Introduction:

The purpose of this project was to create an implementation of a DDD Pacemaker using SCCharts and C. The result was to be programmed on to a NIOS II CPU to use as an emulator.

Mode 1:

For mode 1, 'KEY1' & 'KEY0' stimulates the atrial and ventricular events from the heart to the pacemaker and outputs the AP/VP signals to the green LEDs.

First, we used SCCharts to create the models for the Atrioventricular Interval (AVI), Ventricular Refractory Period (VRP), Post-Ventricular Atrial Refractory Period (PVARP), Atrial Escape Interval (AEI), Upper Rate Interval (URI) and – Lower Rate Interval (LRI).

Secondly, we implemented all the timers for each model in C. We also created the 'start' and 'stop' timers function to initiate and end the state of the timers.

Thirdly, we used polling for the buttons as the CPU continuously checks whether the device needs attention resulting in faster reactiveness compared to interrupts. We also didn't want the signal to change its state mid-way through a tick in the SCCharts.

Finally, we created an AVLEDs function to display the LEDs to showcase the pacing signals of the heart.

Mode 2:

For mode 2, it is an extension to mode 1 where we added the read_uart function. It utilises the non-blocking uart so our code can run continuously without any code blocking.

We initially had a problem with using the heart emulator as access was denied to the COM port. We had no idea why until we realised only one UART communication tool could be used at one time and closed pUTTy.

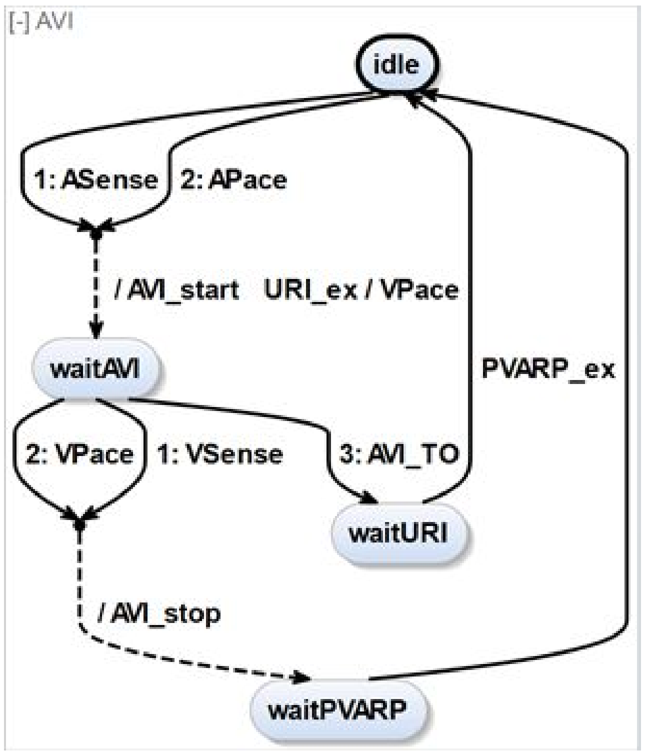
Another problem we had was implementing the non-blocking UART reception of signals. We originally read one char at a time which sets the relevant signal high and starts a timer to set the signal low again after 1ms. However we did not realise that switching to mode 2 mid-way through running results in all the previously received UART signals all being sent in. This was starting the timer multiple times and also sending in a high signal for multiple ticks in the SCCharts. To fix this we added a flag to only send the latest received signal in.

Design Approach:

We chose to design the main functionality of the DDD mode pacemaker in SCCharts with only the timers and IO devices handled in C. In this way, only the only inputs to the SCCharts are ventricular/atrial signals and timer timeouts, while the SCCharts output the timer starting signals and pacing signals.

