COMPSYS 303 Assignment 1

Designing a DDD Mode Pacemaker with SCCharts and Nios II

Due on 25th August (Week 6)

Learning outcomes

- Students will understand the synchronous approach to concurrency.
- Students will be able to design a real-time system using the synchronous approach.
- Students will understand the "model driven" approach of designing an embedded system.
- Students will be able to design a "reactive interface" to interface the code generated by the SCCharts tool with the Altera Nios II IOs.
- Students will understand the design of a biomedical embedded system.

For this assignment, you are required to create an implementation of a DDD Mode pacemaker [1] in SCCharts. This implementation will then need to be run on a Nios II CPU in order to interact with the environment as an actual device, a process known as emulation [2]. Subsequently, you will test your implementation against a virtual heart [3] running on a computer, communicating over UART to your board.

Pacemaker Overview

Pacemakers are devices used to correct bradycardia in the heart, a case where the heart is beating too slowly, by applying electrical stimuli to the heart at certain times. Sensing of the heart is accomplished through two inputs: Atrial Sense (AS) and Ventricular Sense (VS). Meanwhile, actuation of the heart is through two outputs: Atrial Pace (AP) and Ventricular Pace (VP).

Pacemakers are categories through a three or four character name which represent their operation, such as DDD, outlined in Table 1.

Table 1: Pacemaker classifications

Character Index	Meaning	Possible Values
1	Sensing Location	• A – Atrium • V – Ventricle • D – Dual (both)
2	Pacing Location	• A – Atrium • V – Ventricle • D – Dual (both)
3	Pacing Method	• T - Triggered • I – Inhibited • D – Dual (both)
4 (Optional)	Rate responsiveness	• R – Rate responsive

The pacing method options can be simply summaries as the “Triggered” mode meaning that pacing is only performed **when an event is not detected**, while the “Inhibited” mode means that pacing is performed all the time **unless an event is detected**. Dual mode pacemakers typically use multiple timers and conditions that utilise a combination of each of these methods for overall function.

For example, the DDD mode pacemaker will sense in both the atria and ventricles, pace in both the atria and ventricles, and support both Triggered and Inhibited pacing for its corrective actions. Such a pacemaker will make use of both atrial and ventricular signals in order to ensure that both the heart rate is sufficiently high and an acceptable delay is maintained between atrial contractions and ventricular contractions. Alternatively, a basic VVI pacemaker will only have leads in the ventricles (VS and VP only) and only utilise the Inhibited method for its pacing. As such, the VVI pacemaker is only able to ensure that the ventricles beat at an acceptable rate, and not maintain synchronisation between the two chambers. You may refer to the SCCharts design of a VVI mode pacemaker in lecture2.pdf on page numbers 31 and 32.

DDD Mode Timings

As described before, you are asked to design and implement a DDD mode pacemaker which senses and actuates in both the atria and ventricles. The operation of this device is achieved through the use of a series of timers, as outlined below, which perform certain actions when they expire.

- AVI – Atrioventricular Interval
  - The maximum time between an atrial event and its subsequent ventricular event.
- PVARP – Post-Ventricular Atrial Refractory Period
  - The time after a ventricular event where any atrial events are ignored as Atrial Refractory (AR) signals.
- VRP – Ventricular Refractory Period
  - The time after a ventricular event where any other ventricular events are ignored as Ventricular Refractory (VR) signals.
- AEI – Atrial Escape Interval
  - The maximum time between a ventricular event and its subsequent atrial event.
- LRI – Lower Rate Interval
  - The slowest rate at which the heart is allowed to operate. This is measured as the time between ventricular events.
- URI – Upper Rate Interval
  - The fastest rate at which the pacemaker will ever pace the heart at. This is measured as the time between ventricular events.

Figure 1 shows how these timers interact with the heart in a DDD mode pacemaker. The first two graphs are the Electrograms (EGMs) of the heart, from both the atrium and ventricle, which the pacemaker is processing. These graphs have been annotated with whether the events are AS/VS signals that are accepted, AR/VR signals that are ignored, or AP/VP signals that are generated by the pacemaker.

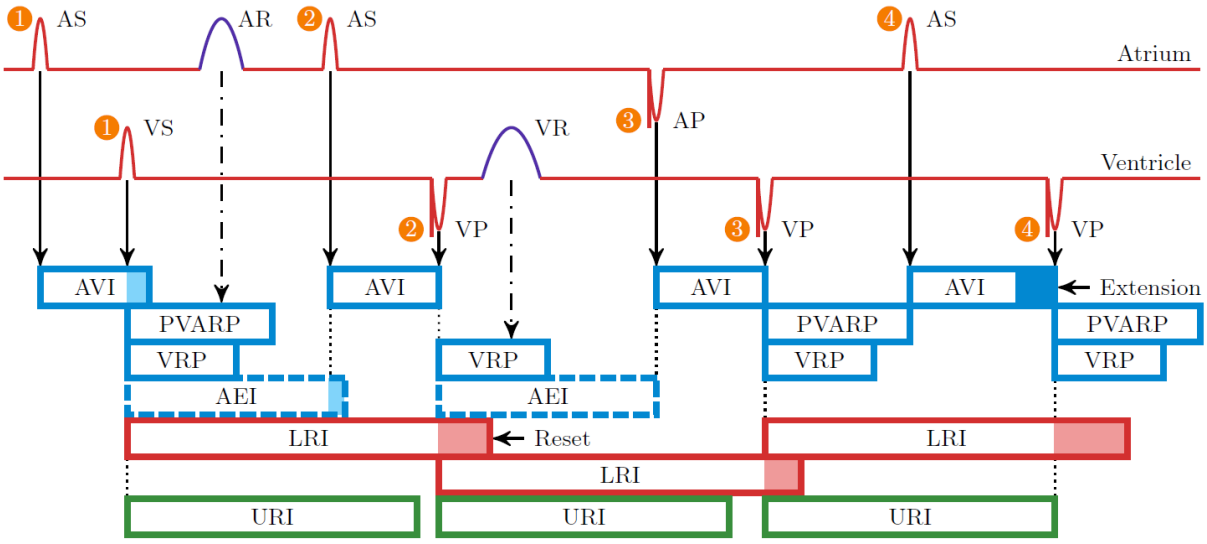

Figure 1: Timing Diagram of a DDD Mode Pacemaker. Note that some timers have been omitted at times for readability. Reproduced from [4]

