# COMPUTER SYSTEMS 303 PACEMAKER ASSIGNMENT

*Hao Lin, Callum McDowell*

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand
hlin784@aucklanduni.ac.nz
cmcd407@aucklanduni.ac.nz

## Abstract

In this project we design and perform simple validation on a DDD mode pacemaker model designed for use on embedded cyber-physical reactive systems. The design is implemented in a sequentially constructed approach to concurrency, using the KIELER SCCharts framework. The code is adapted to interface with Altera Nios II hardware for emulation. This report details the design decisions taken, significant challenges faced, avenues for future work, and a general overview of our project.

## 1. Introduction

A DDD mode pacemaker can sense and independently pace both the atrium and ventricle, with both triggered (pacing after an event is not detected within a period) and inhibited (pacing unless an event is detected) conditions.

Table 1 contains a simple introduction to the relevant timing parameters, and Table 2 the events.

*Table 1:* Timing parameters.

| Period | Definition |
|---|---|
| PVARP | The period after a ventricular event where other atrial events are ignored as refractory (AR). |
| VRP | The period after a ventricular event where other ventricular events are ignored as refractory (VR). |
| AEI | The maximum time between a ventricular event and its subsequent atrial event. |
| AVI | The maximum time between an atrial event and its subsequent ventricular event. |
| LRI | The slowest rate at which the heart may operate. The maximum time between ventricular events. |
| URI | The fastest rate at which the heart may be paced. The minimum time between a ventricular event and VP. |

*Table 2:* Event types.

| Event | Definition |
|---|---|
| AS | A natural atrial event (generated by the heart). |
| VS | A natural ventricular event (generated by the heart). |
| AP | An atrial pace (generated by the pacemaker). |
| VP | A ventricular pace (generated by the pacemaker). |
| AR | A refractory atrial event (occurred within PVARP). |
| VR | A refractory ventricular event (occurred within VRP). |

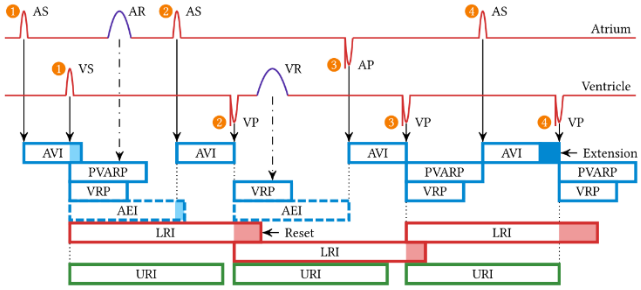


*Figure 1:* Timing diagram of a DDD mode pacemaker. Reproduced from [1].

Fig. 1 indicates how these parameters may control the pacemaker, given certain values. Note that the schedule of events, and indeed the behaviour of the system, will differ (at times significantly) from the pictured example cases.

## 2. Pacemaker Logic Design

### 2.1. Draft 1

*15th August 2021 – 1 Hour*

We initially approached our design from the perspective of a more traditional sequential FSM (finite state machine), where states are extended with 'timer' sub-states, which would allow for concurrent behaviour within each state. However, this thinking proved to be too limiting as the parallel, loosely linked nature of ventricular and atrial signals and paces resulted in an explosion of states for edge-cases.

Fig. 2 demonstrates how the 'Schedule pace VS' and 'Schedule pace AS' states contain timing sub-states. Notably, this created confusing interactions between the other states in the parent FSM and the timer's sub-states. Their scope was unclear, and it was hard to interpret which regions should write to and/or read them.

