METHODOLOGY

We have tried to model the Holter Monitor in different real-world scenarios. A typical Holter Monitor system consists of two basic parts- the hardware for recording the signal, and software for review and analysis. Often there is also a “patient” button, which allows the patient to press it in cases such as sickness. This places the a special mark on the recording, enabling the technicians to pinpoint these areas while analyzing.

The hardware part in our model comprises of the following components – Led, Alarm, Recorder, Internal memory, Leads and Battery. The model specifications are designed based on descriptions from EC-3H 3-Channel Holter ECG system by Labtech, manufacturer of PC-based ECG systems. It has a recording duration of 24-72 hours, with connections to PC via Bluetooth or USB. The complete specification can be looked up at the hyperlink ec-3h-3-channel-holter-ecg.

We have used timed automata based model checker UPPAAL to model the holter monitor. Initially the whole system is in *off* state. The system is started by a button.
Each state of the Holter monitor is depicted by an LED, and each transition from one state to the another by the ringing alarm. The Holter monitor also has a buffer mechanism, where it allows to simultaneously record and store the signals. There are two buffers which work in opposite phases and while one is being filled with recording, the other filled buffer is being flushed into the internal memory.

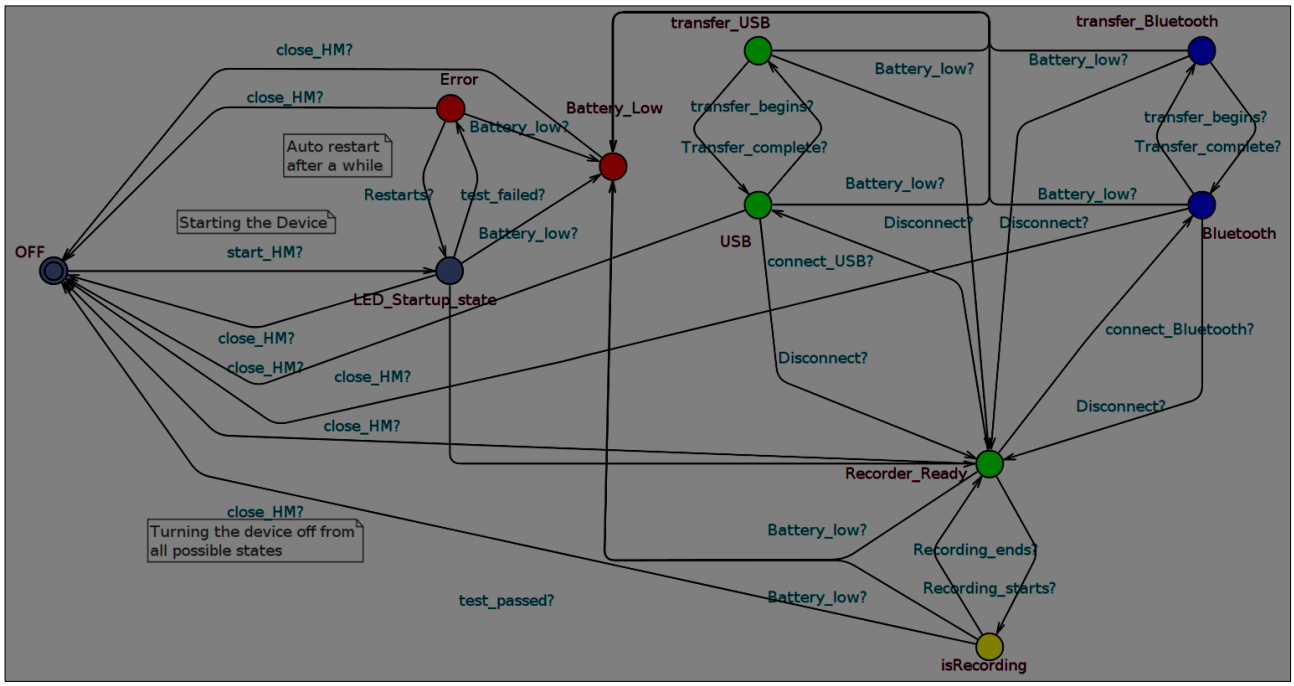
To incorporate functionalities of various components together, we use synchronization channels or signals. The signal ending with ‘!’ generates a signal, which is listened to by all transitions which ends with ‘?’.

The signals used in our model and their use case are mentioned below.

