Programming a Digital Watch In Esterel (Version v5_92)

Gérard Berry

Centre de Mathématiques Appliquées
Ecole des Mines de Paris
and INRIA
BP 93
06902 Sophia-Antipolis
Gerard.Berry@inria.fr

http://www.esterel.org

October 20, 2000

1 Introduction

We study in details how to program a digital wristwatch in Esterel [1,2], how to simulate its behavior using the Esterel v5_92 compiler and simulator, and finally how to execute the C code generated by the compiler in a fullscreen simulation of the wristwatch The full code of the wristwatch example is delivered with the Esterel v5_92 distribution tape.

A digital watch is a typical example of a *reactive system*. Such a system reacts to input signals coming from its environment by sending itself signals to this environment. The Esterel language is especially tailored for programming reactive systems. The watch example is particularly interesting because of its non-trivial modularity and its relative complexity. Numerous commands are folded into a few buttons using command modes and numerous informations are shown on the displays using display modes. Moreover, there is a full range of digital watches, from simple timekeepers to sophisticated devices that include stopwatches, alarms, backtimers, or even more complex features. The one we consider here has a timekeeper, a stopwatch, and an alarm[1]. Its display and command modes are pictured in Figures 1–6 below. Once we have programmed it, we show how to program several variants, to illustrate how easy it is to modify the Esterel code.

Describing correctly the behavior of a watch is by no way an easy task. In the next two sections, we give an informal but precise description. We shall not try to give a formal description different from our Esterel program. This program is actually quite close to what one should call a "specification", being made of rather high-level code. But our code has two advantages over a classical specification: it is executable and analyzable.

- We can *simulate* the wristwatch, which makes debugging easy[2].
- We can *compile* the parallel Esterel code into a small and fast single-threaded C code that can be embedded in an actual device.

[1] It was inspired by the author's CASIO watch, but has some improvements described later on
[2] One should have some doubts about specifications that cannot be executed nor simulated.

- We can use formal verification tools such as the Esterel verifier `xeve` to *verify* properties of the wristwatch (not done in this paper).

In the Esterel distribution, we simulate the watch using the `xes` source code debugger, we embed the code into a real-time Tcl-Tk based application (with the caveat that standard operating systems does not know very much about time interrupts), and we run the Tcl-Tk application with parallel source code animation by `xes`.

To handle a problem of this kind, we can take two different attitudes. We can try to write a compact and clever program, or we can try to build reusable standard components and use them to construct the final program. It is now well-understood that the second approach is better, even if the code is longer and may look heavier. Our Esterel program is 16 pages long, half for declarations and half for executable code. It uses as much as 8 submodules and 28 internal signals for inter-module communication. A compact program could be 4 pages long and use much less internal signals, say 5 to 10. However, we do not pay much for the additional complexity: *only the compiling time is increased*. Once optimized, the generated code is basically the same as for a compact program. This essential advantage of Esterel is due to its synchronous nature and to the way the compiler translates a parallel Esterel program into either a Boolean circuit or an equivalent sequential automaton. Inter-process communication can be resolved *at compile and optimization time*, without run-time overhead. Therefore we can use as many modules and as many local signals as needed for elegance, with no loss of efficiency. Such a phenomenon doesn't exist in classical parallel language, in which increasing the number of processes and of inter-process communications always increases the execution overhead.

We put a special emphasis on an Esterel programming style. In our opinion, the main difficulty when programming in Esterel is to build a neat architecture of which the code should follow in a quite straightforward way. We build reusable components that communicate by internal signals, and we make a systematic use of *signal broadcasting*, which is the primary Exterel communication primitive. Broadcasting has obvious advantages: a receiver doesn't need to know where a signal comes from, an emitter doesn't need to know who is listening to the signal it emits. Broadcasting is done entirely at compile time, with no loss of efficiency. To handle the wristwatch's beeper we use the Esterel *combined signal* facility: a signal can have several simultaneous emitters.

Altogether, we hope to convince the reader that one can write elegant Esterel program generating very good object code.

After giving an informal specification of the wristwatch device in Section 2, we discuss our program architecture in Section 3. We introduce five submodules: a regular watch, a stopwatch, an alarm, a button interpreter, and a display handler. The first three modules are built so as to be easily reusable in different devices. The other modules are easy to modify if needed. We carefully discuss the interfaces and the global behavior. The module interfaces are presented in Section 5, and the module bodies are presented in Section 7

We then study each module individually. We discuss the quality of the generated code. We finally show that modifying our wristwatch is easy. We discuss several possible modifications and their impact on the behavior and on the generated code.

In annex, we present the Esterel programs and a simulation session under Esterel v5_92. The C auxiliary programs for actual simulation and execution of the generated code are given in the Esterel v5_92 distribution tape. They are not listed here.

There is no bibliography in this paper. We encourage the reader to visit *http://www.esterel.org* where all the information about Esterel is available.

2 Rough Description

We start with an informal description of the features of our wristwatch, meant to be understandable by anybody who has used such a device. In the next section, we shall give a more precise description, including details of the displays, beepers, and user commands. Figures 1–6 should help the reader in understanding the intended behavior.

