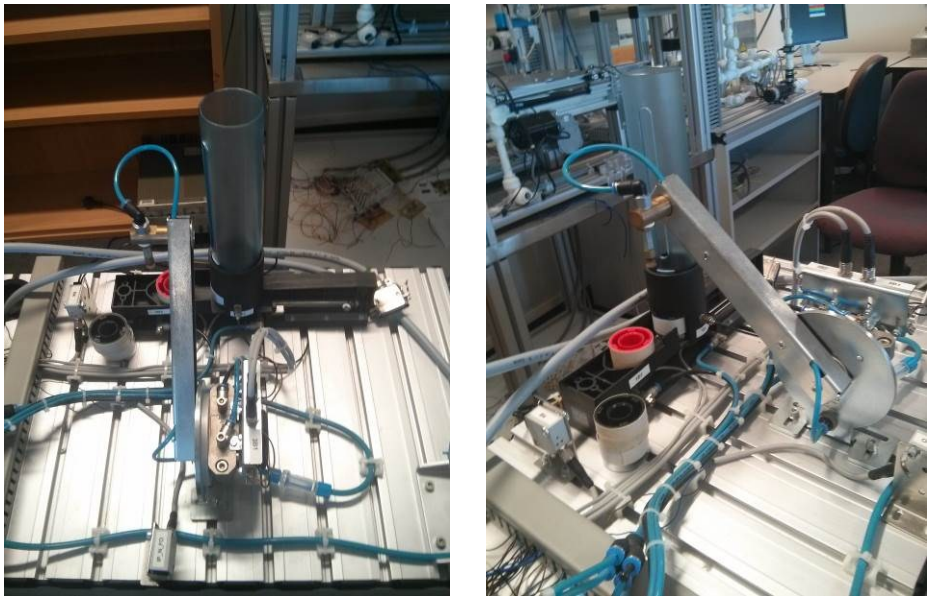

Figure 3: The Loader

This loading station is one of several mechatronics devices in the Industrial Informatics lab. It loads a piece from the magazine onto the conveyor. You can check out the demo video on CECIL if you want to see it in action!

Sequence for operation:

- Initially, the arm is moved to a neutral position
- The controller waits until it detects that there is a work piece in the magazine
- Once it detects a piece, the magazine slider extends, pushing the work piece out
- The magazine slider then retracts
- The loading arm retracts, moving towards the work piece
- Once it's retracted, the vacuum is turned on, picking up the work piece
- The loading arm is then extended, moving towards the conveyor
- Once over the conveyor, the vacuum is turned off, and the work piece is released
- The arm is moved back to the neutral position, and the sequence repeats

The system has five outputs and six inputs.

Input Signal	Role	Output Signal	Role
magEmpty	Indicates if the magazine is empty	extend	Pushes the arm forward towards the conveyor
retracted	Indicates the loading arm is pulled back as far as possible	retract	Pulls arm back towards the magazine
extended	Indicates the loading arm is pushed as far forward as possible	vacOn	Turns the vacuum on
magExtended	Indicates magazine slider has been pushed forward	vacOff	Turns the vacuum off
magRetracted	Indicates magazine slider has been retracted	magExtend	Pushes forward the magazine slider (a low signal retracts it)
icGripped	Indicates that a work piece has been gripped by the vacuum		

Now have a look at ICLoaderController.sysj in *SystemJ-Tools\sysj-examples\IC-Control-Final-JOP*. Go through the code and try to understand how the controller operates.

For the next part of the lab we will run this controller code. Unfortunately, we can't let all of you to run this next example on the mechatronics device at the same time. So you'll have to play the role of the machine for now.

