COMP3241/9245: Real-time Systems

Programming in Esterel — Lab 2

last changed: June 24, 2006

This lab consists of a series of programming exercises designed to increase familiarity and understanding of the Esterel programming language and the synchronous paradigm. The example considered is a simplified backhoe loader.

The sections marked optional are more difficult and cover additional Esterel features. Students who find the other exercises easy (particularly those enrolled in COMP9245) may find these extra challenges stimulating.

Task 1: Understanding the interface specification

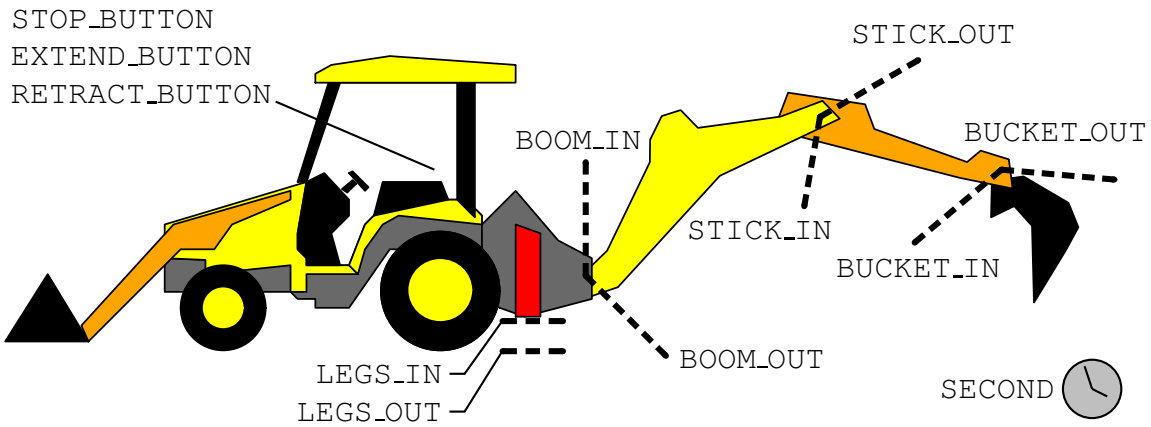

Figure 1: Backhoe Loader input signals

Backhoe loaders are essentially tractors fitted with a loader unit at front, and a backhoe at rear [1]. The backhoe consists of three segments: the *boom*, *stick* and *bucket*. The driver is able to swivel his seat and coordinate the segments using joystick controls.

In this lab we will program a prototype sequencing controller to automate some aspects of backhoe operation. The controller will receive input from the 12 signals shown in Figure 1. Each segment is fitted with two sensors, one triggered when the segment reaches its minimum (`*_IN`) range, the other when at its maximum (`*_OUT`). The sensors only produce output when the associated segment is being driven. Similar sensors exist for a pair of stabilising legs. The cabin is fitted with three buttons, *stop*, *extend*, and *retract*. A signal is associated with each. The controller also receives a timer input, `SECOND`, at regular intervals.

The controller is fitted with the 15 output signals summarised in Figure 2. The segments are fitted with hydraulic pistons that either push or pull depending on the position of an internal valve. The controller can position the valve by emitting either a `*_PUSH` or `*_PULL`

BOOM_PUSH	BOOM_PULL	BOOM_DRIVE
STICK_PUSH	STICK_PULL	STICK_DRIVE
BUCKET_PUSH	BUCKET_PULL	BUCKET_DRIVE
LEGS_EXTEND	LEGS_RETRACT	LEGS_STOP
ALARM_LAMP(bool)	DONE_LAMP(bool)	CANCEL_LAMP(bool)

Figure 2: Backhoe Loader output signals

signal (the effect of simultaneous emission is undefined). Segments move clockwise when pushed by the hydraulics, and counter-clockwise when pulled. Each segment has a closed-loop controller that, by default, holds it in fixed position, or, when a `*_DRIVE` signal is sustained, moves it at constant speed. Motion ceases as soon as a signal becomes absent. The legs operate differently. The `LEGS_EXTEND` and `LEGS_RETRACT` signals start the legs moving until they are out or in, respectively. This movement may be cancelled at any time with the `LEGS_STOP` signal. Three lamps, *alarm*, *done*, and *cancel*, in the cabin are switched on and off by logical (true/false) signals.