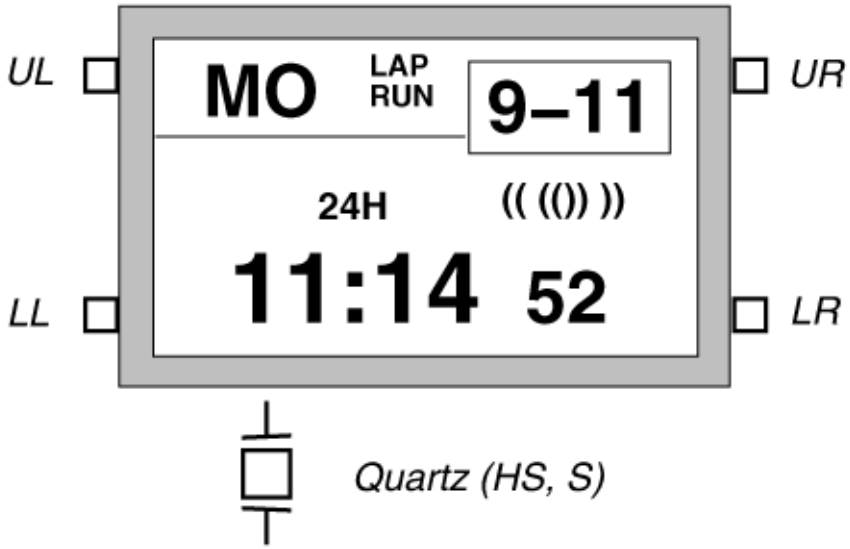

Figure 1: Wristwatch Commands and Displays

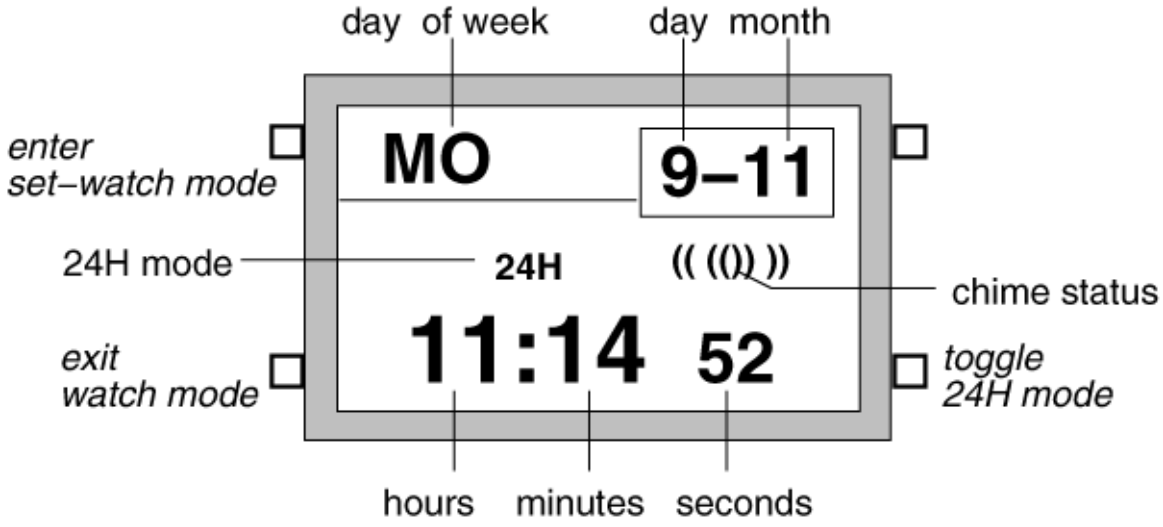

Figure 2: Watch mode

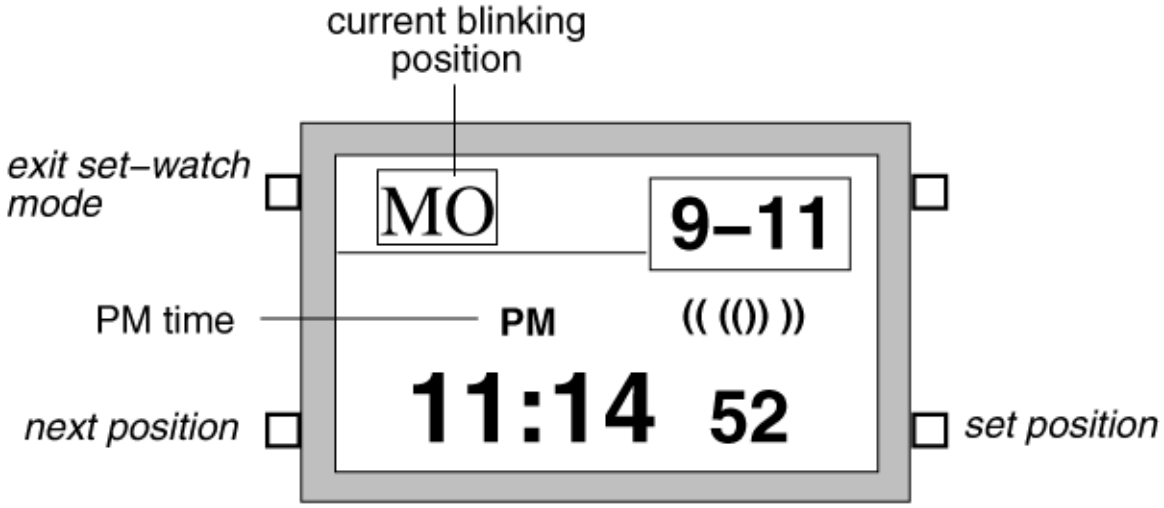

Figure 3: Set-watch mode

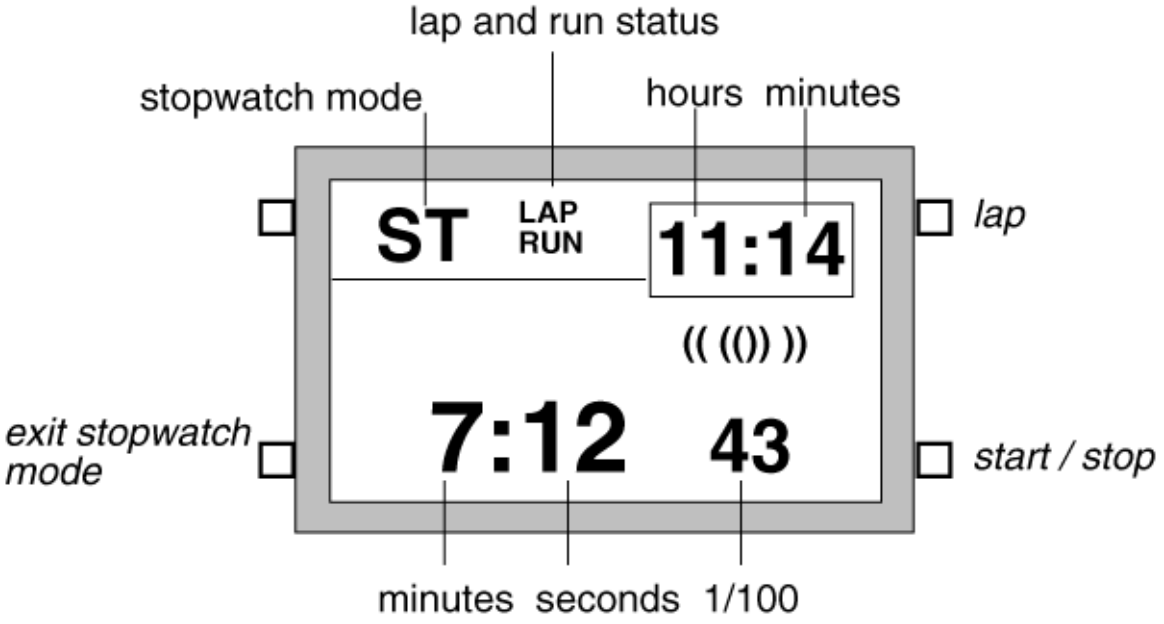

Figure 4: Stopwatch mode

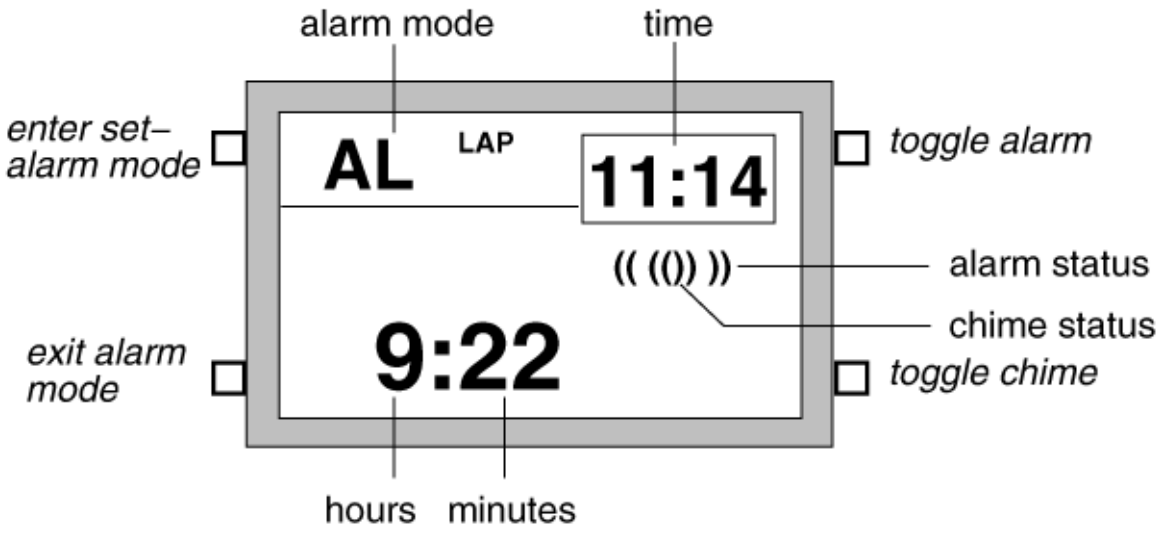

Figure 5: Alarm mode

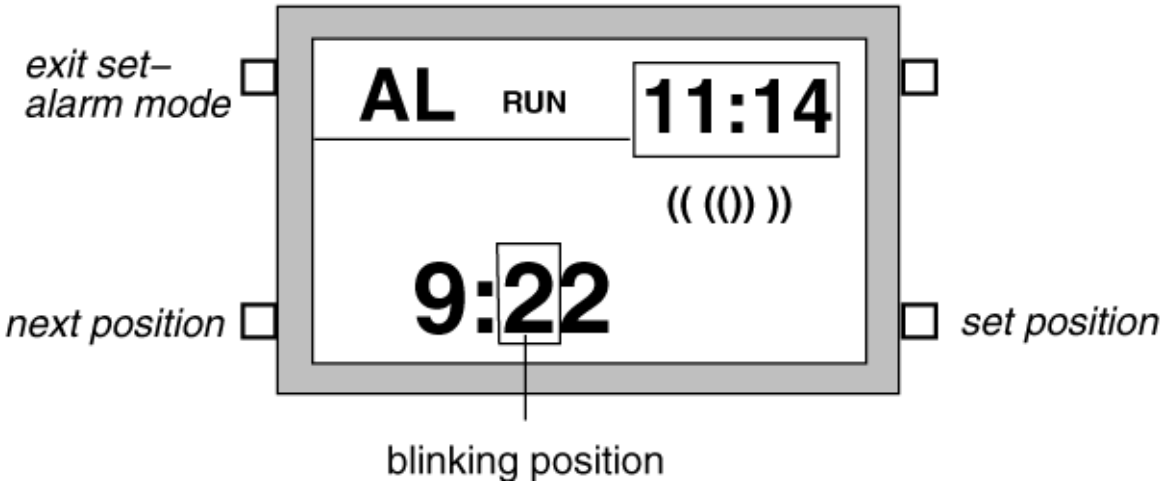

Figure 6: Set-alarm mode

Our wristwatch has three components:

- A regular timekeeper, called simply *the watch* throughout this paper, which displays the time (hours, minutes, seconds), the date (month, day), and the day of the week. There are two time display modes: a 24-hour clock mode (`24H`) and an AM/PM mode (`12H`). A chime can be set to beep every hour. The watch can be set by executing an appropriate setting sequence.
- A stopwatch (minutes, seconds, 1/100 seconds), with split time measurement and "1st-2nd place" time measurement. The stopwatch beeps sounds when the stopwatch is started, stopped, and every 10 minutes when running.
- A daily alarm, which may be set to the minute. The alarm time is shown in `24H` or `12H` mode, depending on the mode used for the regular time. The alarm may be enabled and disabled. The alarm beeps for 30 seconds and may be stopped by depressing a button.

The wristwatch has five modes: watch mode, set-watch mode, stopwatch mode, alarm mode, and set-alarm mode. They correspond to five different display modes, shown in figs. 2–6. The time is shown on the main display when in watch or set-watch mode, and on the mini-display in stopwatch, alarm, and set-alarm mode. Therefore, it is always visible. The date is shown on the mini-display in watch and set-watch mode.

Six on/off indicators display the status of five options (`24H` option, chime option, alarm option, stopwatch-run option, stopwatch-lap option) and the `PM` status when in `12H` mode. In set-watch mode and in set-alarm mode, the position currently being set blinks (hours, minutes, etc).

The wristwatch beeps in three different ways: two beeps per second for the watch chime, one beep per second for the stopwatch, four beeps per second for the alarm. If two units beep simultaneously, their number of bits are added. For example, if the watch and alarm beep simultaneously, the beeper beeps six times per second.

The user controls the wristwatch by four buttons, which change meaning according to the mode. The upper left button `UL` is used for entering and exiting set-watch mode and set-alarm mode. The lower left button `LL` is used for circling between watch mode, stopwatch mode, and alarm mode. It is also used when setting times for changing the position being set (hours, minutes, etc.). The upper right button `UR` is used for toggling the alarm option in alarm mode and is the `LAP` button in stopwatch mode. The lower right button `LR` toggles the `24H` option in watch mode, toggles the chime option in alarm mode, applies a setting command in set-watch or set-alarm mode, and is the start/stop button in stopwatch mode.

3 Detailed Informal Specification

The wristwatch has quite a complex display unit. It includes two numeric displays, containing time and date values. We call them the main display and the mini display see Figure 1. There is an alphabetic display showing the day of the week or the current mode, and six on/off indicators that show some appropriate symbol when a corresponding option is on (and nothing when off). They show respectively the time display option ('`24H`' when on, nothing when off), the `PM` status (only '`PM`' is shown and only when not in `24H` mode), the alarm status, the chime status, the stopwatch run status ('`RUN`' when the stopwatch is running) and the stopwatch lap status ('`LAP`' while in `LAP` mode).

3.1 Details of the Five Modes

3.1.1 Watch Mode

Watch mode is pictured in Figure 2. It corresponds to normal timekeeping. The time (hours, minutes, seconds) is shown on the main display. The date (month, day) is shown on the mini display. The day of the week is shown on the alphabetic display. The six on/off indicators show their current status.

The time is incremented every second, with the usual carry from seconds to minutes, to hours, to days and day of the week, and to months.

Depressing LR switches from 24H to 12H mode. Depressing LL exits watch mode and enters stopwatch mode. Depressing UL exits watch mode and enters set-watch mode. The UR input is ignored.

3.1.2 Set-Watch Mode

Set-watch mode is pictured in Figure 3. It is entered by depressing UL while in regular watch mode. The display only differs by the fact that the current position of the time being set blinks.

The time is incremented every second, but the carry is propagated only to the current position being set. For example, if the 10-minute position is currently being set, then the displays goes from 49mn 59s to 40mn 00s instead of 50mn 00s when a second occurs, to avoid interfering with the user setting.

Depressing LL switches to the next setting position. The order is as follows: seconds, hours, 10-minutes, minutes, months, days, day in week, and back to seconds. Depressing LR applies a setting command that increments the current position value by 1, except for the seconds position, which is reset to 00. Depressing UL exits set-watch mode and returns to watch mode. The UR input is ignored.

3.1.3 Stopwatch Mode

Stopwatch mode is pictured in Figure 4. It is entered by depressing LL while in watch mode. The main display shows the stopwatch time (minutes, seconds, 1/100 seconds). The mini display shows the regular time (hours, minutes) in 24H mode. The alphanumeric display shows the letters 'ST'. The 24H and PM indicators are off. The other on/off indicators show their current status.

The stopwatch maintains two time values, the internal time and the displayed time. The RUN mode, toggled by the LR (START/STOP) button, determines whether the internal time is incremented every 1/100 second. The RUN indicator shows 'RUN' in RUN mode, nothing otherwise. The role of the UR (LAP) button is a bit more complex since it is used both to control the LAP mode and to reset the stopwatch. The LAP mode is entered by depressing UR while in RUN mode. The stopwatch is reset by depressing UR when neither in RUN mode nor in LAP mode. The display doesn't change in LAP mode, whether the stopwatch is running or not (LR still toggles RUN mode while in LAP mode). Depressing UR again exits LAP mode; then, the displayed time is reset to the internal time and is incremented every 1/100 second if in RUN-mode, as if LAP mode had never been entered. The LAP indicator shows 'LAP' while in LAP-mode, nothing otherwise.

Using LAP mode, we can get "1st-2nd place time". The stopwatch is started at the beginning of a race. When the first racer finishes the race, UR (LAP) is depressed. The display then stops, but not the stopwatch proper, which continues counting internally. When the second racer finishes the race, LR (START/STOP) is depressed. One can then record the time of the first racer, which is still on the display, while the stopwatch is internally stopped at the time of the second racer. Depressing UR again shows the time of the second race, since internal time replaces LAP time. Depressing UR for a last time resets the stopwatch.

In stopwatch mode, UL is ignored, while LL exits the mode and enters alarm mode.

3.1.4 Alarm Mode

Alarm mode is pictured in Figure 5. It is entered by depressing LL while in stopwatch mode. The main display shows the alarm time (hours, minutes). The alarm time is shown in 24H mode if and only if the regular time was shown in 24H mode. There is no specific command for toggling the 24H mode in alarm time. The mini display shows the regular time (hours, minutes) in 24H mode. The alphabetic display shows the letters 'AL'. The 24H indicator is on while in 24H mode, and in 12H mode the PM indicator is on if the alarm time is a PM time; The other indicators show their current status.

Depressing UR toggles the alarm status, and accordingly the alarm status indicator. Depressing LR toggles the chime status, and accordingly the chime status indicator. Depresing LL exits alarm mode and enters watch mode. Depressing UL exits alarm mode and enters set-alarm mode.

3.1.5 Set-Alarm Mode

Set-alarm mode is entered by depressing UL when in alarm mode. The display is as in alarm mode, except that the position currently set blinks.

Depressing LL switches to the next setting position, the setting order being hours, minutes. Depressing LR applies a setting command that increments by one the current setting position value. Depressing UL exits set-alarm mode and returns to alarm mode, setting the alarm status to true and turning on the alarm indicator. The UR input is ignored.

3.2 The Beeper

The beeper can be activated by the regular watch, the stopwatch and the alarm. The watch beeps twice a second, the stopwatch beeps once a second, the alarm beeps four times a second. The units may beep at the same time; the actual number of beeps per second is then the sum of the individual numbers. For example if the watch and alarm beep simultaneously, the global effect is six beeps per second.

The watch chime beeps at every full hour when the chime status is on (toggled by button LR in alarm mode), in all modes except in set-watch mode.

The stopwatch beeps each time START/STOP (i.e. LR) is depressed when in stopwatch mode, and also every 10 mn reached by the stopwatch time when the stopwatch is running, this in any mode.

The alarm beeps when the alarm status is on and when the regular time hits the alarm time, in any mode except set-watch and set-alarm modes. The alarm beeps for 30 seconds, and may be stopped by depressing UR. More precisely, a beeping sequence is started and can be terminated only by a 30s delay or by depressing the UR button. Beeping is not terminated by setting the time or alarm to a new value.

3.3 Global Behavior

The time shown on the main display in watch or set-watch modes and in the mini display in all other modes is incremented every second for the main display and every minute for the mini display.

When a mode is exited and later re-entered, the numeric and alphanumeric displays are exactly in the same states as they were when exited, with one exception: an alarm time displayed in 24H mode (resp. 12H mode) is now displayed in 12H mode (resp. 24H mode) if the 24H mode was toggled while in watch mode. The indicators always show their current status, except that the 24H and PM indicators are off in stopwatch mode.

3.4 Initialization

The wristwatch starts in watch mode, the watch shows time 0:00:00 , Sunday 1-1 1900, 24H mode. The chime is off. The alarm time is 0:00 and the alarm is off. The stopwatch time is 0:00:00, RUN and LAP modes off.

3.5 Additional Features

A light is turned on each time UR is depressed. If LL and LR are both hold depressed then the beeper beeps seven times per second, to perform a beeper test.

3.6 Remarks

In the above specification, we have made everything explicit to avoid introducing undescribed features when programming. We must admit that the specification was written after the program and that we had many choices to make that were not easy to detect at start. Should the alarm beep when the alarm time is reached within a watch or alarm setting sequence? Should the alarm keep beeping if we change the time while it is beeping? Should the stopwatch remember its full state when exiting stopwatch mode? The actual watches one can buy have different behaviors, and ours is probably not available on the market! An important point is that all the possible behaviors are equally easy to program in Esterel. Moreover, modifying the program to change a feature is generally very easy, unlike modifying a hand-designed C-code, automaton or circuit. In Section 9, we explain how program several variants.

