Assignment #1: Alarm Clock

BY

Craig Campbell (50111)
And
Paul M. Baillot (#2273596)

Presented to Gregor Bochmann

For the course SEG-2106

University of Ottawa
2008-02-04

Contents

Assignment #1: Software Requirements Specification Critic

Dictionary Of Terms (Definitions)

Alarm Clock: This is the name of the System under design. When the document refers to the "System" or the "Alarm Clock", the document refers to the same entity. The term "Alarm Clock" will be used wherever possible though.

Alarm Time: The "Alarm Time" represents the time of day at which the alarm is triggered.

Alarm Type: One of the two 3-way selector switches present on the Alarm Clock. The three possible selections are either "Off", "Radio" or "Buzzer".

Button (or push button)**:** A device that initiates a response once it is pressed about (i.e.: pressing the "Minute" button to go advance the time of the clock by 1 minute).

Buzzer Volume: One of the two 3-way selector switches present on the Alarm Clock. The "Buzzer Volume" controls the volume at which the buzzer will sound in the case that "Buzzer" is the selected type of alarm and that the alarm is triggered by the Alarm Clock. The buzzer volume can be set to three different levels of intensity, "Low" (level 1), "Medium" (level 2) or "High" (level 3).

Clock: A device that measures intervals of time at a constant rate.

Clock Time: The "clock time" is the device that keeps track of the time of day.

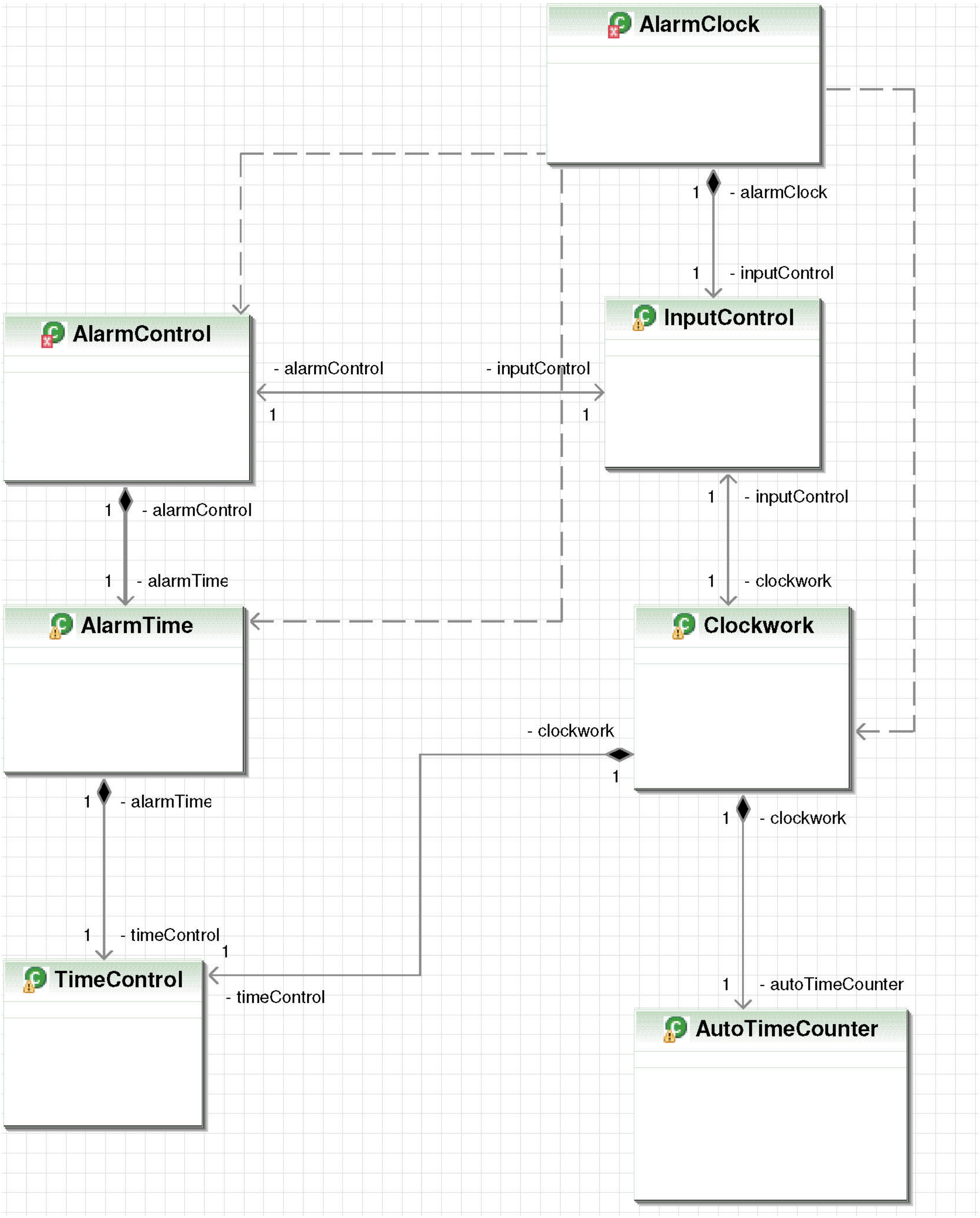
Radio: A communication device that receives electro-magnetic frequencies and then amplifies them into audible sound. *Typical* frequencies can be in the FM or AM range. The Alarm Clock however only plays a single channel on the AM band.

Buzzer: One of the two alarm types in the alarm clock. If the "Buzzer" is the selected alarm type, it will make buzzing sounds once the alarm is triggered by the Alarm Clock.

Selector Switch: A selector switch has a finite amount of possible states. It is always in exactly one of these states. If there are three possible states for example, a selector switch can only be in either state 1, state 2 or state 3. The alarm clock will contain two 3-way selector switches. The first one controls the intensity of the buzzer and can be placed to the "Low", "Medium", or "High" setting. The second selector switch will leave the user select the type of alarm to use, the available options are "Off" to disable the alarm, "Radio" or "Buzzer".

Snooze: A button on the alarm clock, that when pressed while the alarm is sounding, will temporarily stop the alarm from sounding for a period of time (5 minutes on this Alarm Clock). After that certain amount of time has passed, the alarm will resume sounding. Pressing the "snooze" button again will start this process over again.

Conceptual Design



Assumptions And Constraints:

- The 24-hour time format will be used. The time will be printed in the following way: ##:##, hours:minutes respectively.

- The initial *alarm* time will be 00:00.

- The initial *alarm* time, although it is the same as the initial *clock time*, will not trigger the alarm when the *Alarm Clock* (System) is turned on.

- There will be a "Minute" button. Pressing the minute button will increment the *clock time* by one minute. Once this is done, it will take one minute before the clock time increments to the next minute. (i.e.: the seconds counter is reset)

- There will be a "+10 Minutes" button. Pressing the "+10 minutes" button will increment the *clock time* by ten minutes. Once this is done, it will take one minute before the clock time increments to the next minute. (i.e.: the seconds counter is reset)

- When the minute or the +10 minutes button is pressed, if the increment causes the clock time minutes to have a value above 59, then the clock time minutes will now b equal to the current value of minutes minus 60.

- When the minute or the +10 minutes button is pressed, if the increment causes the clock time minutes to have a value above 59, the clock time hour value will be incremented by one. If the clock time hour value becomes higher than 23, than it will start back at 0.

- There will be an "Hour" button. Pressing the hour button will increment the clock time hour value by 1. If the clock time hour value goes beyond 23 than the clock time hour value will start back at 0.

- There will be a "Wake" button. Pressing the wake button will cause the *alarm time* to show on the display.

- Pressing and holding the wake button while then pressing either the minute, +10 minutes or hour button will result in the second button pressed to have almost the same behavior as the actual minute, +10 minutes and hour buttons, with the difference that instead of performing operations on the *clock time*, the operations will be carried out on the *alarm time* instead.

- There will be an "Alarm Type" 3-way *selector*. The three possible settings will be "Off", "Radio" and "Buzzer".

- There will be a "Buzzer Volume" *3-way selector*. The three possible settings will be "Low", "Medium" and "High" (or level 1, 2 and 3 respectively).

- The radio strictly operates on the "580kHz" AM band.

- The radio operates on a unitary volume (the volume cannot be changed).

- If the *alarm type* selected is the radio and the alarm is currently sounding, then switching the *alarm type* to buzzer instead will cause the alarm to immediately cease playing the radio and play the buzzer instead. The contrary is also true.

- If the *alarm type* selected is the buzzer and the alarm is currently sounding, switching the buzzer volume will take effect immediately.

- The only way to turn off the alarm once it is sounding is to place the *alarm type* to "Off" (the alarm type switch can be placed back to the original selection as soon as the alarm ceases to sound).

- When the snooze button is pressed and that the 5 minutes countdown timer is activated, this timer is independent of the clock time timer. (i.e.: If the minutes or hours of the clock time are incremented, this will not affect the snooze feature, which will calmly sound the alarm after 5 minutes regardless, unless the alarm type is turned off…)

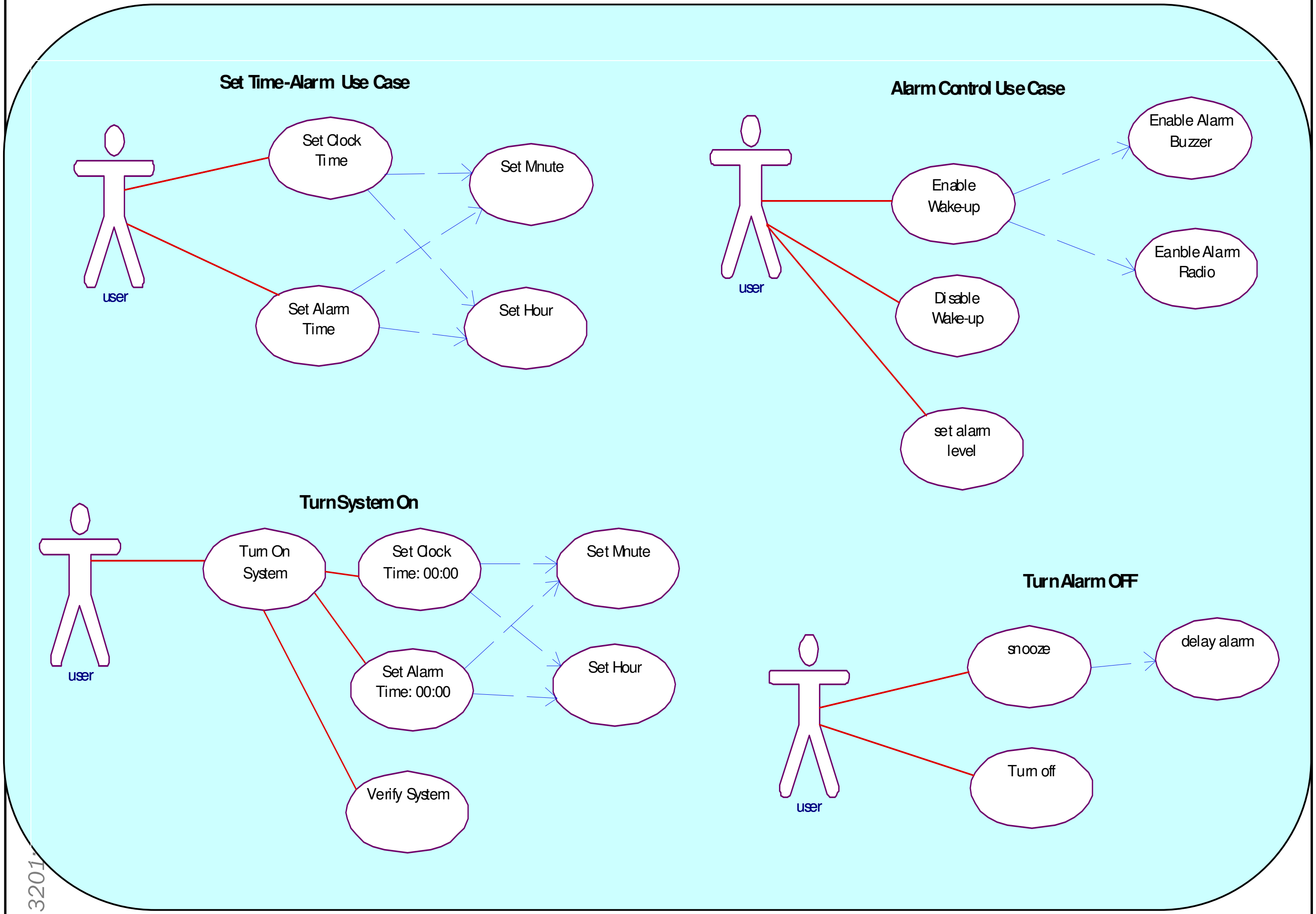
Use Cases

Domain model

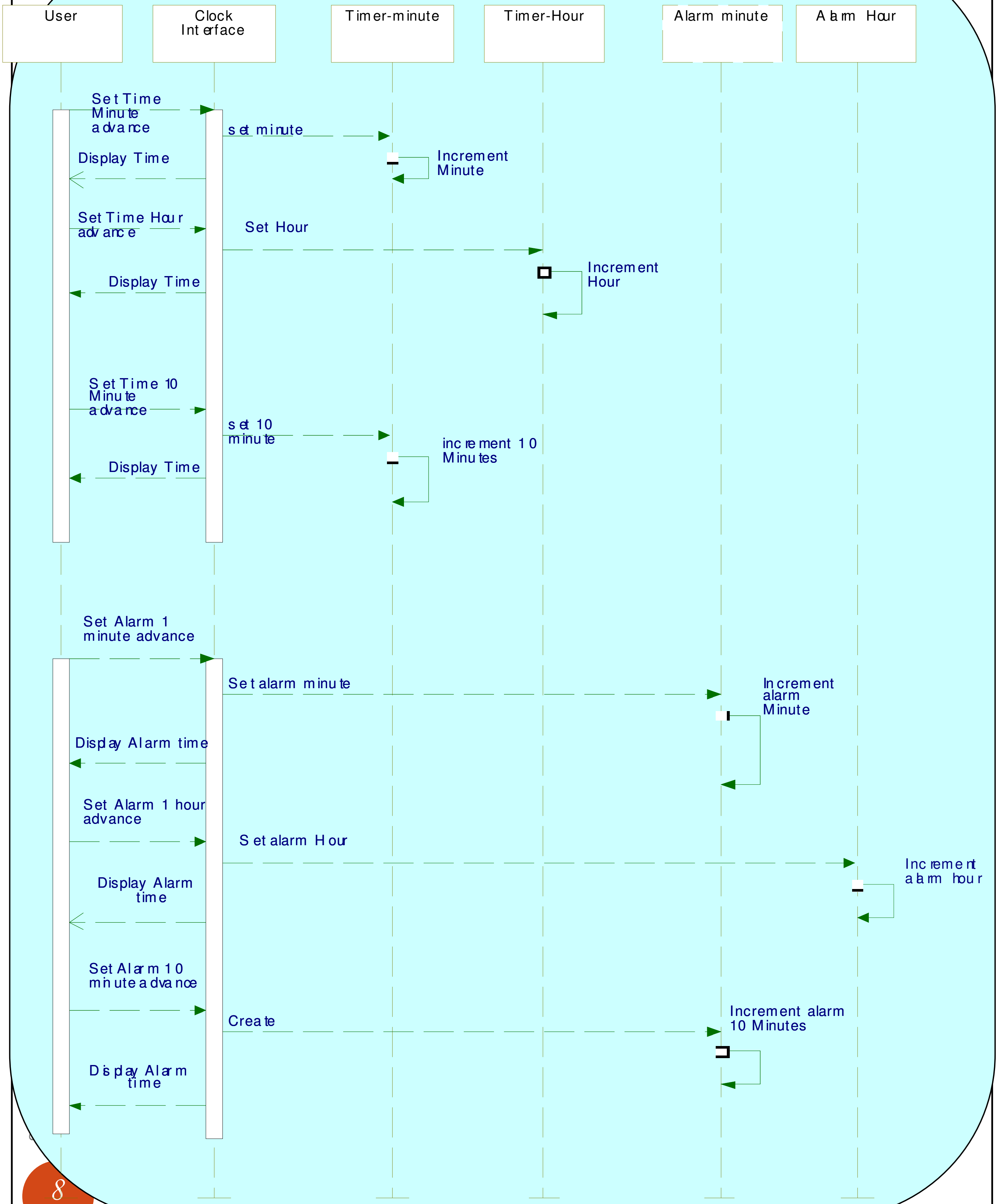
```
domain
  System Concept:System (Activation state of the Alarm Clock)
      Operation Set
      Possible Values Set
          Value:on
          Value:off
  Concept:buzzer volume
      Operation Set
      Possible Values Set
          Value:1
          Value:2
          Value:3
  Concept:alarm type
      Operation Set
      Possible Values Set
          Value:off
          Value:Radio
          Value:Buzzer
Concept:alarm
      Operation Set
      Possible Values Set
          Value:Sounding
          Value:Not Sounding
```

Use case model

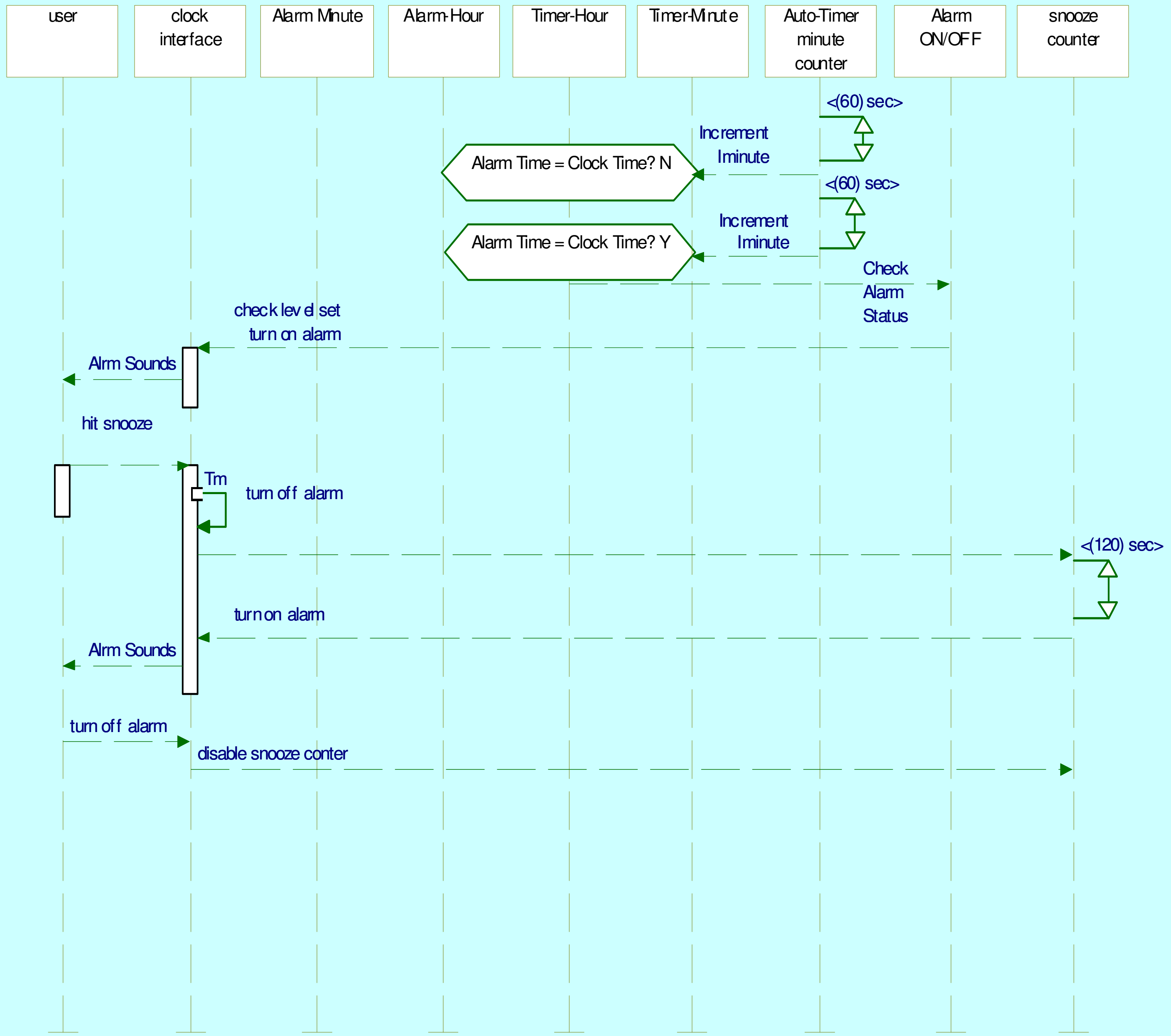
```
use-case-model
  Use Case:Turn On System
  Use Case:Turn Off System
  Use Case:Set Buzzer Volume to Low
  Use Case:Set Buzzer Volume to Medium
  Use Case:Set Buzzer Volume to High
  Use Case:Set Alarm Type to Off
  Use Case:Set Alarm Type to Radio
  Use Case:Set Alarm Type to Buzzer
  Use Case:Press the "Minute" Button
  Use Case:Press the "10 Minute" Button
  Use Case:Press the "Hour" Button
  Use Case:Press the "Wake" button
  Use Case:Press the "Wake+Minute" buttons
  Use Case:Press the "Wake+10 Minute" buttons
  Use Case:Press the "Wake+Hour" buttons
  Use Case:Alarm sounds
  Actor:User
```

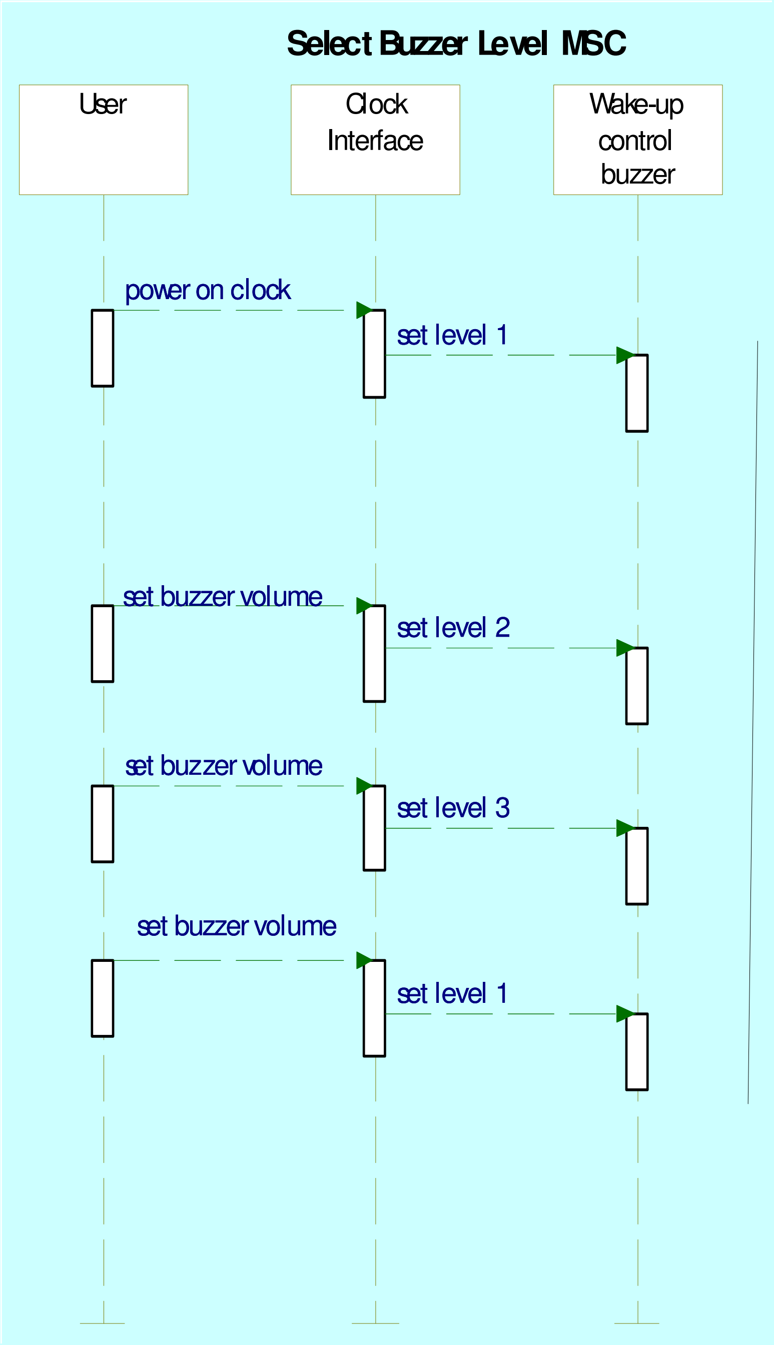


Alarm Clock Time/Alarm Set Conceptual MSC



Alarm Activation/ Snooze & Stop MSC




Select Buzzer Level MSC
User
Clock Interface
Wake-up control buzzer
power on clock
set level 1
set buzzer volume
set level 2
set buzzer volume
set level 3
set buzzer volume
set level 1

Use Case: Turn On System

Title: Turn On System
Description: This is what happens when a user activates the System for the first time. Note that both selector switches are placed on their left most settings when the Alarm Clock is first built, this means that the alarm is "Off" initially and that the buzzer volume is set to the "Low" intensity level (level 1).
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user wants to turn on the System
STEPS
1.User turns on the Alarm Clock
2.Alarm Clock sets the clock time to 00:00
3.Alarm Clock sets the alarm time to 00:00
4.System verifies the buzzer volume selection
5.System verifies the alarm type selection
Success Postcondition: System is on AND Buzzer Volume is 1 AND Alarm Type is Off

