Abstract

In this project we aimed to create a pacemaker implementation in both C and SCCharts for the Nios II softcore processor to run on an FPGA board. This pacemaker acts in DDD mode, which means it senses and paces in both the atria and ventricles of the heart, as well as both inhibiting invalid pulses from the heart, and triggering pulses when necessary for the hearts' function.

Introduction

The purpose of this pacemaker is to treat a few different conditions on a simulated heart, which include; no AV node autorhythm, no SA node autorhythm, and no AV conduction. Combinations of these conditions can also be treated with a DDD mode pacemaker, such as this project.

The project includes multiple modes of operation, two for input/output method, and two for pacemaker model. The project features manual user input mode, where the user can use buttons to simulate external heart stimuli, and the pacemaker's response is shown on LEDs of the FPGA board. The device can also be configured to operate on a simulated heart via UART through the RS232 port on the board. The pacemaker model can either be an implementation in C or in SCCharts, both of which have identical functionality. Modes can be switched by using the on board switches of the FPGA.

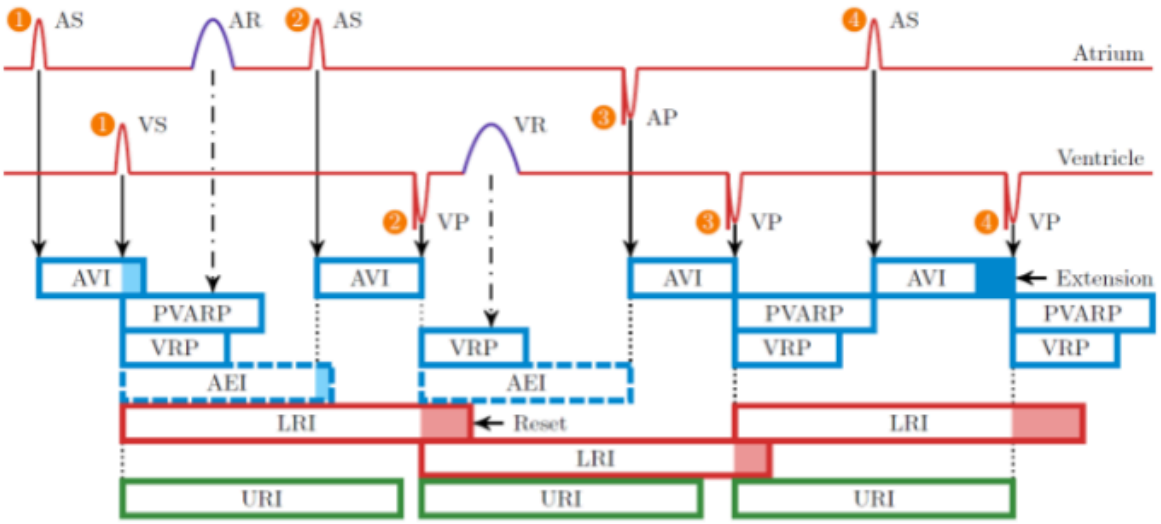

Fig. 1. Timing diagram of a DDD pacemaker. [1]

[1] [1] Srinivas Pinisetty, Partha S Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, Reinhard Von Hanxleden. Runtime Enforcement of Cyber-Physical Systems. The ACM SIGBED International Conference on Embedded Software (EMSOFT), Seoul, South Korea, October, 2017

Approach

SCCharts implementation

The first step in implementing this project was creating an implementation of the pacemaker functionality in SCCharts, this would give us an initial reference for how the pacemaker should operate practically, as well as a baseline for a method of implementation in C.

Our approach for the SCCharts implementation involves having a separate region for the operation of each of the model's timers, in order for them to operate separately and simultaneously. The URI (Upper rate interval) region determines whether the pacemaker can output a ventricular pulse, as URI is the upper board for the ventricular pulsing rate. The LRI (Lower rate interval) region will always create a pulse when timing out, as LRI determines the lower bound for the ventricular pulsing rate. The PVARP (Post ventricular atrial refractory period) and VRP (Ventricular refractory period) regions blocked atrial and ventricular events from occurring too rapidly, by ignoring any atrial or ventricular events that would happen before a fixed timing delay. The AEI (Atrial escape interval) and AVI (Atrioventricular interval) regions pulse the atrium and ventricle respectively if there has not been a natural heart pulse in a fixed amount of time.

Each region is reset by the beginning of a heart pulsing cycle, marked by any ventricular event, whether natural or paced. This is controlled in the pacemaker by a signal 'v' which will be emitted if any ventricular event occurred in the previous tick, this is pre() controlled to eliminate any non-deterministic behaviour caused by the v signal being emitted and consumed in the same tick. All the fixed timer lengths are controlled by an external header file called timing.h which controls the number of milliseconds (or ticks) the timers run for.

