CIS 541: Embedded Software for Life-Critical Applications

Pacemaker Project Milestone 2: Implementation Design

Mauricio Mutai, Jack Harkins, Gabriel Naghi, Sylvester Tate

Setup

Compile heart and pacemaker code bases and upload them to their respective mbeds. Both programs assume the LCD connection demonstrated by the course material and used in the Timer assignment.

Heart/Pacemaker Interface

Pacemaker

Vpace p8

Apace p7
Vget p6
Aget p5

Display Pin Mappings

Display	**mBed**
1	GND
2	Vo (5 V)
3	GND via 1.8 kΩ resistor
4	P9
5	GND
6	P10
11	P11
12	P12
13	P13
14	P14

Heart

Vpace p10
Apace p11
Vget p13
Aget p12

Code Logic

Pacemaker

In our pacemaker code, we use 8 threads: five for the timing cycles and three for peripherals. The threads are initialized in pacemaker.cpp (note that we had to decrease the stack size of each thread in order to allow them to run).

URI
This thread is tasked with controlling the global system clock (clk) and playing the speaker if the heart is beating too fast (ventricular paces or senses before clk reaches TIME_URI) or if the heart is beating too slowly (the timer reaches TIME_LRI and a ventricular sense has not occurred). The “speaker” in our system is not a sound-producing speaker, but rather text that is displayed on the LCD when a heart rate warning has been triggered.

LRI
The LRI thread’s job is to send an atrial pace (Apace) when clk has reached TIME_LRI - TIME_AVI. This Apace is timed in this manner in order to allow for the pacemaker to send a Vpace, if necessary, once clk has reached TIME_LRI.

In both the lri and Ased cases, we first check to see if SIG_FORCEAPACE is high. This signal is triggered when the user presses the ‘a’ key on the keyboard to force an Apace in manual mode. The idea behind this is that we want the pacemaker to operate normally in the absence of keypresses, and when a pacing key is pressed, the user’s action should take precedence.

VRP
The VRP thread is tasked with converting Vget signals (from the heart) into Vsense signals as long as the time since the last ventricular event (Vget or Vpace) is greater than or equal to TIME_VRP. The Vsense signal is then used by other threads in the pacemaker to determine when a ventricular signal sent by the heart (but not a ventricular pace) has occurred. To check the time since the last ventricular event, we use a Timer (vrp_timer) that is reset upon a ventricular event occurring.

PVARP
This thread's job is similar to that of the VRP thread. It waits for an Aget signal from the heart and only triggers an Asense (analogous to the Vsense described above) when an internal timer (pvarp_timer) is greater than TIME_PVARP. This is done to ignore noise on the Aget line that may result in the refractory period after the atrium and ventricle have been sensed or paced.

AVI
The AVI thread is responsible for sending Vpace signals when a local timer (avi_timer) has exceeded TIME_AVI or TIME_URI, depending on which state this thread is in. If this thread is in state AVI, which means that it is waiting for a venticular event (VSense), and no VSense signal is received by the timer the timer reaches TIME_AVI, then the thread sends a Vpace to enforce the AVI timing constraint between atrial and ventricular events. If this thread is in state WAIT_URI instead, it is waiting for the URI time period to be up before sending the Vpace signal. This prevents the AVI thread from violating the URI timing constraint, which could happen if it sent a Vpace without checking this condition.

Keyboard
The keyboard thread handles all keyboard input for the pacemaker. If a character is ready to be processed, it handles the input according to whether the pacemaker is in manual mode.

If the pacemaker is not in manual mode, the keyboard handles key presses according to the following rules:

- N: sets the timing parameters (TIME_URI and TIME_LRI) to normal pacing mode
- S: sets the timing parameters (TIME_URI and TIME_LRI) to sleep mode
- E: sets the timing parameters (TIME_URI and TIME_LRI) to exercise (sports) mode
- M: enables manual mode (or disables it, if it has already been enabled)
- O: allows the user to start setting the observation window using the number keys.