Use Case: Turn Off System

Title: Turn Off System
System Under Design: Alarm Clock
Primary Actor: User
Goal: User de-activates the system.
Precondition: System is on
STEPS
1.User turns the System off.
Success Postcondition: System is off

Use Case: Set Buzzer Volume to Low

Title: Set Buzzer Volume to Low
System Under Design: Alarm Clock
Primary Actor: User
Goal: Set the buzzer volume to the "Low" setting.
Precondition: System is on AND Buzzer Volume is not 1
STEPS
1.User moves the buzzer volume selector switch to the "Low" setting
Success Postcondition: Buzzer Volume is 1

Use Case: Set Buzzer Volume to Medium

Title: Set Buzzer Volume to Medium
System Under Design: Alarm Clock
Primary Actor: User
Goal: Set the buzzer volume to the "Medium" setting.
Precondition: System is on AND Buzzer Volume is not 2
STEPS
1.User moves the buzzer volume selector switch to the "Medium" setting
Success Postcondition: Buzzer Volume is 2

Use Case: Set Buzzer Volume to High

Title: Set Buzzer Volume to High
System Under Design: Alarm Clock
Primary Actor: User
Goal: User moves the buzzer volume selector switch to the "High" setting
Precondition: System is on AND Buzzer Volume is not 3
STEPS
1.User moves the buzzer volume selector switch to the "High" setting.
Success Postcondition: Buzzer Volume is 3

Use Case: Set Alarm Type to Off

Title: Set Alarm Type to Off
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user wants to turn the alarm feature off.
Precondition: System is on AND Alarm Type is not Off
STEPS
1.The user moves the "Alarm Type" selector switch to the "Off" position.
2.If the alarm is sounding, then the alarm is turned off.
Success Postcondition: Alarm Type is Off AND Alarm is Not Sounding

Use Case: Set Alarm Type to Radio

Title: Set Alarm Type to Radio
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user wants to switch the "Alarm Type" selector switch to the "Radio" setting.
Precondition: System is on AND Alarm Type is not Radio
STEPS
1.The user moves the "Alarm Type" selector switch to the "Radio" position
2.If the alarm is sounding, then the alarm is switched to the Radio immediately.
Success Postcondition: Alarm Type is Radio

Use Case: Set Alarm Type to Buzzer

Title: Set Alarm Type to Buzzer
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user wants to switch the "Alarm Type" selector switch to the "Buzzer" setting.
Precondition: System is on AND Alarm Type is not Buzzer
STEPS
1.The user moves the "Alarm Type" selector switch to the "Buzzer" position
2.If the alarm is sounding, then the alarm is switched to the Buzzer immediately.
Success Postcondition: Alarm Type is Buzzer

Use Case: Press the "Minute" Button

Title: Press the "Minute" Button
System Under Design: Alarm Clock
Goal: The user presses the "Minute" button in order to increment the clock time by one minute.
Precondition: System is on
STEPS
1.The user presses the "Minute" button.
2.The Alarm Clock increments the clock time minute counter by 1 minute.
3.The Alarm Clock second counter is reset and starts counting from 0 seconds again.
4.If the user passes the 59 minute mark, then the minutes become "0" and the hour is incremented by 1
5.If the hour counter passes the 23 mark, then hour becomes "0"

Use Case: Press the "10 Minute" Button

Title: Press the "10 Minute" Button
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user presses the "10 Minute" button in order to increment the clock time by ten minutes.
Precondition: System is on
STEPS
1.The user presses the "Minute" button.
2.The Alarm Clock increments the clock time minute counter by 10 minutes.
3.The Alarm Clock second counter is reset and starts counting from 0 seconds again.
4.If the user passes the 59 minute mark, then the minutes become "minutes-60" and the hour is incremented by 1
5.If the hour counter passes the 23 mark, then hour becomes "0"

Use Case: Press the "Hour" Button

Title: Press the "Hour" Button
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user presses the "Hour" button in order to increment the clock time by 1 hour.
Precondition: System is on
STEPS
1.The user presses the "Hour" button.
2.The Alarm Clock increments the clock time by 1 hour
3.if the clock time's hour>23 then hour becomes "0".
4.The Alarm Clock second counter is reset and starts counting from 0 seconds again.

Use Case: Press the "Wake" button

Title: Press the "Wake" button
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user presses the "Wake" button in order to look at the current alarm time.
Precondition: System is on
STEPS
1.The user presses the "Wake" button.
2.The Alarm Clock displays the alarm time.

Use Case: Press the "Wake+Minute" buttons

Title: Press the "Wake+Minute" buttons
System Under Design: Alarm Clock
Primary Actor: user
Goal: The user wants to increment the alarm time by one minute.
Precondition: System is on
STEPS
1.The user presses and holds the "Wake" button and then presses the "Minute" button.
2.The alarm time is incremented by 1 minute.
3.if the alarm time minute counter>59 then minute becomes 0 and the alarm time hour counter is incremented by 1
4.if the alarm time hour counter>23 then the alarm time hour counter becomes "0".

Use Case: Press the "Wake+10 Minute" buttons

Title: Press the "Wake+10 Minute" buttons
System Under Design: Alarm Clock
Primary Actor: User
Goal: The user wants to increment the alarm time by one minute.
Precondition: System is on
STEPS
1.The user presses and holds the "Wake" button and then presses the "10 Minute" button.
2.The alarm time is incremented by 10 minutes.
3.if the alarm time minute counter>59 then minute becomes "minute-60" and the alarm time hour counter is incremented by 1
4.if the alarm time hour counter>23 then the alarm time hour counter becomes "0".

Use Case: Press the "Wake+Hour" buttons

Title: Press the "Wake+Hour" buttons
System Under Design: Alarm Clock
Primary Actor: User
Precondition: System is on
STEPS
1.The user presses and holds the "Wake2 button while then pressing the "hour" button.
2.The alarm time hour counter is incremented by 1.
3.If the alarm time hour counter>23 then the alarm time counter becomes "0".

Description Of Simulations Performed

Before we talk about the simulations we performed, I feel that it is important to note that we didn't jump in the Telelogic TAU software making boxes everywhere and then hoping everything would magically work. In actuality we had such a difficult time trying to figure out exactly how it worked that we performed multiple tasks BEFORE attempting to make the SDL diagram and actually (surprisingly even) didn't "find" many problems through the simulation process.

We first started by loading the LAB1 SDL diagram into TAU to study how it was done. Barely understanding anything at that point, we read a great deal from the book about SDL and some points troubled me. One area said that SDL was perfectly able to work in parallel, meaning to send 2 signals at 2 different places at once. Reading further into the book it then warned about this saying that there is no way to really safe guard from bad things happening when different signals simultaneously try to access the same process for example. This really scared us in the sense where we were afraid that if the clock was busy updating its time for example and that just at that time the user would press the "Hour" button for example, that the signal would never reach the clock because the clock was not in an idle state (we come back to this further in the text).

Having completed our list of assumptions, constraints and the use cases, it was now time to "try" and implement this system, however, we still were uncomfortable with SDL at this point. Instead of jumping into it and making a mess that would leave the program in serious need of repair after, we decided to start by designing a UML state chart diagram (see **Appendix A**). While we were drawing this diagram, we were thinking on "how" SDL worked, trying to make the state diagram work in a way that would ease the implementation to SDL. While we were doing this, we found that we had made some seriously wrong assumptions in the way we wanted the system to work. Indeed the mess we were afraid was going to happen while implementing this system in SDL was happening in front of our eyes. First there was the problem on how to synchronize everything together. If you look at the "Alarm Handler" details page on the left side in the box that says "Set WaitTime ← ….", we're using what seems to be a very complicated way of just waiting for the clock to advance to the next minute by use of temporary variables and continuously comparing against the clock time. This was just one problem; the next was the difficulty we were having getting the alarm to act in the way we wanted it to. In the state diagram, the alarm turns off as soon as the Alarm Type is switched, even if it is not switched to "Off". Then there is the problem of the "Snooze" button which persists even if the Alarm Type selector was switched to "Off" and then to something else again. Then there is of course the problem of the "Input Handler" not working as we would have liked, especially if you follow it into the "Wake button pressed" branch.

After having seen all the possible problems we could run into while working on the actual SDL diagram, we then loaded LAB 2, studied the way *it* was done and fixed ALL the problems with it. Once we were done doing that we then finally started the actual SDL diagram for the Alarm Clock.

The first unit we worked on was the "Time Updater" process. We started simply by making a simple timer and making it send a current time output every 60 seconds. The analyzer found quite some syntax problems while just trying to do this, but the simulation ran exactly as desired. We then completed the "Time Updater" process to automatically increment the hours and when to start back at 00:00. At this point, we wanted as few wires going from the "Alarm Clock Block" to the outside, and yet, we wanted this to be as user friendly (or clear) as possible. We needed an output for the time display. We needed a wire to carry in all the input signals, and then finally, we needed an output for the speaker to play the radio or the buzzer. To avoid the problems the book stated, we sought out to create an Input Handler that would receive ALL inputs from the Alarm Clock and then pass them one by one along to the other processes to avoid any synchronization problems. Two alarm processes were then created, one to take care of holding the alarm time and triggering the alarm, and the other to handle the actual alarm such as playing the radio, the buzzer, handling the snooze button as well as playing with the 3-way selectors while the alarm was sounding.

To avoid a deluge of bugs, all the processes created after having finished the "Time Updater" basically all started by containing only a single "Dud" state. A few at a time, inputs were added to the signal list, directed to the input handler and from there the input handler was given added functionality, as well, some implementation started in the processes that originally contained nothing. Each time the implementation of a few signals was done, a full analysis was performed. This is where errors were flying everywhere, mostly syntactical or typo errors though. Once the "Minute", "+10 Minute" and "Hour" inputs were implemented, we ran a simulation and to our great surprise, everything ran as expected. We were grateful for this, because we had paid so much attention to our implementation that a problem at this stage would have baffled us.

Implementation of the "AlarmMan" process was now taking its course, we wanted to get the "Wake" button fully operational and at least be able to set the alarm time. Once the implementation was done, an analysis was ran. Again, errors were flying from everywhere. We were still learning how to use the proper syntax for everything, but we were definitely starting to understand a bit more by now and easily correcting them. A new simulation was run, again like the "Time Updater", everything ran as expected, we had again paid good attention to how we were going about the implementation.

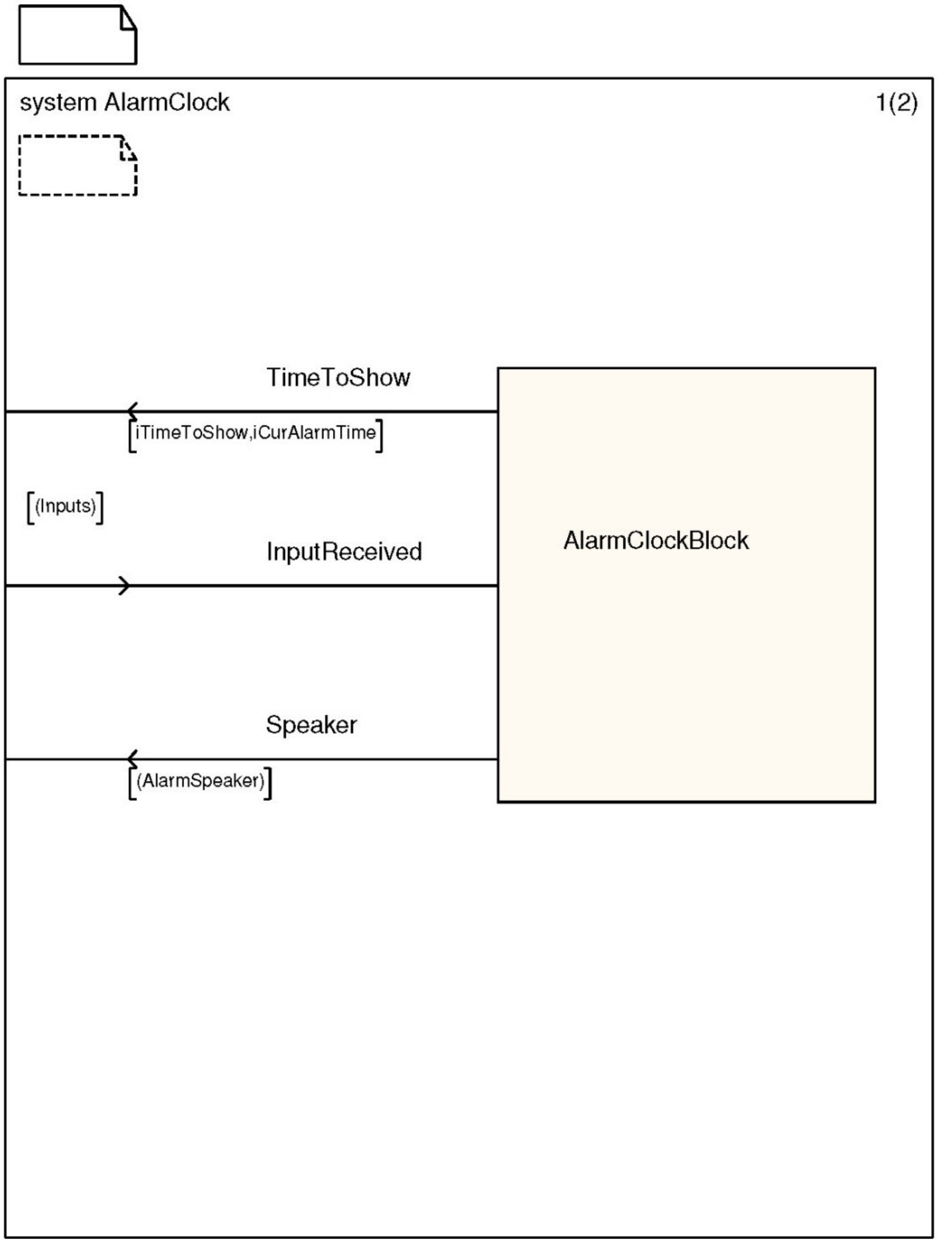
By this time, we were now arriving to the most difficult part, the implementation of the actual alarm and we needed to know "what signal went where and did what when". This is the point where we realized just how much we had despised it in LAB2 in the way that they had used the "Signal lists". Sure, the signal lists removes a lot of the clutter off the screen but does nothing, I mean NOTHING to someone trying to understand what is going on and trying to read and follow the diagram. Our diagram at this point was making use of no "signal lists" and we must say we really liked it this way, we could easily see where everything went, determine how things worked by the name of the signals going to different places, we think at this point is where we really no longer wanted to use signal lists, at least not for the time being. We remembered how hard it was to debug the LAB2 assignment while trying to decipher what signal went where and we didn't want to go through this in my own SDL diagram.

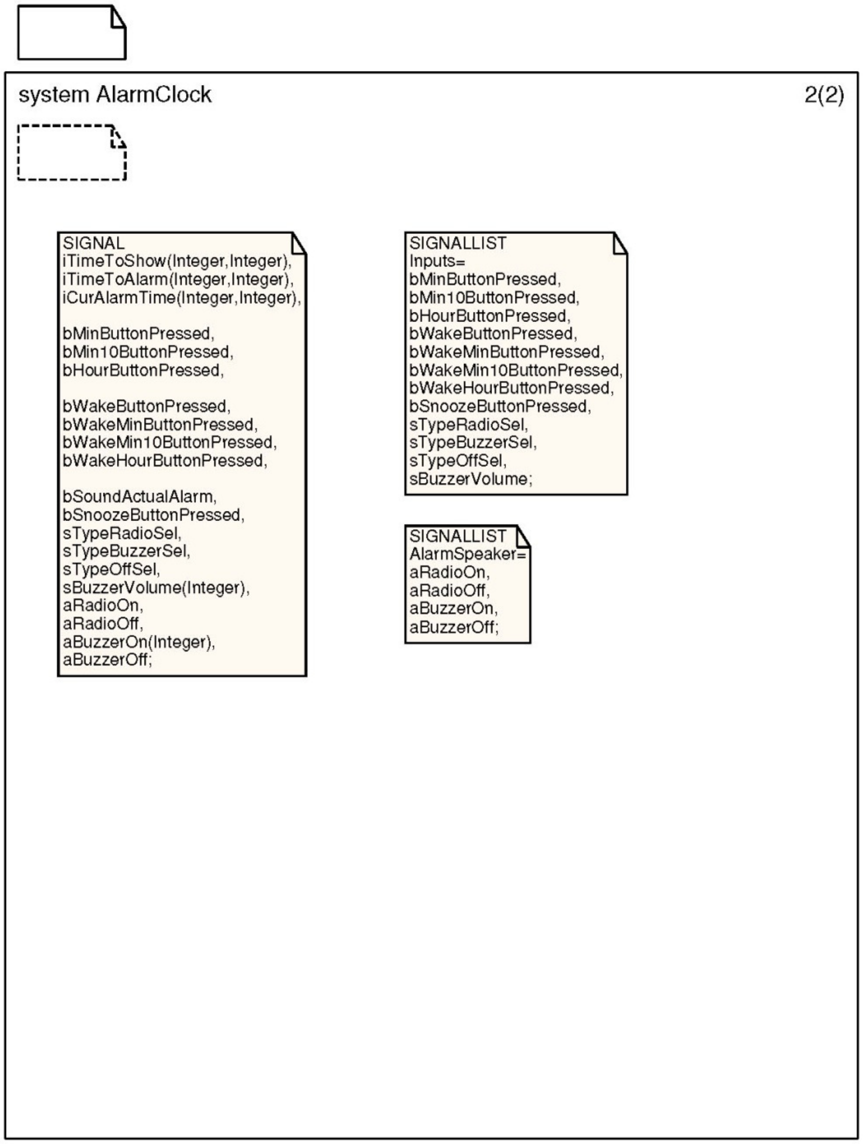
While performing the implementation for the "ActualAlarm" process, errors made in the UML State Diagram were kept in check, however it was still going to be hard not to make any mistakes with the amount of Booleans floating around, and having to perfectly keep all of them in check at all time, where failing to do so at any time would cause serious problems. By this time, we had learned from LAB2, that when the "RESET" function wasn't used to absorb a timer, that a warning would appear in the validation saying that the timer was consumed implicitly every time is timed out and that the signal was ignored. To avoid any warnings at all, we made an effort to reset the snooze timer whenever possible and applicable. To try to minimize problems related to the snooze feature actually, snooze was the second function we worked on in the "Actual Alarm" process so that we had a good idea on how it was supposed to work and keep it in mind while implementing the other parts. The rest of the "Actual Alarm" process was implemented and a full analysis was ran again. This time we were experimenting with the "*", the "-", the "else" keyword and such and we "did" run into some problems with them, they were corrected and then we ran a simulation. For the first time since beginning the implementation process, we saw a definite error occur. We asked (sent a signal to) the system to increase the buzzer volume to 2, and it stayed at 1. The error was an oversight on our parts, we actually wrote "1" three times where it should have been "1", "2" and "3". We also found that the time display would start displaying at 00:01 instead of 00:00, so we added the necessary output for that as well. We re-ran the simulation and this time we gave it the full Monty. We went through absolutely every button, set the alarm, waited for the alarm to sound, switched the buzzer volume while it sounded, switched to radio and back to buzzer while the alarm was on, tried the snooze feature repeatedly while performing different operations in the back (which led us to catching an assumptions we had not anticipated) and everything ran just as expected.

We should also note (this is really a side note) a general revelation we found while performing the various simulations; as stated earlier, we were afraid of what would happen if a signal got to a busy process, or a process currently performing certain tasks and not in any particular states. After looking at how the system behaved when we sent plenty of signals at once, and after a bit more reading, we found that while a process is in between states, it is uninterruptable and signals must wait in a queue so the program ends up running normally in any case. It's a bit strange though because the presence of this magic "buffer" is not explicitly defined anywhere nor is its size…

Now that everything was done, it was time to make a decision whether or not we were going to implement the Signal Lists to "beatify" (so to speak) the SDL diagram and after speaking to the T.A., we finally chose to add two of them even though we still feel that it makes the System harder to read and understand.

SDL System Design





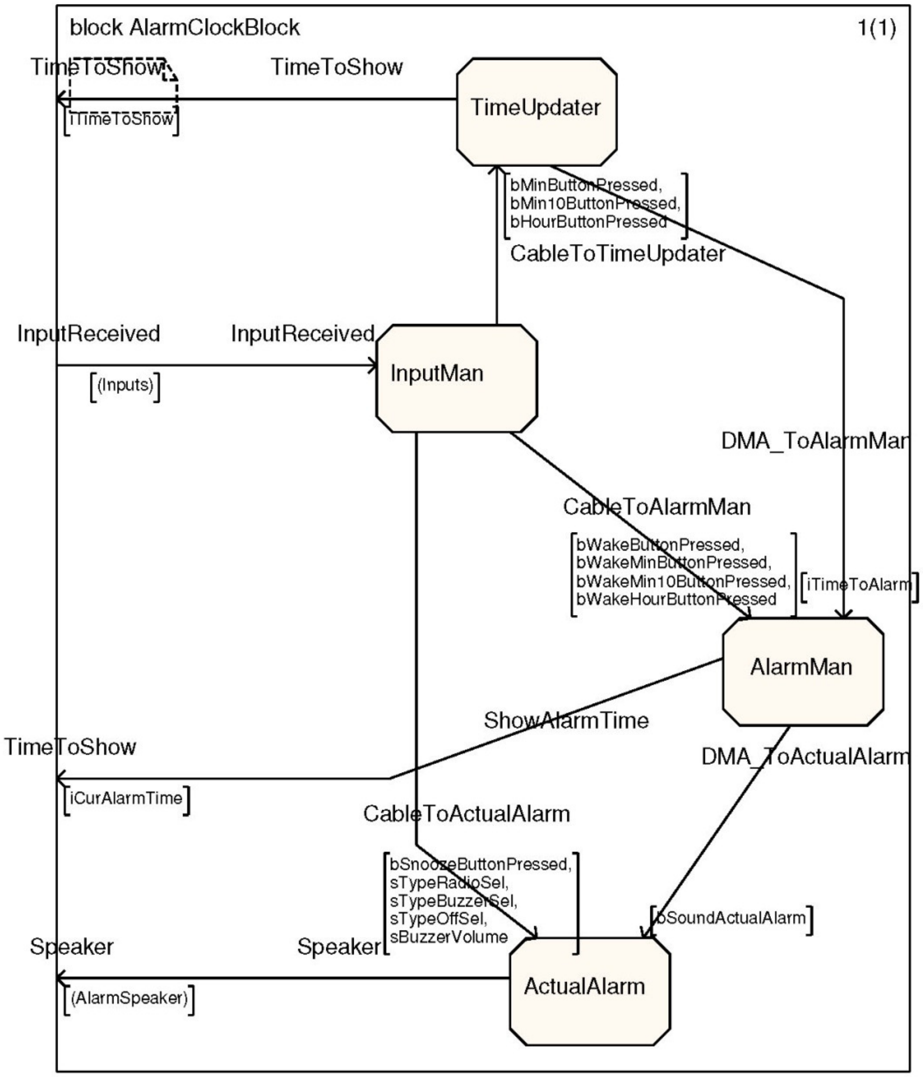
system AlarmClock

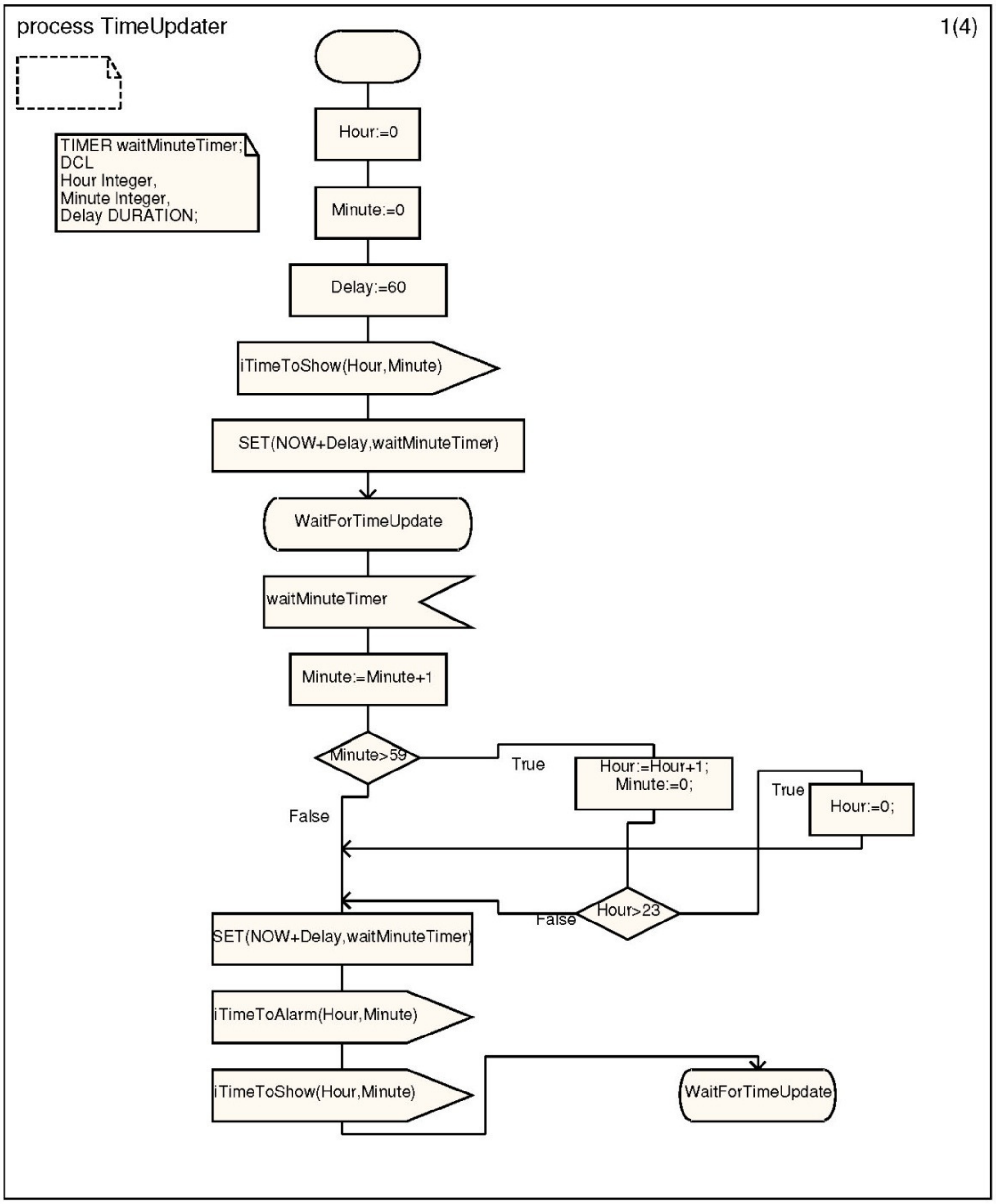
2(2)