Task 2: Running the backhoe simulation

A simulator and Esterel interface may be found at:
`/home/esterel/examples/backhoe`

The contents should be copied into your home directory. You will need write permissions to the directory (`chmod u+w .`). Typing `make` in the new directory will compile the `backhoe.strl` program, launch `xes` in animation mode, and display the simulator window. The Makefile essentially executes the Esterel and C compilers as per the previous lab, with some additional details for interfacing with the simulation tools.

The upper-half of the simulation window contains the Backhoe Loader and cabin interface (buttons and indicator lamps). The lower-right corner contains buttons for controlling the simulation:

continue/pause	Simulation time is frozen when paused (red).
disconnect/connect	Inputs signals are sent to `xes` for code animation when connected (red).
reset	Resets the simulation and controlling program.
quit	Terminates the simulation.

Additional simulation options are at lower-left. The simulation is paused when first started.

Task 3: Extending the stick, then the bucket

Implement a simple controller (in `backhoe.strl`) that first extends the *stick* segment, and then the *bucket* segment. The segments must not be in motion simultaneously. The *done* lamp should be illuminated when the movements are complete.

Task 4: Retracting the stick and bucket together

Modify the previous program so that (only) after the *stick* and *bucket* are fully extended, pushing the *retract* button causes both to be retracted simultaneously. The *done* lamp

should *not* be illuminated while the backhoe is moving, but should come on again as soon as the motion is complete.

Task 5: Flashing light during movement

Modify the previous program so that the *alarm* light blinks on and off with a period of two seconds whenever the backhoe is moving.

Optionally: Break the code into two or three separate modules (within the one file) in order to make the program more adaptable. The lamp to be flashed should be passed as a parameter.

Optionally: Use `suspend` and `pre` (which may necessitate initializing signal values) to implement the lamp blinking.

Task 6: Leg Control

Create a copy of the previous file, and start a new program. Allow the three buttons to control leg movement directly, i.e. pressing *extend* should start lowering the legs, *stop* should stop them, and *retract* should cause them to lift. The controller should illuminate the *done* lamp and halt if/when the legs are fully extended.

Optionally: You can assume that the legs are always raised when the controller starts. It should now be possible for the operator to raise, stop and lower the legs any number of times. To avoid strain on the equipment, it is desirable to ensure that the legs are fully lowered within 10 seconds of the extend button having been pressed for the first time.

Add a watchdog timer in parallel to the leg control loop. If, after 10 seconds have elapsed, the legs are not fully extended the alarm lamp should be illuminated and the legs should be completely raised, they should then be held for a further 5 seconds before the alarm lamp is switched off and normal operation resumes — all operator input is to be ignored during the alarm period. The watchdog timer can be reset each time the legs are fully extended or retracted.

Task 7: Dig your own hole

Enhance the previous program. Pressing *extend* once the legs have been lowered should result in a series of movements:

1. Raise the *stick* and *bucket*.
2. Then, completely lower the *boom* (into the ground).
3. Then, retract both the *stick* and *bucket*.
4. Wait for operator to press *retract*.
5. Bring the *boom* back into the starting position.

During the first two steps pushing the *retract* button should illuminate the *cancel* lamp and retract all three segments (simultaneously) to the starting position. The other buttons may

be ignored during the retraction sequence. The *retract* button is to be ignored in steps 3 and 5.

Pressing the *stop* button at any time should halt all movement. The interrupted motion should resume when the button is pressed a second time. The *alarm* lamp should be illuminated during such interruptions, and the other buttons may be ignored.

The *done* lamp should only be illuminated when the backhoe is stationary and the controller is waiting for operator input. It should not, however, be illuminated when in the leg movement mode.

After the backhoe returns to the starting position, the buttons should control the legs again, as per the previous task. At this stage all of the lamps should be dimmed. The digging motion may be resumed when the legs are fully lowered, i.e. pressing *extend* once would illuminate the *done* lamp and pressing it again would start the movement sequence.

Optionally: Flash the *cancel* lamp as the three segments are being retracted.

Optionally: Divide the program into modules, reusing any common behaviours by giving signals as parameters.

Optionally: The controller is damaging the valve seals by applying the throttle too soon after the direction has been changed. Have the controller pause for at least three seconds between changing the valve direction and applying the drive signal so as to increase the longevity of the prototype equipment. **warning:** Minimising the delay is quite challenging (i.e. time consuming) if one considers interactions with the stopping feature, and 'expectant' direction changes.

Reflection: Consider the issues/work involved if implementing the same behaviour using an explicit Finite State Machine or a C program.

References

[1] M. Brain and T. Harris. How caterpillar backhoe loaders work. http://www.howstuffworks.com/backhoe-loader.html, accessed: Oct. 2005.