If the pacemaker is in manual mode, in addition to the keys above:

- A: forces an Apace signal
- V: forces a Vpace signal

After pressing 'O', the user can set the observation interval in seconds (max. 256) by typing three integer digits and then pressing the enter key. The display will now start to update at this new observation interval as soon as it refreshes again.

Heart Rate Display
This thread periodically updates the heart rate display every obs_interval seconds with the new heart rates on the LCD using a timer heart_rate_timer. This update is triggered as a callback when the timer has counted up to the observation interval. In our current implementation, for simplicity, we are simply displaying the number of atrial (A_ticks) and ventricular events (V_ticks) that have occurred in the observation interval.

External Signals
This thread acts as the interface between the RTOS thread signals and the mBed's external GPIO pins. Specifically, it is used for capturing Aget and Vget from the heart and translating these physical signals into the RTOS signals for the pacemaker, as well as for capturing the pacemaker's Apace and Vpace RTOS signals and translating them into external Apace and Vpace pin toggles for the heart.

For capturing the Aget and Vget signals from the heart, we declared Aget and Vget as rising edge interrupt-driven GPIO pins. When either of these pins receives a rising edge from the heart, this thread emits the appropriate signal (Aget or Vget) to all threads. The external signals from the heart arrive in the form of a 10 ms pulse that is controlled by a timer (see Heart section below for details).

For translating the Apace and Vpace signals into physical pin toggles, this thread listens for these signals in its main loop and, upon receiving one of them, enables a 10 ms timer and sets the appropriate pin to high. When the timer expires, the callback function associated with the timer (reset_vpace() or reset_apace()) then sets the pin to low. The effect of this is that when an Apace or a Vpace is sent by a thread in the pacemaker, the appropriate pin is set high for 10 ms.

Heart

The heart code was generated from the UPPAAL model deterministically, modeling each significant automaton as a single thread. The two threads which diverge from this pattern are the Keyboard automaton, which was replaced by a human operator, and the LED automata, which were combined into a single led manager thread. There are 5 main threads which were implemented:

Generator
The generator is responsible for detecting pacing signals internal to the heart and broadcasting them as Aget and Vget to the pacemaker. The pacemaker is initialized in a Waiting state, where it checks for Asignal or Vsignal signals. If it gets one, it transitions to the respective gotA or gotV state. If the heart is in one of these states, it increments its respective counter, sets high the external signal, schedules a timer to disable it after some time, and transitions back to the waiting state.

Responder
The Responder thread rotates between three main states based on keypresses. These are Random, Manual and Test. Each of these states have sub-states which send and receive heartbeat signals. In random, the heart can be force-paced by Vpace or Apace, or after some time delay, randomly transitions to a send state. Any of these four sub-states then fires its respective Asignal or Vsignal and transitions back to Random state. Manual mode has similar behavior except, instead of force pacing complimented by random transitions, it is complimented by keypress-induced transitions to Asignal and Vsignal firing states. Last, the Test state waits for a keypress of 1-10 to begin execution of the

corresponding test. The tests traverse a number of states which together verifiy the behavior of the pacemaker. See the Testing companion document for more information about the test cases.

Display
The display automaton waits at least some observation interval, after which it prints the number of Asignals and Vsignals that were generated over the time period to the LCD. It then resets the counts and the interval timer. There is only one state in this automaton: the Display state.

Led
While the original design was to eliminate LED automata entirely and flip LEDs inline with events, we eventually decided to create a LED manager thread. The LED manager is responsible for capturing the signals corresponding to various LED events and flipping the corresponding LED. It does this by waiting in its singular state and upon capturing such a signal, toggling the corresponding led.

External Signals
This thread is identical to that implemented in the pacemaker, with the exception that instead of the led flipping inline, we separated that functionality into a separate thread as described above.

References

https://canvas.upenn.edu/courses/1336112/files/folder/Project/Pacemaker%20References?preview=60098294

https://canvas.upenn.edu/courses/1336112/files/folder/Project/Pacemaker%20References?preview=60098454

https://canvas.upenn.edu/courses/1336112/files/folder/Project/Pacemaker%20References?preview=60098495