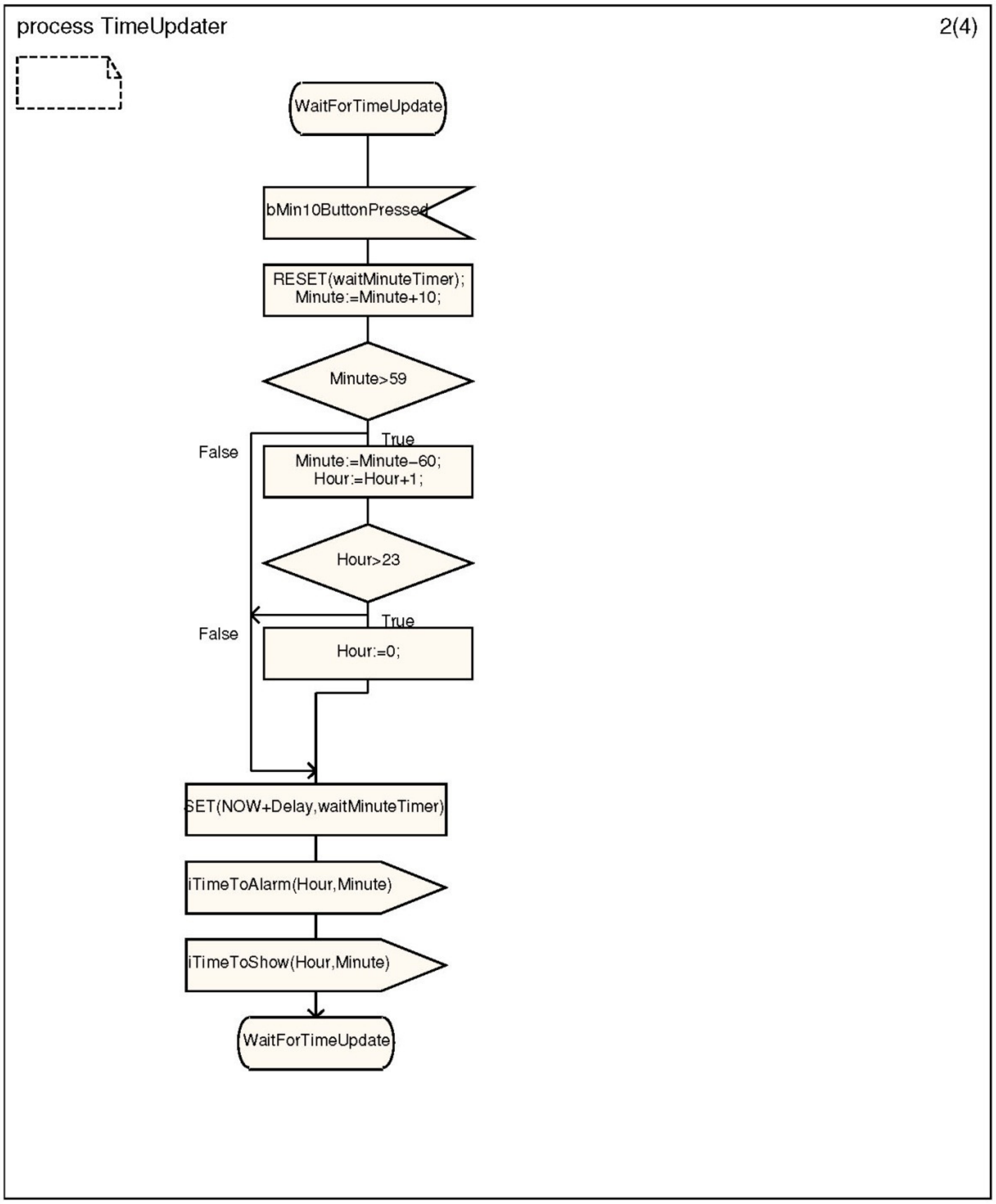
```
SIGNAL
iTimeToShow(Integer,Integer),
iTimeToAlarm(Integer,Integer),
iCurAlarmTime(Integer,Integer),

bMinButtonPressed,
bMin10ButtonPressed,
bHourButtonPressed,

bWakeButtonPressed,
bWakeMinButtonPressed,
bWakeMin10ButtonPressed,
bWakeHourButtonPressed,

bSoundActualAlarm,
bSnoozeButtonPressed,
sTypeRadioSel,
sTypeBuzzerSel,
sTypeOffSel,
sBuzzerVolume(Integer),
aRadioOn,
aRadioOff,
aBuzzerOn(Integer),
aBuzzerOff;
```

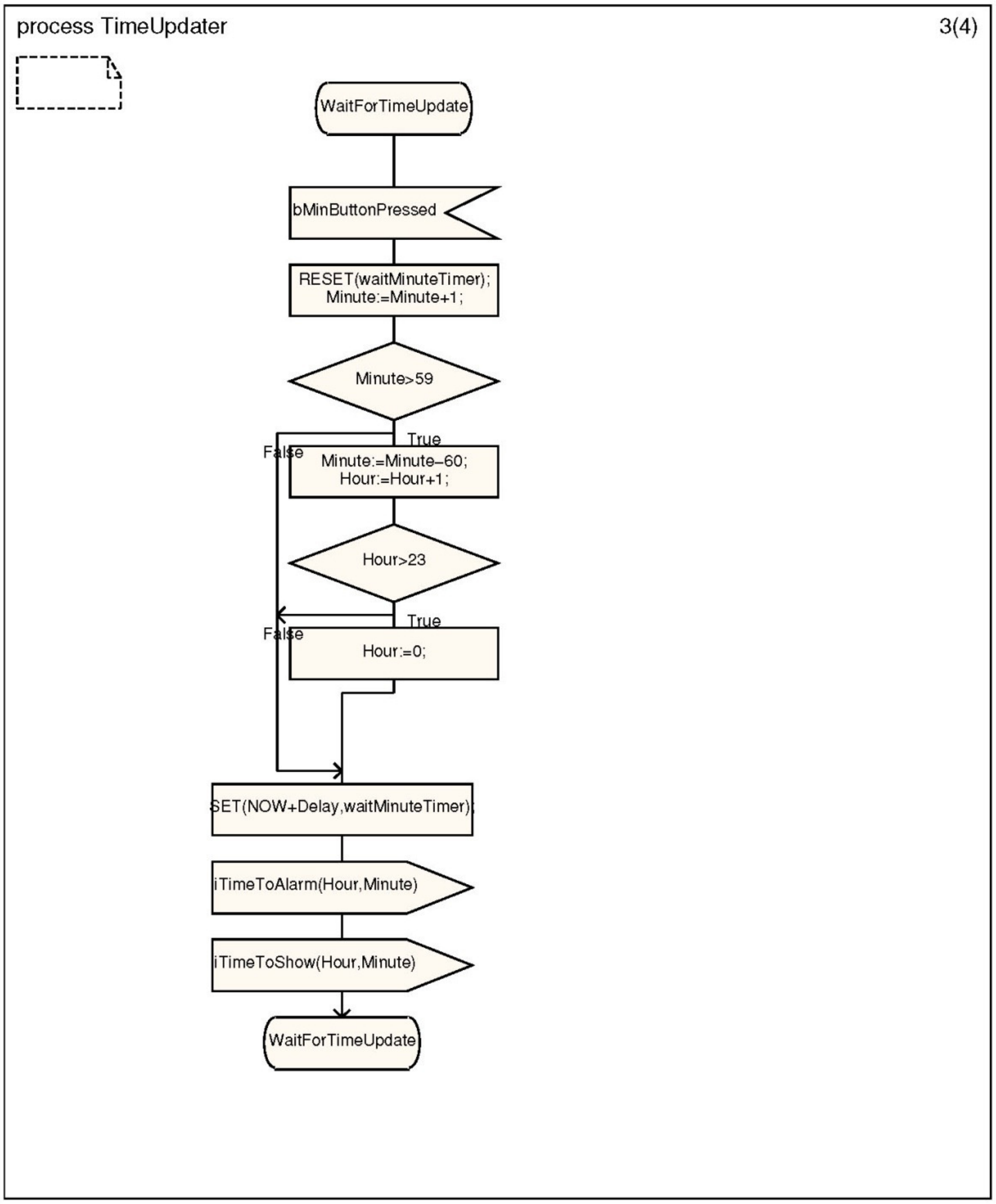
```
SIGNALLIST
Inputs=
bMinButtonPressed,
bMin10ButtonPressed,
bHourButtonPressed,
bWakeButtonPressed,
bWakeMinButtonPressed,
bWakeMin10ButtonPressed,
bWakeHourButtonPressed,
bSnoozeButtonPressed,
sTypeRadioSel,
sTypeBuzzerSel,
sTypeOffSel,
sBuzzerVolume;
```

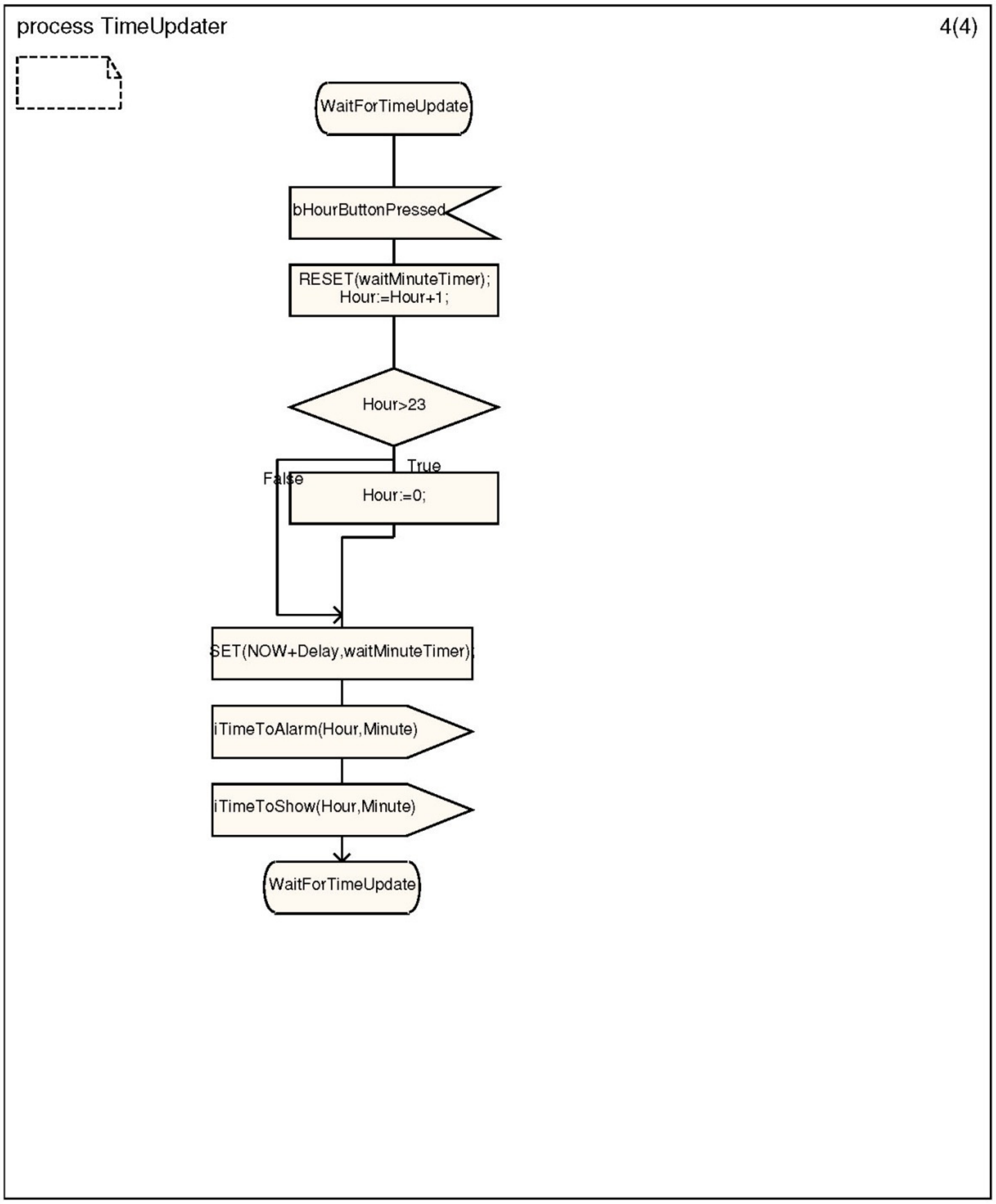
```
SIGNALLIST
AlarmSpeaker=
aRadioOn,
aRadioOff,
aBuzzerOn,
aBuzzerOff;
```

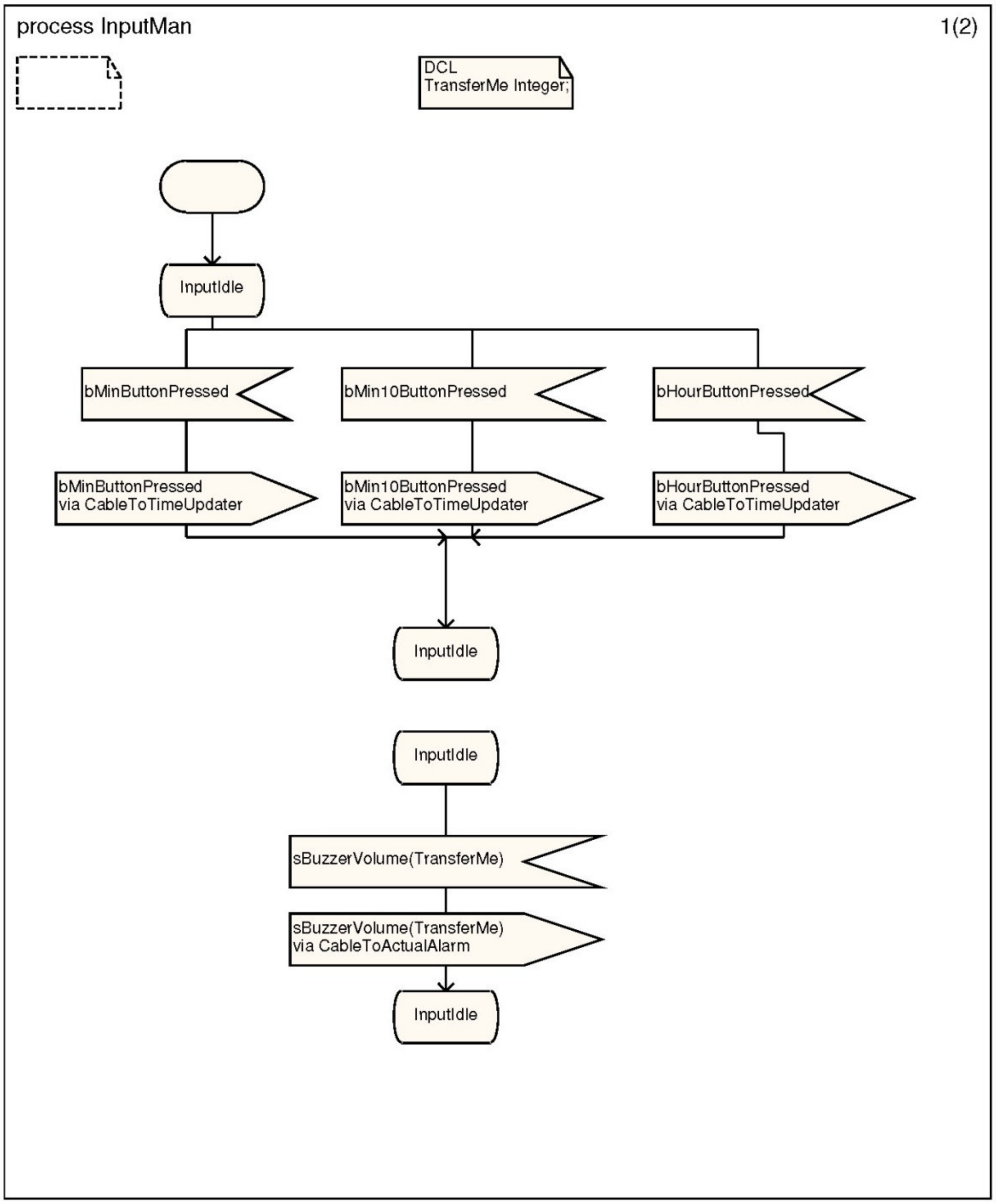

block AlarmClockBlock
1(1)
TimeToShow
TimeToShow
iTimeToShow
TimeUpdater
bMinButtonPressed,
bMin10ButtonPressed,
bHourButtonPressed
CableToTimeUpdater
InputReceived
InputReceived
(Inputs)
InputMan
DMA_ToAlarmMan
CableToAlarmMan
bWakeButtonPressed,
bWakeMinButtonPressed,
bWakeMin10ButtonPressed,
bWakeHourButtonPressed
iTimeToAlarm
AlarmMan
ShowAlarmTime
TimeToShow
DMA_ToActualAlarm
iCurAlarmTime
CableToActualAlarm
bSnoozeButtonPressed,
sTypeRadioSel,
sTypeBuzzerSel,
sTypeOffSel,
sBuzzerVolume
bSoundActualAlarm
Speaker
Speaker
(AlarmSpeaker)
ActualAlarm

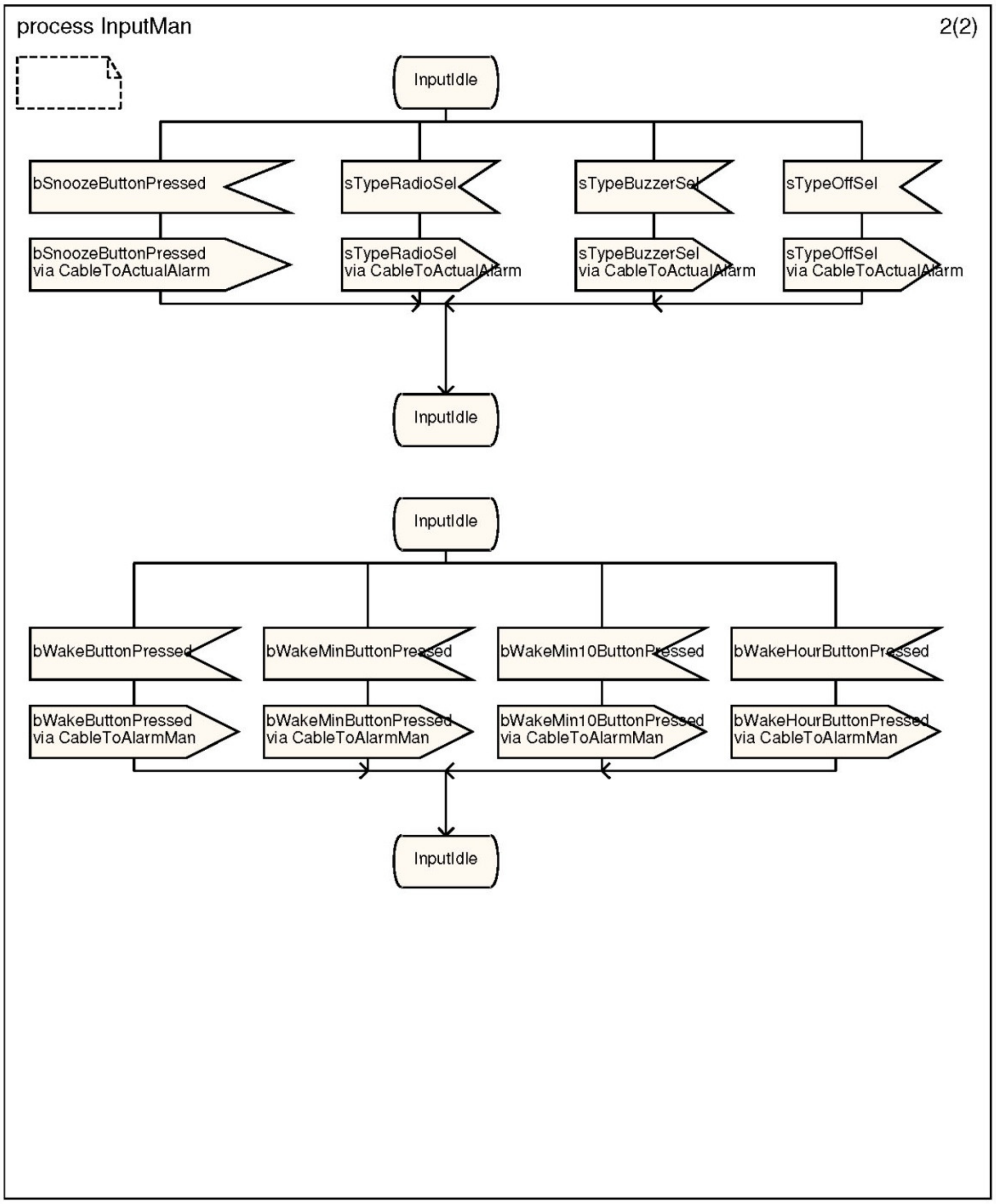

process TimeUpdater
1(4)
TIMER waitMinuteTimer;
DCL
Hour Integer,
Minute Integer,
Delay DURATION;
Hour:=0
Minute:=0
Delay:=60
iTimeToShow(Hour,Minute)
SET(NOW+Delay,waitMinuteTimer)
WaitForTimeUpdate
waitMinuteTimer
Minute:=Minute+1
Minute>59
True
Hour:=Hour+1;
Minute:=0;
True
Hour:=0;
False
Hour>23
False
SET(NOW+Delay,waitMinuteTimer)
iTimeToAlarm(Hour,Minute)
iTimeToShow(Hour,Minute)
WaitForTimeUpdate

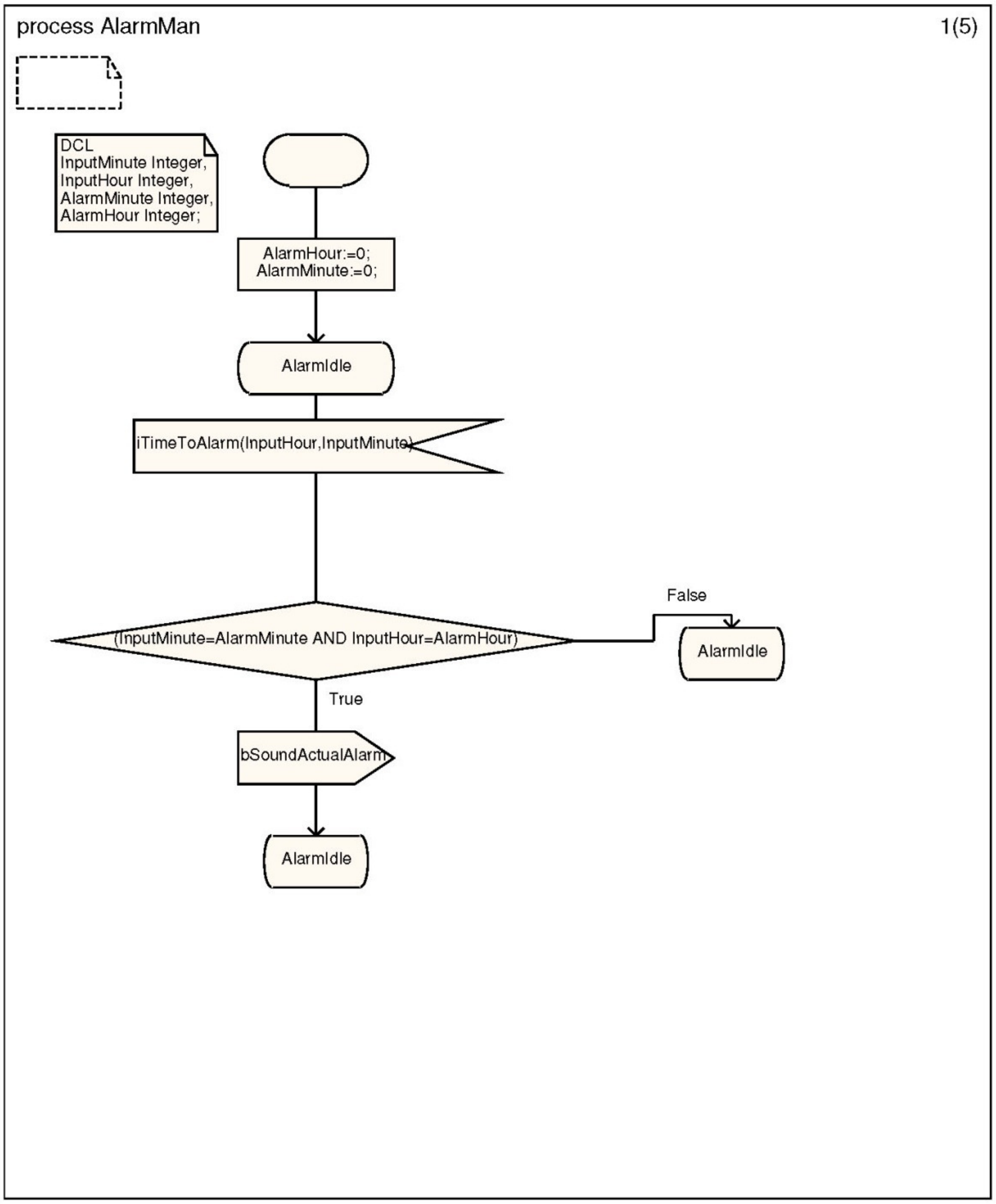

process TimeUpdater
2(4)
WaitForTimeUpdate
bMin10ButtonPressed
RESET(waitMinuteTimer);
Minute:=Minute+10;
Minute>59
True
False
Minute:=Minute−60;
Hour:=Hour+1;
Hour>23
True
False
Hour:=0;
SET(NOW+Delay,waitMinuteTimer)
iTimeToAlarm(Hour,Minute)
iTimeToShow(Hour,Minute)
WaitForTimeUpdate