3.7 What We Program and What We Leave Out

In the Esterel program, we shall leave out three features of our wristwatch:

- The light, which is trivially handled by an electrical contact.
- The blinking mechanism of setting positions. Making a position blink is just a way of enhancing it. In some other device the position could be set in another color, or shown by some sign. Therefore it is unwise to program a fine-grain blinking mechanism at the level of the source Esterel code.
- The beeper test, which is trivial and has no interaction with the rest of the watch. There is no problem in programming it, but there is a penalty in some compiling modes such as automaton mode, where any proper wristwath state should be coupled with one of four beeper states (each button depressed or released). A better idea is to write a separate (trivial) Esterel program for the beeper test.

4 Architecture of the Esterel Program

We construct our wristwatch as a set of five cooperating modules: a watch, a stopwatch, an alarm, a button interpreter, and a display handler. They communicate by exchanging local signals possibly carrying values in appropriate data types. We make an extensive use of two essential Esterel features: signal broadcasting and instantaneous control transmission.

We use as many local signals as needed for convenient programming, remembering that emission and reception of local signals is done mostly at compile-time and produces almost no overhead at run-time. In particular, the `WATCH`, `STOPWATCH` and `ALARM` modules have their own set of commands (for example, `START_STOP_COMMAND` and `LAP_COMMAND` for the stopwatch), and ignore the four actual buttons `UL`, `UR`, `LL`, and `LR`. The role of the button interpreter is to transform button commands into actual watch, stopwatch, and alarm commands, according to the current mode. This makes our modules reusable in other contexts, such as wristwatches with three or five input buttons and different command layouts.

4.1 Overall Architecture

The `WATCH` module takes care of the regular time, which belongs to a type `WATCH_TIME_TYPE`. Its functions are incrementing the time, setting the time, toggling the `24H` and `12H` mode in time representation, and handling the chime. The `WATCH` module broadcasts the time value whenever this value is modified, as the value of a signal called `WATCH_TIME`. Synchronously with this signal, `WATCH` broadcasts a pure signal `WATCH_BEING_SET` when the watch is in set mode. It also broadcasts the chime status whenever this status changes, the chime beep value every second (either `WATCH_BEEP_VALUE` or

NO_BEEP_VALUE). When changing setting position in set-watch mode, WATCH broadcasts two signals for enhancing time positions, which belong to a type WATCH_TIME_POSITION. These signals are called START_ENHANCING and STOP_ENHANCING.

The STOPWATCH module handles the stopwatch time, which belongs to a type STOPWATCH_TIME_TYPE. It handles the RUN and LAP modes. It broadcasts the visible stopwatch time value whenever this value is modified, through a signal called STOPWATCH_TIME. The STOPWATCH module broadcasts the RUN and LAP status whenever they change. It also broadcasts a beep value, either STOPWATCH_BEEP_VALUE or NO_BEEP_VALUE. The stopwatch module is internally composed of three submodules, see Section 7.2.

The ALARM module takes care of the alarm time, which belongs to a type ALARM_TIME_TYPE, of the alarm time setting, and of the alarm beep sequence. It assumes the existence of an external watch that broadcasts a WATCH_TIME signal carrying the regular time value, possibly synchronously with a pure signal WATCH_BEING_SET telling that the external watch is currently in set-watch mode. It broadcasts the alarm time and the alarm status whenever modified, by broadcasting signals ALARM_TIME and ALARM_STATUS. The ALARM module broadcasts two signals for enhancing alarm time positions, which belong to a type ALARM_TIME_POSITION. These signals are called START_ENHANCING and STOP_ENHANCING. It finally broadcasts the beep value ALARM_BEEP_VALUE when the alarm beeps.

The BUTTON module handles the command modes. It broadcasts the mode changes by appropriate signals WATCH_MODE_COMMAND, STOPWATCH_MODE_COMMAND, and ALARM_MODE_COMMAND. In each mode, it renames the signals UL, UR, LL, and LR into adequate watch, stopwatch or alarm commands; for example, any reception of LR is instantaneously relayed into an emission of START_STOP_COMMAND when in stopwatch mode.

The DISPLAY module handles the main display and the mini display. It receives mode-switching commands from the button interpreter, time values from the watch, stopwatch, and alarm modules, and time positions to be enhanced from the WATCH and ALARM modules. In each mode, it converts time values and positions to display values and positions and updates the display.

The main module consists basically in putting the five modules in parallel. However some signal renaming has to be done. For example both the WATCH and ALARM modules emit the signals START_ENHANCING and STOP_ENHANCING, but with values of distinct types WATCH_TIME_POSITION and ALARM_TIME_POSITION. These signals are renamed into WATCH_START_ENHANCING etc.

Notice that the five modes described in the specification make sense only for the button interpreter and the display. The watch, stopwatch, and alarm ignore them.

5 Input-Output Interface

The input-output interface must be known precisely before starting the programming process; it determines how to insert our Esterel program into other programs receiving the actual physical input events, updating the actual physical display, and activating the physical beeper.

The Esterel style undoubtedly induces some choices that we shall try to make explicit, and we do not claim that we are making the design top-down (although we use a top-down presentation here). In any real application, one has to consider several event levels, from the electrical level (e.g. pure interrupts or electrical signals) to a logical level (double click on a mouse button, or "second" signal). The role of an interrupt-handling system is to convert the electrical level into a logical level. For Esterel programs, we think that the right way is to start at a rather high logical level, leaving most of the trivial tasks to the low-level interrupt handling routines.

5.1 Input Interface

The input declarations are:

- `input UL, UR, LL, LR;`
  The four control buttons.

- `input HS;`
  The 1/100 second

- `input S;`
  The second, always synchronous with `HS`.

Notice that no "physical time" is built-in in Esterel; here, we describe the interface with an external quartz, assuming that the quartz handler delivers two distinct signals `HS` and `S`. We could of course produce internally `S` from `HS`, but we prefer to do it externally, i.e. at the level of the execution system that will run our code, where it can be done easily and most efficiently.

Besides `HS` and `S`, we shall assume that all input signals are pairwise incompatible. Except for the beeper test which is not handled here, we did not specify what to do when receiving simultaneously two button signals, and this is certainly pointless. At the operating system (or interrupt handling) level, we just assume that the `UL`, `UR`, `LL`, `LR`, and `HS` signals are serialized in some way. The input relations are

```
relation UL # UR # LL # LR # HS,
         S => HS;
```

5.2 Output Interface

For the output interface, we need to introduce some signals carrying values in appropriate data types. We associate a type `MAIN_DISPLAY_TYPE` with the main display and a type `MINI_DISPLAY_TYPE` with the mini display. For position enhancing, we introduce a type `DISPLAY_POSITION`. The alphabetic display is handled by using the predefined `string` type. The `24H` and `PM` on/off indicators are related to the way a time value is shown on the main display. Therefore, they will be handled as parts of the main display, and their status will be included in the type `MAIN_DISPLAY_TYPE`, for example as two Boolean fields. For the remaining four on/off indicators, we have two equivalent solutions. We can either use two signals `ON` and `OFF` per indicator, or a single signal conveying a `boolean` value, e.g. `true` for on. We choose the second solution. Finally we introduce a type `BEEP_TYPE` for the beeper. A `BEEP_TYPE` value tells how to beep; it could be implemented as an integer telling how much physical beeps should be produced in the next second. Elements of `BEEP_TYPE` can be combined by a function `COMBINE_BEEPS`; it is very convenient to introduce a special dummy value `NO_BEEP_VALUE`, ignored by the actual beeper and acting as an identity element for `COMBINE_BEEPS`.
The output declarations are:

- `output MAIN_DISPLAY : MAIN_DISPLAY_TYPE;`
  Towards the main display.

- `output MINI_DISPLAY : MINI_DISPLAY_TYPE;`
  Towards the mini display.

- `output ALPHABETIC_DISPLAY : string;`
  Towards the alphabetic display.

- `output START_ENHANCING : DISPLAY_POSITION;`
  To enhance a display position.

- `output STOP_ENHANCING : DISPLAY_POSITION;`
  To stop enhancing a display position.

- `output CHIME_STATUS : boolean;`
  Towards the chime status indicator.

- `output STOPWATCH_RUN_STATUS : boolean;`
  Towards the stopwatch run status indicator.
- `output STOPWATCH_LAP_STATUS : boolean;`
  Towards the stopwatch lap status indicator.
- `output ALARM_STATUS : boolean;`
  Towards the alarm status indicator.
- `output BEEP : combine BEEP_TYPE with COMBINE_BEEPS;`
  Towards the beeper.

Notice that the types mentioned here are "abstract" or "private" types. Their exact implementation is not known at the Esterel level. An output signal such as `START_ENHANCING` should tell the external system to start enhancing the specified `DISPLAY_POSITION`, but no detail is given on how do do it. This of course improves portability.

6 Module Interfaces

We have fixed the external interface. We now describe the interface of each module. Once this interface is well-understood, the modules themselves are easy to program, see the next section.

6.1 The WATCH Module Interface

The `WATCH` module handles both the watch and set-watch modes. The types involved are `WATCH_TIME_TYPE`, `WATCH_TIME_POSITION`, and `BEEP_TYPE`. The input declarations are:

- `input S;`
  The second.
- `input TOGGLE_24H_MODE_COMMAND;`
  Go from `24H` mode to `AM/PM` mode and conversely.
- `input TOGGLE_CHIME_COMMAND;`
  Toggle the chime between on and off.
- `input ENTER_SET_WATCH_MODE_COMMAND;`
  Start a setting sequence.
- `input SET_WATCH_COMMAND;`
  Apply a setting command.
- `input NEXT_WATCH_TIME_POSITION_COMMAND;`
  Go to the next setting position.
- `input EXIT_SET_WATCH_MODE_COMMAND;`
  Terminate the setting sequence.

All input signals are assumed to be pairwise incompatible.
The output declarations are:

- `output WATCH_TIME : WATCH_TIME_TYPE;`
  The current time.
- `output WATCH_BEING_SET;`
  A pure signal, always synchronous with `WATCH_TIME`, which tells that the watch is currently in a setting sequence.

- `output START_ENHANCING : WATCH_TIME_POSITION;`
  Emitted in setting sequences when a position becomes the currently set position.
- `output STOP_ENHANCING : WATCH_TIME_POSITION;`
  Emitted in setting sequences when a position stops being the currently set position or when set-watch mode is exited.
- `output CHIME_STATUS : boolean;`
  Towards the chime status indicator.
- `output BEEP : BEEP_TYPE;`
  Towards the beeper.

6.2 The STOPWATCH Module Interface

The `STOPWATCH` module handles the stopwatch time, which belongs to the type `STOPWATCH_TIME_TYPE`. Its input declarations are:

- `input HS;`
  The 1/100 second.
- `input START_STOP_COMMAND;`
  Toggles the `RUN` mode.
- `input LAP_COMMAND;`
  Toggles the `LAP` mode, also used to reset the stopwatch.

All input signals are assumed to be pairwise incompatible.
The output declarations are:

- `output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  The current value of the visible stopwatch time.
- `output STOPWATCH_RUN_STATUS : boolean;`
  Towards the `RUN` status indicator.
- `output STOPWATCH_LAP_STATUS : boolean;`
  Towards the `LAP` status indicator.
- `output BEEP : BEEP_TYPE;`
  Towards the beeper.

6.3 The ALARM Module Interface

The `ALARM` module is in charge of handling the alarm time, which belongs to the type `ALARM_TIME_TYPE`, of setting this time, and of starting the alarm beep sequence when needed. Its input declarations are:

- `input TOGGLE_24H_MODE_COMMAND;`
  Switch between `24H` and `12H` mode.
- `input ENTER_SET_ALARM_MODE_COMMAND;`
  Start a setting sequence.
- `input SET_ALARM_COMMAND;`
  Apply a setting command.

- `input NEXT_ALARM_TIME_POSITION_COMMAND;`
  Go to the next setting position.
- `input EXIT_SET_ALARM_MODE_COMMAND;`
  Terminate the setting sequence.
- `input WATCH_TIME : WATCH_TIME_TYPE;`
  The regular time, broadcasted by an external watch.
- `input WATCH_BEING_SET;`
  A signal present synchronously with `WATCH_TIME` if the external watch is currently in a setting sequence (remember that the alarm should not beep in this case).
- `input TOGGLE_ALARM_COMMAND;`
  Toggle the alarm.
- `input S;`
  The second, used in the beeping sequence.
- `input STOP_ALARM_BEEP_COMMAND;`
  Stop the alarm beep sequence.

We have the relation:

- `relation WATCH_BEING_SET => WATCH_TIME`

The signal `STOP_ALARM_BEEP_COMMAND` can appear at any time. Otherwise all input signals are supposed to be incompatible.

The output declarations are:

- `output ALARM_TIME : ALARM_TIME_TYPE;`
  The current value of the alarm time.
- `output START_ENHANCING : ALARM_TIME_POSITION;`
  Used in setting sequences.
- `output STOP_ENHANCING : ALARM_TIME_POSITION;`
  Used in setting sequences.
- `output ALARM_STATUS : boolean;`
  Towards the alarm status indicator.
- `output BEEP : BEEP_TYPE;`
  Towards the beeper.