1. Battery_low – Signal generated when the battery level falls below some threshold. The Holter monitor can be expected to run for sometime before it is focibly switched off.
2. *start_HM, close_HM* – Signals generated to start and close the Holter monitor respectively.
3. *test_failed, test_passed* – These two signals enable us to take the transition corresponding to success or failure of the initialization test of holter monitor system after it has been switched on.
4. *Restarts* – When the test fails, this signal is generated which takes the system again to the testing state after some time.
5. *Recording_starts, Recording_ends* – Signals to enable the transitions which makes the recorder start or stop the recording.
6. *connect_USB, connect_Bluetooth* – Signals which enables the system to connect to the PC either via USB or Bluetooth. This allows data transfer to take place between the holter monitor and PC.
7. *transfer_begins, transfer_complete* – Signals which enables starting and ending of transfer of data between holter monitor and PC. This transfer can be either by USB or Bluetooth depending on the connection.
8. *Disconnect* – After the transfer of data is complete, this signals disconnects the holter monitor from PC to a *ready* state, from where it is ready to do further recordings.
9. *Buffer1, Buffer2* – These signals shows which buffer of the holter monitor is being filled with recording, while the other is storing it in the internal memory.
10. *Patient_uncomfortable* – This signal work by human intervention. This alert signal is generated by when the patient feels uncomfortable. While recording, this places a mark at particular timestamp of recording.
11. *Leads_off* – This signal is generated when the leads are not properly clamped onto the skin of the patient. This takes also takes the system to the alerted state as the previous state does.

12. *alert, alert_dealt* – Thes signals are used to take the system in and out of the alerted state.
13. ring_off – This signal marks the end of ringing of alarm. The alarm rings for sometime between 2-3 time units for every transition it takes from one state to another.

[I] This section gives detailed explanation of the LED component.



The recorder component acts as the control of the Led and Alarm component(explained next).

Each coloured state has a blinking pattern.
Red Led indicates that the system is in Error or Battery Low state.
For data transfer is taking place between the holter monitor memory and PC – Green Led indicates that transfer takes place via USB, and Blue Led indicates that transfer takes place via Bluetooth.
A differently blinking green led shows the recorder to be in ready state to start recordings or data transfer,
A blinking Yellow Led indicates that the recording is recording that ecg data.

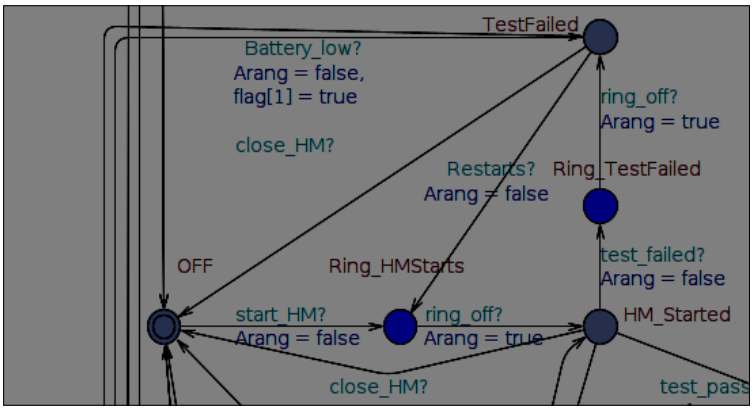
The Led process like every other process is started by the “start_HM?” signal. This takes it to LED_startup_state, where initialization test of recorder takes place.

When the system startup test is successful, it goes to Recorder_Ready state, from where recordings or data transfer can take place. When the test is unsuccessful, it goes to an Error state. The recorder then restarts after sometime, which is shown by “Restarts?” signal.

For the possibility of low battery, each state can listen to “Battery_low?” signal which takes the control to Battery_Low state. Here the Led remains red as long as holter monitor is switched off. Also each state listens to “close_HM?” signal, to take back the Led to the initial state in case the system is switched off when some process is going on.

[II] This section explains the Alarm Controller component.

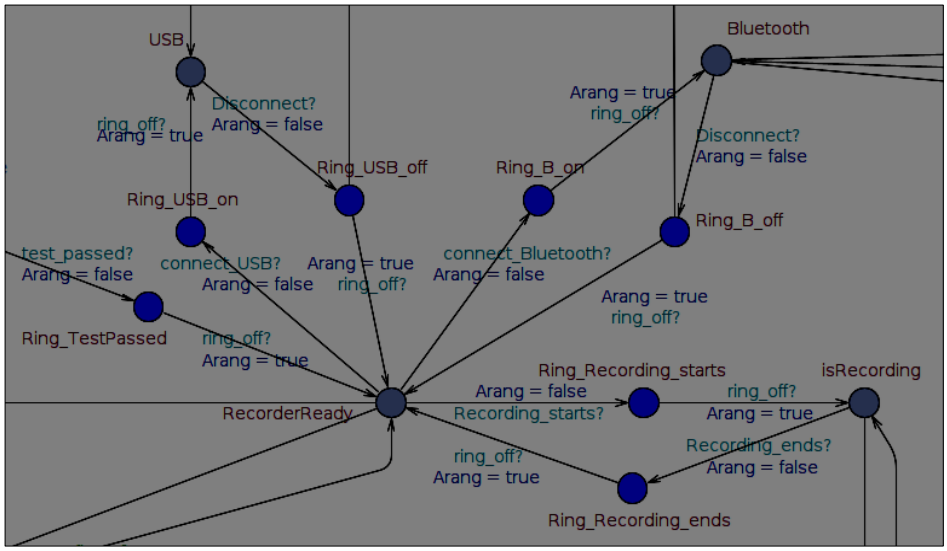
Blue state indicates ringing state. The pattern of the alarm ring for different transitions will be different.