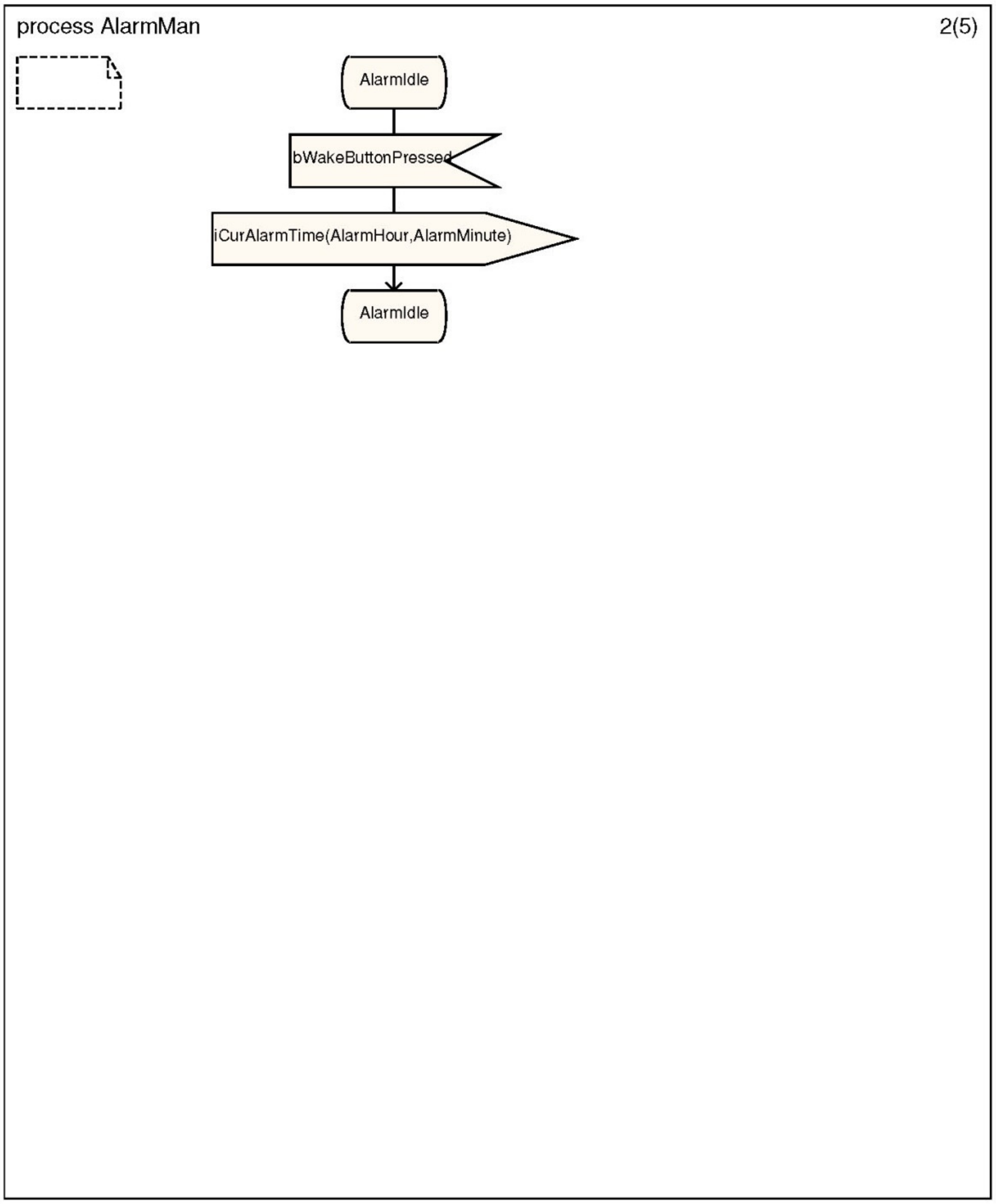

process TimeUpdater
3(4)
WaitForTimeUpdate
bMinButtonPressed
RESET(waitMinuteTimer);
Minute:=Minute+1;
Minute>59
True
False
Minute:=Minute−60;
Hour:=Hour+1;
Hour>23
True
False
Hour:=0;
SET(NOW+Delay,waitMinuteTimer);
iTimeToAlarm(Hour,Minute)
iTimeToShow(Hour,Minute)
WaitForTimeUpdate

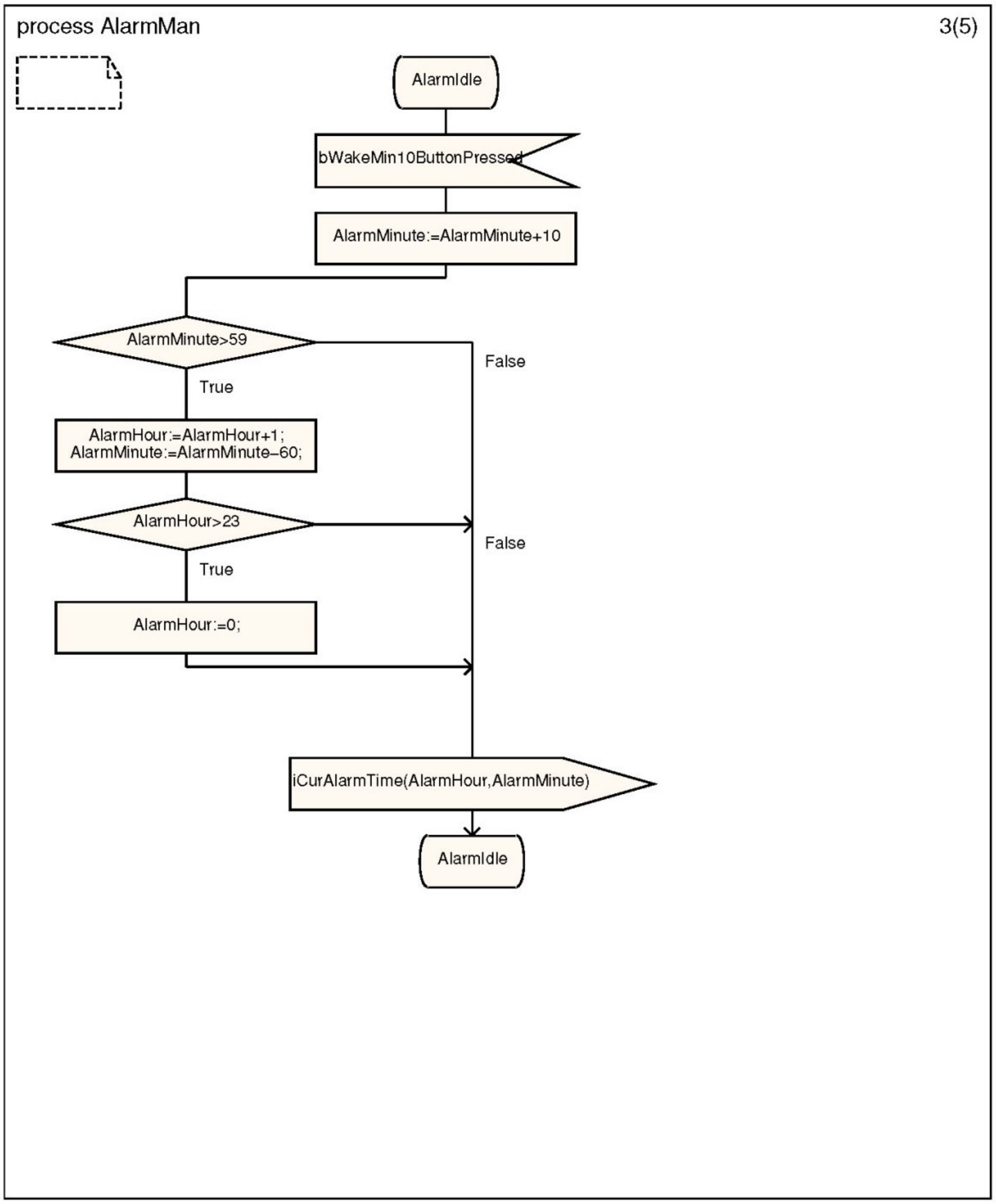

process TimeUpdater
4(4)
WaitForTimeUpdate
bHourButtonPressed
RESET(waitMinuteTimer);
Hour:=Hour+1;
Hour>23
True
False
Hour:=0;
SET(NOW+Delay,waitMinuteTimer);
iTimeToAlarm(Hour,Minute)
iTimeToShow(Hour,Minute)
WaitForTimeUpdate

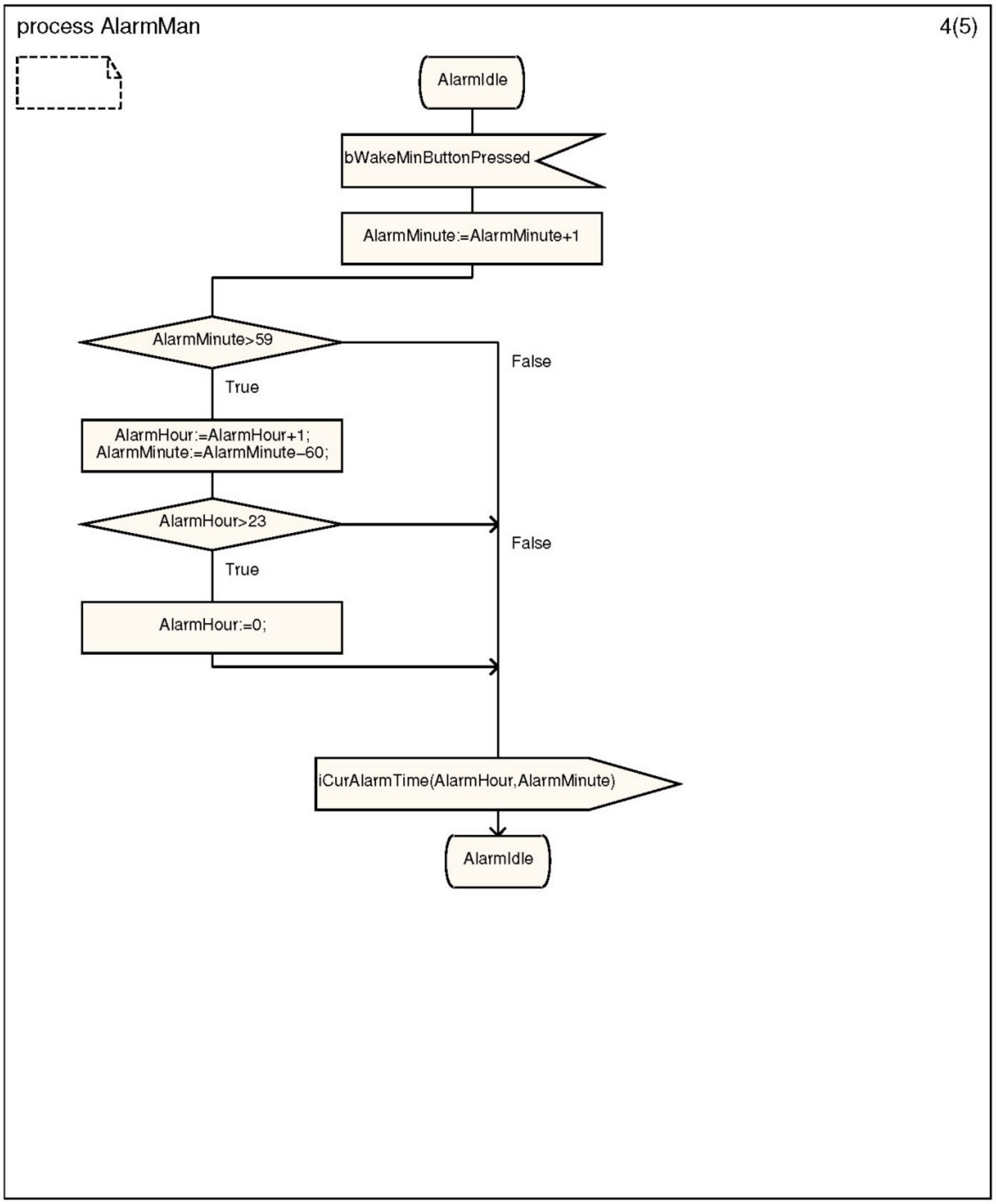

process InputMan
1(2)
DCL
TransferMe Integer;
InputIdle
bMinButtonPressed
bMin10ButtonPressed
bHourButtonPressed
bMinButtonPressed
via CableToTimeUpdater
bMin10ButtonPressed
via CableToTimeUpdater
bHourButtonPressed
via CableToTimeUpdater
InputIdle
InputIdle
sBuzzerVolume(TransferMe)
sBuzzerVolume(TransferMe)
via CableToActualAlarm
InputIdle

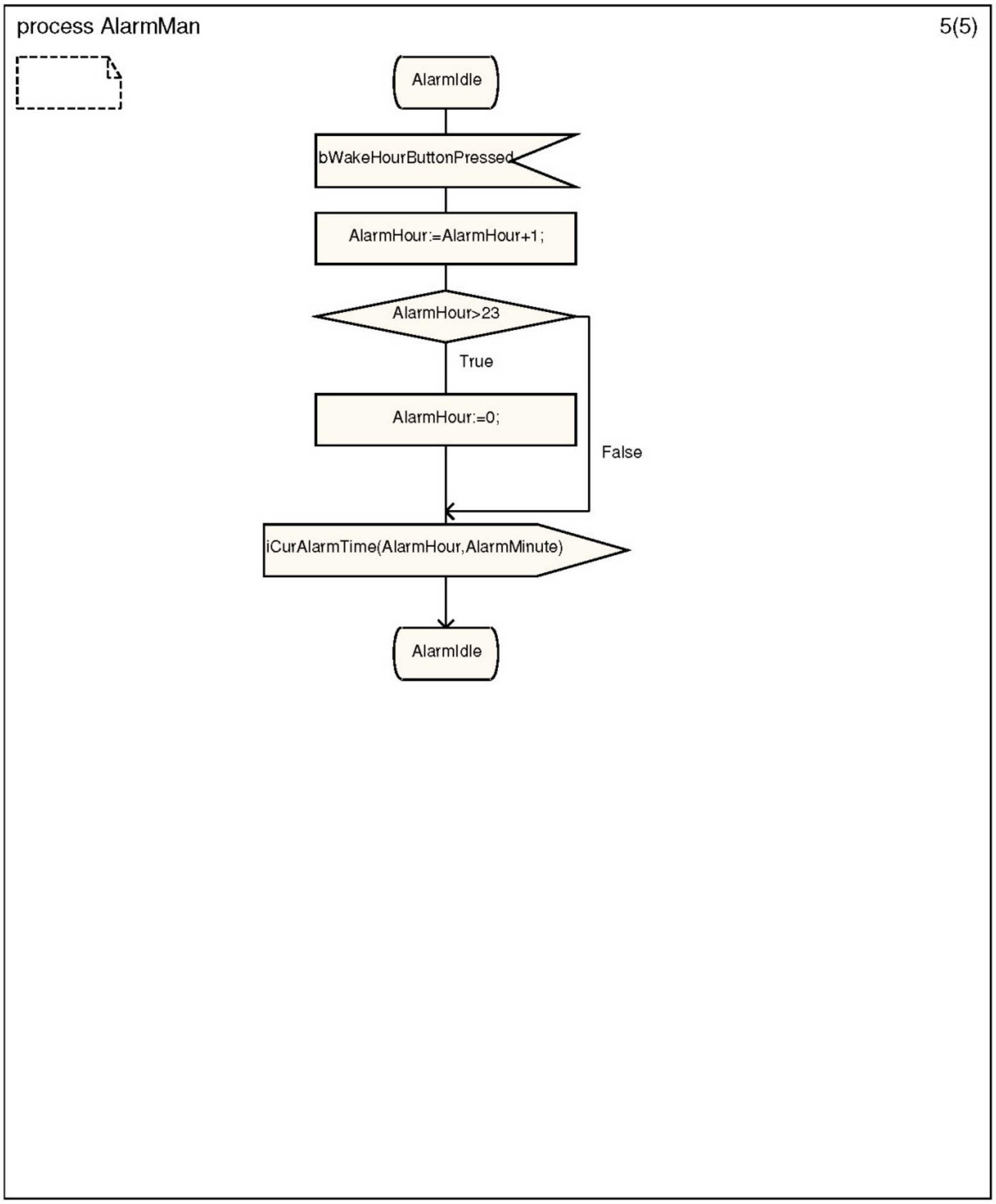

process InputMan
2(2)
InputIdle
bSnoozeButtonPressed
sTypeRadioSel
sTypeBuzzerSel
sTypeOffSel
bSnoozeButtonPressed via CableToActualAlarm
sTypeRadioSel via CableToActualAlarm
sTypeBuzzerSel via CableToActualAlarm
sTypeOffSel via CableToActualAlarm
InputIdle
InputIdle
bWakeButtonPressed
bWakeMinButtonPressed
bWakeMin10ButtonPressed
bWakeHourButtonPressed
bWakeButtonPressed via CableToAlarmMan
bWakeMinButtonPressed via CableToAlarmMan
bWakeMin10ButtonPressed via CableToAlarmMan
bWakeHourButtonPressed via CableToAlarmMan
InputIdle

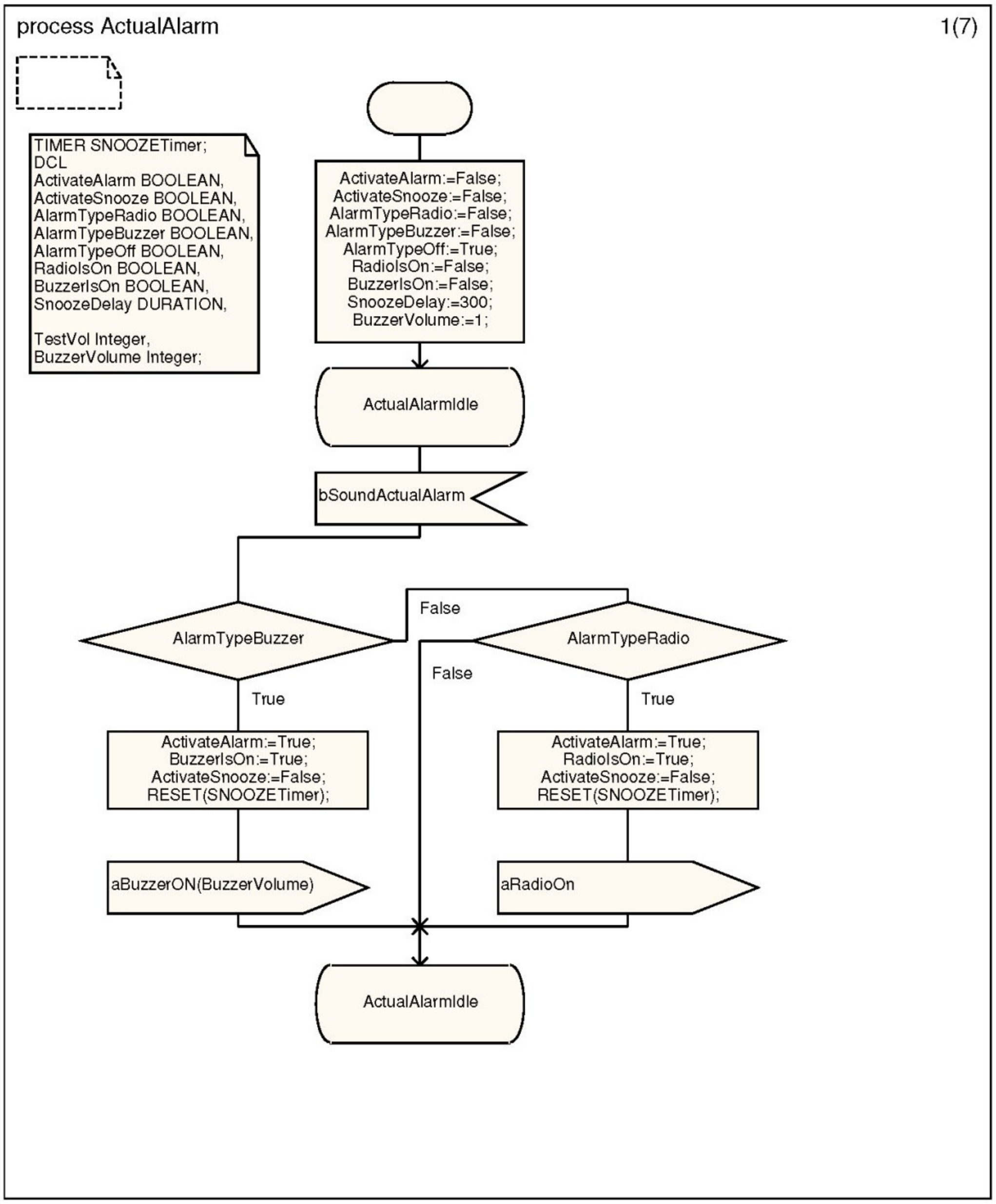

process AlarmMan
1(5)
DCL
InputMinute Integer,
InputHour Integer,
AlarmMinute Integer,
AlarmHour Integer;
AlarmHour:=0;
AlarmMinute:=0;
AlarmIdle
iTimeToAlarm(InputHour,InputMinute)
False
(InputMinute=AlarmMinute AND InputHour=AlarmHour)
AlarmIdle
True
bSoundActualAlarm
AlarmIdle

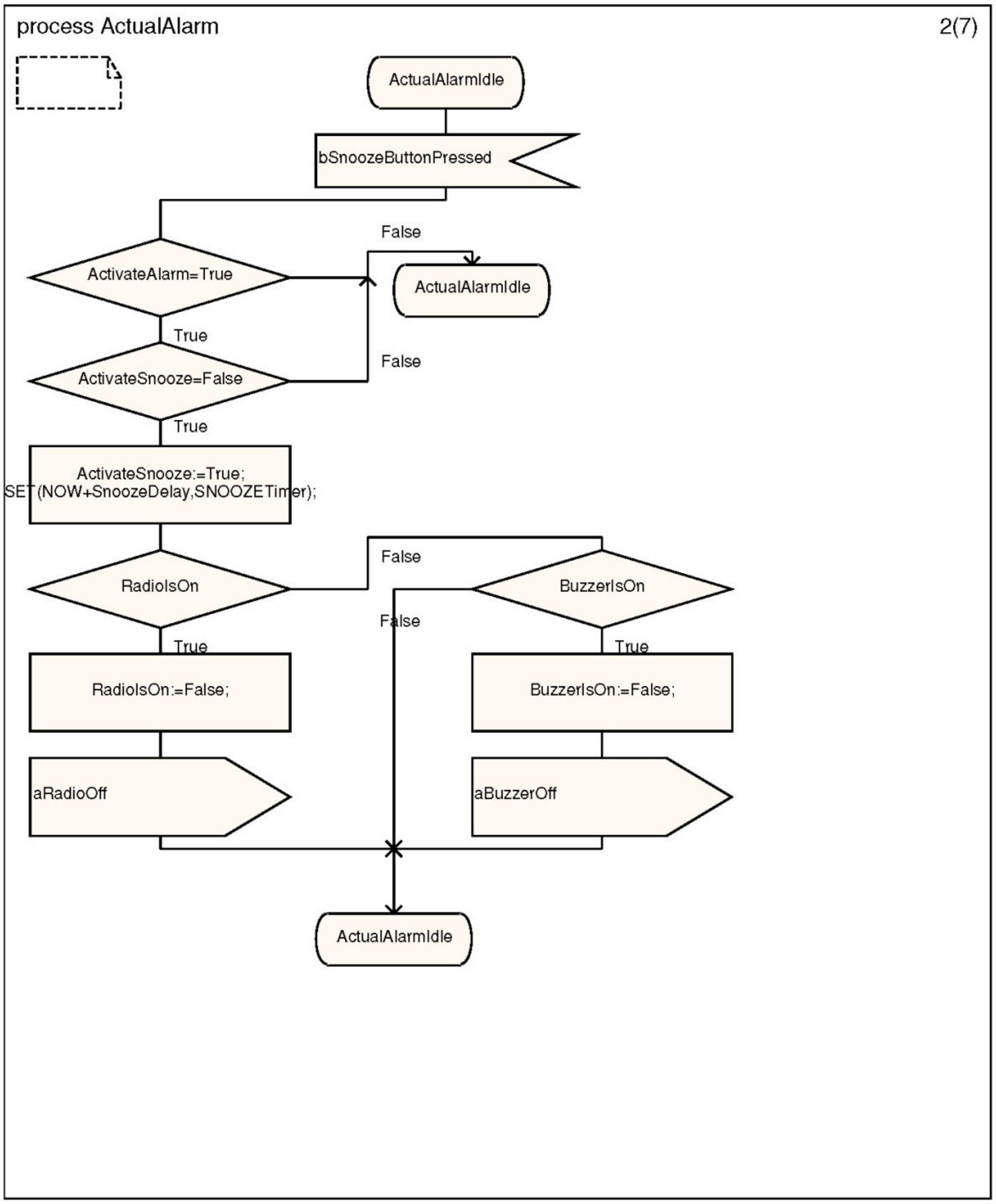

process AlarmMan
2(5)
AlarmIdle
bWakeButtonPressed
iCurAlarmTime(AlarmHour,AlarmMinute)
AlarmIdle