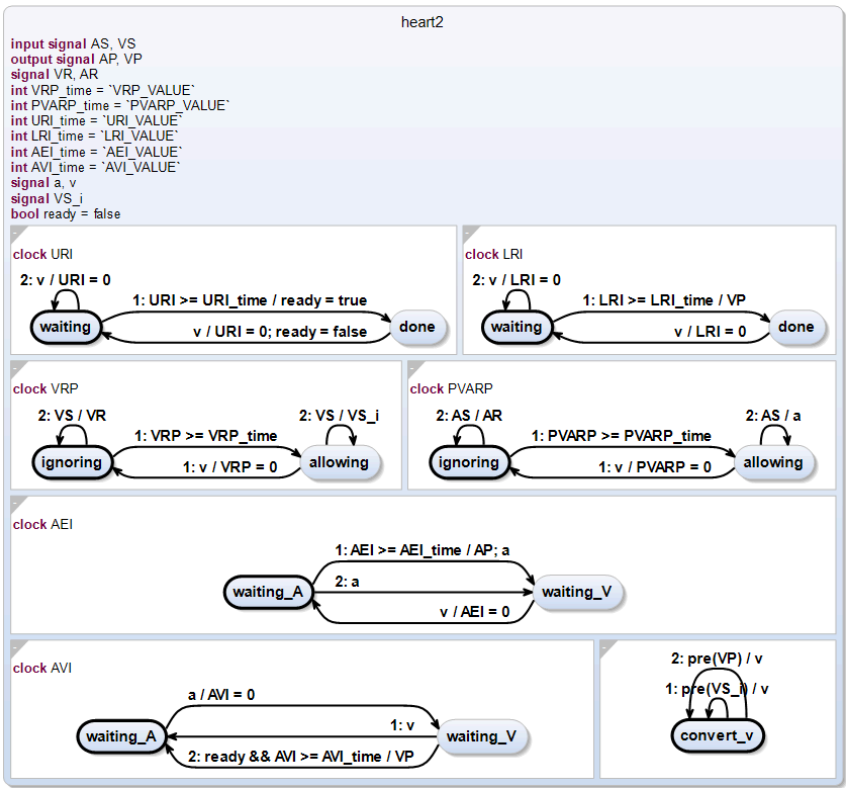

Fig. 2. Our SCCharts implementation.

C implementation

The pacemaker implementation in C was very similar to our SCCharts implementation in structure, except for a few modifications to how timers are used. The URI and LRI regions in the SCChart were combined into one ventricular rate timer, with a combined ISR. Identical functionality was achieved by setting the timer to timeout after a URI delay, then timing out for a second time after the remaining amount of time of the LRI delay. This was possible due to the fact that the URI and LRI timers will always begin at the same time. The AVI and AEI regions of the SCChart were combined into one region due to the fact that AVI and AEI will never be active simultaneously, as therefore the timer will alternate between AVI and AEI timeouts depending on whether an atrial or ventricular event happened most recently. PVARP and VRP are implemented in the same way as the SCCharts, each using a separate timer.

The timers are reset in a similar way to the SCChart implementation, resetting on any ventricular event. This is implemented by setting a flag on any ventricular event, and polling this flag in our main while loop, calling our reset function and resetting the flag when triggered. This approach has less timing issues then the implementation in SCCharts due to the computational demand of the C implementation being lower.

Functional overhead

In order for our pacemaker to function, it needs to be able to handle input and output of simulated atrial and ventricular events, as well as the mode switching between our different pacemaker implementations. Timing parameters are stored in a header file, which contains values in the number of milliseconds for each of the pacemakers timer timeout values. This timing information is used for both the C mode and SCCharts mode.

Our main function has 3 main steps per loop; poll mode switches, poll input signals (atrial sense and ventricular sense), and update the active pacemaker model. Polling the mode switched just involves calling the Nios II IORD functionality to check the value of the switches. Then, depending on this result, the program will know to check either the hardware buttons, or the UART receive buffer for the presence of an input signal. If an input signal is received, it will set the appropriate field when in SCChart mode, and call the respective atrial and ventricular functions for C mode. Once input is handled, it will reset the C timers if the flag is set. The SCChart is ticked in an ISR with a period of 1 millisecond in order to have high execution priority.