(i)The Alarm is initially in Off state. Like Led, the component starts with the "start_HM?" signal. After every transition, the alarm is supposed to ring. So we make a boolean Arang, which tells whether the alarm has rang or not, as false.

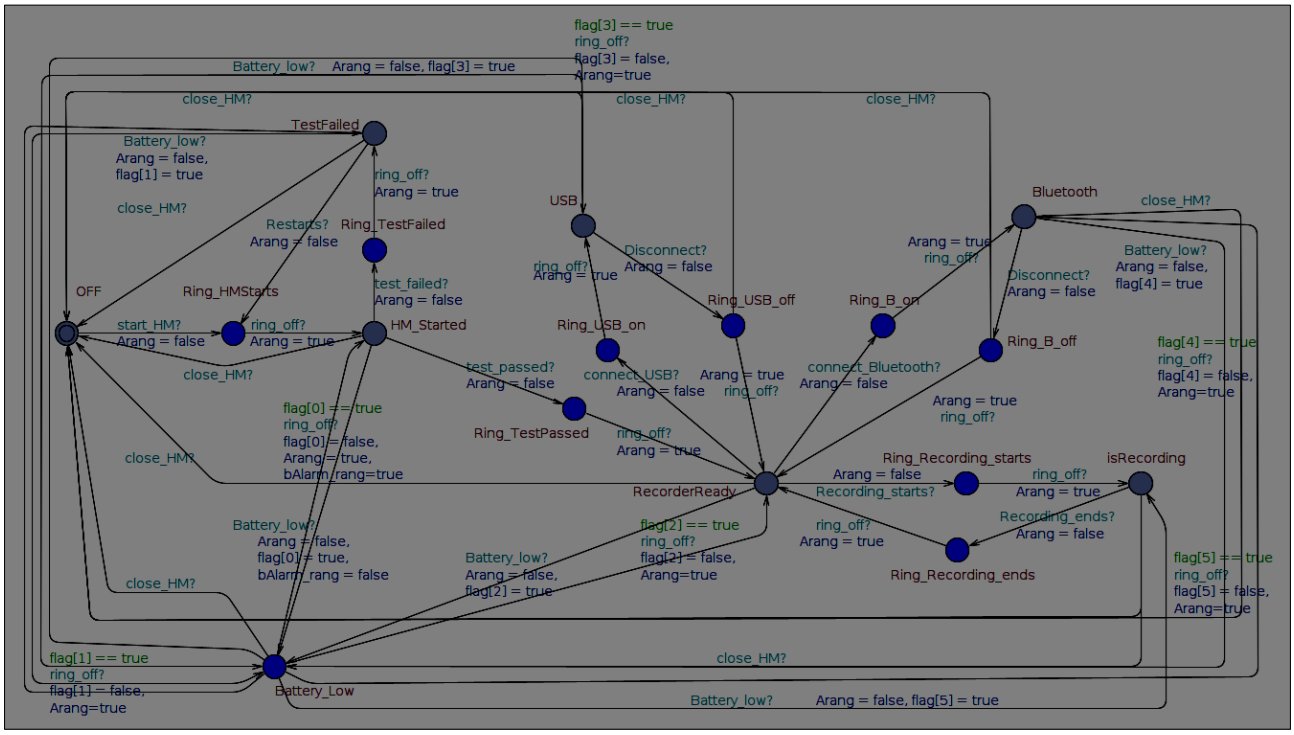
To make the alarm stop ringing, the ring_off signal is received. It takes the control to the HM_started state. It also makes boolean Arang as true.

The same pattern is followed for ringing of alarm in other states as well.
Like Led, each state also listens to "close_HM?" signal which takes the alarm to initial state.



(ii) When the startup test is passed, an alarm rings for it, and we reach the RecorderReady state. Now depending on whether new Recording is started, or connection is established to USB/Bluetooth, alarm rings for them at the start/ end of the action.