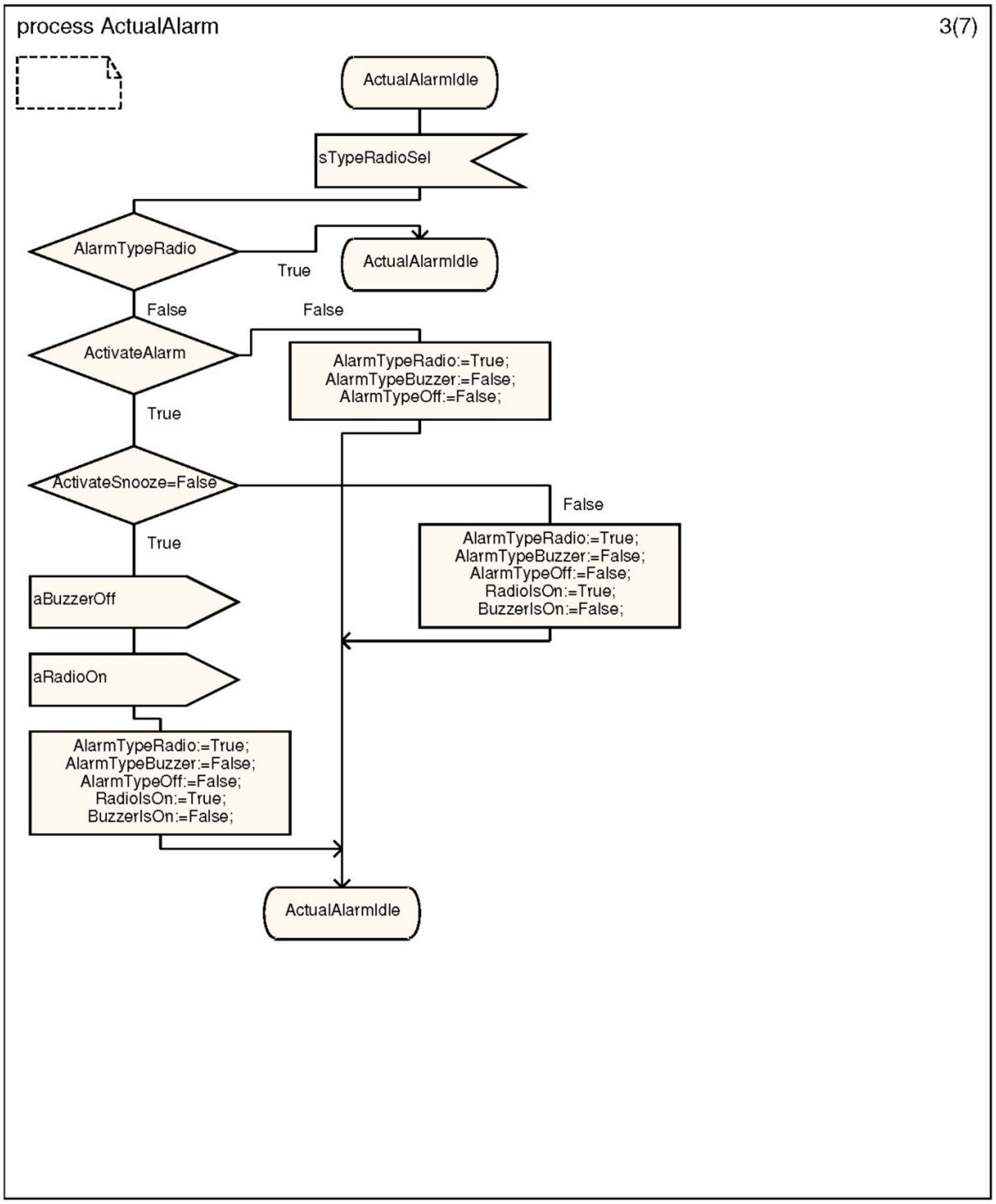

process AlarmMan
3(5)
AlarmIdle
bWakeMin10ButtonPressed
AlarmMinute:=AlarmMinute+10
AlarmMinute>59
False
True
AlarmHour:=AlarmHour+1;
AlarmMinute:=AlarmMinute-60;
AlarmHour>23
False
True
AlarmHour:=0;
iCurAlarmTime(AlarmHour,AlarmMinute)
AlarmIdle

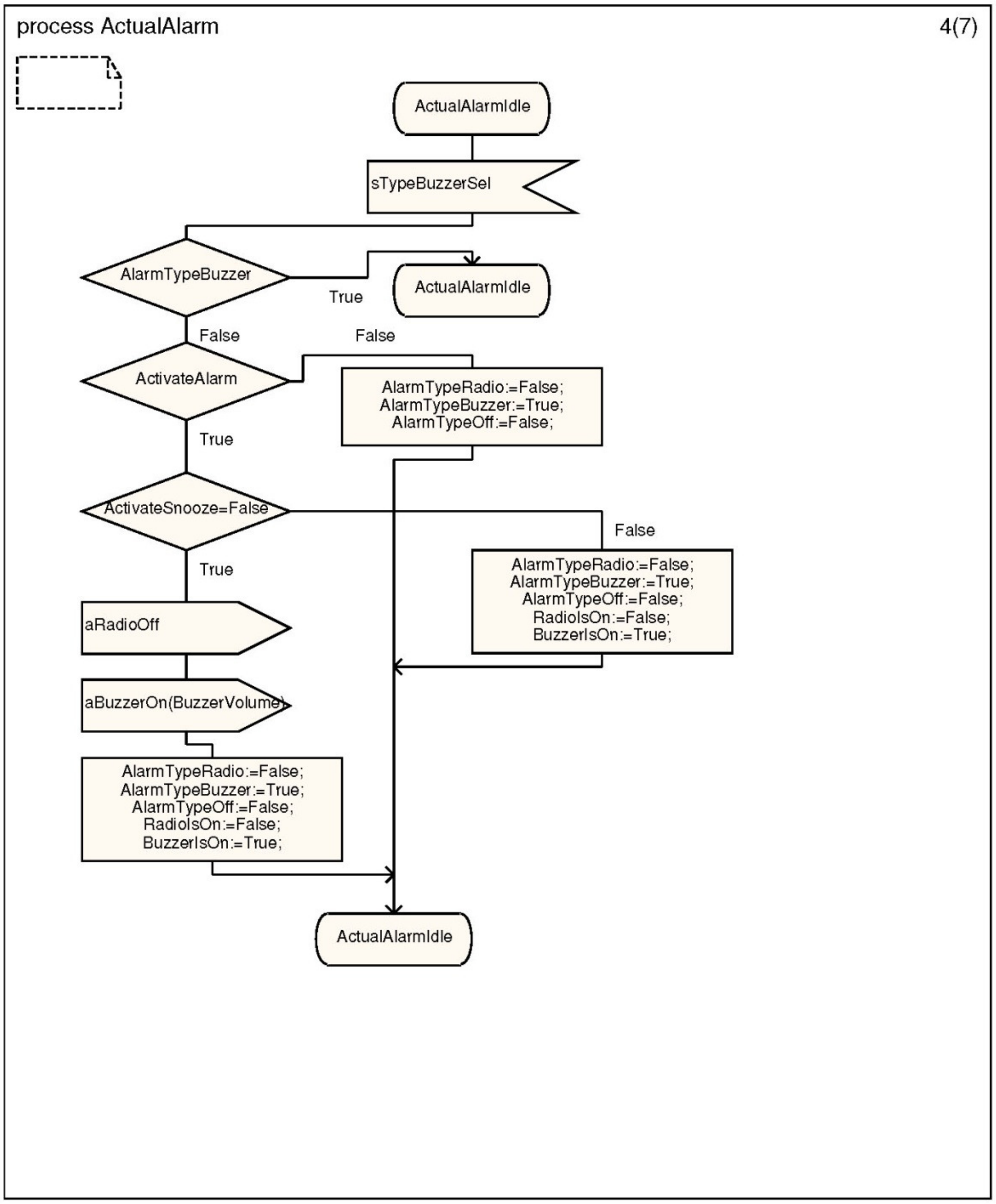

process AlarmMan
4(5)
AlarmIdle
bWakeMinButtonPressed
AlarmMinute:=AlarmMinute+1
AlarmMinute>59
False
True
AlarmHour:=AlarmHour+1;
AlarmMinute:=AlarmMinute-60;
AlarmHour>23
False
True
AlarmHour:=0;
iCurAlarmTime(AlarmHour,AlarmMinute)
AlarmIdle

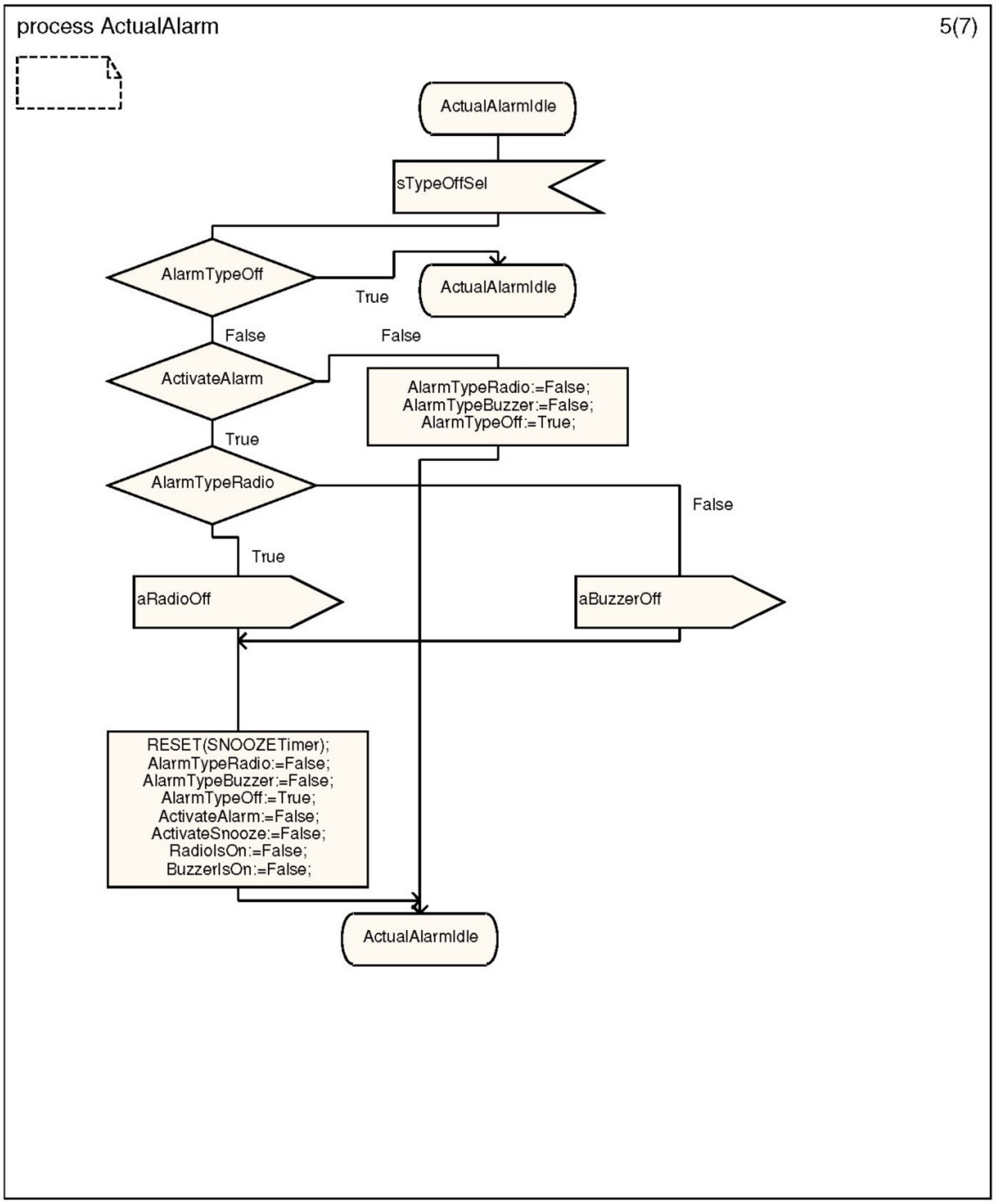

process AlarmMan
5(5)
AlarmIdle
bWakeHourButtonPressed
AlarmHour:=AlarmHour+1;
AlarmHour>23
True
False
AlarmHour:=0;
iCurAlarmTime(AlarmHour,AlarmMinute)
AlarmIdle

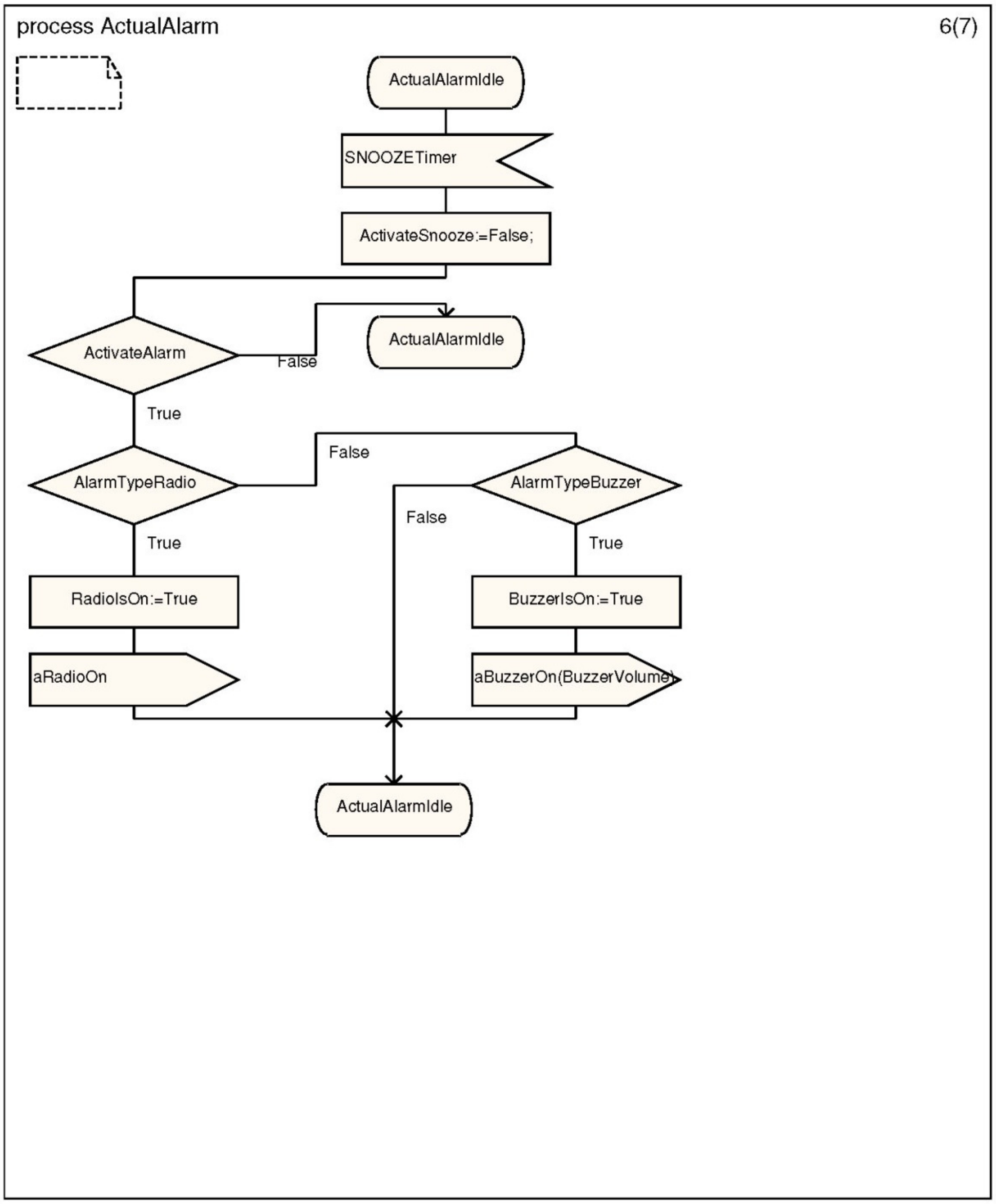

process ActualAlarm
1(7)
TIMER SNOOZETimer;
DCL
ActivateAlarm BOOLEAN,
ActivateSnooze BOOLEAN,
AlarmTypeRadio BOOLEAN,
AlarmTypeBuzzer BOOLEAN,
AlarmTypeOff BOOLEAN,
RadioIsOn BOOLEAN,
BuzzerIsOn BOOLEAN,
SnoozeDelay DURATION,
TestVol Integer,
BuzzerVolume Integer;
ActivateAlarm:=False;
ActivateSnooze:=False;
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=True;
RadioIsOn:=False;
BuzzerIsOn:=False;
SnoozeDelay:=300;
BuzzerVolume:=1;
ActualAlarmIdle
bSoundActualAlarm
AlarmTypeBuzzer
False
AlarmTypeRadio
False
True
True
ActivateAlarm:=True;
BuzzerIsOn:=True;
ActivateSnooze:=False;
RESET(SNOOZETimer);
ActivateAlarm:=True;
RadioIsOn:=True;
ActivateSnooze:=False;
RESET(SNOOZETimer);
aBuzzerON(BuzzerVolume)
aRadioOn
ActualAlarmIdle

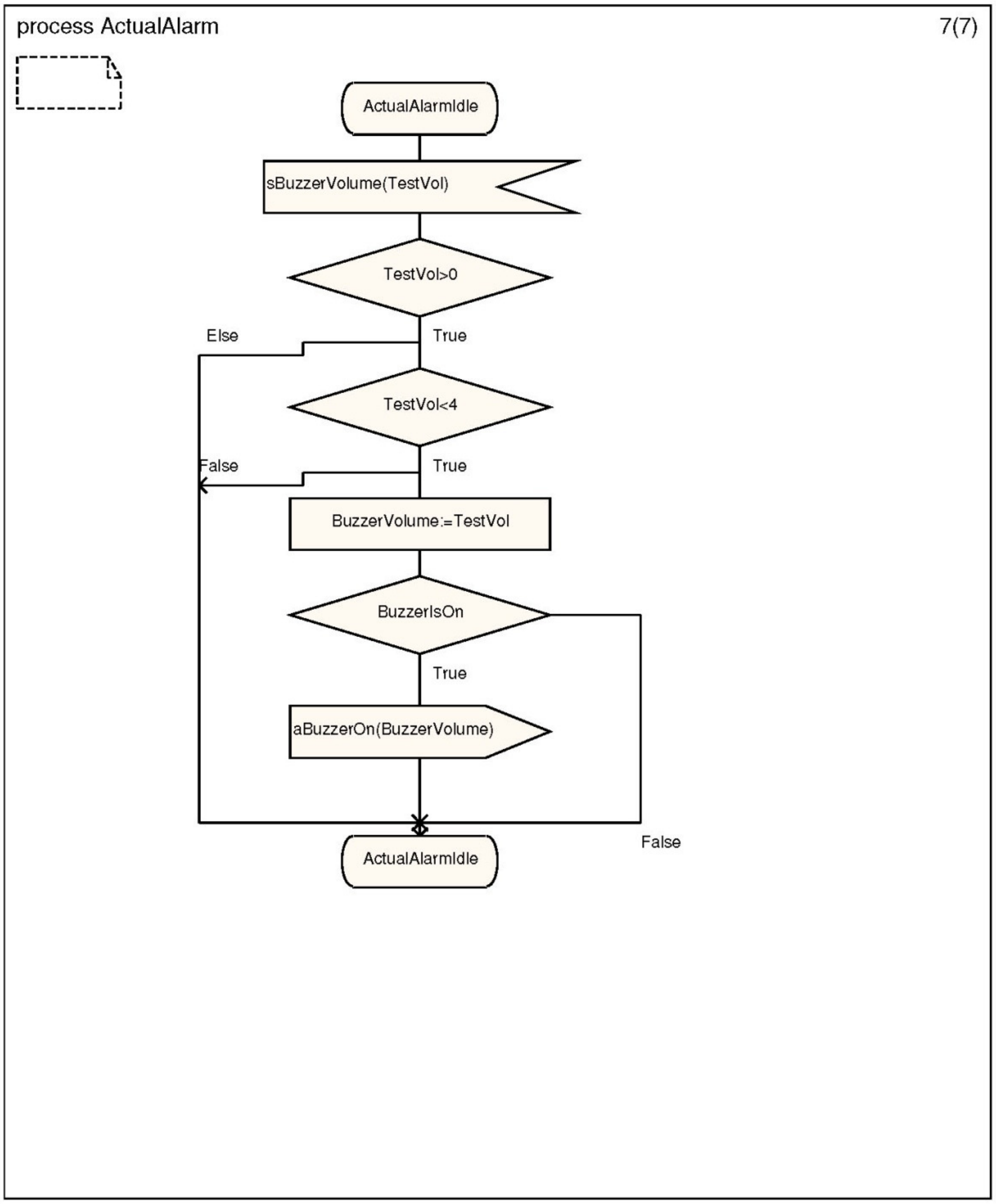

process ActualAlarm
2(7)
ActualAlarmIdle
bSnoozeButtonPressed
ActivateAlarm=True
False
ActualAlarmIdle
True
ActivateSnooze=False
False
True
ActivateSnooze:=True;
SET(NOW+SnoozeDelay,SNOOZETimer);
RadioIsOn
False
BuzzerIsOn
False
True
True
RadioIsOn:=False;
BuzzerIsOn:=False;
aRadioOff
aBuzzerOff
ActualAlarmIdle


process ActualAlarm
3(7)
ActualAlarmIdle
sTypeRadioSel
AlarmTypeRadio
True
ActualAlarmIdle
False
False
ActivateAlarm
AlarmTypeRadio:=True;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=False;
True
ActivateSnooze=False
False
True
AlarmTypeRadio:=True;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=False;
RadioIsOn:=True;
BuzzerIsOn:=False;
aBuzzerOff
aRadioOn
AlarmTypeRadio:=True;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=False;
RadioIsOn:=True;
BuzzerIsOn:=False;
ActualAlarmIdle


process ActualAlarm
4(7)
ActualAlarmIdle
sTypeBuzzerSel
AlarmTypeBuzzer
True
ActualAlarmIdle
False
False
ActivateAlarm
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=True;
AlarmTypeOff:=False;
True
ActivateSnooze=False
False
True
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=True;
AlarmTypeOff:=False;
RadioIsOn:=False;
BuzzerIsOn:=True;
aRadioOff
aBuzzerOn(BuzzerVolume)
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=True;
AlarmTypeOff:=False;
RadioIsOn:=False;
BuzzerIsOn:=True;
ActualAlarmIdle


process ActualAlarm
5(7)
ActualAlarmIdle
sTypeOffSel
AlarmTypeOff
True
ActualAlarmIdle
False
False
ActivateAlarm
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=True;
True
AlarmTypeRadio
False
True
aRadioOff
aBuzzerOff
RESET(SNOOZETimer);
AlarmTypeRadio:=False;
AlarmTypeBuzzer:=False;
AlarmTypeOff:=True;
ActivateAlarm:=False;
ActivateSnooze:=False;
RadioIsOn:=False;
BuzzerIsOn:=False;
ActualAlarmIdle