6.4 The BUTTON Module Interface

The button interpreter has four input signals `UL`, `UR`, `LL`, and `LR`, which are supposed to be incompatible. It has many output signals. The first ones are related to the watch:

- `output WATCH_MODE_COMMAND;`
  Emitted when watch mode is entered.
- `output TOGGLE_24H_MODE_COMMAND;`
- `output ENTER_SET_WATCH_MODE_COMMAND;`
- `output SET_WATCH_COMMAND;`

- `output NEXT_WATCH_TIME_POSITION_COMMAND;`
- `output EXIT_SET_WATCH_MODE_COMMAND;`
- `output TOGGLE_CHIME_COMMAND;`

The next ones are related to the stopwatch:

- `output STOPWATCH_MODE_COMMAND;`
  Emitted when stopwatch mode is entered.
- `output START_STOP_COMMAND;`
- `output LAP_COMMAND;`

The last ones are related to the alarm:

- `output ALARM_MODE_COMMAND;`
  Emitted when alarm mode is entered.
- `output ENTER_SET_ALARM_MODE_COMMAND;`
- `output SET_ALARM_COMMAND;`
- `output NEXT_ALARM_TIME_POSITION_COMMAND;`
- `output EXIT_SET_ALARM_MODE_COMMAND;`
- `output TOGGLE_ALARM_COMMAND;`
- `output STOP_ALARM_BEEP_COMMAND;`

6.5 The DISPLAY Module Interface

The `DISPLAY` module receives mode commands from the button interpreter and signals from the `WATCH`, `STOPWATCH`, and `ALARM` modules. These signals carry times or time positions. The module converts them into output signals to be sent to the display unit.

The input declarations are:

- `input WATCH_MODE_COMMAND;`
  Tells that watch mode is entered.
- `input WATCH_TIME : WATCH_TIME_TYPE;`
  The time broadcast by the watch.
- `input WATCH_START_ENHANCING : WATCH_TIME_POSITION;`
  Used in set-watch mode, to be converted to a display position.
- `input WATCH_STOP_ENHANCING : WATCH_TIME_POSITION;`
  Used in set-watch mode, to be converted to a display position.
- `input STOPWATCH_MODE_COMMAND;`
  Tells that stopwatch mode is entered.
- `input STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  The time broadcast by the stopwatch.
- `input ALARM_MODE_COMMAND;`
  Tells that alarm mode is entered.

- `input ALARM_TIME : ALARM_TIME_TYPE;`
  The time broadcast by the alarm.
- `input ALARM_START_ENHANCING : ALARM_TIME_POSITION;`
  Used in set-watch mode, to be converted to a display position.
- `input ALARM_STOP_ENHANCING : ALARM_TIME_POSITION;`
  Used in set-watch mode, to be converted to a display position.

For relations, we suppose that the three mode commands `WATCH_MODE_COMMAND`, `STOPWATCH_MODE_COMMAND`, and `ALARM_MODE_COMMAND` are pairwise incompatible; There is no relation between signal pairs such as `WATCH_START_ENHANCING` and `WATCH_STOP_ENHANCING` that can either appear separately (at the beginning or end of a setting sequence) or simultaneously (when going from one position to another one). A complete set of relations is hard to give here.
The output declarations are:

- `output MAIN_DISPLAY : MAIN_DISPLAY_TYPE;`
- `output MINI_DISPLAY : MINI_DISPLAY_TYPE;`
- `output ALPHABETIC_DISPLAY : string;`
- `output START_ENHANCING : DISPLAY_POSITION;`
- `output STOP_ENHANCING : DISPLAY_POSITION;`

7 Module Codes

We now detail the code of the individual modules.

7.1 The WATCH Module

7.1.1 Declarations of WATCH

There are three groups of declarations: the first group concerns the watch time handling, the second group concerns the watch time position handling for setting sequences, and the third group concerns the beeper interface. The input-output declarations were already described in the watch interface section and are omitted here.
To handle the watch time:

- `type WATCH_TIME_TYPE;`
  The type of time values.
- `function GET_INITIAL_WATCH_TIME (): WATCH_TIME_TYPE;`
  Get the initial watch time, which is displayed when the watch is initialized.
- `function INCREMENT_WATCH_TIME (WATCH_TIME_TYPE) : WATCH_TIME_TYPE`
  The standard watch time incrementation function.
- `function TOGGLE_24H_MODE_IN_WATCH_TIME (WATCH_TIME_TYPE) : WATCH_TIME_TYPE`
  Toggles the `24H` and `12H` modes.

To set the watch time:

- `type WATCH_TIME_POSITION;`
  The type of watch time setting positions.

- `constant INITIAL_WATCH_TIME_POSITION : WATCH_TIME_POSITION;`
  Denotes the starting position of setting sequences.
- `function NEXT_WATCH_TIME_POSITION (WATCH_TIME_POSITION) : WATCH_TIME_POSITION;`
  Returns the next setting position from a given position.
- `function SET_WATCH_TIME (WATCH_TIME_TYPE, WATCH_TIME_POSITION) : WATCH_TIME_TYPE;`
  Applies a setting command to a watch time at the current position (for example resets the seconds to 00, or increments the day).
- `function INCREMENT_WATCH_TIME_IN_SET_MODE (WATCH_TIME_TYPE) : WATCH_TIME_POSITION;`
  Increments the time as required in set-watch mode, that is up to the position being currently set.

To beep:

- `type BEEP_TYPE;`
  The type of beeper commands carried by the `BEEP` output signal.
- `function WATCH_BEEP (WATCH_TIME_TYPE, boolean) : BEEP_TYPE;`
  Returns the value `WATCH_BEEP_VALUE` if the time is a full hour and if the Boolean is true, the value `NO_BEEP_VALUE` otherwise; the Boolean is of course the chime status.

7.1.2 Body of WATCH

The exact code is given in Section 11.1. We explain how it is built. We start by emitting the initial time obtained from the environment and the initial chime status:

```
emit WATCH_TIME (GET_INITIAL_WATCH_TIME());
emit CHIME_STATUS (false)
```

Then, we enter an infinite loop. The body of this loop is a sequence of the instruction corresponding to watch mode and of the instruction corresponding to set-watch mode:

```
loop
   abort
      <watch-mode>
   when ENTER_SET_WATCH_MODE_COMMAND;
   abort
      <set-watch-mode>
   when EXIT_SET_WATCH_MODE_COMMAND
end loop
```

The watch-mode instruction is simply an infinite loop having as body an `await-case` on three signals:

- `case S`
  Emit the new time, computed as the result of applying the `INCREMENT_WATCH_TIME` function to the previous time obtained by using the `?pre` operator; emit the `BEEP` signal with the value obtained by calling the `WATCH_BEEP` function.
- `case TOGGLE_24H_MODE_COMMAND`
  Emit the time obtained from the previous time by applying the mode toggling function.
- `case TOGGLE_CHIME_COMMAND`
  Emit the signal `CHIME_STATUS` with value the negation of its previous value.

When entering set-watch mode, we first emit the `START_ENHANCING` signal with value the ininital enhancing position; we then enter a loop over an `await-case` statement on three signals:

- `case S`
  Emit the time incremented by calling the `INCREMENT_WATCH_TIME_IN_SET_MODE` function with arguments the previous time and the current setting position; emit `WATCH_BEING_SET`.
- `case SET_WATCH_COMMAND`
  Emit the new time obtained by calling the `SET_WATCH_TIME` function with arguments the previous time and the current setting position; emit the new time and `WATCH_BEING_SET`.
- `case NEXT_WATCH_TIME_POSITION_COMMAND`
  Emit the `STOP_ENHANCING` signal with value the previous value of `START_ENHANCING` and emit `START_ENHANCING` with value the result of `NEXT_WATCH_TIME_POSITION` applied to its previous value.

When set-watch mode is exited, i.e. upon reception of `EXIT_SET_WATCH_MODE_COMMAND`, we emit the `STOP_ENHANCING` signal with argument the current value of `START_ENHANCING`.

7.1.3 Remarks

The signals `TOGGLE_24H_MODE_COMMAND` and `TOGGLE_CHIME_COMMAND` are taken into account only in watch mode. There is no difficulty in handling them also in set-watch mode, by copying the two corresponding cases of the first `await` into the second one. The obtained watch is certainly better, even if the two new cases may never be used in our global wristwatch. There is no penalty in doing that since any Esterel optimizer will wipe away the dead code.

Notice finally that all signal must be incompatible, otherwise there would be some trouble with the `await-case` statement, which takes the cases in order.

7.2 The STOPWATCH module

7.2.1 Architecture of STOPWATCH

The stopwatch behavior is a bit complex, because of the `RUN` and `LAP` modes, and also because of the particular command used to reset the stopwatch : `LAP_COMMAND` when neither in `RUN` mode nor in `LAP` mode. We break the complexity down by introducing submodules.

First, we treat separately the reset command and program a more natural stopwatch with three distinct buttons, start/stop, lap, and reset. The initially specified stopwatch is recovered by the code:

```
signal RESET_STOPWATCH_COMMAND in
   <THREE_BUTTON_STOPWATCH>
||
   run STOPWATCH_RESET_HANDLER  % generates RESET_STOPWATCH_COMMAND
end signal
```

Now we notice that the reset command is very easy to handle in the three-button stopwatch. We just have to introduce a simpler two-button stopwatch with only `RUN` and `LAP` modes, hence without resetting. Call it `NO_RESET_STOPWATCH`. The above program becomes

```
signal RESET_STOPWATCH_COMMAND in
   loop
      <NO_RESET_STOPWATCH>
   each RESET_STOPWATCH_COMMAND
||
   run STOPWATCH_RESET_HANDLER  % produces RESET_STOPWATCH_COMMAND
end signal
```

We further simplify `NO_RESET_STOPWATCH` by dividing it into two submodules: a `BASIC_STOPWATCH` module that only knows about `RUN` mode and a `LAP_FILTER` module that only knows about `LAP` mode. The `LAP_FILTER` module filters the time broadcast by `BASIC_STOPWATCH` according to the current `LAP` mode in order to produce the visible stopwatch time. Hence `BASIC_STOPWATCH` handles what we called the “internal stopwatch time” and `LAP_FILTER` handles the “visible stopwatch time”.

7.2.2 The BASIC_STOPWATCH module

To handle the stopwatch time, we write the following declarations:

- `type STOPWATCH_TIME_TYPE;`
  For stopwatch time values.
- `constant ZERO_STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  To initialize the stopwatch.
- `function INCREMENT_STOPWATCH_TIME (STOPWATCH_TIME) : (STOPWATCH_TIME);`
  Increments a stopwatch time.

To handle the beeper, we write:

- `type BEEP_TYPE;`
- `constant STOPWATCH_BEEP_VALUE : BEEP_TYPE;`
- `function STOPWATCH_BEEP (STOPWATCH_TIME) : BEEP_TYPE;`
  Takes a stopwatch time as argument and returns either `NO_BEEP_VALUE` or `STOPWATCH_BEEP_VALUE`, the latter being returned when the stopwatch time is a beeping time (say a multiple of 10 minutes — to be defined in the host language).

The input signals are `HS` and `START_STOP_COMMAND`. They are assumed to be pairwise incompatible. We output three signals:

- `output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  Broadcasts the current stopwatch time.
- `output STOPWATCH_RUN_STATUS : boolean;`
  Broadcasts a boolean value representing the current run status (to be used for example by a display).
- `output BEEP : BEEP_TYPE;`
  Broadcasts the current beep value.

The body of `BASIC_STOPWATCH` is very simple, see the code in annex. We enter an infinite loop, which starts with the instantaneous emissions of the initial `false` `RUN` status and of the initial stopwatch time. Since we are not in `RUN` mode, we wait for `START_STOP_COMMAND`. When `START_STOP_COMMAND` occurs, we enter `RUN` mode. We emit the new `true` run status and beep. Run mode lasts up to the next occurrence of `START_STOP_COMMAND`, and consists in incrementing the time every `HS`.

7.2.3 The LAP_FILTER module

We need to declare the `STOPWATCH_TIME_TYPE` type, but no constants, functions or procedures. There are two input signals:

- `input LAP_COMMAND;`
  Toggle the `LAP` mode.
- `input BASIC_STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  A stopwatch time issued by some basic stopwatch.

There are two output signals:

- `output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;`
  Broadcasts the visible stopwatch time.

- `output STOPWATCH_LAP_STATUS : boolean;`
  broadcasts the lap status, presumably to some display unit.

The body of `LAP_FILTER` is an infinite loop, which starts by emitting the initial `false` lap status. Then we are not in `LAP` mode up to the next occurrence of `LAP_COMMAND`. During that time, whenever we receive a time value broadcast by `BASIC_STOPWATCH_TIME`, we re-emit it as the value of `STOPWATCH_TIME` (we use `loop...each` and not `every` in order to catch the first value when the stopwatch is started and the current value when `LAP` mode is exited).

When receiving `LAP_COMMAND`, we enter `LAP` mode. We emit the `true` lap status and wait for the next occurrence of `LAP_COMMAND` that will exit `LAP` mode. See the code in annex.