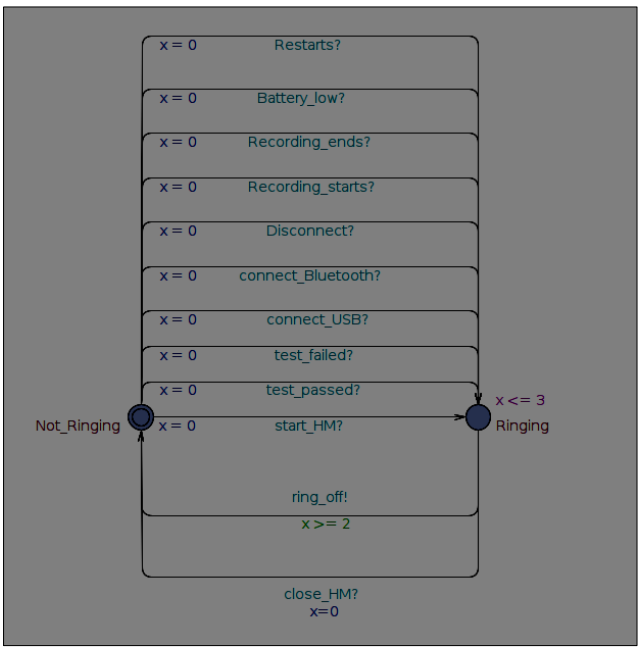
Each state of the Alarm component also listens to "Battery_low?" signal. When this is encountered, our alarm is supposed to ring for sometime and then control goes back to the state from where we got "Battery_low?". The normal action of alarm continues till the whole system is closed due to battery depletion.



This shows the complete template of Alarm part.

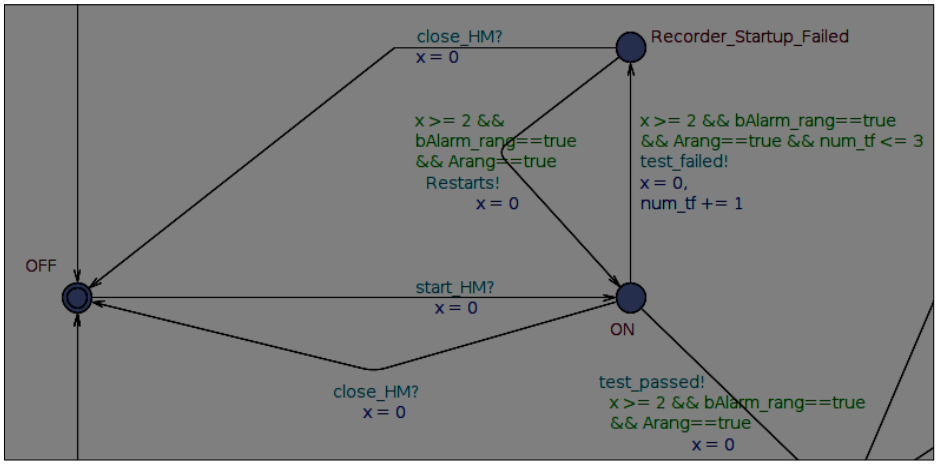
There is another process to handle the duration for which the alarm rings. Typically each alarm ring is unique, but here we model each beep to be somewhere between 2-3 time units. After this much time, it generates the “ring_off?” signal which is listened to by the alarm controller.

The template of the process to handle ringing of the Alarm is shown below.

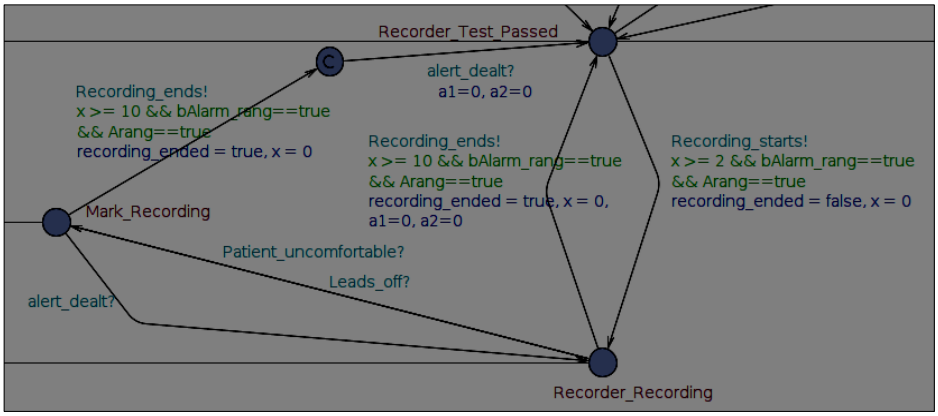


[III] This section shows the recorder part in detail.