The lower half of Figure 1 shows each of the previously described timers in operation. Here, the length of each box is its maximum value, while any lightly coloured portion of the box is any un-used part, after the timer has been reset early. Examples of this include resetting the AVI timer when a VS signal is received (1), resetting AEI when an atrial signal is received (2), or resetting LRI when a subsequent ventricular event occurs (2).

The figure shows how AR and VR signals are generated due to atrial or ventricular events happening within their respective refractory periods (PVARP or VRP). Also take note of how AEI and AVI timers can be used to generate AP and VP signals across beats 2, 3, and 4. One further point can be seen about the AVI timer in beat 4, which is labelled as “Extension” and denoted by solid shading. In this case, the AVI timer expiration would have caused a pacing of the ventricle at a rate faster than the URI timer would allow. Instead of this, the AVI timer is extended up until the point where pacing is allowed by the URI timer. This behaviour happens since the URI bound has higher priority over AVI.

You may also notice that in this example diagram it may be observed that $AEI + AVI < LRI$. Note that this does not have to be true, and most often the inverse should be true ($AEI + AVI > LRI$) in order for the LRI timer to actually have any meaning. This is merely an example, and your implementation should be able to deal with both of the cases.

In modern pacemakers the values for each of these timers are configurable, as different patients require different treatments, hence the above diagram should not be treated as the only possible set of timings – it is merely an example.

Your Task

For this assignment, you are requested to design (using SCCharts) and implement (using Nios II) a model of a DDD mode pacemaker. The logic for your design should be implemented in SCCharts, while the mapping of internal signals to external outputs should be performed in C, similar to what was done in Lab 2, using proper use of multiple C files and header files.

Additionally, your design should have the ability to easily change the timeout values through a series of #define statements in a header file. A set of example timeouts is provided below, where each time is presented in milliseconds. Feel free to change these to test various features of your design.

```
// Example timeout values in milliseconds
#define AVI_VALUE 300
#define AEI_VALUE 800
#define PVARP_VALUE 50
#define VRP_VALUE 150
#define LRI_VALUE 950
#define URI_VALUE 900
```

Your design should be able to operate in two modes, selected using one of the switches.

Mode 1

This mode will use KEY1 and KEY0 to simulate atrial and ventricular events respectively from the heart to the pacemaker, and output AP or VP signals to the green LEDs. This will be useful when testing your design, as you will be able to provide any arbitrary timing inputs and test your SCCharts design is operating as you expect without much extra logic.

Some simple test cases that you may want to perform include pacing include the following:

- Using only KEY1 (atrium) should cause the pacemaker to pace the ventricles at the same rate.
- Stopping all key presses should cause the pacemaker to pace both the atria and ventricles.

Note that you may need to add in some extra logic to extend the amount of time the LEDs are illuminated in order to make them visible.

Mode 2

Here, the pacemaker will communicate with a virtual heart running on a PC over UART. This heart will "beat" as a normal heart would, with the ability to exhibit several disease states at the press of a button. For communication, bi-directional UART communication will be used where the characters "A" and "V" from the heart to the pacemaker mean "AS" and "VS" respectively. Similarly, the characters "A" and "V" from the pacemaker to the heart mean "AP" and "VP" respectively.

This mode will require you to use non-blocking UART reads, as you will not know when signals will be arriving ahead of time.

The program containing this virtual heart model is provided on Canvas.

Report

A report should be written which chronicles your design and implementation process through this assignment. This report should be structured as a "design report" and can be either single column or double column using 12-point font. As a guideline, we suggest that this report should be around 2-3 pages in length.

The report should allow the reader to understand the design decisions taken during this project, along with adequate implementation details to allow for the reconstruction of your assignment. Additionally, any hurdles or roadblocks that were encountered should be described, along with any perceived avenues for future work. Finally, we expect to see a breakdown of how long (hours) the different aspects of the project took to implement (each mode, planning, documentation, etc.).

Submission

The assignment will be demonstrated on **Wednesday 25th August** in **405-541** from **2pm – 4pm**. There will be support sessions announced on Canvas in the week(s) before where students can practice their assignment and ask any unresolved questions.

Final code is to be **submitted through Canvas** before **Wednesday 25th August at 11:59pm** as a **zip file** named by your group number (e.g. group_1.zip or group_2.zip) **containing the following**:

- Your SCCharts (.sct) file(s) used for your design
- The Nios II Project Folder for your implementation
- The BSP Project Files
- The .sof and .sopcinfo files
- Any documentation (e.g. README)
- Report PDF

Good Luck!

References

1. Jiang, Zhihao, Miroslav Pajic, Rajeev Alur, and Rahul Mangharam. “Closed-loop verification of medical devices with model abstraction and refinement.” *International Journal on Software Tools for Technology Transfer* 16, no. 2 (2014): 191-213.
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.
3. Allen, N., Andalam, S., Roop, P., Malik, A., Trew, M., & Patel, N. (2016, March). “Modular code generation for emulating the electrical conduction system of the human heart.” In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe* (pp. 648-653). EDA Consortium.
4. 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.