Practical session: Prelude and SchedMCore

Julien Forget[*]

January 17, 2014

The goal of this practical session is to illustrate how real-time constraints can be handled in a synchronous language context.

1 Basics of Prelude

The first exercise is based on the `sampling_loop` example of Prelude distribution. Its structure is recalled in Figure 1.

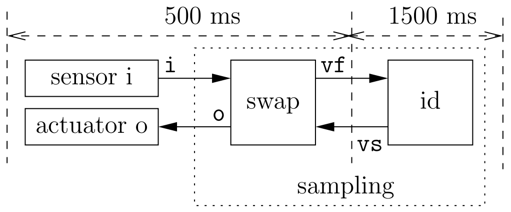

Figure 1: Sampling loop

1.1 Build with Prelude compiler

- Copy the `sampling` example[1] to your home directory. Compile the file with `preludec -node sampling sampling.plu`.

The compiler creates two files in a directory named `sampling_c`: `sampling.h` and `sampling.c`. The latter includes a file `sampling_includes.h`, which must be provided by the user and defines the C prototypes for the imported nodes, the sensors, and the actuators. Imported nodes have the same name in Prelude and C. The functions for sensors are prefixed with `input_` in C, and the functions for actuators are prefixed with `output_`.

Several options are available to print the results of the static analyses. Try the following:

[*]Based on a practical session by Wolfgang Puffitsch, Julien Forget, Eric Noulard and Claire Pagetti
[1]Located in `/usr/local/prelude-1.3/Examples/sampling_loop`.

- `preludec -node sampling -print_types sampling.plu`.
- Same with `-print_clocks`.

1.2 Simulation with SchedMCore

The actual implementation of the imported nodes can be found in the file `sampling_includes.h`. This file and the file `sampling.c` have to be compiled and linked together in a shared library. As this part is tedious to perform manually and has little educational value, it can be left to a build tool like `cmake`.

The following steps are necessary to set up building with `cmake`:

- Create a build directory and go there: `mkdir build; cd build`
- Run `cmake` to create the actual Makefile:
```
cmake -DPRELUDE_PATH_HINT=<path_to_prelude> \
<path_to_prelude_lab>/sampling_loop
```
- Build a library that is used to execute the example: `make`. In case you are interested in the precise commands executed by make, you can alternatively use `make VERBOSE=1`.

The build created the shared library `libsampling-endoced.so`, which implements the model and contains the necessary information for SchedMCore. After these steps, simply execute `make` to re-build the library.

The library can be executed with `lsmc_run-nort -l ./libsampling-encoded.so`. The duration of a tick can be controlled with the option `-b <N>`. Set the duration to one milisecond (`-b 1000`) as the default is far too slow for our example. By default, SchedMCore uses EDF scheduling, but for simulation any scheduler fits.

Exercise 1 *Try some simple modifications of the example:*

- *Change the input period to 300, execute, then retry with a period of 200;*
- *Fall back to the initial period of 500 and modify your program so that it prints 5 times the same value instead of 3 times;*
- *Play with the clock calculus, for instance, see what happens when the factors of `*^` and `/^` are different;*
- *Change the sensor so that it counts from 9 to 0 and again.*

Exercise 2 Prelude *can automatically add tracing information to the generated code. Modify the file `CMakeLists.txt` in the folder `sampling_loop`. Add `TRACING values` (if you want to see the values exchanged) or `TRACING instances` (if you want to see additionally which task instance produced a value) after `NOENCODING`. Re-build the example and execute it.*

2 Modeling with PRELUDE

Exercise 3 (Simplified flight control) *We consider the simplified Flight Control System of Fig. 2. This system controls the attitude, the trajectory and the speed of an airplane. It consists of 7 tasks which execute repeatedly at a periodic rate. The fastest sub-system executes at 10 ms, it acquires the state of the system (angles, position, acceleration) and computes the feedback law of the system. The order is then sent to the flight control surfaces. The intermediate sub-system is the piloting loop, it executes at 40 ms and determines the angle to apply. The slowest sub-system is the navigation loop, it executes at 120 ms and determines the acceleration to apply. The required position of the airplane is acquired at the slow rate.*