7.2.4 The STOPWATCH_RESET_HANDLER module

We declare the two incompatible input signals, `START_STOP_COMMAND` and `LAP_COMMAND`, and the output signal `RESET_STOPWATCH_COMMAND`.

The body is an interesting example of using a `sustain` statement that continuously emits a signal to tell in which state a submodule is. We enter an infinite loop of the form:

```
loop
   trap RESET in
      <exit RESET when detecting the reset condition>
   end trap;
   emit RESET_STOPWATCH_COMMAND
end loop
```

To detect the reset condition, we run two loops in parallel, which respectively handle `START_STOP_COMMAND` and `LAP_COMMAND`. Whenever the basic stopwatch is stopped, the first loop sends continuously a signal `STOPWATCH_STOPPED`. Whenever `LAP_COMMAND` is received when not in `LAP` mode, the second loop tests for the presence of `STOPWATCH_STOPPED`. If it is present, the received `LAP_COMMAND` signals provokes a reset by exiting the `RESET` trap. The code of the trap body is:

```
signal STOPWATCH_STOPPED in
   loop
      abort
         sustain STOPWATCH_STOPPED
      when START_STOP_COMMAND;
      await START_STOP_COMMAND
    end loop
||
   loop
      await LAP_COMMAND do
         % LAP_COMMAND received when not in LAP mode
         present STOPWATCH_STOPPED then
            exit RESET
         end
      end await;
      await LAP_COMMAND
   end loop
end signal
```

Notice that the structure of each loop is similar to the structure of the bodies of `BASIC_STOPWATCH` and `LAP_FILTER`; we could as well detect the reset condition in the same way in these modules, but the Esterel code would be much heavier for no run-time gain (after optimization)

7.2.5 The main STOPWATCH module

Its structure follows directly from what we said above.

7.3 The ALARM Module

The full code can be read in Section 11.3

7.3.1 Declarations of ALARM

To handle the alarm time, we write:

- `type ALARM_TIME_TYPE;`
- `constant INITIAL_ALARM_TIME : ALARM_TIME_TYPE;`
  The initial value of the alarm time, displayed when the module is started.
- `function TOGGLE_24H_MODE_IN_ALARM_TIME (ALARM_TIME_TYPE) : ALARM_TIME_TYPE;`
  Switches from `24H` mode to `12H` mode and conversely.

To handle the alarm time setting sequences, we write:

- `type ALARM_TIME_POSITION;`
  Used in set-alarm mode for the currently set position.
- `constant INITIAL_ALARM_TIME_POSITION : ALARM_TIME_POSITION;`
  Defines the starting alarm time position in setting sequences.
- `function NEXT_ALARM_TIME_POSITION (ALARM_TIME_POSITION) : ALARM_TIME_POSITION;`
  Returns the next setting position from a given position.
- `function SET_ALARM_TIME (ALARM_TIME_TYPE, ALARM_TIME_POSITION) : ALARM_TIME_TYPE;`
  Applies a setting command to the time at the current position (for example increments the hours).

To communicate with the external watch, we declare:

- `type WATCH_TIME_TYPE;`
  For the watch time value broadcast by the watch.

To know when and how to beep, we declare

- `type BEEP_TYPE;`
- `constant ALARM_BEEP_VALUE : BEEP_TYPE;`
  Gives the beep value of the alarm beeping sequence.
- `constant ALARM_DURATION : integer;`
  Defines the maximal alarm beep sequence duration (in seconds).
- `function COMPARE_ALARM_TIME_TO_WATCH_TIME (ALARM_TIME_TYPE, WATCH_TIME_TYPE) : boolean;`
  Tests whether the alarm should start beeping

The input and output declarations follow directly from the interface described in the previous section.

7.3.2 Body of ALARM

We declare a local signal START_BEEPING used to start a beeping sequence. We then enter two statement in parallel. The first one handles the time and determines when to start beeping, the second one handles the beeping sequence.

In the first statement, we emit the initial values of ALARM_TIME and CHIME_STATUS. We then enter an infinite loop, the body of which is a sequence of the statement corresponding to alarm mode and of the statement corresponding to set-alarm mode:

```
loop
   abort
      <alarm mode>
   when ENTER_SET_ALARM_MODE_COMMAND;
   abort
      <set-alarm mode>
   when EXIT_SET_ALARM_MODE_COMMAND
end loop
```

The alarm mode instruction is simply an infinite loop having as body a `await-case` on three signals:

- `case` TOGGLE_24H_MODE_COMMAND
  Emit the alarm time obtained from the previous one by calling the 24H and 12H toggling function.

- `case` TOGGLE_ALARM_COMMAND
  Emit the ALARM_STATUS signal with value the negation of the previous one.

- `case` WATCH_TIME
  Test for the presence of the WATCH_BEING_SET signal, using a `present` statement. If this signal is present, the watch is in a setting sequence and there is nothing to do. Otherwise, compare the alarm and watch times. If they match, emit the local signal START_BEEPING that starts the beeping sequence

Notice that the order of cases is important here. In a `await-case` statement, only the first case statement is executed if several case occurrences occur simultaneously. Hence, we must put WATCH_TIME in the *last* case. Otherwise a command like TOGGLE_24H_MODE_COMMAND would not be taken into account, since it is certainly synchronous with WATCH_TIME. The given ordering is easily checked to be safe: one should not start a beeping sequence when receiving TOGGLE_24H_MODE_COMMAND or TOGGLE_ALARM_COMMAND.

The set-watch mode statement is similar. We declare a local variable ALARM_TIME_POSITION initially set to INITIAL_ALARM_TIME_POSITION. We first emit a START_ENHANCING signal carrying this position. We then enter a loop over a `await-case` on two signals:

- `case` SET_ALARM_COMMAND
  Apply a setting command by calling the SET_ALARM_TIME procedure with arguments the current time and setting position; emit the new alarm time.

- `case` NEXT_ALARM_TIME_POSITION_COMMAND
  Set ALARM_TIME_POSITION to the next position by calling the NEXT_ALARM_TIME_POSITION function

The beeping sequence is simple. We beep every second upto the next STOP_ALARM_BEEP_COMMAND, with a maximum of ALARM_DURATION seconds. The appropriate statement is '`abort...case`'.

```
every START_BEEPING do
   abort
      loop emit BEEP (ALARM_BEEP_VALUE) each S
   when
      case STOP_ALARM_BEEP_COMMAND
      case ALARM_DURATION S
   end abort
end every
```

7.3.3 Remarks

We chose that the signals `TOGGLE_24H_MODE_COMMAND` and `TOGGLE_ALARM_COMMAND` are taken into account only in alarm mode. There would be no difficulty to accept them also in set-alarm mode, by copying the two corresponding cases of the first await into the second one.

7.4 The BUTTON Module

Since this module only handles pure signals there are no data declarations; the interface has been already described in the previous section. The full code is given in Section 11.4.

We do two things in parallel: handling the modes and renaming permanently the `UR` input into `STOP_ALARM_BEEP_COMMAND` using an `every` statement. The mode handling has the following structure:

```
emit WATCH_MODE_COMMAND;
loop
   trap WATCH_MODE in
      loop
         abort
            <watch mode -- exit WATCH_MODE on LL>
         when UL;
         emit ENTER_SET_WATCH_MODE_COMMAND;
         abort
            <set-watch mode>
         when UL;
         emit EXIT_SET_WATCH_MODE_COMMAND
      end loop
   end trap
end loop;
 emit STOPWATCH_MODE_COMMAND;
abort
   <stopwatch mode>
when LL;
 emit ALARM_MODE_COMMAND;
loop
   trap ALARM_MODE in
      loop
         abort
            <alarm mode -- exit ALARM_MODE on LL>
         when UL;
         emit ENTER_SET_ALARM_MODE_COMMAND;
         abort
            <set-alarm mode>
         when UL;
         emit EXIT_SET_ALARM_MODE_COMMAND
      end loop
   end trap
end loop
```

The WATCH_MODE and ALARM_MODE traps are necessary for exiting watch mode and alarm mode. Each individual mode consists in simple button renamings, using every statements. For example, LR and UR are respectively renamed into START_STOP_COMMAND and LAP_COMMAND when in stopwatch mode:

```
    every LR do emit START_STOP_COMMAND end
||
    every UR do emit LAP_COMMAND end
```

7.5 The DISPLAY Module

The full code is given in Section 11.5.

7.5.1 Declarations of DISPLAY

We declare the types related to the displays:

- type MAIN_DISPLAY_TYPE;
  For the main display.
- type MINI_DISPLAY_TYPE;
  For the mini display.
- type DISPLAY_POSITION;
  For main, mini, or alphabetic display positions.

To handle the watch, we write the following declarations:

- type WATCH_TIME_TYPE;
- function WATCH_TIME_TO_MAIN_DISPLAY (WATCH_TIME_TYPE) : MAIN_DISPLAY_TYPE;
  Converts a watch time to MAIN_DISPLAY_TYPE. Hours, minutes, and seconds are displayed on the main display, together with the current 24H or PM status.
- function WATCH_TIME_TO_MINI_DISPLAY (WATCH_TIME_TYPE) : MINI_DISPLAY_TYPE;
  Converts a watch time to MINI_DISPLAY_TYPE. Used in stopwatch or alarm mode, where the hours and minutes are displayed on the mini display.
- function WATCH_DATE_TO_MINI_DISPLAY (WATCH_TIME_TYPE) : MINI_DISPLAY_TYPE;
  Converts the date in a watch time to MINI_DISPLAY_TYPE. Month and day are displayed on the mini display when in watch mode.
- function WATCH_DAY_TO_ALPHABETIC_DISPLAY (WATCH_TIME_TYPE) : string;
  Converts the day of the week in a watch time into a string to be displayed on the alphabetic display.
- type WATCH_DISPLAY_POSITION;
- function WATCH_DISPLAY_POSITION (WATCH_TIME_POSITION) : DISPLAY_POSITION;
  Converts a watch time position into a display position to be enhanced.

To handle the stopwatch, we write:

- type STOPWATCH_TIME_TYPE;
- function STOPWATCH_TIME_TO_MAIN_DISPLAY (STOPWATCH_TIME_TYPE) : MAIN_DISPLAY_TYPE;
  Converts a stopwatch time to MAIN_DISPLAY_TYPE. Minutes, seconds, and 1/100 seconds are displayed on the main display.

To handle the alarm, we write:

- `type ALARM_TIME_TYPE;`
- `function ALARM_TIME_TO_MAIN_DISPLAY (ALARM_TIME_TYPE) : MAIN_DISPLAY_TYPE;`
  Converts an alarm time to `MAIN_DISPLAY_TYPE`. Hours and minutes are displayed on the main display, together with the current `24H` or `PM` status.
- `type ALARM_DISPLAY_POSITION;`
- `function ALARM_DISPLAY_POSITION (ALARM_TIME_POSITION) : DISPLAY_POSITION;`
  Converts an alarm time position into a display position to be enhanced.

The input-output interface was declared in the previous section.

7.5.2 Body of DISPLAY

The structure of the body is as follows:

```
loop
   abort
      <watch on display>
   when STOPWATCH_MODE_COMMAND;
   abort
      <watch time on mini display>
   ||
      abort
         <stopwatch on display>
      when ALARM_MODE_COMMAND;
      abort
         <alarm on display>
      when WATCH_MODE_COMMAND
   when WATCH_MODE_COMMAND
end loop
```

Notice that the internal "`abort...when WATCH_MODE_COMMAND`" is useless, since it is preempted by the external one. It is there only for elegance and better extensibility: we can add more easily another alarm or a backtimer.

We only detail the `<watch mode>` statement, the other ones being similar. We have three independent things to do, which correspond to three statements in parallel:

- Displaying the watch time; we use a statement of the form "`loop...each WATCH_TIME`" (we need "`loop...each`" and not `every` to display the watch time whenever entering watch mode); we emit the values to be displayed in the three displays; they are computed by applying the suitable conversion functions described above.
- Starting enhancing the display positions; we use an `every` statement and the appropriate conversion function from watch time positions to display positions
- Stopping enhancing display positions, in the same way

Notice that we use three statements in parallel, *not* a `await-case` statement: the three operations are really independent and share no variable, so that three parallel statements are more natural than an `await-case`. Moreover, the signals `WATCH_START_ENHANCING` and `WATCH_STOP_ENHANCING` are quite often simultaneous (whenever we go to the next setting position), so that an `await-case` wouldn't work properly — remember that the cases are taken *sequentially and up to the first success only*.

7.6 The Main WRISTWATCH Module

Since they appear as types of internal signals. we declare all the types related with times, positions, and displays. We also delare the `BEEP_TYPE` type used for the beeper.

The input-output interface is as described in the previous section. The only remark we make here concerns the `BEEP` signal, which is declared to be a *combined* signal, with `COMBINE_BEEPS` as combination function. This signal can be emitted independently by the `WATCH`, `STOPWATCH`, and `ALARM` modules, and it can be emitted *simultaneously* by them. Therefore, `BEEP` must be a combined signal in the main `WRISTWATCH` module, although it is a single signal in each submodule.

We declare all the required local signals and copy the five submodules in parallel. To avoid name clashes, we rename the signals related to watch and alarm positions enhancing.