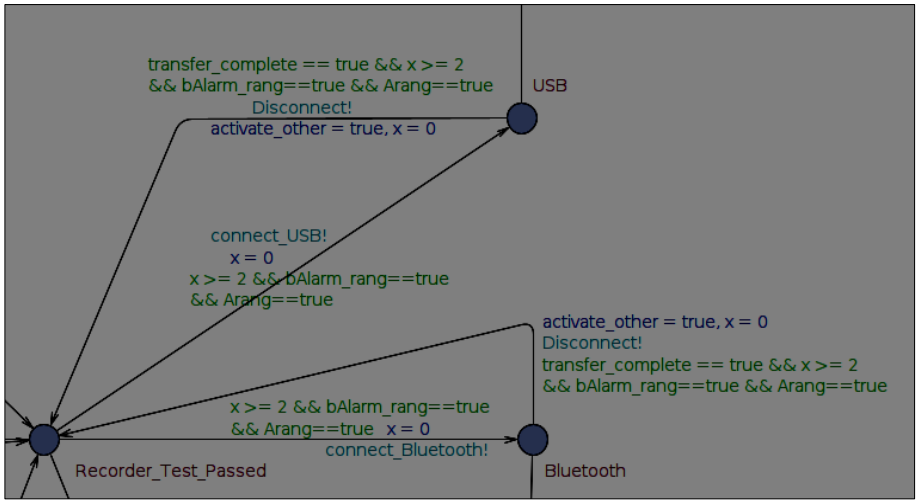
The recorder starts by listening to the “start_HM?” signal.



(i) Upon every transition, a clock is reset. Also following transitions has a guard condition of x >= 2. This allows the transition to be taken only when the alarm has rang which eliminates inconsistencies between different interleaved processes.



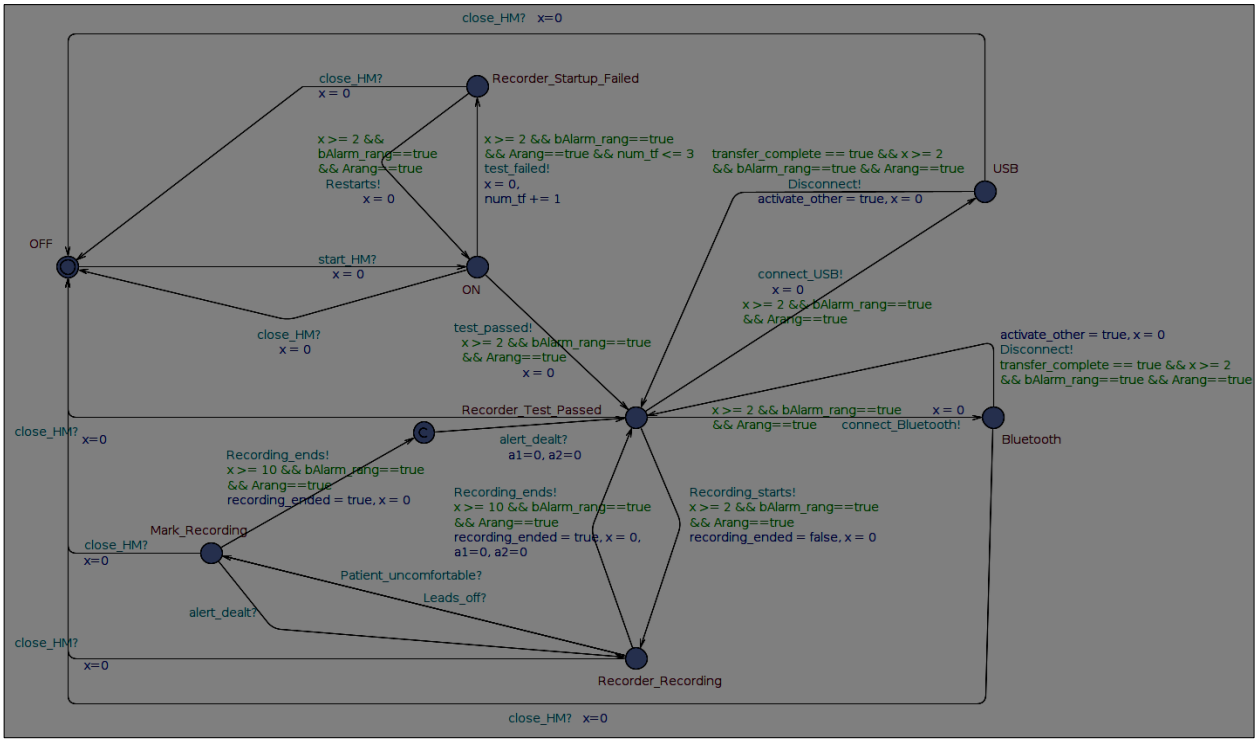
(ii) While recording is taking place, if we encounter alert signals such as “Lead_off” or “Patient_uncomfortable”, it takes the process to a Mark_Recording state. This is to indicate the timstamp when during the recording, the alert signals were generated.
If the alert is dealt during the recording, it returns to original Recorder_recording state. If recording_ends when the alert was on, it ends the recording from there.
In the updates, we make a boolean “recording_ended” as false when the recording starts, and make it true when it ends.