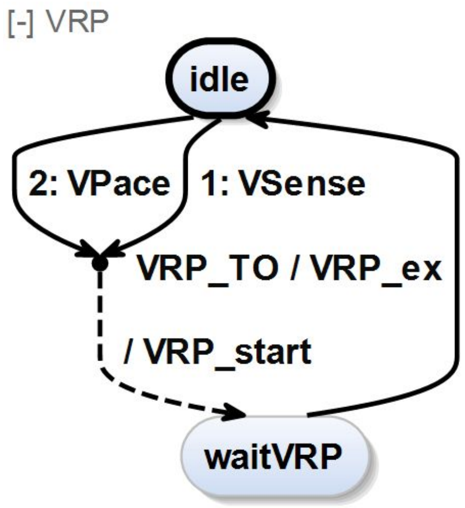
We began designing the SCCharts by identifying all of the signal's roles and how they interact with each other, for example, which timers relate to the ventricular (V) only, atrial (A) only or both. By doing this, we could start creating the skeleton code for the SCCharts with the basic functionality. E.g. (V) events start the VRP timer, and a VRP timeout signal sends a VRP expired signal output.

The next step included taking into account the timers, which affect the execution of other timers. This is because some timers have a role of ignoring the (V) or (A) inputs for its duration. We did this by studying the timing diagram and implementing our interpretation of how the timers should run.



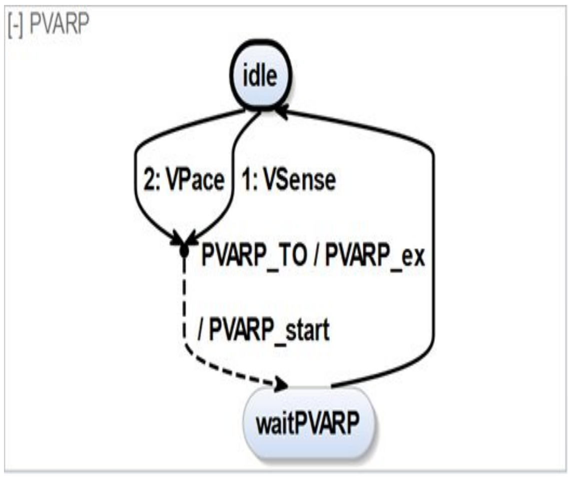
AVI: The maximum time between an atrial event (A) and its subsequent ventricular event (V).

1. [a] events start the AVI timer.
2. [v] events stop the AVI timer.
3. Wait for PVARP to expire before returning to the initial state to ignore [a] signals in this period.
4. If no [v] even is detected and the timer runs out:
5. Wait for URI to expire before pacing the ventricular as URI is the fastest rate a [v] pace can be generated.



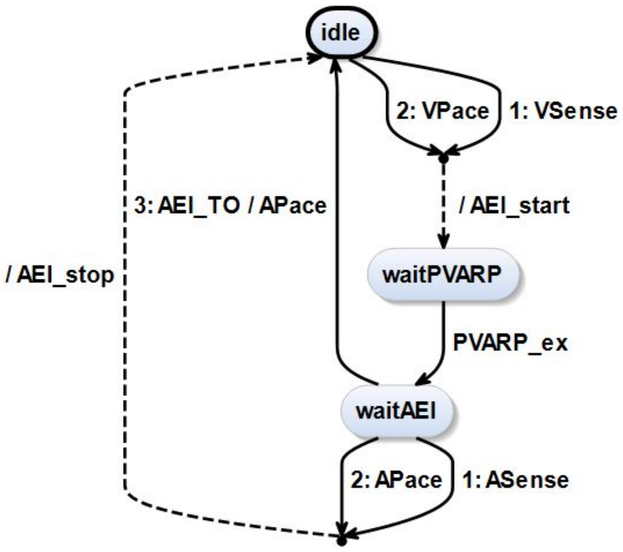
VRP: The time after a ventricular event where any other ventricular events are ignored as Ventricular Refractory (VR) signals.

1. [v] events start the VRP timer.
2. The timeout signal outputs a VRP expired signal to the rest of the SCCharts.