8 Running and Simulating the Wristwatch

The wristwatch contained in the Esterel ditribution can be run and simulated. Go to the `wristwatch` directory in the distribution, and read the README.txt file. After typing the appropriate `make` command described there, the following happens:

1. A Tcl/Tk based real-time wristwatch `tkwatch` is built and started. Click on the buttons.
2. Immediately after that, a graphical `xes`-based simulator `xeswatch` is built and started. This simulator uses a fake time type to make simulation faster (2 seconds per minute, etc.). Play with it. In particular, call the recorder and play `sww.esi`, using the `Load` button in the upper part of the recorder panel.
3. Then `tkwatch` and `xeswatch` are run in parallel, `tkwatch` sending the events it receives to `xeswatch` (running with the true time). You can view source code animation while you are running the real watch.
4. Finally A tty-based simulator called `swatch` is built and run.

We suggest exploring the different compiling styles explained in the manual. Try for example setting "`ESTEREL_FLAGS = -A`" in `Makefile` to generate the Tcl/Tk watch `tkww` in automaton mode.

To run a batch simulation, type

```
sww < sww.esi > sww.eso
```

The contents of the simulation output file `sww.eso` should be as follows:

```
WRISTWATCH> ; % empty event, for initializations;
--- Output: MAIN_DISPLAY(1:00:00 24H) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU") CHIME_STATUS(false)
            STOPWATCH_RUN_STATUS(false) STOPWATCH_LAP_STATUS(false)
            ALARM_STATUS(false)

WRISTWATCH> HS, S; % a second
--- Output: MAIN_DISPLAY(1:00:01 24H) MINI_DISPLAY(1-1)
ALPHABETIC_DISPLAY("SU")
BEEP(0)

WRISTWATCH> LL; % enter stopwatch mode
--- Output: MAIN_DISPLAY(0:00:00) MINI_DISPLAY(1:00)
ALPHABETIC_DISPLAY("ST")

WRISTWATCH> LR; % start the stopwatch
--- Output: STOPWATCH_RUN_STATUS(true) BEEP(1)
```

```
WRISTWATCH> HS; % a hundredth
--- Output: MAIN_DISPLAY(0:00:01) BEEP(0)

WRISTWATCH> ! trace signals;
--- Awaited: UL UR LL LR HS S

WRISTWATCH> UR; % lap
--- Output: STOPWATCH_LAP_STATUS(true)
--- Local: LAP_COMMAND STOP_ALARM_BEEP_COMMAND
--- Trap:
--- Awaited: UL UR LL LR HS S

WRISTWATCH> HS; % a hundredth
--- Output: BEEP(0)
--- Local: BASIC_STOPWATCH_TIME(0:01:00)
--- Trap:
--- Awaited: UL UR LL LR HS S

WRISTWATCH> UR; % lap
--- Output: MAIN_DISPLAY(0:01:00) STOPWATCH_LAP_STATUS(false)
--- Local: LAP_COMMAND STOPWATCH_TIME(0:01:00)
           STOP_ALARM_BEEP_COMMAND
--- Trap:
--- Awaited: UL UR LL LR HS S

WRISTWATCH> ! untrace signals;

WRISTWATCH> LL ; % enter alarm mode
--- Output: MAIN_DISPLAY(0:00 24H) ALPHABETIC_DISPLAY("AL")

WRISTWATCH> ! show variables;
--- Source Variables:
--- Signal Variables:
V6 = 0:00 24H  (value of signal MAIN_DISPLAY)
V7 = 1:00  (value of signal MINI_DISPLAY)
V8 = "AL"  (value of signal ALPHABETIC_DISPLAY)
V9 = -*-  (value of signal START_ENHANCING)
V10 = -*-  (value of signal STOP_ENHANCING)
V11 = false  (value of signal CHIME_STATUS)
V12 = true  (value of signal STOPWATCH_RUN_STATUS)
V13 = false  (value of signal STOPWATCH_LAP_STATUS)
V14 = false  (value of signal ALARM_STATUS)
V15 = 0  (value of signal BEEP)
V17 = SU 1-1 1:0:1 24H  (value of signal WATCH_TIME)
V18 = -*-  (value of signal WATCH_START_ENHANCING)
V19 = -*-  (value of signal WATCH_STOP_ENHANCING)
V20 = 0:01:00  (value of signal STOPWATCH_TIME)
V21 = 0:00 24H  (value of signal ALARM_TIME)
V22 = -*-  (value of signal ALARM_START_ENHANCING)
V23 = -*-  (value of signal ALARM_STOP_ENHANCING)
V27 = 0:01:00  (value of signal BASIC_STOPWATCH_TIME)
--- Counters:
V32 = -*- [line: 137, column: 15 of file: "alarm.strl" (ALARM#7)]

WRISTWATCH> UL; % enter set-alarm mode
--- Output: START_ENHANCING(hours)
```

```
WRISTWATCH> LR; % set command (setting hours)
--- Output: MAIN_DISPLAY(1:00 24H)

WRISTWATCH> LL; % next position
--- Output: START_ENHANCING(10 minutes) STOP_ENHANCING(hours)

WRISTWATCH> LL; % next position
--- Output: START_ENHANCING(minutes) STOP_ENHANCING(10 minutes)

WRISTWATCH> LR; % set command (setting minutes)
--- Output: MAIN_DISPLAY(1:01 24H)

WRISTWATCH> UL; % back to alarm mode
--- Output: STOP_ENHANCING(minutes) ALARM_STATUS(true)

WRISTWATCH> LL; % back to watch mode
--- Output: MAIN_DISPLAY(1:00:01 24H) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU")

WRISTWATCH> UL; % enter set watch mode
--- Output: START_ENHANCING(seconds)

WRISTWATCH> LL; % next position
--- Output: START_ENHANCING(hours) STOP_ENHANCING(seconds)

WRISTWATCH> LR; % set hours
--- Output: MAIN_DISPLAY(0:00:01 24H) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU")

WRISTWATCH> UL; % back to watch mode
--- Output: STOP_ENHANCING(hours)

WRISTWATCH> HS, S; % a correct second
--- Output: MAIN_DISPLAY(0:01:00 24H) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU") BEEP(0)

WRISTWATCH> S; % incorrect, HS should be there too
"stdin", line 49: *** Error: implication violated: S => HS

WRISTWATCH> LR; % go to 12H mode
--- Output: MAIN_DISPLAY(1:01:00) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU")

WRISTWATCH> ! reset; % We start a new simulation to get multiple beeps
--- Automaton WRISTWATCH reset
WRISTWATCH>          % in the simulation, the watch beeps every minute,
WRISTWATCH>          % a minute is 2 seconds, and the stopwatch
WRISTWATCH>          % beeps every 2 hundredth of a second!

WRISTWATCH> ; % empty event, for initializations
--- Output: MAIN_DISPLAY(1:00:00 24H) MINI_DISPLAY(1-1)
            ALPHABETIC_DISPLAY("SU") CHIME_STATUS(false)
            STOPWATCH_RUN_STATUS(false) STOPWATCH_LAP_STATUS(false)
            ALARM_STATUS(false)
```

```
WRISTWATCH> LL; % to stopwatch mode
--- Output: MAIN_DISPLAY(0:00:00) MINI_DISPLAY(1:00)
            ALPHABETIC_DISPLAY("ST")

WRISTWATCH> LR; % starts the stopwatch
--- Output: STOPWATCH_RUN_STATUS(true) BEEP(1)

WRISTWATCH> LL; % to alarm mode
--- Output: MAIN_DISPLAY(0:00 24H) ALPHABETIC_DISPLAY("AL")

WRISTWATCH> LR; % sets chime on
--- Output: CHIME_STATUS(true)

WRISTWATCH> UR; % sets alarm on
--- Output: ALARM_STATUS(true)

WRISTWATCH> HS, S; % time passes
--- Output: MINI_DISPLAY(1:00) BEEP(0)

WRISTWATCH> HS, S; % again
--- Output: MINI_DISPLAY(1:01) BEEP(0)

WRISTWATCH> HS, S; % and again
--- Output: MINI_DISPLAY(1:01) BEEP(0)

WRISTWATCH> HS, S; % a big beep: watch, stopwatch,
                     and alarm beep together
--- Output: MINI_DISPLAY(0:00) BEEP(7)

WRISTWATCH> HS,S; % the alarm keeps beeping
--- Output: MINI_DISPLAY(0:00) BEEP(4)

WRISTWATCH> UR; % we stop it
--- Output: ALARM_STATUS(false)

WRISTWATCH> HS, S; % to check that beeping is over
--- Output: MINI_DISPLAY(0:01) BEEP(0)
```

9 Variants of the Wristwatch

Since our architecture is modular, it is very easy to make several variants of the wristwatch. For example, we can remove the alarm or the stopwatch by removing or slightly modifying the corresponding lines of the `BUTTON`, `DISPLAY`, and `WRISTWATCH` modules.

However, there is a problem when removing the alarm, because of an anomaly in the wristwatch *specification*. The `TOGGLE_CHIME_COMMAND` signal is issued by `BUTTON` in `ALARM` mode and *not* in `WATCH` mode, which means that removing the alarm would make the chime inoperative. This feature actually appears in the author's watch, and we kept it to show an example of non-modular specification. A simple way to solve the problem is to modify `BUTTON` by emitting `TOGGLE_CHIME_COMMAND` when receiving `UR` in `WATCH` mode, a one-line change.

Such variants are rather trivial and will not be discussed further. The auxiliary C code written for the full wristwatch needs not be modified for the smaller ones.

A more interesting variant concerns the stopwatch. In the author's wristwatch, the stopwatch does not remember its `LAP` mode when exited. Let us give and example: enter the stopwatch, start

it, and then depress the LAP button to enter LAP mode; exit the stopwatch by depressing LL; re-enter the stopwatch by depressing LL twice. Then the stopwatch has forgotten LAP mode and its display keeps running. In fact, LAP mode is exited as soon as stopwatch mode is exited, with LAP indicator turned off at that moment.

To obtain this behavior, we introduce a new signal EXIT_SET_WATCH_MODE_COMMAND, declared as output in BUTTON, as input in STOPWATCH, STOPWATCH_RESET, and as local in WRISTWATCH. We emit it in BUTTON when exiting stopwatch mode. In STOPWATCH, we replace

```
run LAP FILTER
```

by

```
loop
   run LAP_FILTER
each EXIT_STOPWATCH_MODE_COMMAND
```

In STOPWATCH_RESET, we enclose the second branch of the parallel in

```
loop
   ...
each EXIT_STOPWATCH_MODE_COMMAND
```

Call our original stopwatch $stopwatch_1$ and the new one $stopwatch_2$. When using the Esterel v5 '-A' automaton generation option, $stopwatch_1$ has 42 states while $stopwatch_2$ has only 26, since it needs to remember less when exiting stopwatch mode. This is the (only) interest of the modification.

10 Conclusion

We have completely programmed a reasonably complex Esterel application together with its simulation and execution interfaces. We hope that the present paper will help the reader to understand the programming style we try to promote; this style is actually quite close to the "object programming" style, but uses parallel composition of synchronous processes instead of sequential message passing. We tried to follow the following rules:

- Put the main effort on architecture. Programming is quite easy once the modules and their interfaces are well-understood. More precisely programming *should* be easy. If it is not the case, the architecture is probably not good enough.
- Never hesitate to introduce additional modules or signals. A small clever program does not produce a better object code than a longer but more understandable one. The compiling algorithms perform deep optimizations and many source instructions do not produce code: they just give more work to the compiler and optimizer.
- Use signals to handle the control instead of Booleans and `if-then-else-fi` statements. These statements generate code that cannot currently be simplified, unlike pure signal handling; they should be avoided or kept only if the test generates really different temporal behaviors. There is only one indispensable test in our watch, the test for alarm beeping. The stopwatch reset procedure could be done with booleans, but is much better done with signals as in the text. When a signal is pure output, it is also wise to emit dummy values instead of testing whether a value should be emitted or not, leaving the test to the external output routine and thus factoring it out (as for the BEEP signal here).
- Use parallel statements as much as possible. Reserve the `await-case` statement to situations where it is really necessary, that is to situations where the different cases read or update the same set of variables.

The simulation and execution interfaces raised no particular difficulties. In real situations where the host system is not a workstation system but an embedded real-time system and where speed is required, much more effort should be put in system interfaces. Esterel gives no specific tool at that level.