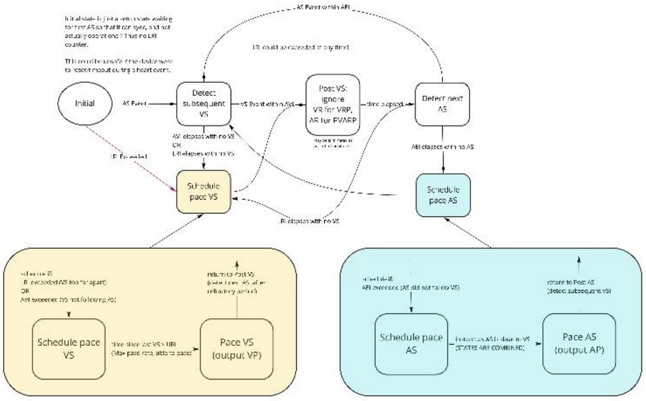

AS Event within AEI
Initial state is just a return state waiting for first AS so that it can sync, and not actually operational? Thus no LRI counter.
This could be unsafe if the device were to reset/timeout during a heart event.
LRI could be exceeded at any time!
Initial
AS Event
Detect subsequent VS
VS Event within AVI
Post VS: ignore VR for VRP, AR for PVARP
time elapsed
Detect next AS
AVI elapses with no VS OR LRI elapses with no VS
LRI Exceeded
AEI elapses with no AS
Schedule pace VS
Schedule pace AS
LRI elapses with no VS
schedule if: LRI exceeded (VS too far apart) OR AVI exceeded (VS not following AS)
return to Post VS (detect next AS after refractory period)
Schedule pace VS
time since last VS > URI (Max pace rate, able to pace)
Pace VS (output VP)
schedule if: AEI exceeded (AS did not follow VS)
return to Post AS (detect subsequent VS)
Schedule pace AS
Instant as AS is slave to VS (STATES ARE COMBINED)
Pace AS (output AP)

*Figure 2:* Draft 1, a sequential approach. Appendix 1.

### 2.2. Draft 2

*15th August 2021 – 1 Hour*

Our second approach attempted to solve these parallelism issues by running two FSM concurrently, one dedicated to ventricular signals (VS and VP) and the other to atrial signals (AS and AP). This is demonstrated in Fig. 3.

To avoid the timer complexity from our first draft we also attempted to restrict timers to the domain of only one FSM. AVI, URI, LRI, and VRP were dedicated to the ventricular FSM while AEI was dedicated to the atrial FSM. This became problematic when PVARP needed to be read in the ventricular FSM to ensure refractory atrial signals (AR) were not incorrectly triggering state transitions.

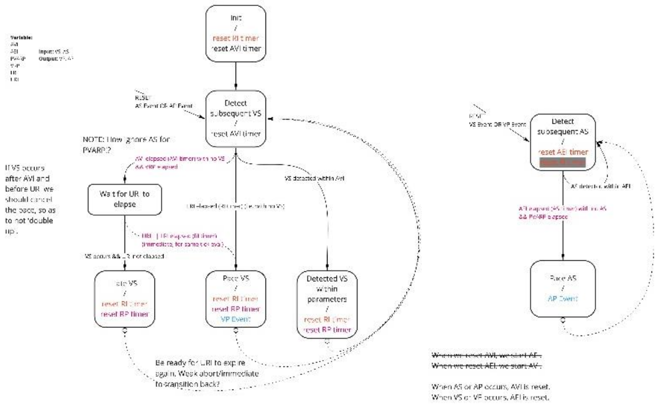

Variable:
AVI
AEI
PVARP
VRP
URI
LRI
Input: VS, AS
Output: VP, AP
Init
/
reset RI timer
reset AVI timer
RESET
AS Event OR AP Event
Detect
subsequent VS
/
reset AVI timer
NOTE: How ignore AS for PVARP?
AVI elapsed (AVI timer) with no VS && VRP elapsed
VS detected within AVI
If VS occurs after AVI and before URI we should cancel the pace, so as to not 'double up'.
Wait for URI to elapse
URI elapsed (RI timer) (i.e. with no VS)
URI || LRI elapsed (RI timer) (immediate, for same tick eval)
VS occurs && URI not elapsed
Late VS
/
reset RI timer
reset RP timer
Pace VS
/
reset RI timer
reset RP timer
VP Event
Detected VS within parameters
/
reset RI timer
reset RP timer
Be ready for URI to expire again. Weak abort/immediate to transition back?
RESET
VS Event OR VP Event
Detect
subsequent AS
/
reset AEI timer
AS detected within AEI
AEI elapsed (AS timer) with no AS && PVARP elapsed
Pace AS
/
AP Event
~~When we reset AVI, we start AEI.~~
~~When we reset AEI, we start AVI.~~
When AS or AP occurs, AVI is reset.
When VS or VP occurs, AEI is reset.