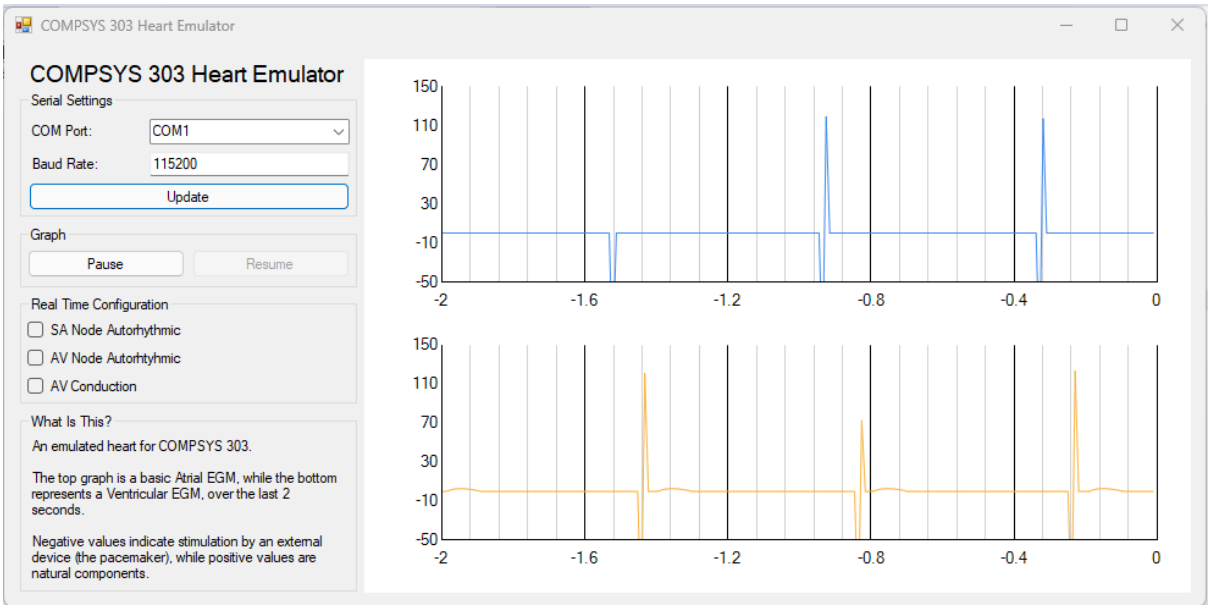

Fig. 3. Our pacemaker pacing the emulated heart.[2]

Difficulties encountered

While doing the project, there were a few issues concerning timing in a real-time embedded system. While developing the SCCharts design, critical compilation errors arose with initially unclear causes. Root-cause analysis traced the issue to the evaluation of certain conditional expressions. Because an input could drive the ventricular pulse (VP) to produce different outcomes, the system became non-deterministic, and the design was deemed non-constructive, leading to the compilation failure. This issue was addressed by using pre() to introduce a single tick(1 ms).

Although the initial SCchart design was proven to be functional by passing all the test cases, the output on the heart emulator did not match the intended behaviour. The discrepancy resulted from the host system assigning equal priorities to competing protocols, introducing timing delays. We resolved this by enabling ISR-based ticking, which assigns higher priority to I/O polling and eliminated the timing interference.

For a time-critical embedded system, missing a tick could result in severe outcomes. Because of the nature of the pacemaker device, all the possible measures were taken to emulate realistic cardiac behaviour by controlling various factors. However, with our limited heart knowledge, it was often difficult to recognise whether or not the behaviour was desirable. By thoroughly examining a given project specification, we determined the nature of the pacemaker's behaviour.

When implementing UART, we were not aware that using the RS-232 serial port was required. After being informed about the RS-232 serial port, it was still believed that it is not working as intended because no new communication port was showing up on the Windows device manager. However, later it came to our knowledge that the communication port always showed up as COM1.

---

[2] Avinash Malik, Partha S Roop, Nathan Allen, and Theo Steger, “Emulation of Cyber-Physical Systems using IEC61499”, IEEE Transactions on Industrial Informatics, July 2017.

Conclusion

Through this project, we have successfully demonstrated a synchronous approach to concurrency by creating and implementing a DDD mode pacemaker in both SCCharts and C. It was observed that functionally, an implementation using SCcharts and an implementation using C work very similarly. Through this project, we were able to understand the “model-driven” approach of designing an embedded system and interface the code with the Altera Nios II IOs. We gained a better understanding of embedded systems through this project.

We have used the given header files to input timing parameters manually. Going forward, being able to input timing parameters using a user interface would be desirable, as it would allow us to simulate more heart conditions without having to program the code.
Subsequently, introducing different pacing modes would increase the support for different conditions.

Time taken

Breaking down the project, SCcharts design took about 3 hours to complete and C implementation also took about 3 hours to complete. Implementing mode switching and SCcharts took about 3 hours to complete. Report writing and preparing for the presentation took about 4 hours to complete.