11 The Full Esterel code

11.1 The WATCH module

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Module WATCH : the timekeeper %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module WATCH :

% To handle the watch time

type WATCH_TIME_TYPE;

function GET_INITIAL_WATCH_TIME () : WATCH_TIME_TYPE;
function INCREMENT_WATCH_TIME (WATCH_TIME_TYPE) : WATCH_TIME_TYPE;
function TOGGLE_24H_MODE_IN_WATCH_TIME (WATCH_TIME_TYPE) : WATCH_TIME_TYPE;

% To set the watch time

type WATCH_TIME_POSITION;
constant INITIAL_WATCH_TIME_POSITION : WATCH_TIME_POSITION;
function NEXT_WATCH_TIME_POSITION (WATCH_TIME_POSITION)
                                  : WATCH_TIME_POSITION;
         % say from seconds to hours to 10 minutes to minutes to month to
         % day to day in week and circularly ! (not relevant for ESTEREL)
function SET_WATCH_TIME
           (WATCH_TIME_TYPE, WATCH_TIME_POSITION) : WATCH_TIME_TYPE;
          % applies a setting command to the current time and position
function INCREMENT_WATCH_TIME_IN_SET_MODE
           (WATCH_TIME_TYPE, WATCH_TIME_POSITION) : WATCH_TIME_TYPE;
          % increments the time only to the position being currently set

% To beep

type BEEP_TYPE;
function WATCH_BEEP (WATCH_TIME_TYPE, boolean) : BEEP_TYPE;
         % returns either the value WATCH_BEEP_VALUE if the watch has to beep
         % and the boolean (CHIME_STATUS) is true,
         % or the value NO_BEEP_VALUE otherwise

% Interface

input S;
input TOGGLE_24H_MODE_COMMAND;

output WATCH_TIME : WATCH_TIME_TYPE;

input ENTER_SET_WATCH_MODE_COMMAND,
      SET_WATCH_COMMAND,
      NEXT_WATCH_TIME_POSITION_COMMAND,
      EXIT_SET_WATCH_MODE_COMMAND;

output WATCH_BEING_SET;
       % Synchronous with WATCH_TIME when the watch is set
output START_ENHANCING : WATCH_TIME_POSITION,
       STOP_ENHANCING : WATCH_TIME_POSITION;
```

```
input TOGGLE_CHIME_COMMAND;
output CHIME_STATUS : boolean;
output BEEP : BEEP_TYPE;

relation   S
         # TOGGLE_24H_MODE_COMMAND
         # TOGGLE_CHIME_COMMAND
         # ENTER_SET_WATCH_MODE_COMMAND
         # SET_WATCH_COMMAND
         # NEXT_WATCH_TIME_POSITION_COMMAND
         # EXIT_SET_WATCH_MODE_COMMAND;

% initializations

emit WATCH_TIME (INITIAL_WATCH_TIME);
emit CHIME_STATUS (false);

% main loop

loop
   % normal mode
   abort % when ENTER_SET_WATCH_MODE_COMMAND
      loop
         await
            case S do
               emit WATCH_TIME (INCREMENT_WATCH_TIME(pre(?WATCH_TIME)));
               emit BEEP (WATCH_BEEP (?WATCH_TIME, ?CHIME_STATUS))
            case TOGGLE_24H_MODE_COMMAND do
               emit WATCH_TIME
                      (TOGGLE_24H_MODE_IN_WATCH_TIME (pre(?WATCH_TIME)))
            case TOGGLE_CHIME_COMMAND do
               emit CHIME_STATUS (not pre(?CHIME_STATUS))
         end await
      end loop
   when ENTER_SET_WATCH_MODE_COMMAND;

   % set-watch mode
   % (in set-watch mode one might as well accept the commands
   % TOGGLE_24H_MODE_COMMAND and TOGGLE_CHIME_COMMAND; for
   % this one just could copy the corresponding cases above into
   % the  select!)

   abort % when  EXIT_SET_WATCH_MODE_COMMAND
      emit START_ENHANCING (INITIAL_WATCH_TIME_POSITION);
      loop
         await
            case S do
               emit WATCH_TIME
                        (INCREMENT_WATCH_TIME_IN_SET_MODE(pre(?WATCH_TIME),
                                                          ?START_ENHANCING));
               emit WATCH_BEING_SET
            case SET_WATCH_COMMAND do
               emit WATCH_TIME (SET_WATCH_TIME(pre(?WATCH_TIME),
                                               ?START_ENHANCING));
               emit WATCH_BEING_SET
            case NEXT_WATCH_TIME_POSITION_COMMAND do
```

```
                emit STOP_ENHANCING (pre(?START_ENHANCING));
                emit START_ENHANCING
                        (NEXT_WATCH_TIME_POSITION(pre(?START_ENHANCING)))
          end await
       end loop
    when  EXIT_SET_WATCH_MODE_COMMAND;
    emit STOP_ENHANCING (pre(?START_ENHANCING))
end loop

end module
```

11.2 The STOPWATCH module

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The Esterel split-time stopwatch %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% There are three submodules : a basic stopwatch, which only handles
% the start/stop command, a lap filter, which handles the lap command,
% and a reset handler, which determines when to reset the stopwatch.
% These modules are put in parallel in the main STOPWATCH module,
% with suitable renamings.

% The BASIC_STOPWATCH module
%---------------------------

module BASIC_STOPWATCH :

% To handle the stopwatch time:

type STOPWATCH_TIME_TYPE;
constant ZERO_STOPWATCH_TIME : STOPWATCH_TIME_TYPE;
function INCREMENT_STOPWATCH_TIME (STOPWATCH_TIME_TYPE) : STOPWATCH_TIME_TYPE;

% To beep:

type BEEP_TYPE;
constant STOPWATCH_BEEP_VALUE : BEEP_TYPE;
function STOPWATCH_BEEP (STOPWATCH_TIME_TYPE) : BEEP_TYPE;
         % returns either the value STOPWATCH_BEEP_VALUE if the stopwatch has
         % to beep or the value NO_BEEP_VALUE otherwise

% Interface

input HS;
input START_STOP_COMMAND;

relation   HS
         # START_STOP_COMMAND;

output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;
output STOPWATCH_RUN_STATUS : boolean;
output BEEP : BEEP_TYPE;

% Body

emit STOPWATCH_TIME (ZERO_STOPWATCH_TIME);

loop

   % stopwatch not running
   emit STOPWATCH_RUN_STATUS (false);
   await START_STOP_COMMAND;

   % starting the stopwatch
   emit STOPWATCH_RUN_STATUS (true);
   emit BEEP (STOPWATCH_BEEP_VALUE);
```

```
   abort
      every HS do
         emit STOPWATCH_TIME (INCREMENT_STOPWATCH_TIME(pre(?STOPWATCH_TIME)));
         emit BEEP (STOPWATCH_BEEP (?STOPWATCH_TIME))
      end every
   when START_STOP_COMMAND;

   % stopping the stopwatch
   emit BEEP (STOPWATCH_BEEP_VALUE)
end loop
end module

% The LAP_FILTER module
%----------------------

module LAP_FILTER :

type STOPWATCH_TIME_TYPE;

% Interface

input LAP_COMMAND;
input BASIC_STOPWATCH_TIME : STOPWATCH_TIME_TYPE;

output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;
output STOPWATCH_LAP_STATUS : boolean;

% Body

loop
   emit STOPWATCH_LAP_STATUS (false);

   % not in LAP mode
   abort
      loop
         emit STOPWATCH_TIME (? BASIC_STOPWATCH_TIME)
      each BASIC_STOPWATCH_TIME
   when LAP_COMMAND;

   % LAP_COMMAND received, enter LAP mode
   emit STOPWATCH_LAP_STATUS (true);
   await LAP_COMMAND

end loop
end module

% The STOPWATCH_RESET_HANDLER module
%-----------------------------------

module STOPWATCH_RESET_HANDLER :

% Interface

input START_STOP_COMMAND,
      LAP_COMMAND;

relation START_STOP_COMMAND # LAP_COMMAND;
```

```
output RESET_STOPWATCH_COMMAND;

% Body

loop
   trap RESET in
      signal STOPWATCH_STOPPED in
         loop
            abort
               sustain STOPWATCH_STOPPED
            when START_STOP_COMMAND;
            await START_STOP_COMMAND
         end loop
      ||
         loop
            await LAP_COMMAND do
               % LAP_COMMAND received when not in LAP mode
               present STOPWATCH_STOPPED then
                  exit RESET
               end
            end await;
            await LAP_COMMAND
         end loop
      end signal
   end trap;
   emit RESET_STOPWATCH_COMMAND
end loop
end module

% The main STOPWATCH module
%-------------------------

module STOPWATCH :

% To handle the stopwatch time:

type STOPWATCH_TIME_TYPE;

% To beep:

type BEEP_TYPE;

% Interface

input HS;
input START_STOP_COMMAND,
      LAP_COMMAND;

relation   HS
         # START_STOP_COMMAND
         # LAP_COMMAND;

output STOPWATCH_TIME : STOPWATCH_TIME_TYPE;

output STOPWATCH_RUN_STATUS : boolean,
       STOPWATCH_LAP_STATUS : boolean;
```

```
output BEEP : BEEP_TYPE;

% Body

signal RESET_STOPWATCH_COMMAND,
       BASIC_STOPWATCH_TIME : STOPWATCH_TIME_TYPE in
   loop
      run BASIC_STOPWATCH [signal BASIC_STOPWATCH_TIME / STOPWATCH_TIME]
   ||
      run LAP_FILTER
   each RESET_STOPWATCH_COMMAND
||
   run STOPWATCH_RESET_HANDLER
end signal
end module
```

11.3 The ALARM module

```
%%%%%%%%%%%%%%%%%%%%
% The ALARM module %
%%%%%%%%%%%%%%%%%%%%

module ALARM :

% To handle the alarm time:

type ALARM_TIME_TYPE;
constant INITIAL_ALARM_TIME : ALARM_TIME_TYPE;
function TOGGLE_24H_MODE_IN_ALARM_TIME (ALARM_TIME_TYPE) : ALARM_TIME_TYPE;

% To set the alarm time:

type ALARM_TIME_POSITION;
constant INITIAL_ALARM_TIME_POSITION : ALARM_TIME_POSITION;
function NEXT_ALARM_TIME_POSITION (ALARM_TIME_POSITION) : ALARM_TIME_POSITION;
         % say from hours to 10-minutes to minutes and circularly
         % (not relevant for ESTEREL)
function SET_ALARM_TIME (ALARM_TIME_TYPE, ALARM_TIME_POSITION)
                        : ALARM_TIME_TYPE;
         % applies a setting command

% To communicate with a watch:

type WATCH_TIME_TYPE;
function COMPARE_ALARM_TIME_TO_WATCH_TIME
                          (ALARM_TIME_TYPE, WATCH_TIME_TYPE) : boolean;

% To beep:

type BEEP_TYPE;
constant ALARM_BEEP_VALUE : BEEP_TYPE;
constant ALARM_DURATION : integer;

% Interface

input TOGGLE_24H_MODE_COMMAND;

output ALARM_TIME : ALARM_TIME_TYPE;

input ENTER_SET_ALARM_MODE_COMMAND,
      SET_ALARM_COMMAND,
      NEXT_ALARM_TIME_POSITION_COMMAND,
      EXIT_SET_ALARM_MODE_COMMAND;

output START_ENHANCING : ALARM_TIME_POSITION,
       STOP_ENHANCING : ALARM_TIME_POSITION;

input WATCH_TIME : WATCH_TIME_TYPE;
input WATCH_BEING_SET;

input S;
input TOGGLE_ALARM_COMMAND,
      STOP_ALARM_BEEP_COMMAND;
```

```
output ALARM_STATUS : boolean;
output BEEP : BEEP_TYPE;

relation WATCH_BEING_SET => WATCH_TIME;

         % all the other signals are pairwise incompatible,
         % except STOP_ALARM_BEEP_COMMAND that may appear anytime
relation   S
         # TOGGLE_24H_MODE_COMMAND
         # TOGGLE_ALARM_COMMAND
         # ENTER_SET_ALARM_MODE_COMMAND
         # SET_ALARM_COMMAND
         # NEXT_ALARM_TIME_POSITION_COMMAND
         # EXIT_SET_ALARM_MODE_COMMAND;

% Body

signal START_BEEPING in

   % initializations

   emit ALARM_TIME (INITIAL_ALARM_TIME);
   emit ALARM_STATUS (false);

   % main loop

   loop
      % normal mode
      abort % when ENTER_SET_ALARM_MODE_COMMAND
         loop
            await
               case TOGGLE_24H_MODE_COMMAND do
                  emit ALARM_TIME
                         (TOGGLE_24H_MODE_IN_ALARM_TIME(pre(?ALARM_TIME)))
               case TOGGLE_ALARM_COMMAND do
                  emit ALARM_STATUS (not(pre(?ALARM_STATUS)))
               case WATCH_TIME do
                  present WATCH_BEING_SET else
                     if COMPARE_ALARM_TIME_TO_WATCH_TIME
                                          (?ALARM_TIME, ? WATCH_TIME)
                        and ?ALARM_STATUS
                     then
                        emit START_BEEPING
                     end if
                  end present
            end await
         end loop
      when ENTER_SET_ALARM_MODE_COMMAND;

      % set-alarm mode
      % the currently set position is the value of START_ENHANCING
      % (one might also accept TOGGLE_24H_MODE_COMMAND
      % and TOGGLE_ALARM_COMMAND; for this one just has to
      % copy the corresponding cases above into the next await).
      % Notice that the alarm does not ring in set mode

      abort % when EXIT_SET_ALARM_MODE_COMMAND
```

```
         emit START_ENHANCING (INITIAL_ALARM_TIME_POSITION);
         loop
            await
               case SET_ALARM_COMMAND do
                   emit ALARM_TIME (SET_ALARM_TIME (pre(?ALARM_TIME),
                                                    ?START_ENHANCING))
               case NEXT_ALARM_TIME_POSITION_COMMAND do
                   emit STOP_ENHANCING (pre(?START_ENHANCING));
                   emit START_ENHANCING
                         (NEXT_ALARM_TIME_POSITION(pre(?START_ENHANCING)))
            end await
         end loop
      when EXIT_SET_ALARM_MODE_COMMAND;
      emit STOP_ENHANCING (?START_ENHANCING);
      emit ALARM_STATUS (true)
   end loop
||
   % beeping sequence
   every START_BEEPING do
      abort
         loop emit BEEP (ALARM_BEEP_VALUE) each S
      when
         case STOP_ALARM_BEEP_COMMAND
         case ALARM_DURATION S
      end abort
   end every
end signal
end module
```