*Figure 3:* Draft 2, a concurrent approach. Appendix 2.

A new design with better described timer interactions was required.

## 2.3. Draft 3

*15th August 2021 – 2 Hours*

The third draft continued the changes made in the second, with a focus on decomposing each element to its simplest parts and running it in parallel with the other elements. Timers were simplified and made more independent to reflect how in function they controlled the transitions of the ventricular and atrial FSM.

Functionality was greatly simplified but timer scope and scheduling had still not been addressed. Timers were used between both FSM, requiring them to have a scope greater than either of the FSM. However, a few timers were also read and reset within both parallel FSMs, resulting in potential race conditions and issues ensuring that all write operations could be done before read operations.

Note that in the changes from Draft 2 to Draft 3 the AVI and AEI timers have been implemented incorrectly! For Draft 2, the AEI FSM would enter a 'serviced' state ('AP||AS Event') if an AS event was detected while waiting for AEI to elapse. In Draft 3 the AEI timer does not reset at all, meaning AEI will always elapse unless reset (ignoring AS events). Fig. 4 elaborates on the design.

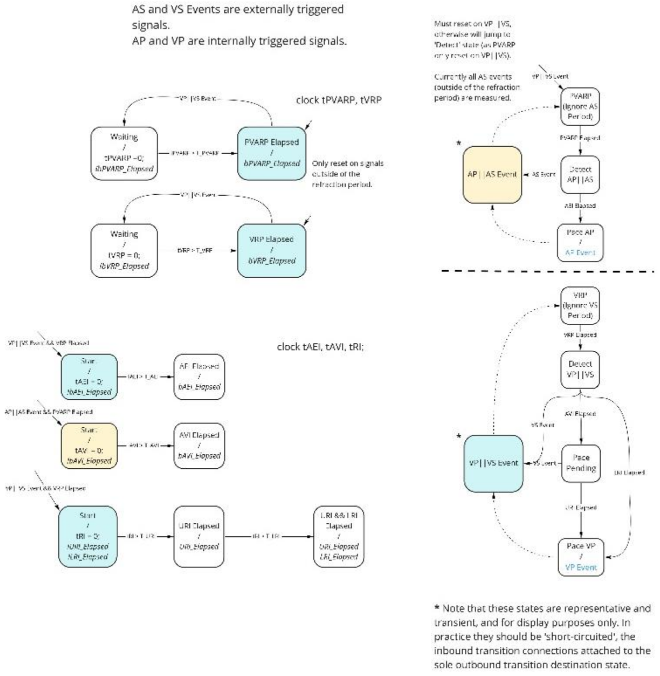


*Figure 4:* Draft 3, a decomposed approach. Appendix 3.

## 2.4. Draft 3 in SCCharts (SCCharts Rev2)

*16th-21st August 2021 – 3 Hours*

Our timer scheduling concurrency issues were immediately brought to light when implementing the FSM in SCCharts (Fig. 5). The logic had been designed from a synchronous hardware perspective more suited to implementation in VHDL on an FPGA than sequentially constructed code on a microcontroller. Additional issues arose from a beginner understanding of concurrent formalisms, and naive use of strong abort transitions.

The tick delay introduced by our use of pre() to attempt to get around scheduling issues created unmanaged impractical and potentially unacceptable delays, wherein the VRP_Elapsed and PVARP_Elapsed values viewed by the model would always be one tick behind their actual values.

This would cause refractory signals VR and AR to be accepted as VS or AS signals in the case that they occurred a tick after the actual VS or AS signals. At this point a VR or AR signal being registered as VS or AS currently does not have implications beyond re-triggering strong abort transitions (and introducing a further one tick inaccuracy), but could become a major issue if related logic or timing were to change in the future.