(iii) The recorder goes to USB and Bluetooth state indicating that a connection has been setup between HM and PC. During “Disconnect?”, we activate other signals such as battery_low, or alert signals. Also a boolean transfer_complete is set to true.

Here also each state can listen to the “close_HM?” signal to turn off the recorder.

This show complete template of the Recorder part.

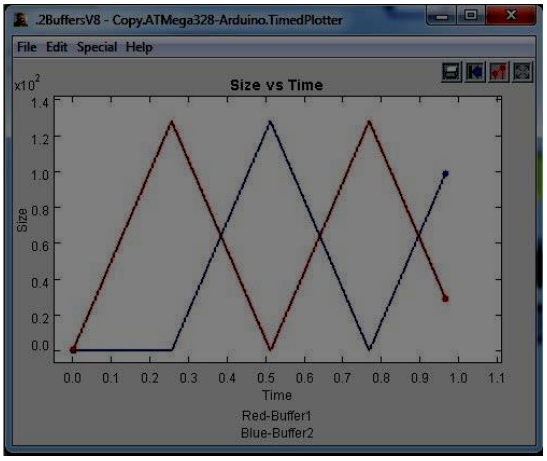


[IV] We have a buffer system to simultaneously record and store the recordings in internal memory. We have two buffers – one of which will be filled by ecg recording, while the second filled buffer will be cleared onto the internal memory storage.

is

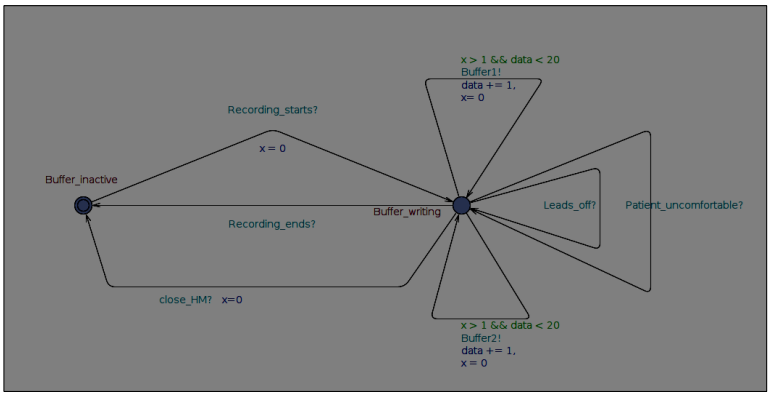
to

is



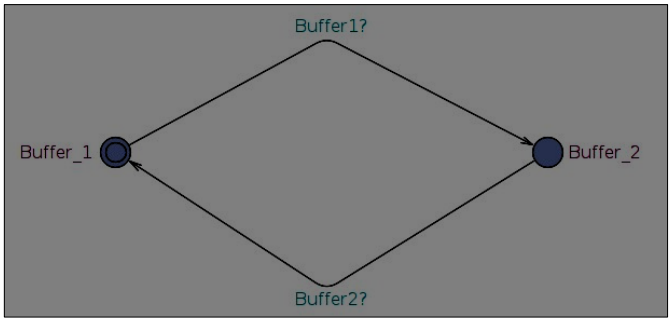
Two buffers are used to store and record the ECG signals simultaneously. When one buffer is is being filled with recording, the other being flushed in the local storage(which can record for upto 24 72 hours).

The buffers are active when the holter in recording state.



Template for the buffer write state. This components becomes active when the recording is on. While recording, it can listen to the different alert signals such as “Leads_off?” and “Patient_uncomfortable?”.

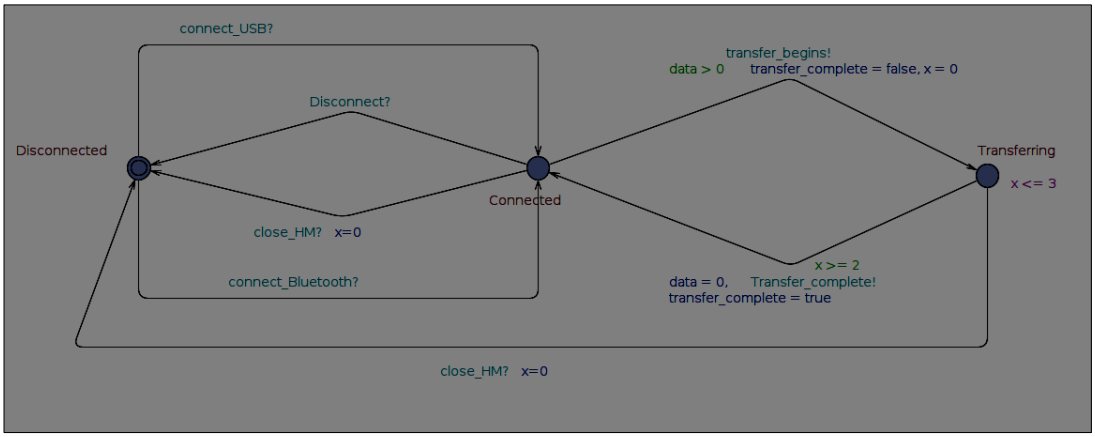
By using the guard condition x > 1, we say that at least 1 time unit is spent before one buffer is filled. Also a guard of data < 20 maintains an upper bound of data stored in the memory.