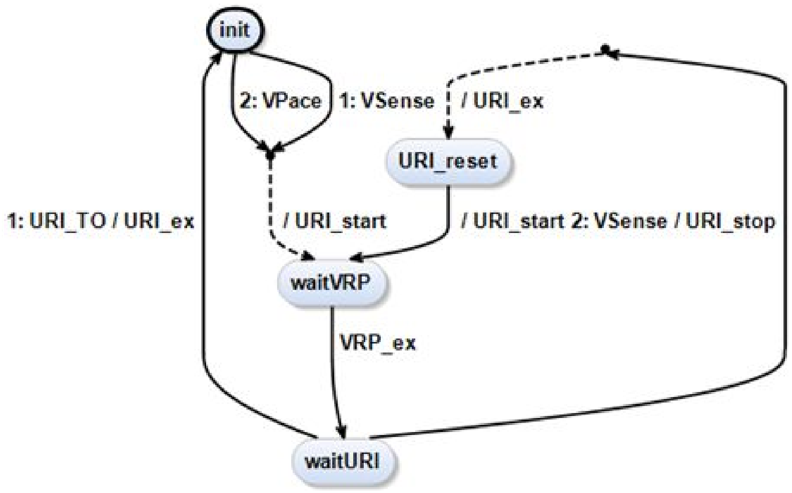
PVARP: The time after a ventricular event where any atrial events are ignored.

1. [v] events start the PVARP timer.
2. The timeout signal outputs a PVARP expired signal to the rest of the SCCharts.



AEI: The maximum time between a ventricular event and its subsequent atrial event.

1. [v] events start the AEI timer.
2. Wait for PVARP to timeout to ignore [a] signals in this period.
3. [a] events stop the AEI timer.
4. If no [a] even is detected and the timer runs out: pace the atrial.
5.

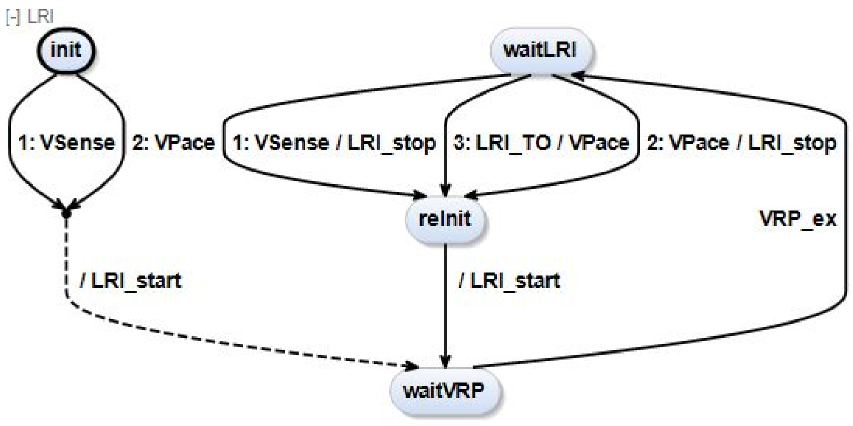


URI: The fastest rate at which the pacemaker will ever pace the heart at. This is measured as the time between ventricular events

1. [v] events start the URI timer.
2. Waits until VRP expires to ignore any [v] events.
3. When a [v sense] occurs after the VRP, the URI is reset.
4. Else waits until the URI timer finishes.
5. The timeout signal outputs a URI expired signal to the rest of the SCCharts.

LRI: The slowest rate at which the heart is allowed to operate. This is measured as the time between ventricular events.

1. [v] events start the LRI timer.
2. Waits for VRP to timeout to ignore any [v] events in this period.
3. If a [v] event is detected stop the LRI timer.
4. Else when the LRI timer runs out, pace the ventricular.
5. Goes to reinitialise state and starts the LRI timer again.



Future Work: In the future, we would like to make our project even more readable and concise. For example, all of the timers have the same start-stop functionality in the C code. There may be a way to initialise and set the functionality through a defined way where only the name of the timer changes. This would get rid of all the repeated code of the timers. For the SCCharts our LRI block could be reworked to merge the separated init part. Our project could also utilise more features of the board; making use of all of the LEDs to show the state of the timers.

Task	**Time**	**Reasoning**
Planning	1 hr	Planning to implement the SCCharts & C code.
SCCharts	4 hr	Making the models of the DDD pacemaker.
Mode 1	4 hr	Making the timers in C code to work with the SCCharts correctly.
Mode 2	4 hr	Non blocking UART debugging and signal propagating Issues.
Testing	3 hr	Testing the mode switch and making sure the operation runs smoothly.
Report	3 hr	Reflection & Editing