11.4 The BUTTON module

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The BUTTON module for watch, stopwatch1, alarm %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module BUTTON :

% Input interface

input UL, % upper-left button
      UR, % upper-right button
      LL, % lower-left button
      LR; % lower-right button

relation UL # UR # LL # LR;    % all buttons are incompatible

% Outputs for the watch mode

output WATCH_MODE_COMMAND;

output ENTER_SET_WATCH_MODE_COMMAND,
       SET_WATCH_COMMAND,
       NEXT_WATCH_TIME_POSITION_COMMAND,
       EXIT_SET_WATCH_MODE_COMMAND;

output TOGGLE_24H_MODE_COMMAND; % also to the alarm
output TOGGLE_CHIME_COMMAND;

% Outputs for the stopwatch mode

output STOPWATCH_MODE_COMMAND;

output START_STOP_COMMAND,
       LAP_COMMAND;

% Outputs for the alarm mode

output ALARM_MODE_COMMAND;

output ENTER_SET_ALARM_MODE_COMMAND,
       SET_ALARM_COMMAND,
       NEXT_ALARM_TIME_POSITION_COMMAND,
       EXIT_SET_ALARM_MODE_COMMAND;

output TOGGLE_ALARM_COMMAND,
       STOP_ALARM_BEEP_COMMAND;
```

```
% Body

[
   loop

      % Watch and set-watch mode (exit by LL in watch mode only,
      %                           not in set-watch mode)

      emit WATCH_MODE_COMMAND;
      trap WATCH_AND_SET_WATCH_MODE in
         loop

            % watch mode

            abort       % when UL, that turns to set-watch mode
                  await LL do exit WATCH_AND_SET_WATCH_MODE end
               ||
                  every LR do emit TOGGLE_24H_MODE_COMMAND end
            when UL;

            % set-watch mode

            emit ENTER_SET_WATCH_MODE_COMMAND;
            abort       % when UL, that brings back to watch mode
                  every LL do emit NEXT_WATCH_TIME_POSITION_COMMAND end
               ||
                  every LR do emit SET_WATCH_COMMAND end
            when UL;
            emit EXIT_SET_WATCH_MODE_COMMAND
         end loop
      end trap;

      % Stopwatch mode (exit by LL); LR is START/STOP, UR is LAP

      emit STOPWATCH_MODE_COMMAND;
      abort       % when LL
            every LR do emit START_STOP_COMMAND end
         ||
            every UR do emit LAP_COMMAND end
      when LL;

      % Alarm and set-alarm mode (exit by LL in alarm mode only,
      %                           not in set-alarm mode)

      trap ALARM_AND_SET_ALARM_MODE in
         emit ALARM_MODE_COMMAND;
         loop

            % alarm mode

            abort       % when UL, that turns to set-alarm mode
                  await LL do exit ALARM_AND_SET_ALARM_MODE end
               ||
                  every LR do emit TOGGLE_CHIME_COMMAND end
               ||
                  every UR do emit TOGGLE_ALARM_COMMAND end
            when UL;
```

```
            % set-alarm mode

            emit ENTER_SET_ALARM_MODE_COMMAND;

            abort % when UL, that returns to alarm mode
                  every LL do emit NEXT_ALARM_TIME_POSITION_COMMAND end
               ||
                  every LR do emit SET_ALARM_COMMAND end
            when UL;
            emit EXIT_SET_ALARM_MODE_COMMAND
         end loop
      end trap
   end loop

||

   % transforms permanently UR into STOP_ALARM_BEEP_COMMAND

   every UR do emit STOP_ALARM_BEEP_COMMAND end
]
end module
```

11.5 The DISPLAY module

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The DISPLAY module for watch, stopwatch, alarm %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module DISPLAY :

% For the main display:

type MAIN_DISPLAY_TYPE;

output MAIN_DISPLAY : MAIN_DISPLAY_TYPE;

% For the mini display:

type MINI_DISPLAY_TYPE;

output MINI_DISPLAY : MINI_DISPLAY_TYPE;

% For the alphabetic display:

output ALPHABETIC_DISPLAY : string;

% For display positions:

type DISPLAY_POSITION;

output START_ENHANCING : DISPLAY_POSITION,
       STOP_ENHANCING : DISPLAY_POSITION;

% To handle the watch:

type WATCH_TIME_TYPE;
function WATCH_TIME_TO_MAIN_DISPLAY (WATCH_TIME_TYPE) : MAIN_DISPLAY_TYPE,
         WATCH_TIME_TO_MINI_DISPLAY (WATCH_TIME_TYPE) : MINI_DISPLAY_TYPE,
         WATCH_DATE_TO_MINI_DISPLAY (WATCH_TIME_TYPE) : MINI_DISPLAY_TYPE,
         WATCH_DAY_TO_ALPHABETIC_DISPLAY (WATCH_TIME_TYPE) : string;


type WATCH_TIME_POSITION;
function WATCH_DISPLAY_POSITION (WATCH_TIME_POSITION) : DISPLAY_POSITION;


input WATCH_MODE_COMMAND;
input WATCH_TIME : WATCH_TIME_TYPE;
input WATCH_START_ENHANCING : WATCH_TIME_POSITION,
      WATCH_STOP_ENHANCING : WATCH_TIME_POSITION;

% To handle the stopwatch:

type STOPWATCH_TIME_TYPE;
function STOPWATCH_TIME_TO_MAIN_DISPLAY
                              (STOPWATCH_TIME_TYPE) : MAIN_DISPLAY_TYPE;


input STOPWATCH_MODE_COMMAND;
input STOPWATCH_TIME : STOPWATCH_TIME_TYPE;
```

```
% To handle the alarm:

type ALARM_TIME_TYPE;
function ALARM_TIME_TO_MAIN_DISPLAY (ALARM_TIME_TYPE) : MAIN_DISPLAY_TYPE;

type ALARM_TIME_POSITION;
function ALARM_DISPLAY_POSITION (ALARM_TIME_POSITION) : DISPLAY_POSITION;

input ALARM_MODE_COMMAND;
input ALARM_TIME : ALARM_TIME_TYPE;
input ALARM_START_ENHANCING : ALARM_TIME_POSITION,
      ALARM_STOP_ENHANCING : ALARM_TIME_POSITION;

% Global input relations; the 3 modes are mutually incompatible:

relation WATCH_MODE_COMMAND # STOPWATCH_MODE_COMMAND # ALARM_MODE_COMMAND;

% Body of DISPLAY

loop

   % In watch mode, the main display shows the regular time
   % and the mini display shows the date

   abort  % when STOPWATCH_MODE_COMMAND
      loop
         emit MAIN_DISPLAY (WATCH_TIME_TO_MAIN_DISPLAY (? WATCH_TIME));
         emit MINI_DISPLAY (WATCH_DATE_TO_MINI_DISPLAY (? WATCH_TIME));
         emit ALPHABETIC_DISPLAY
                     (WATCH_DAY_TO_ALPHABETIC_DISPLAY (? WATCH_TIME))
      each WATCH_TIME
   ||
      every WATCH_START_ENHANCING do
         emit START_ENHANCING (WATCH_DISPLAY_POSITION
                                    (? WATCH_START_ENHANCING))
      end
   ||
      every WATCH_STOP_ENHANCING do
         emit STOP_ENHANCING (WATCH_DISPLAY_POSITION
                                    (? WATCH_STOP_ENHANCING))
      end
   when STOPWATCH_MODE_COMMAND;

   % Stopwatch and alarm modes

   abort  % when  WATCH_MODE_COMMAND

      % The mini display contains the regular watch time
      loop
         emit MINI_DISPLAY (WATCH_TIME_TO_MINI_DISPLAY (? WATCH_TIME))
      each WATCH_TIME
   ||
```

```
      % Stopwatch mode

      abort
         emit ALPHABETIC_DISPLAY("ST");
         loop
            emit MAIN_DISPLAY (STOPWATCH_TIME_TO_MAIN_DISPLAY
                                              (? STOPWATCH_TIME))
         each STOPWATCH_TIME
      when ALARM_MODE_COMMAND;

      % alarm mode
      abort
         emit ALPHABETIC_DISPLAY ("AL");
         loop
            emit MAIN_DISPLAY (ALARM_TIME_TO_MAIN_DISPLAY (? ALARM_TIME))
         each ALARM_TIME
      ||
         every ALARM_START_ENHANCING do
            emit START_ENHANCING (ALARM_DISPLAY_POSITION
                                           (? ALARM_START_ENHANCING))
         end
      ||
         every  ALARM_STOP_ENHANCING do
            emit STOP_ENHANCING (ALARM_DISPLAY_POSITION
                                           (? ALARM_STOP_ENHANCING))
         end
      when WATCH_MODE_COMMAND % for easy extensibility!
   when WATCH_MODE_COMMAND
end loop
end module
```

11.6 The WRISTWATCH module

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The main WRISTWATCH module for watch, stopwatch1, alarm %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module WRISTWATCH :

% Data and signal interface
%--------------------------

% The wristwatch buttons

input UL, % upper-left button
      UR, % upper-right button
      LL, % lower-left button
      LR; % lower-right button

% The time units

input HS, % quartz - 1/100 second
      S;  % quartz - second

% The input relations

relation UL # UR # LL # LR # HS,
         S => HS;

% The main display

type MAIN_DISPLAY_TYPE;

output MAIN_DISPLAY : MAIN_DISPLAY_TYPE;

% The mini display

type MINI_DISPLAY_TYPE;

output MINI_DISPLAY : MINI_DISPLAY_TYPE;

% The alphabetic display

output ALPHABETIC_DISPLAY : string;

% The display positions

type DISPLAY_POSITION;

output START_ENHANCING : DISPLAY_POSITION,
       STOP_ENHANCING : DISPLAY_POSITION;

% The watch boolean indicators

output CHIME_STATUS : boolean;
% The stopwatch boolean indicators:

output STOPWATCH_RUN_STATUS : boolean,
       STOPWATCH_LAP_STATUS : boolean;
```

```
% The alarm boolean indicators

output ALARM_STATUS : boolean;

% The beeper and the beep combination function

type BEEP_TYPE;
function COMBINE_BEEPS (BEEP_TYPE, BEEP_TYPE) : BEEP_TYPE;

output BEEP : combine BEEP_TYPE with COMBINE_BEEPS;

% Internal types used in submodule communication
%-----------------------------------------------

% For the watch

type WATCH_TIME_TYPE;
function GET_INITIAL_WATCH_TIME : WATCH_TIME_TYPE;
type WATCH_TIME_POSITION;
constant Il.tex
NITIAL_WATCH_TIME_POSITION : WATCH_TIME_POSITION;

% For the stopwatch:

type STOPWATCH_TIME_TYPE;

% For the alarm:

type ALARM_TIME_TYPE;
type ALARM_TIME_POSITION;

% Body of WRISTWATCH
%-------------------

signal WATCH_MODE_COMMAND,
       STOPWATCH_MODE_COMMAND,
       ALARM_MODE_COMMAND,

       TOGGLE_24H_MODE_COMMAND,
       TOGGLE_CHIME_COMMAND,

       ENTER_SET_WATCH_MODE_COMMAND,
       SET_WATCH_COMMAND,
       NEXT_WATCH_TIME_POSITION_COMMAND,
       EXIT_SET_WATCH_MODE_COMMAND,

       WATCH_TIME : WATCH_TIME_TYPE,
       WATCH_BEING_SET,

       WATCH_START_ENHANCING : WATCH_TIME_POSITION,
       WATCH_STOP_ENHANCING : WATCH_TIME_POSITION,

       START_STOP_COMMAND,
       LAP_COMMAND,
       STOPWATCH_TIME : STOPWATCH_TIME_TYPE,
```

```
       TOGGLE_ALARM_COMMAND,

       ENTER_SET_ALARM_MODE_COMMAND,
       NEXT_ALARM_TIME_POSITION_COMMAND,
       EXIT_SET_ALARM_MODE_COMMAND,
       SET_ALARM_COMMAND,
       STOP_ALARM_BEEP_COMMAND,

       ALARM_TIME : ALARM_TIME_TYPE,

       ALARM_START_ENHANCING : ALARM_TIME_POSITION,
       ALARM_STOP_ENHANCING : ALARM_TIME_POSITION in

   run BUTTON
||
   run WATCH [ signal WATCH_START_ENHANCING / START_ENHANCING,
                      WATCH_STOP_ENHANCING / STOP_ENHANCING ]
||
   run STOPWATCH
||
   run ALARM [ signal ALARM_START_ENHANCING / START_ENHANCING,
                      ALARM_STOP_ENHANCING / STOP_ENHANCING ]
||
   run DISPLAY
end signal

end module
```