While the buffer switches state, we increment a counter in data to indicate the amount of data stored in the buffer. This counter will be used while transferring data to PC as a sanity check that we don’t go into transferring while there is no data in memory.

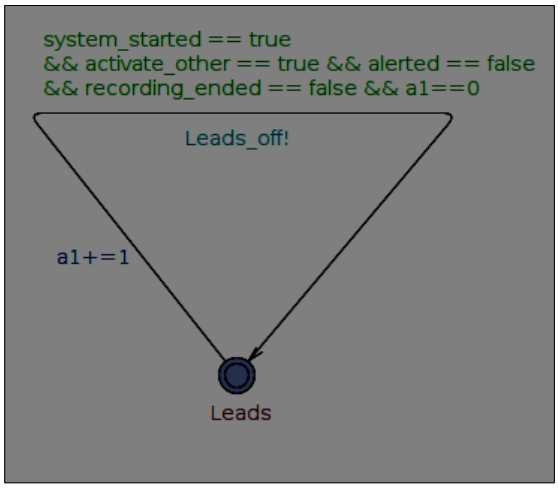
[V] Here we have shown that Data_transfer process, which is activated when the recorder is connected to Pc. It receiving the “transfer_begin!” / “transfer_complete!” signal to start/stop the transfer of recordings. Keeping in mind the memory constraint of internal storage, we have said that it takes somewhere in between 2 to 3 time units to complete on transfer of data.

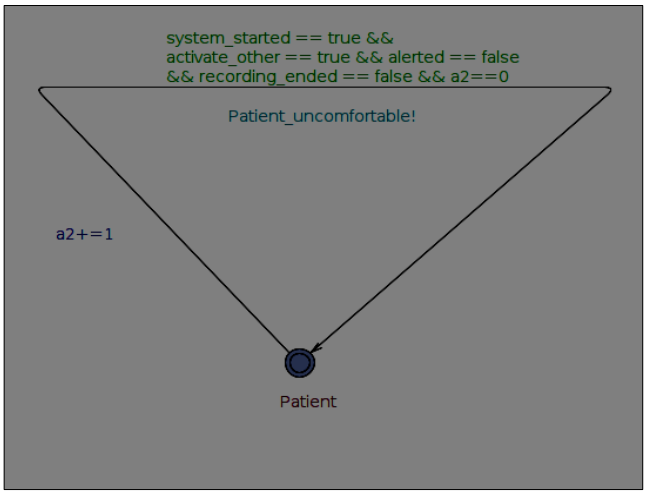
Here we update a

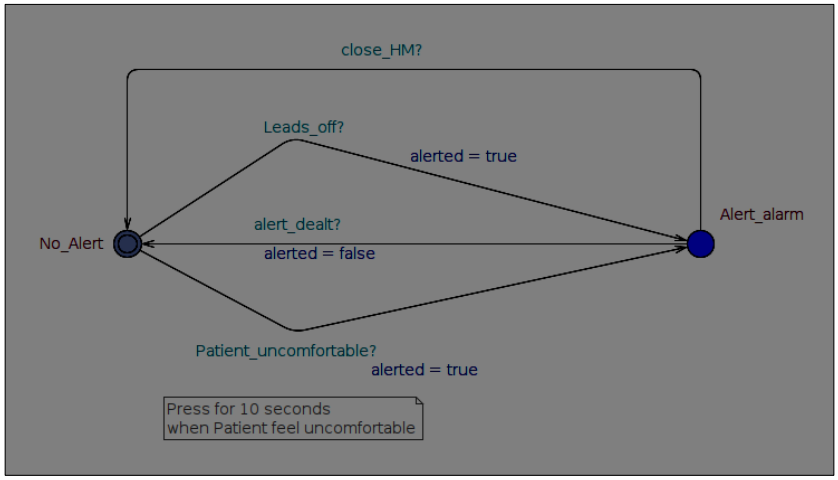


boolean transfer_complete as true/false as we start and end the data transfer, which is used by the recorder to only disconnect once the transfer is complete.