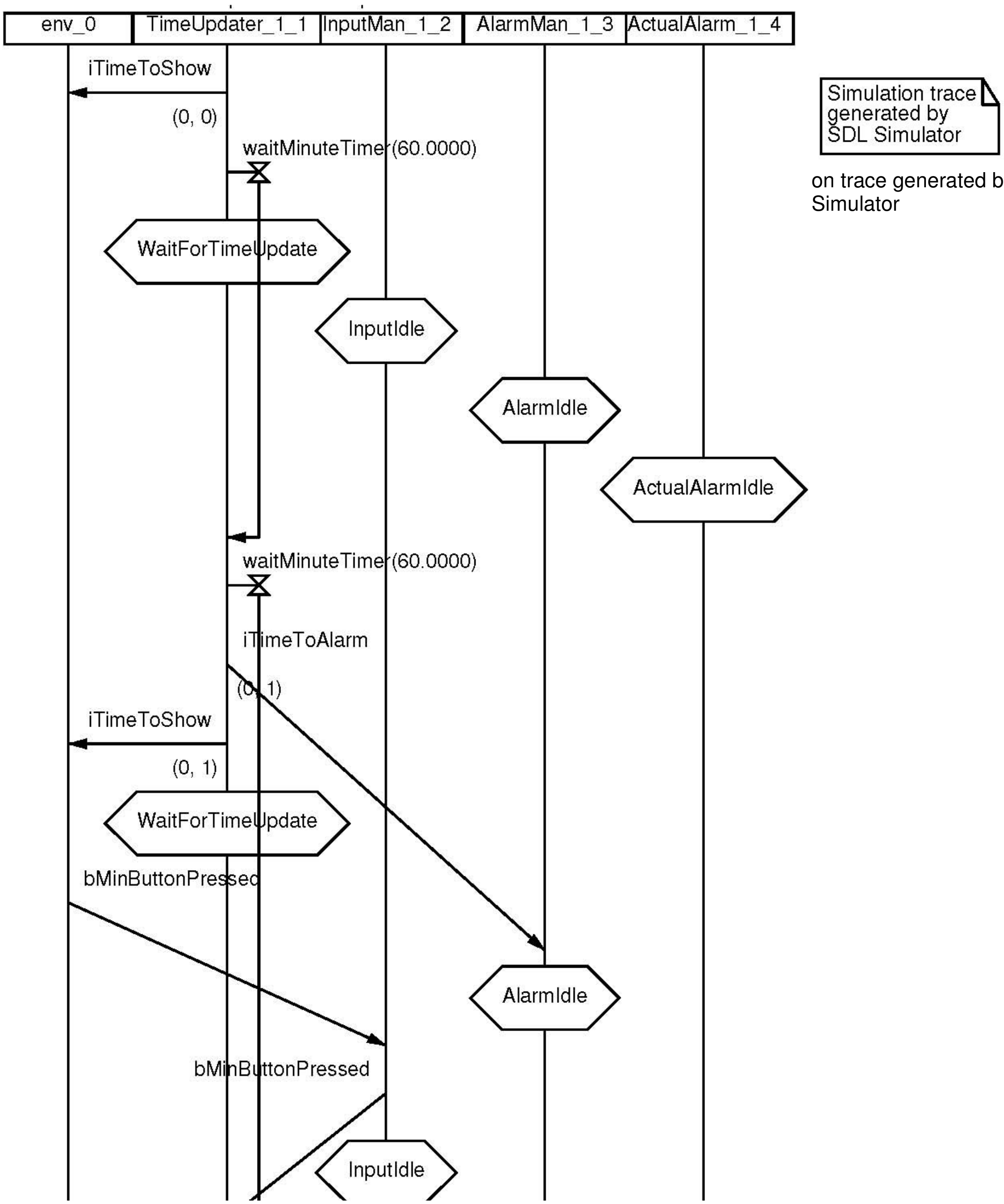

process ActualAlarm
6(7)
ActualAlarmIdle
SNOOZETimer
ActivateSnooze:=False;
ActivateAlarm
False
ActualAlarmIdle
True
AlarmTypeRadio
False
AlarmTypeBuzzer
False
True
True
RadioIsOn:=True
BuzzerIsOn:=True
aRadioOn
aBuzzerOn(BuzzerVolume)
ActualAlarmIdle


process ActualAlarm
7(7)
ActualAlarmIdle
sBuzzerVolume(TestVol)
TestVol>0
Else
True
TestVol<4
False
True
BuzzerVolume:=TestVol
BuzzerIsOn
True
aBuzzerOn(BuzzerVolume)
False
ActualAlarmIdle

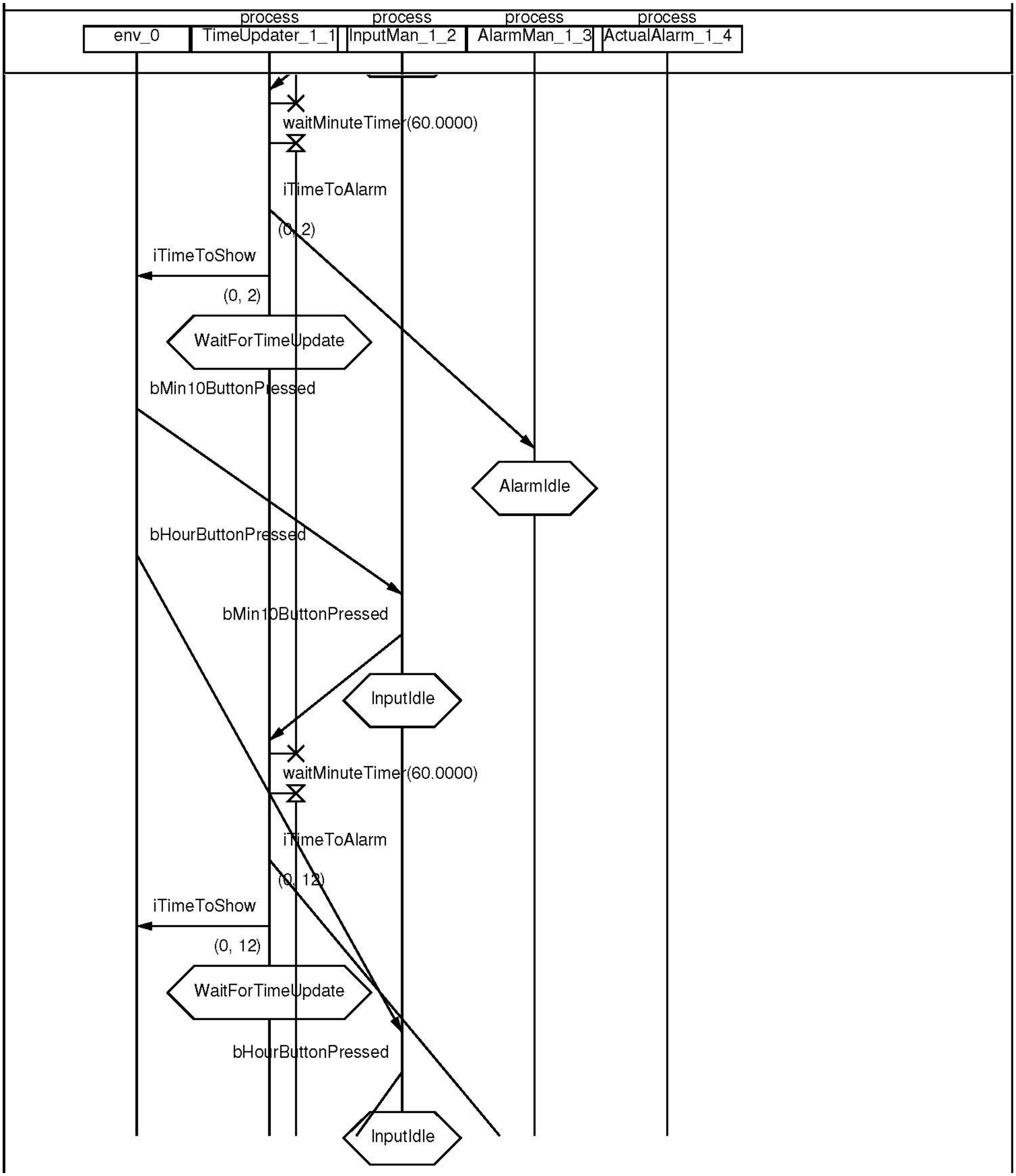
Message Sequence Chart

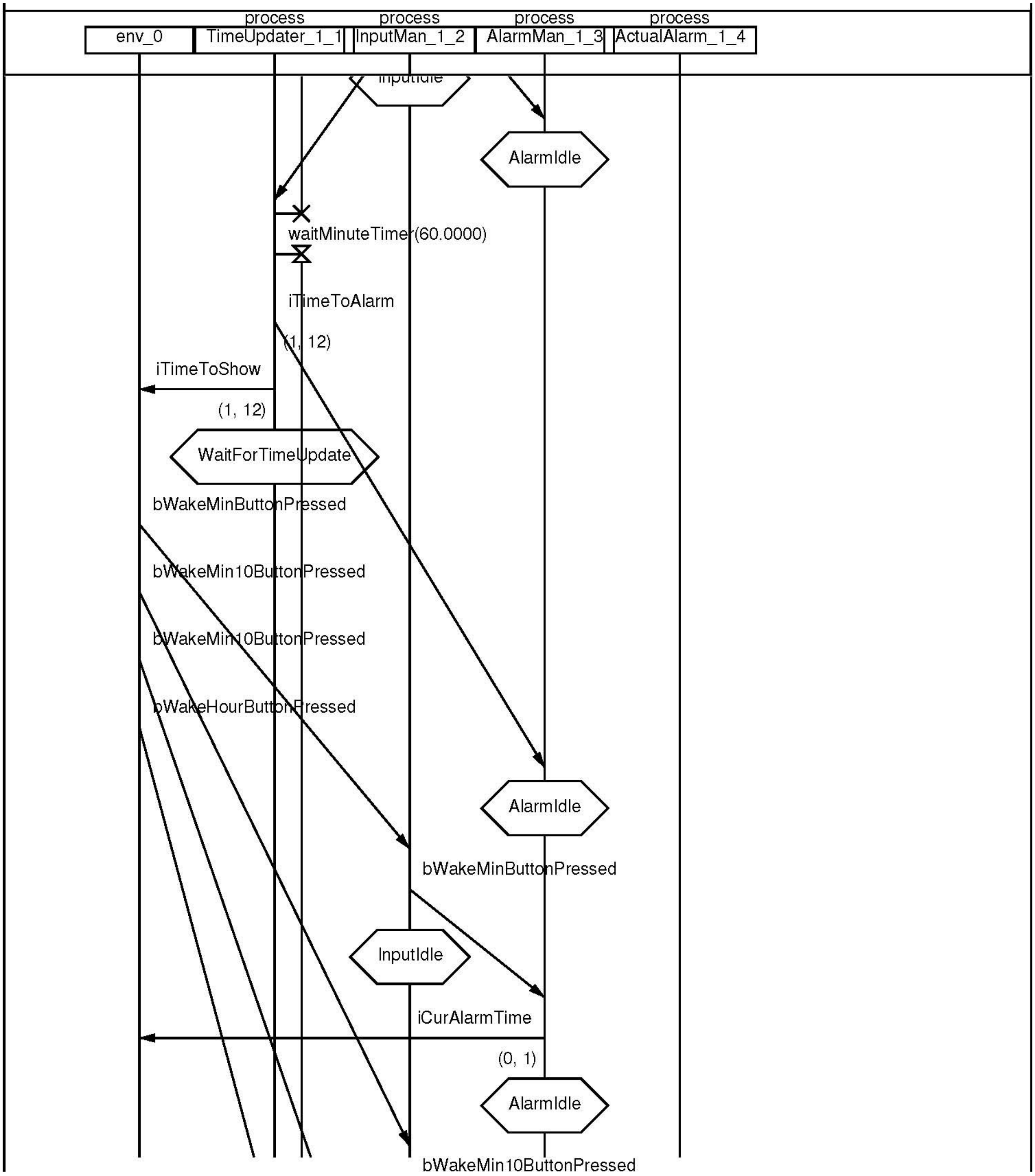
The following message sequence diagram goes through every features listed in the use cases and even goes a bit further.

MSC SimulatorTrace

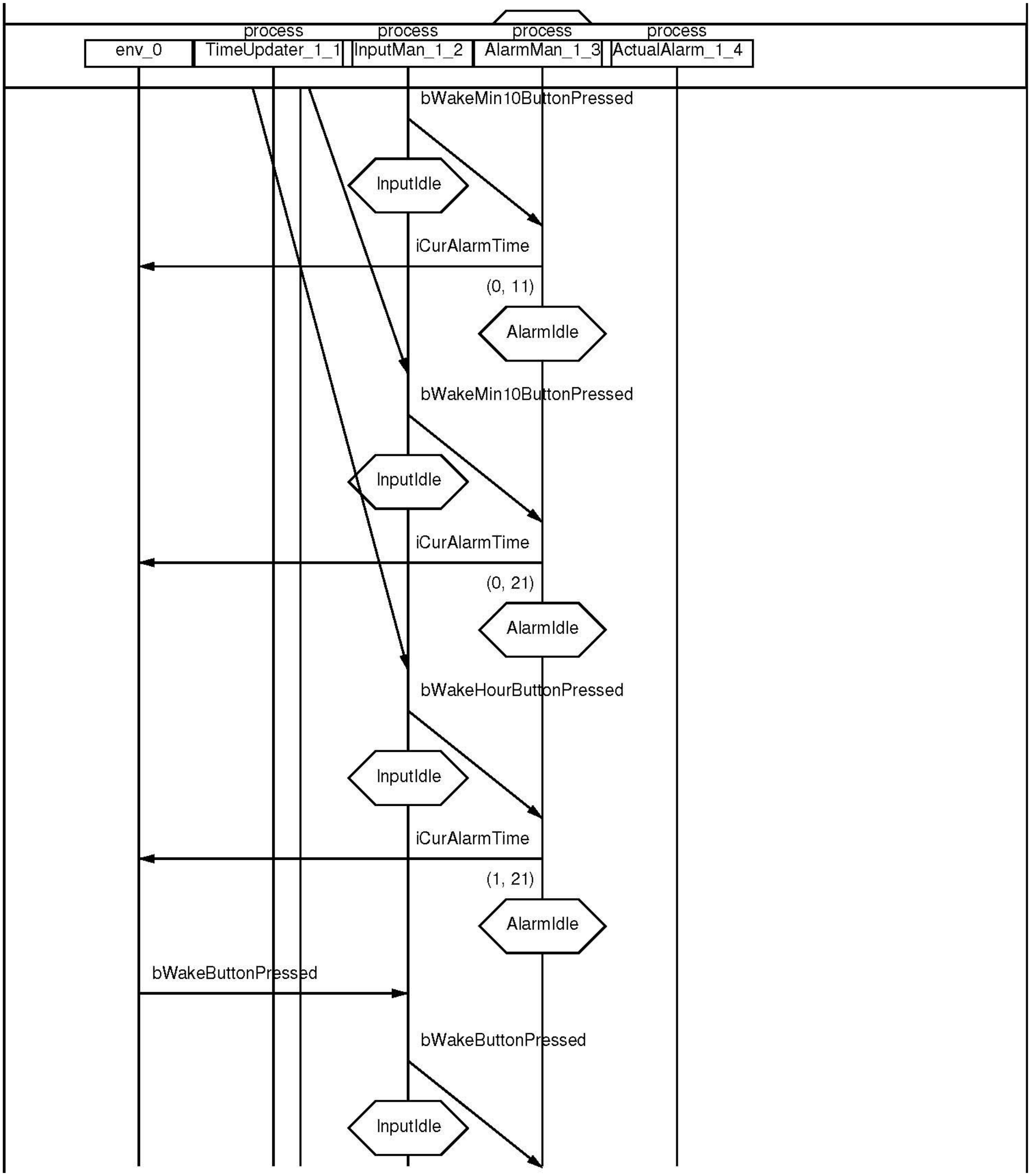


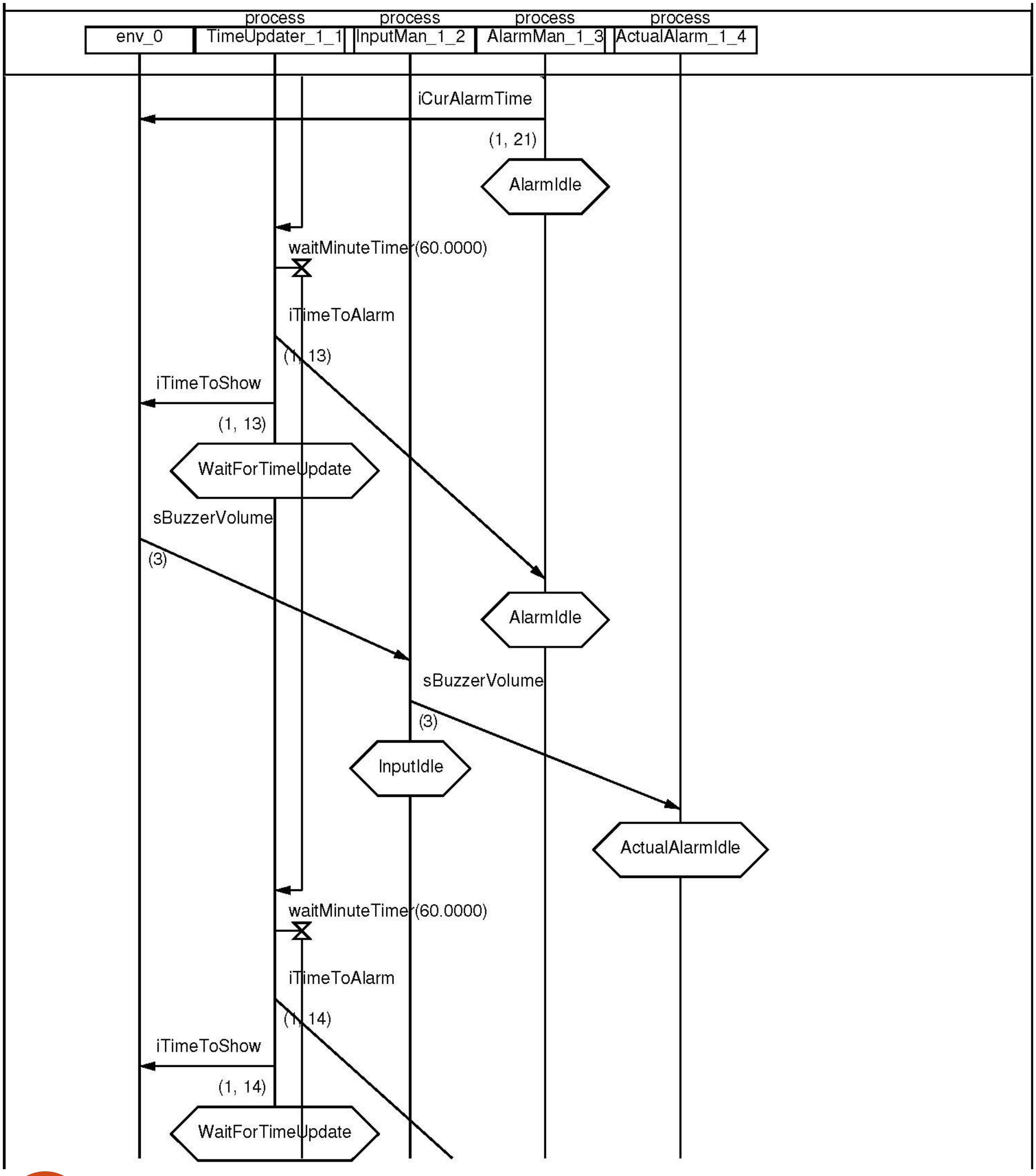
Simulati
on trace generated by SDL
Simulator

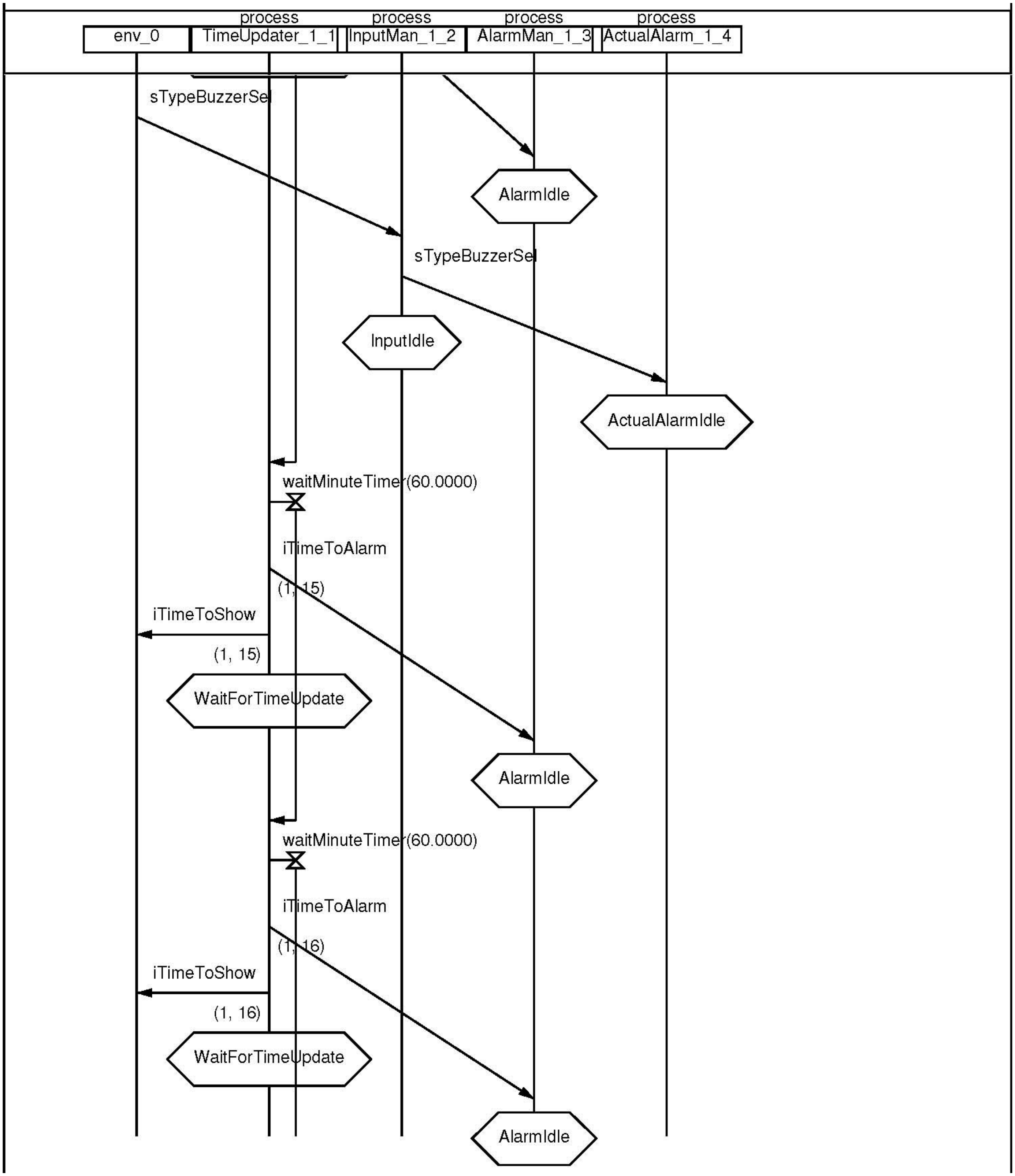

process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
waitMinuteTimer(60.0000)
iTimeToAlarm
(0, 2)
iTimeToShow
(0, 2)
WaitForTimeUpdate
bMin10ButtonPressed
AlarmIdle
bHourButtonPressed
bMin10ButtonPressed
InputIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(0, 12)
iTimeToShow
(0, 12)
WaitForTimeUpdate
bHourButtonPressed
InputIdle