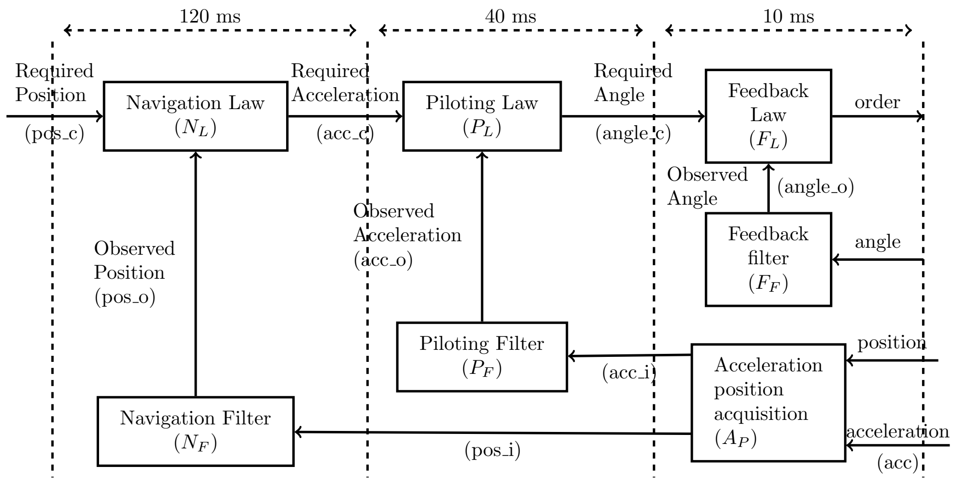

Figure 2: Flight control system

In directory `fcs`, you find a project that contains a cmake file and C files to implement the (imported) nodes of the case study and simulate the sensors/actuators.

Table 1: Task parameters

Task	N_L	N_F	P_L	P_F	F_L	F_F	A_P
Period	120	120	40	40	10	10	10
WCET	6	5	4	4	1	1	1

- *Edit the file `fcs.plu` to implement the flight control system as shown in Figure 2. Use the task parameters shown in Table 1; assume that the WCET of all sensors and actuators is 1 ms. Assume moreover that the actuator `order` has a deadline of 7 ms. In this first step, use direct communication between the nodes, i.e., do not use `fby`.*

- PRELUDE *modifies the deadlines of tasks to enforce precedences. Use the option* `-print_deadlines` *to display the deadlines for the encoded task set. Is the system schedulable ?*
- *The* `fby` *operator can be used to break direct dependencies. Use this operator to delay the flows from N_L to P_L and from P_L to F_L. Use an initial value of 1. Again, use* `-print_deadlines` *to examine the deadlines for the task set with encoded precedences.*
- *Examine the output of the* PRELUDE *compiler when using the option* `-print_protocols`*. How large are the buffers between the different tasks?*
- *Execute the task set with the* SCHEDMCORE *runner.* `lsmc_run-nort -l ./libfcs.so -b 10000 -c2` *(we will see later why we need option -c2). Note that if you want to reuse the same* `build` *folder, you must remove the file* `CMakeCache.txt`*. What output is generated by the program?*

3 Schedulability analysis with converter

The SCHEDMCORE converter can be used to check the schedulability of a task set generated by PRELUDE (or encoded in other formats). The task set is translated into a C model and the execution of this model tells us whether the system is schedulable or not.

The `lsmc_converter` supports several policies, which can be specified with the option `-p` (including the classic GEDF, GLLF, LLREF, along with some more specific policies).

Exercise 4 *Perform a schedulability analysis of the flight control system for EDF scheduling on a uniprocessor with the C model.*

- *Generate the model:* `lsmc_converter -c 1 -m c -p GEDF -l ./libfcs.so`
- *Compile the model:* `gcc -o model libfcs.so_GEDF.c`
- *Run the model:* `./model`*. Is the task set schedulable?*
- *If you want some details on the execution, compile with the option* `gcc -o model -DDEBUG libfcs.so_GEDF.c` *and run again;*
- *Try again, with two cores (*`-c 2`*).*