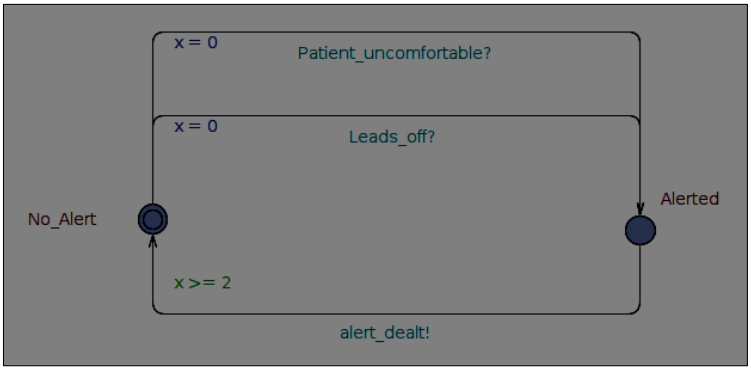
[VI] There are few signals which are under control of human or patient. These are the alert buttons or start/stop button.





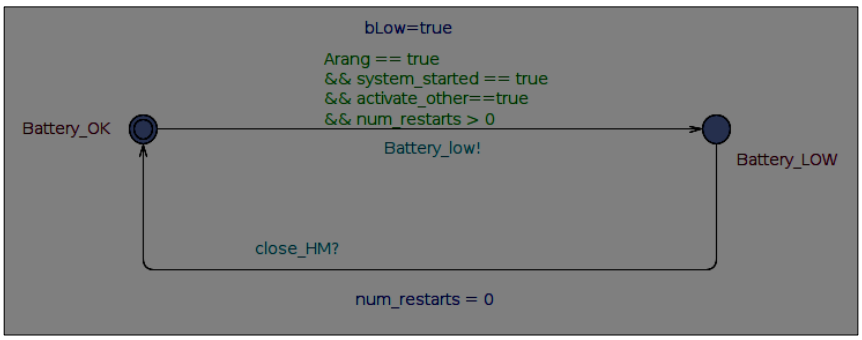


For alert signals, we use a different alarm than the one for other processes because it need not follow the normal timing constraints of other transitions. The alert being in control of the patient will be switched off by them only.



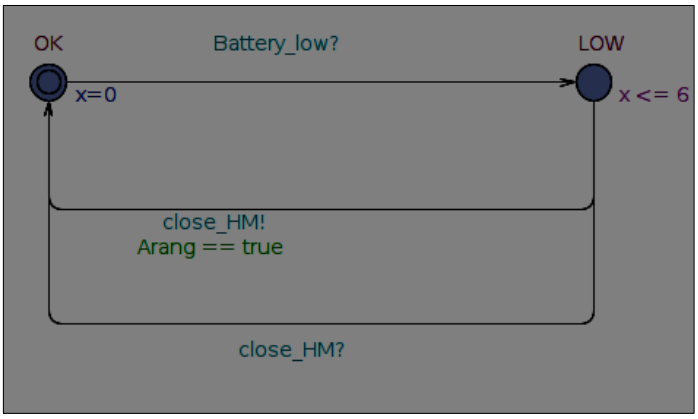
We will also prevent the system from going into alert once it comes out of it, by enabling it only once. We only put a lower bound on the timings of alert alarm as 2 time units.

[VII] The last component of the model is the battery.



The system is able to generate the battery low signal when the system has restarted for some n number of times. We can also model the battery as a clock which will become low once it reaches some value.

Once the battery becomes low, the holter monitor system will continue to work normally for at most some fixed time units(6 in our case) by putting an invariant on the LOW state. After this invariant becomes false, the process must exit the state and doing so, it will forcibly close the Holter monitor, will all other processes also going back to the initial state.



VERIFICATION

The main purpose of a model is to verify the model with respect to a requiement specification. UPPAAL uses simplified version of Computation Tree Logic (CTL) about how the state of a system can evolve over time. It consists of atomic propositional logic formulas, plus temporal connectives.

Some queries which we verified in our model are -

1. A<> (Data_transfer_process.Connected and data==0) imply Data_transfer_process.Disconnected
- In every path of the computational tree, there is a state where if the Recorder is connected to USB/Bluetooth for data transfer, and there is no recorded data in memory, the immediate next state is Disconnected. This means that no transfer of data of data ever takes place when there is no recorded data to transfer in the first place.
This query is found to be satisfied.

2. A<> (Battery_process.Battery_LOW) imply (Battery_low_process.x <= 6 and LED_process.OFF)
- We have set an upper bound time on running the Holter monitor when the battery is low. Here we verify that the LED comes back to initial state within 6 time units after Battery_low is encountered.
This query is found to be satisfied.