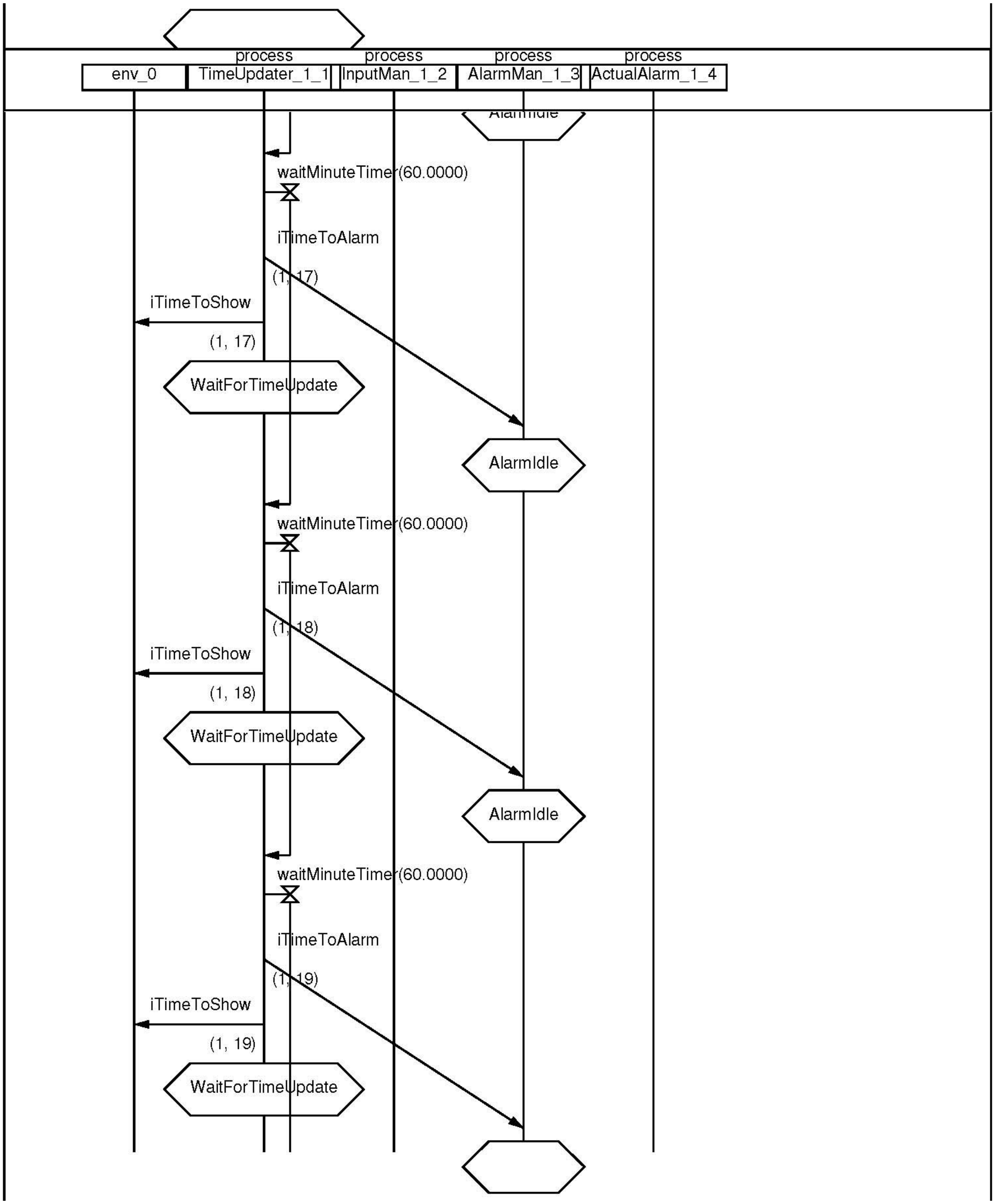


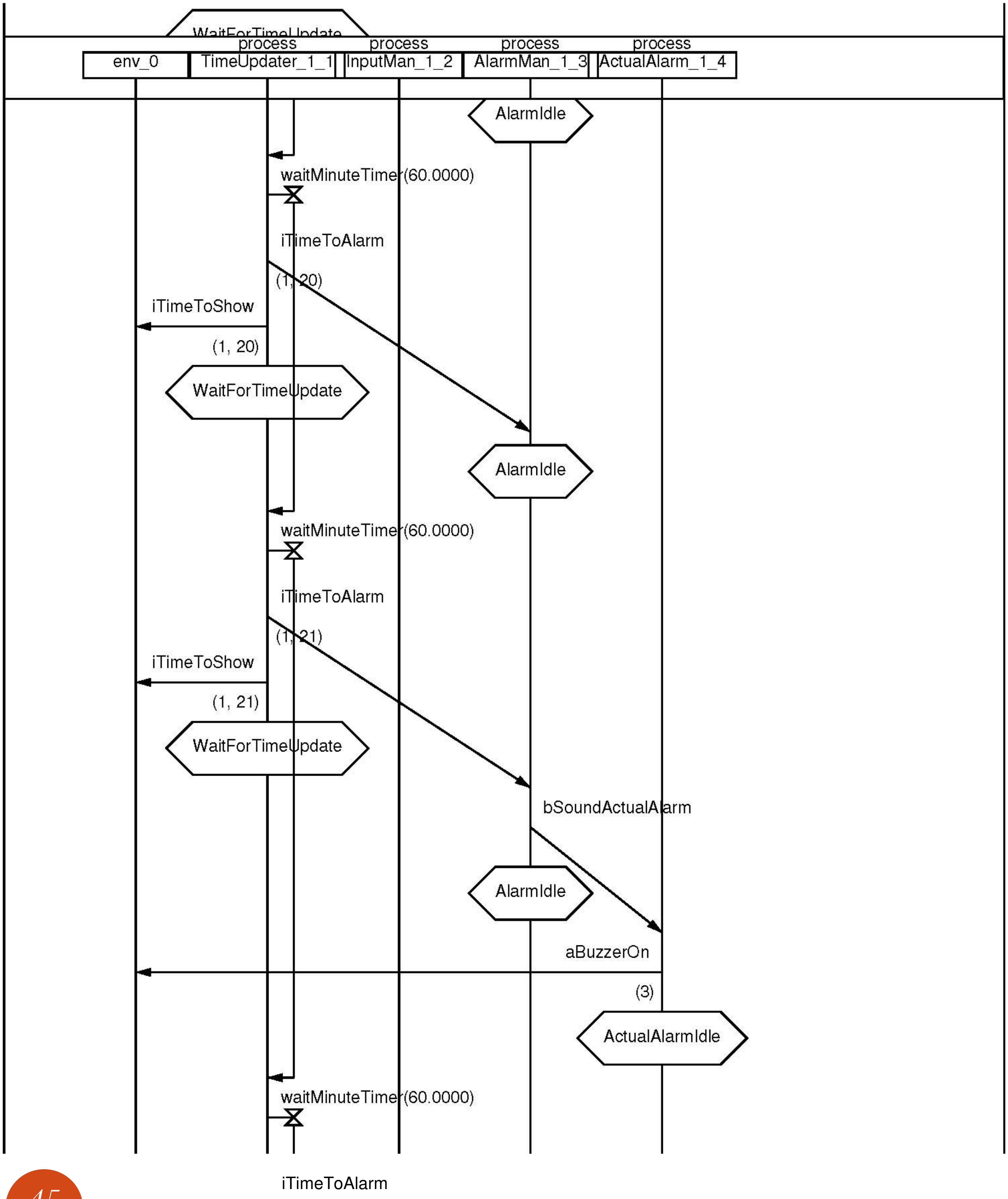
process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
InputIdle
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 12)
iTimeToShow
(1, 12)
WaitForTimeUpdate
bWakeMinButtonPressed
bWakeMin10ButtonPressed
bWakeMin10ButtonPressed
bWakeHourButtonPressed
AlarmIdle
bWakeMinButtonPressed
InputIdle
iCurAlarmTime
(0, 1)
AlarmIdle
bWakeMin10ButtonPressed

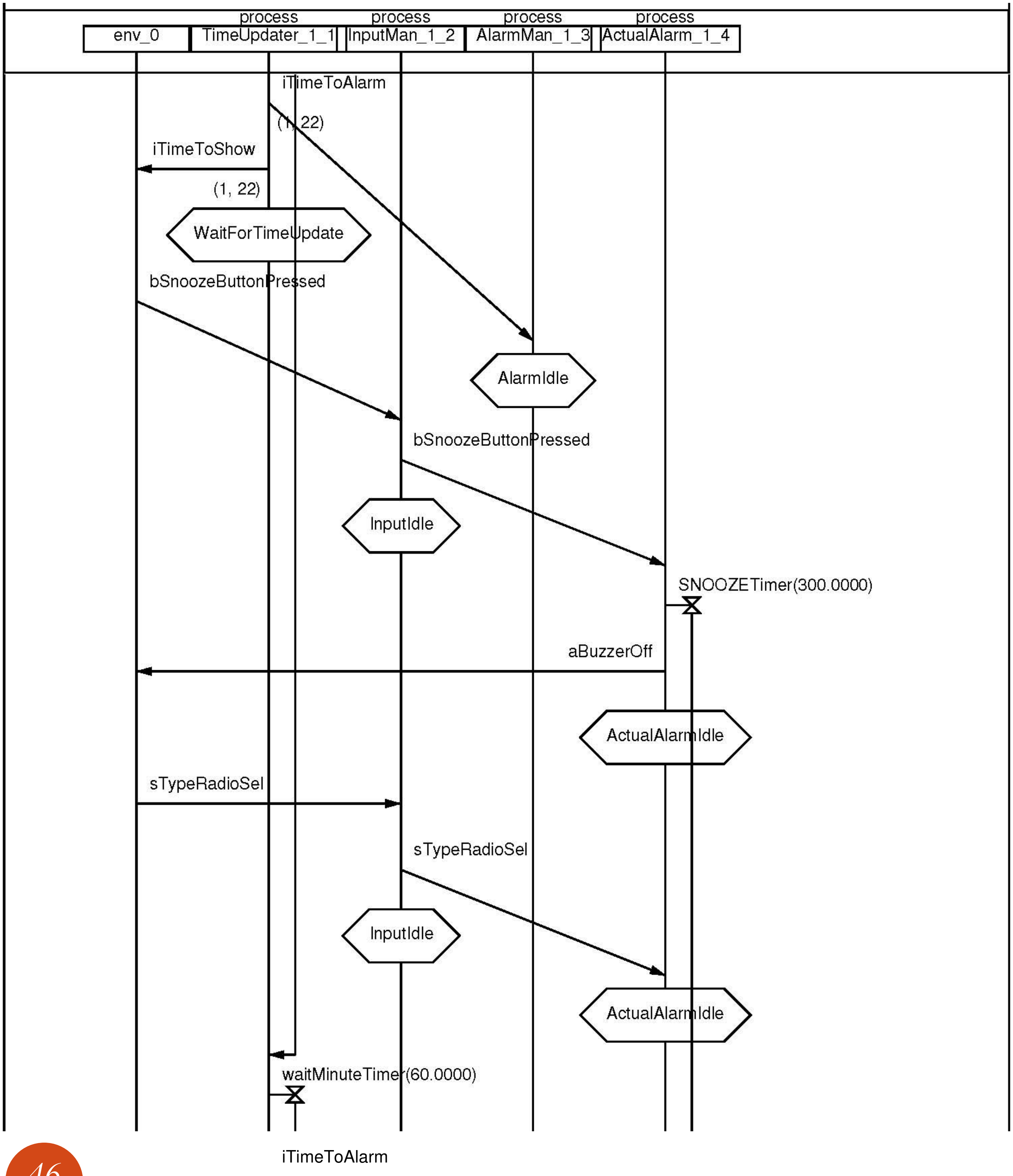

env_0
process
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
bWakeMin10ButtonPressed
InputIdle
iCurAlarmTime
(0, 11)
AlarmIdle
bWakeMin10ButtonPressed
InputIdle
iCurAlarmTime
(0, 21)
AlarmIdle
bWakeHourButtonPressed
InputIdle
iCurAlarmTime
(1, 21)
AlarmIdle
bWakeButtonPressed
bWakeButtonPressed
InputIdle

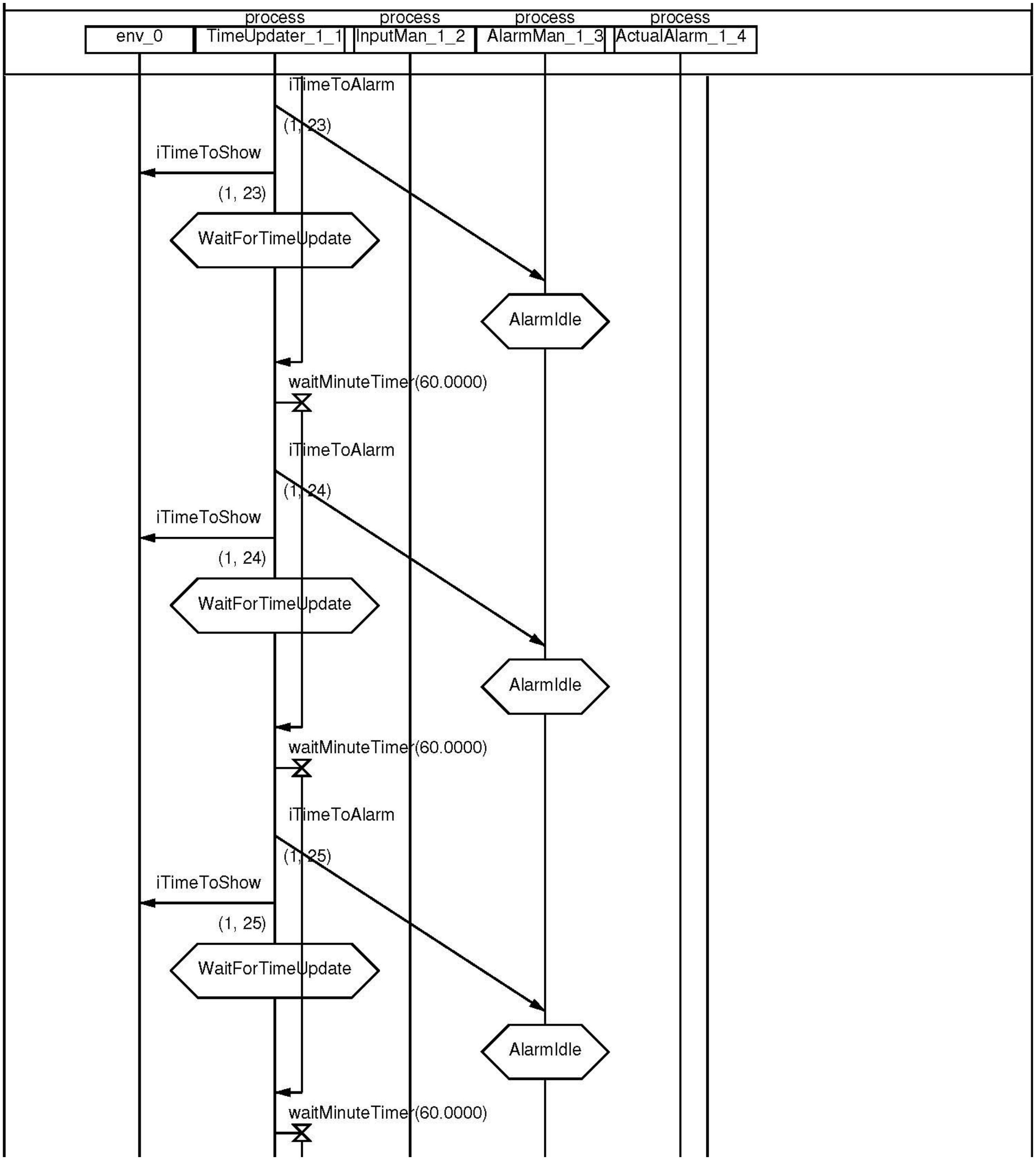

env_0
process
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
iCurAlarmTime
(1, 21)
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 13)
iTimeToShow
(1, 13)
WaitForTimeUpdate
sBuzzerVolume
(3)
AlarmIdle
sBuzzerVolume
(3)
InputIdle
ActualAlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 14)
iTimeToShow
(1, 14)
WaitForTimeUpdate

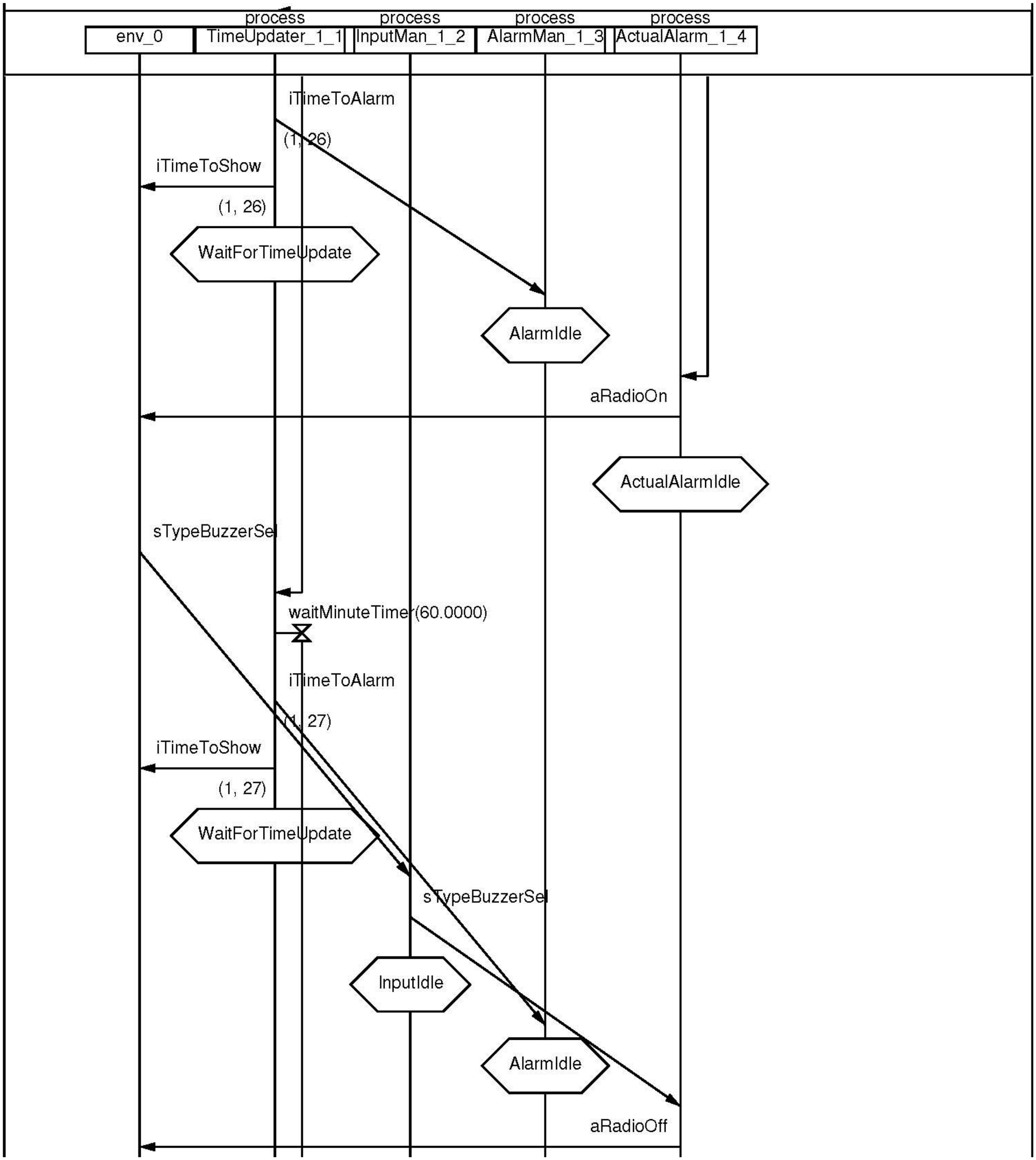

process
env_0
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
sTypeBuzzerSel
AlarmIdle
sTypeBuzzerSel
InputIdle
ActualAlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 15)
iTimeToShow
(1, 15)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 16)
iTimeToShow
(1, 16)
WaitForTimeUpdate
AlarmIdle

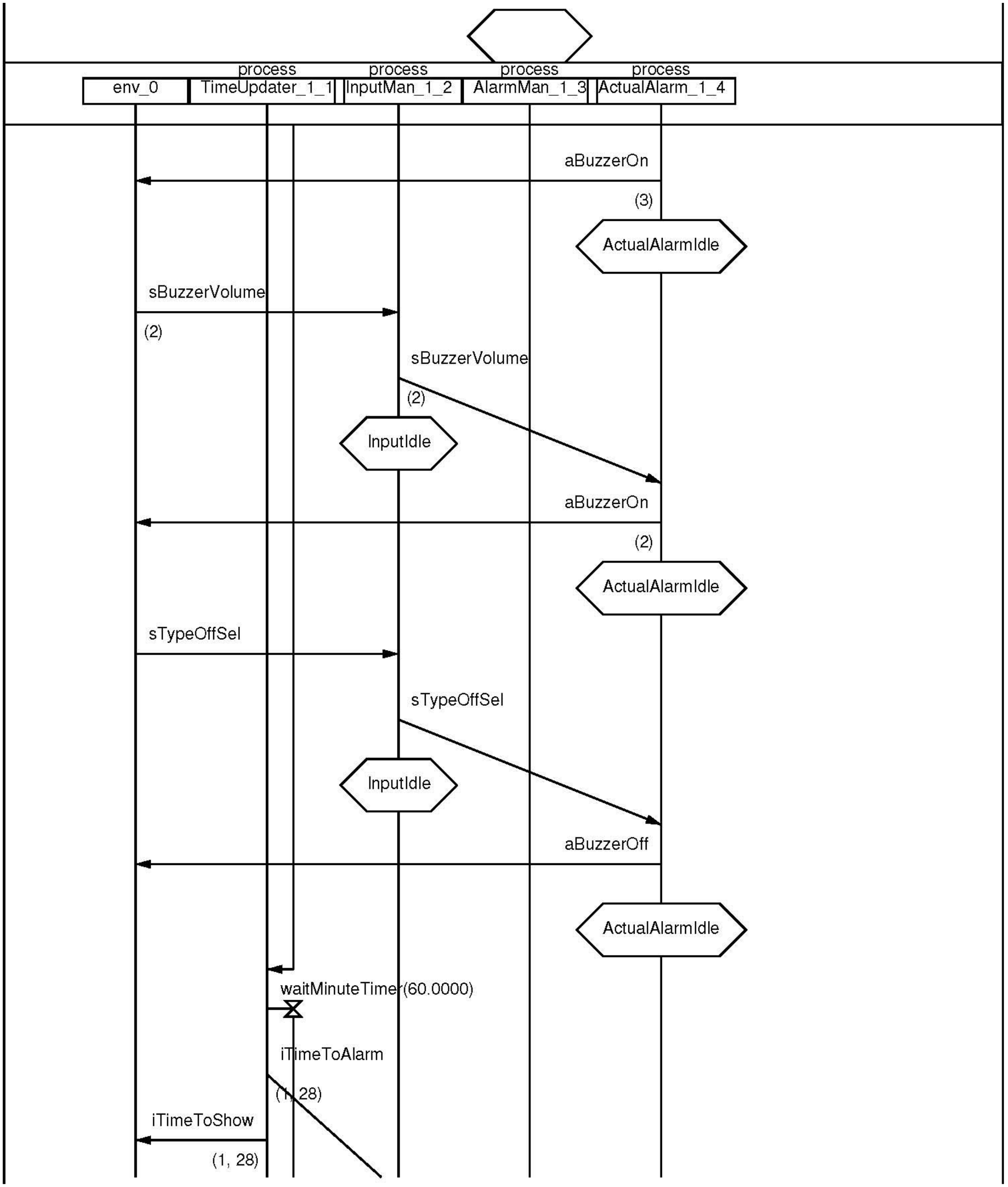

process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 17)
iTimeToShow
(1, 17)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 18)
iTimeToShow
(1, 18)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 19)
iTimeToShow
(1, 19)
WaitForTimeUpdate

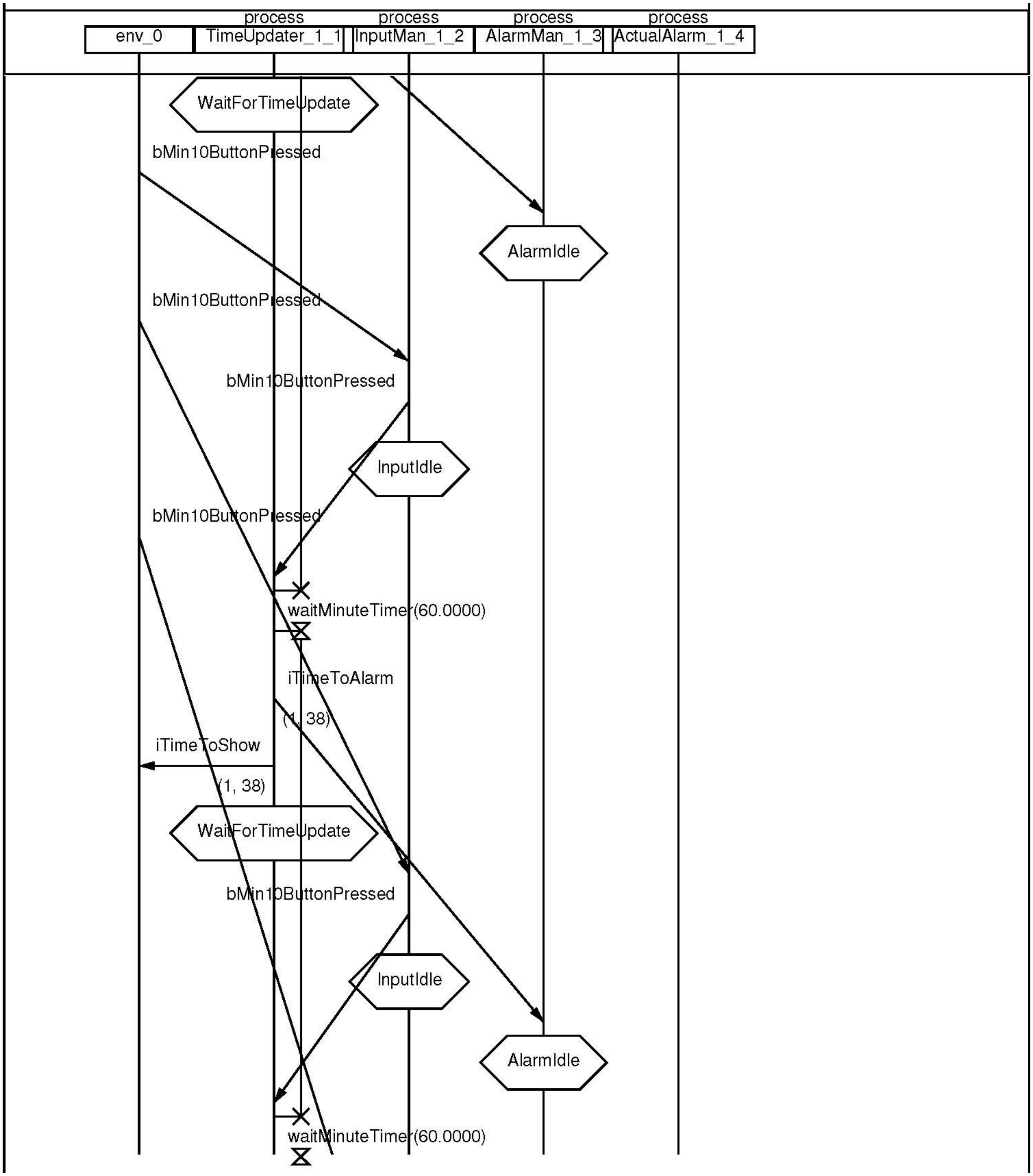

WaitForTimeUpdate
process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 20)
iTimeToShow
(1, 20)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 21)
iTimeToShow
(1, 21)
WaitForTimeUpdate
bSoundActualAlarm
AlarmIdle
aBuzzerOn
(3)
ActualAlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm