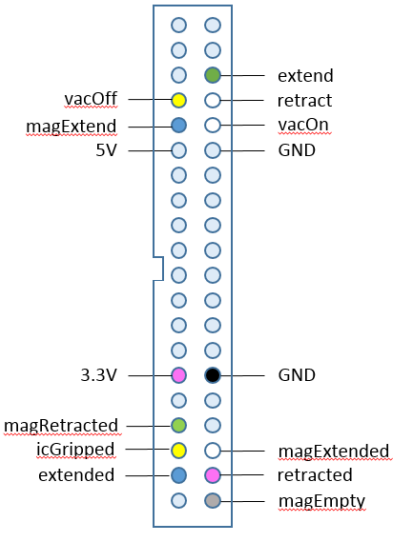

Figure 5: The GPIO expansion port on the DE2-115

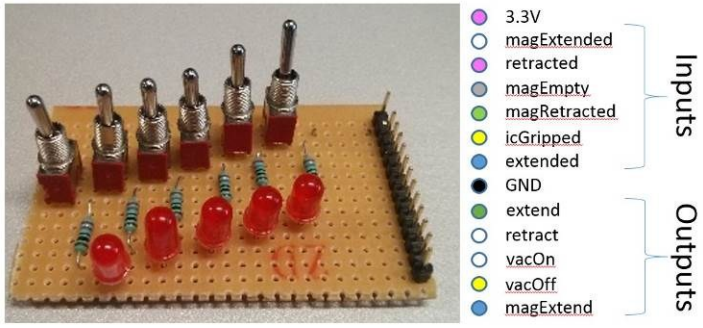

Figure 4: Prototype Board (yours might be a little different) with Suggested Cable Order

Hook up the wires to connect the prototype board and GPIO port. Make sure you put the 3.3V and GND in the correct place. The other locations are simply suggested; you don't even have to follow the same colour scheme. Just keep track of which signals are connected to which pins.

Compile ICLoaderController.sysj and makemain with ICmodified.xml in the usual way, and program it on the board.

See if you can work your way through the control code sequence, by recreating the sequence:

Sequence for operation, with signals (Start by having the switch corresponding to *magEmpty* set to high. Also set *extended* to high):

- Initially, the arm is moved to a neutral position
- The controller waits until it detects that there is a work piece in the magazine *(magEmpty becomes low)*
- Once it detects a piece, the magazine slider extends, pushing the work piece out *(magExtend is made high until magExtended goes high)*
- The magazine slider then retracts *(magExtend is made low, the controller waits until magRetracted goes high)*
- The loading arm retracts (magRetracted, moving towards the work piece *(retract is made high)*
- Once it's retracted *(retracted goes high)*, the vacuum is turned on *(vacOn is made high)*, picking up the work piece *(icGripped goes high)*
- The loading arm is then extended *(extend is made high)*, moving towards the conveyor
- Once over the conveyor *(extended becomes high)*, the vacuum is turned off *(vacOn goes low, vacOff goes high)*, and the work piece is released *(icGripped goes low)*
- The arm is moved back to the neutral position, and the sequence repeats

As you progress through the sequence, you should see the LEDs change on and off. Console output should also help you find your way. This board is now yours to use for the remainder of the project, and may be useful in debugging your processor.

Part IV – The Future

We've seen this control program execute on the JOP alone. However, SystemJ can be partitioned into control flow and data computation parts.

For this last part of the lab, we will compile the same control code for a TP-JOP target.

Copy ICLoaderController.sysj and ICmodified.xml from *SystemJ-Tools\sysj-examples\IC-Control-Final-TP-JOP* to *SystemJ-Tools\compiler-tpjop\bin*. Run:

```
./compile-s ICLoaderController.sysj ICmodified.xml
```

A new directory is created, called something like ICLoaderControlleroptimizedmp. Have a look inside. There should be a set of java files, and an .asm file.

The ReCOP you are developing will work in tandem with the JOP to execute this generated SystemJ file. The .asm will need to be assembled and loaded into the program memory of your processor, and the remaining java will have to be compiled and downloaded to the board in the usual way.

IMPORTANT -- Running SystemJ programs on TP-JOP

In order to compile and run SystemJ programs on TP-JOP you need to modify *Makefile*, which is in **CS701JOP.zip**. Open the file, remove the path set to classpath paramter for the variable TARGET_JFLAGS (it is **line 223** if you did not modify the file). So after the modification, TARGET_JFLAG should be set to:

```
TARGET_JFLAGS=-d $(TARGET)/dist/classes -sourcepath $(TARGET_SOURCE) -bootclasspath "" -extdirs "" -classpath "" -source 1.5 -target 1.5 -encoding Latin1
```

Additionally, **remove line 426** in the Makefile, which is "`(cd java/target/dist/classes ; jar -xvf ../../../../System*.jar)`".

References

[1] G. Berry, "The Esterel v5 Language Primer Version v5.91," 2000. [Online]. Available: http://francois.touchard.perso.esil.univmed.fr/3/esterel/primer.pdf.

[2] R. Connolly, A. Malik, P. S. Roop, Z. Salcic and J. Stott, *SystemJ Programming Manual*, University of Auckland, 2010.