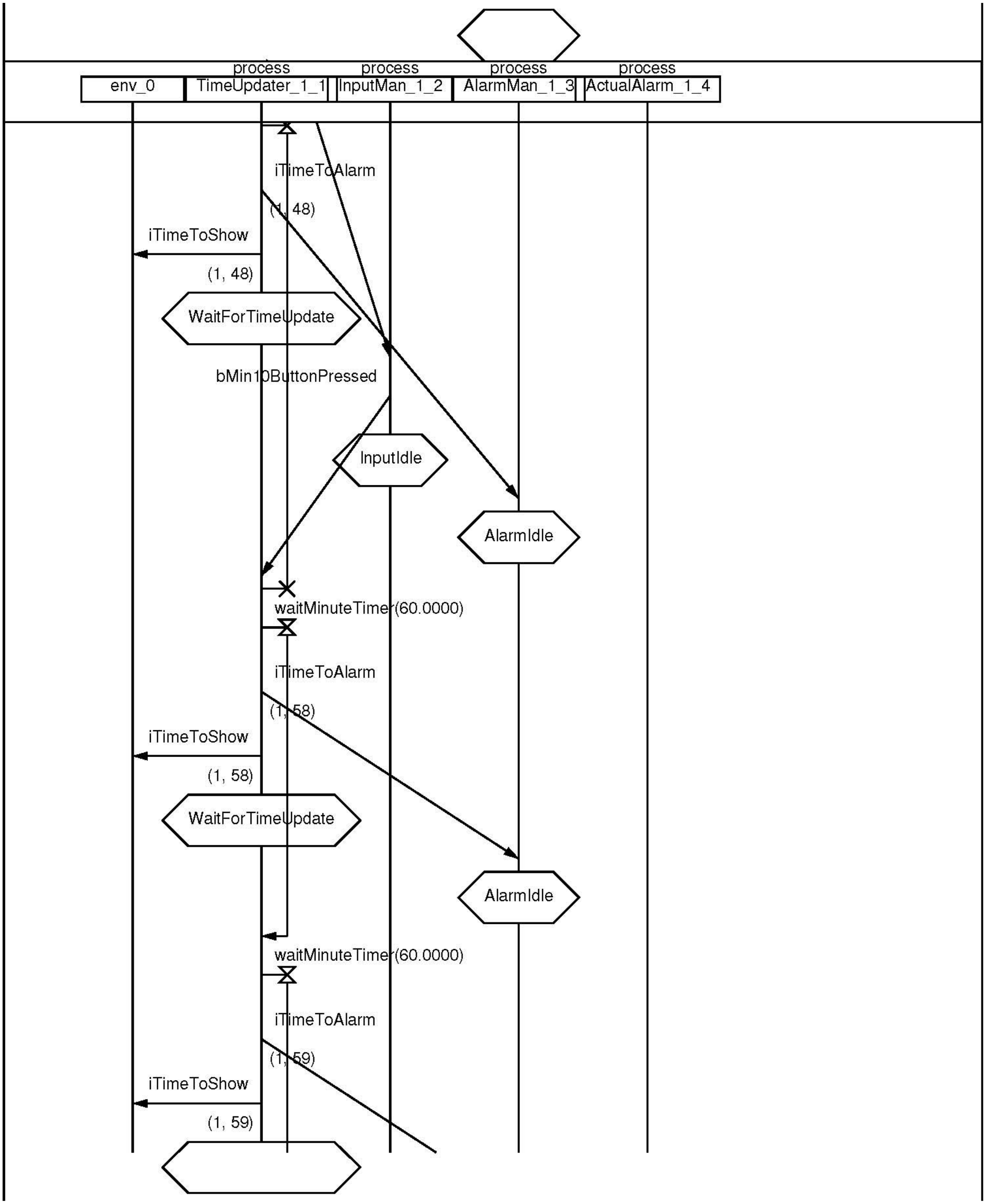

process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
iTimeToAlarm
(1, 22)
iTimeToShow
(1, 22)
WaitForTimeUpdate
bSnoozeButtonPressed
AlarmIdle
bSnoozeButtonPressed
InputIdle
SNOOZETimer(300.0000)
aBuzzerOff
ActualAlarmIdle
sTypeRadioSel
sTypeRadioSel
InputIdle
ActualAlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm

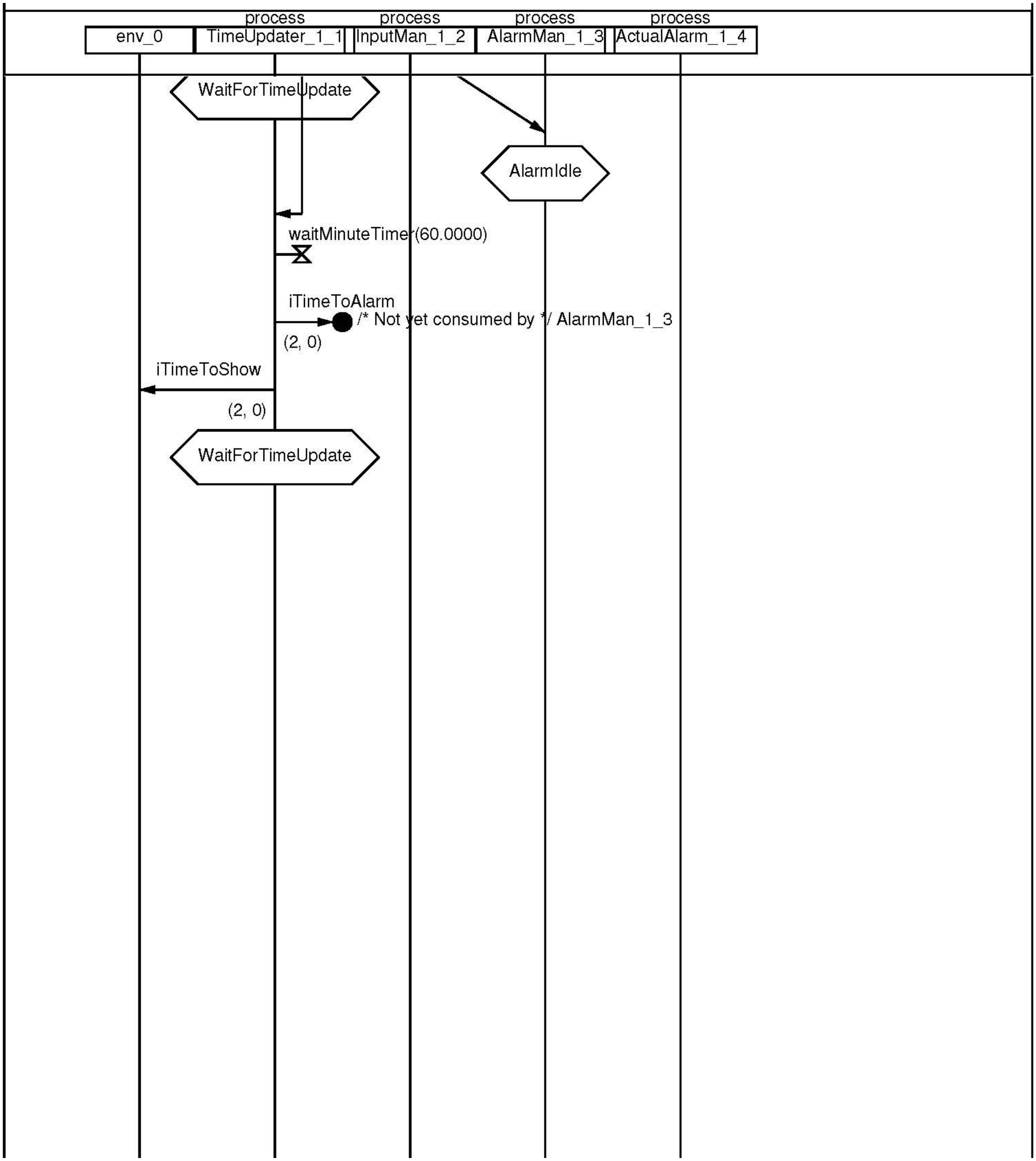

process
env_0
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
iTimeToAlarm
(1, 23)
iTimeToShow
(1, 23)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 24)
iTimeToShow
(1, 24)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 25)
iTimeToShow
(1, 25)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)


env_0
process
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
iTimeToAlarm
(1, 26)
iTimeToShow
(1, 26)
WaitForTimeUpdate
AlarmIdle
aRadioOn
ActualAlarmIdle
sTypeBuzzerSel
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 27)
iTimeToShow
(1, 27)
WaitForTimeUpdate
sTypeBuzzerSel
InputIdle
AlarmIdle
aRadioOff


env_0
process
TimeUpdater_1_1
process
InputMan_1_2
process
AlarmMan_1_3
process
ActualAlarm_1_4
aBuzzerOn
(3)
ActualAlarmIdle
sBuzzerVolume
(2)
sBuzzerVolume
(2)
InputIdle
aBuzzerOn
(2)
ActualAlarmIdle
sTypeOffSel
sTypeOffSel
InputIdle
aBuzzerOff
ActualAlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 28)
iTimeToShow
(1, 28)

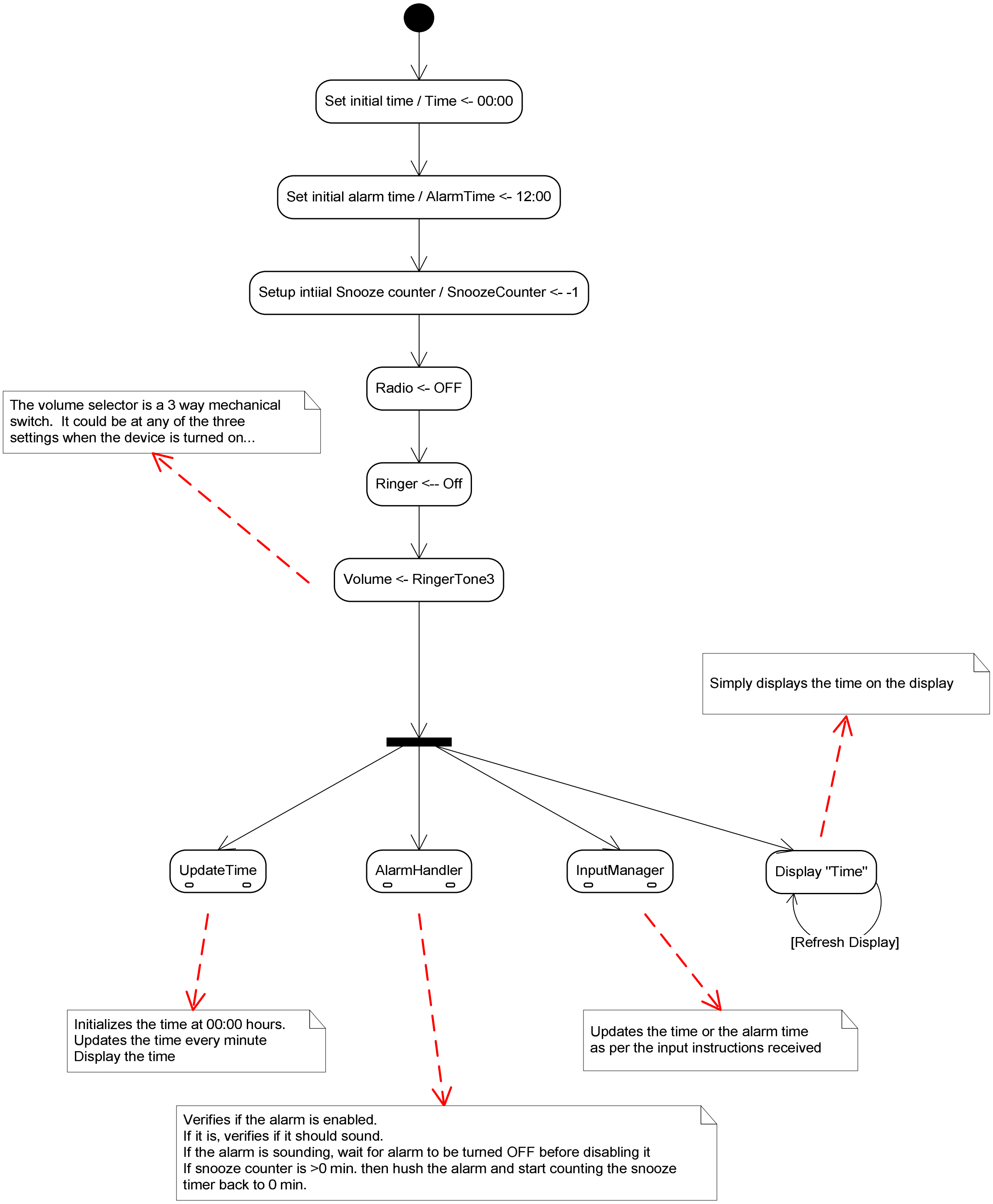

process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
WaitForTimeUpdate
bMin10ButtonPressed
AlarmIdle
bMin10ButtonPressed
bMin10ButtonPressed
InputIdle
bMin10ButtonPressed
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 38)
iTimeToShow
(1, 38)
WaitForTimeUpdate
bMin10ButtonPressed
InputIdle
AlarmIdle
waitMinuteTimer(60.0000)


process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
iTimeToAlarm
(1, 48)
iTimeToShow
(1, 48)
WaitForTimeUpdate
bMin10ButtonPressed
InputIdle
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 58)
iTimeToShow
(1, 58)
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
(1, 59)
iTimeToShow
(1, 59)


process
process
process
process
env_0
TimeUpdater_1_1
InputMan_1_2
AlarmMan_1_3
ActualAlarm_1_4
WaitForTimeUpdate
AlarmIdle
waitMinuteTimer(60.0000)
iTimeToAlarm
/* Not yet consumed by */ AlarmMan_1_3
(2, 0)
iTimeToShow
(2, 0)
WaitForTimeUpdate

Appendix A

The UML state diagram below (starting on the next page) shows the first attempt I took at designing the behavior of the Alarm Clock. I say design because these diagrams were a way to try and plan the implementation of the SDL diagram. While doing this state diagram, I found a lot of problems that I definitely not want in the SDL diagram and learned from those mistakes before even starting the SDL diagram in the first place. Doing this, in my belief, seriously minimized running into problems while doing the SDL diagram and having to “redo” entire sections (loosing time fixing and debugging like was the case in LAB2).


Set initial time / Time <- 00:00
Set initial alarm time / AlarmTime <- 12:00
Setup intiial Snooze counter / SnoozeCounter <- -1
Radio <- OFF
Ringer <-- Off
Volume <- RingerTone3
UpdateTime
AlarmHandler
InputManager
Display "Time"
[Refresh Display]

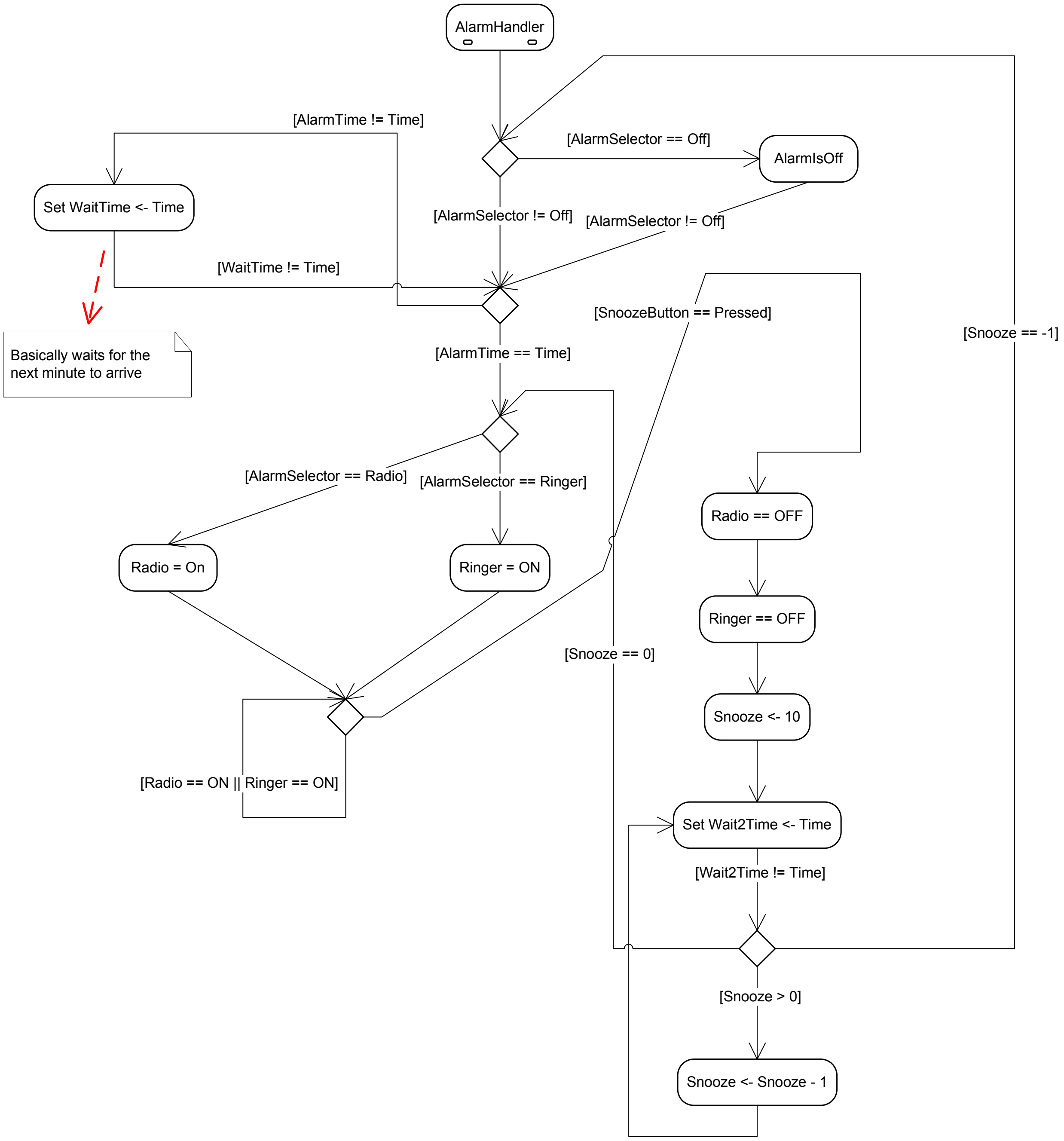
The volume selector is a 3 way mechanical switch. It could be at any of the three settings when the device is turned on...

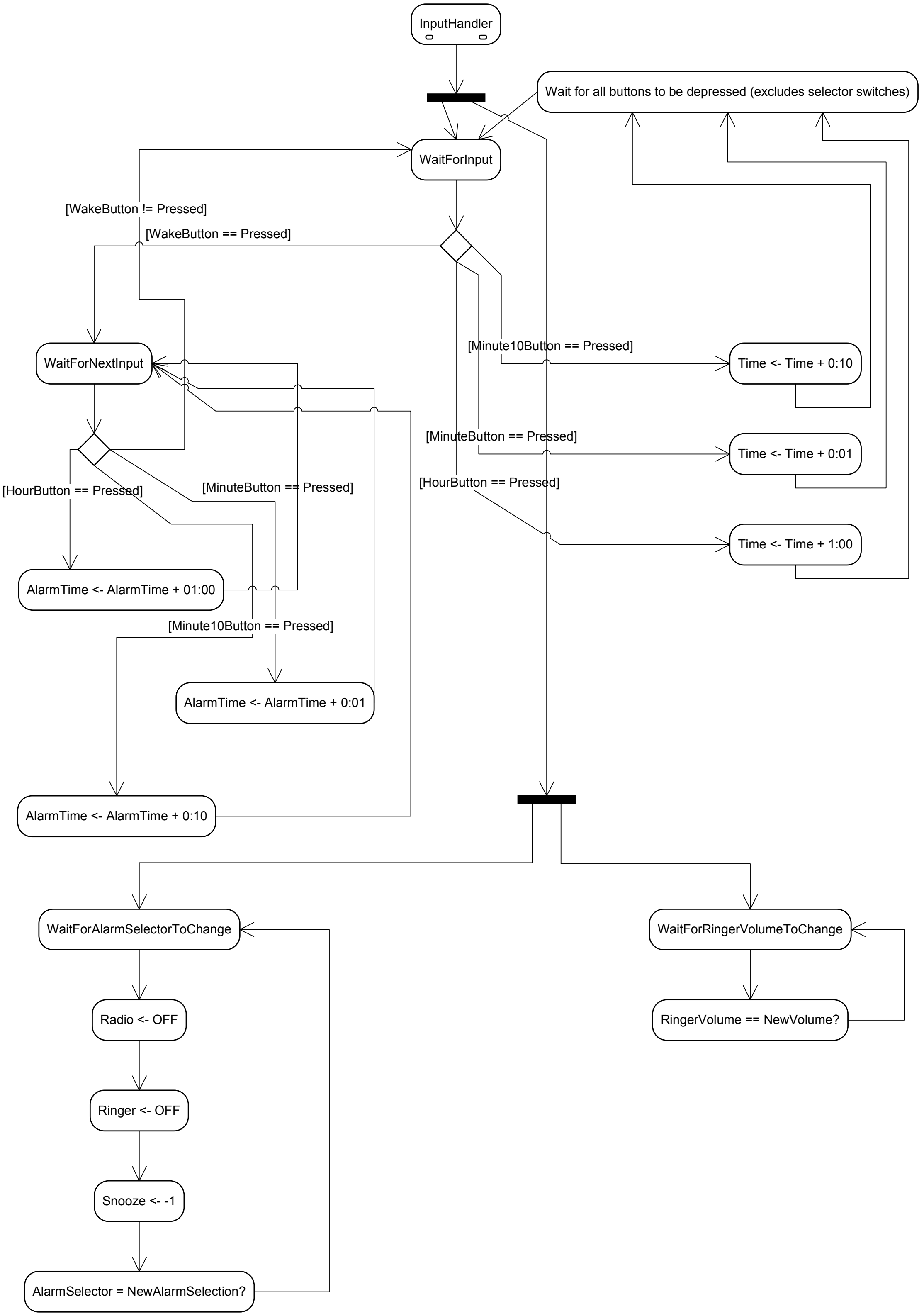
Simply displays the time on the display

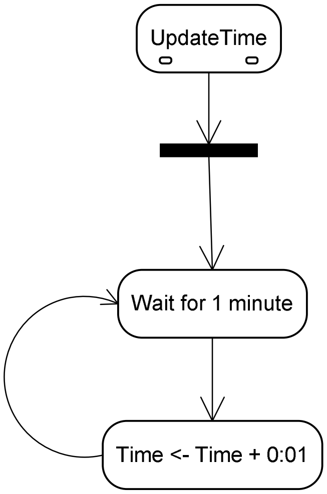
Initializes the time at 00:00 hours.
Updates the time every minute
Display the time

Updates the time or the alarm time as per the input instructions received

Verifies if the alarm is enabled.
If it is, verifies if it should sound.
If the alarm is sounding, wait for alarm to be turned OFF before disabling it
If snooze counter is >0 min. then hush the alarm and start counting the snooze timer back to 0 min.


AlarmHandler
[AlarmSelector == Off]
AlarmIsOff
[AlarmTime != Time]
Set WaitTime <- Time
[AlarmSelector != Off]
[AlarmSelector != Off]
[WaitTime != Time]
Basically waits for the next minute to arrive
[SnoozeButton == Pressed]
[Snooze == -1]
[AlarmTime == Time]
[AlarmSelector == Radio]
[AlarmSelector == Ringer]
Radio == OFF
Radio = On
Ringer = ON
Ringer == OFF
[Snooze == 0]
Snooze <- 10
[Radio == ON || Ringer == ON]
Set Wait2Time <- Time
[Wait2Time != Time]
[Snooze > 0]
Snooze <- Snooze - 1


InputHandler
Wait for all buttons to be depressed (excludes selector switches)
WaitForInput
[WakeButton != Pressed]
[WakeButton == Pressed]
WaitForNextInput
[Minute10Button == Pressed]
Time <- Time + 0:10
[MinuteButton == Pressed]
Time <- Time + 0:01
[HourButton == Pressed]
Time <- Time + 1:00
[HourButton == Pressed]
[MinuteButton == Pressed]
AlarmTime <- AlarmTime + 01:00
[Minute10Button == Pressed]
AlarmTime <- AlarmTime + 0:01
AlarmTime <- AlarmTime + 0:10
WaitForAlarmSelectorToChange
WaitForRingerVolumeToChange
Radio <- OFF
RingerVolume == NewVolume?
Ringer <- OFF
Snooze <- -1
AlarmSelector = NewAlarmSelection?



UpdateTime
Wait for 1 minute
Time <- Time + 0:01