While in practice a maximum delay of two ticks between the ideal signal output and the model signal output would not be a critical issue given an appropriately small tick period, this would be a cause for concern for future revisions involving different tick periods or dynamic tick logic – in addition to the error being entirely avoidable.

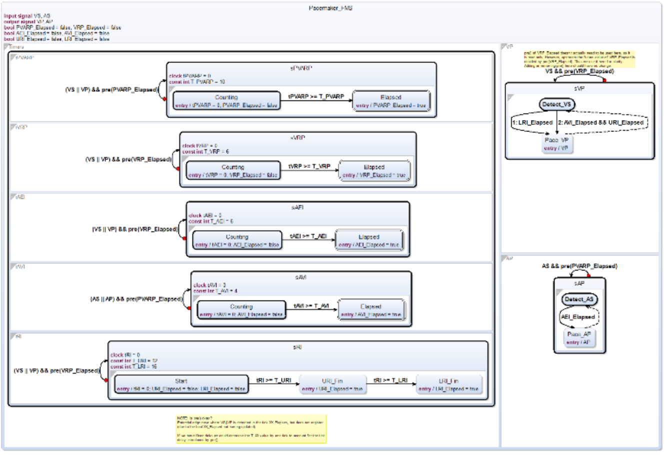

Pacemaker_FMS
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
Timers
tPVARP
sPVARP
clock tPVARP = 0
const int T_PVARP = 10
(VS || VP) && pre(PVARP_Elapsed)
Counting
entry / tPVARP = 0, PVARP_Elapsed = false
tPVARP >= T_PVARP
Elapsed
entry / PVARP_Elapsed = true
tVRP
sVRP
clock tVRP = 0
const int T_VRP = 6
(VS || VP) && pre(VRP_Elapsed)
Counting
entry / tVRP = 0, VRP_Elapsed = false
tVRP >= T_VRP
Elapsed
entry / VRP_Elapsed = true
tAEI
sAEI
clock tAEI = 0
const int T_AEI = 6
(VS || VP) && pre(VRP_Elapsed)
Counting
entry / tAEI = 0, AEI_Elapsed = false
tAEI >= T_AEI
Elapsed
entry / AEI_Elapsed = true
tAVI
sAVI
clock tAVI = 0
const int T_AVI = 4
(AS || AP) && pre(PVARP_Elapsed)
Counting
entry / tAVI = 0, AVI_Elapsed = false
tAVI >= T_AVI
Elapsed
entry / AVI_Elapsed = true
tRI
sRI
clock tRI = 0
const int T_URI = 12
const int T_LRI = 16
(VS || VP) && pre(VRP_Elapsed)
Start
entry / tRI = 0, URI_Elapsed = false, LRI_Elapsed = false
tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP
VS && pre(VRP_Elapsed)
sVP
Detect_VS
1: LRI_Elapsed
2: AVI_Elapsed && URI_Elapsed
Pace_VP
entry / VP
AP
AS && pre(PVARP_Elapsed)
sAP
Detect_AS
AEI_Elapsed
Pace_AP
entry / AP

*Figure 5:* Draft 3 in SCCharts (Rev2). Appendix 4.

## 2.5. Analysis of the timing challenges

We found the core of the timer scheduling issues to be due to the incompatible nature of the ventricular and atrial logic for controlling AP and VP. They are both consumers of (i.e., read) the timer values, but also emit signals (VP and AP) that act as resets for the timers. Thus, the VP and AP logic has both a consumer and producer relationship with the timer logic.

This creates a scheduling dependency issue in which VP and AP are assigned an output value depending on the value of other signals– while those other signals are also a function of VP and AP.

This situation is not possible in a sequentially constructed system. We must resolve the scheduling issue by introducing a timing delay (in our case, with the use of pre()), while also being sure to avoid introducing unacceptable side-effects or violating timing requirements.

From Revision 3 onwards we chose to use pre() values for all internally generated signals (PVARP_Elapsed, VRP_Elapsed, AP, etc.) in read operations taking place outside of the scope of the timer FSM (the timer FSM being responsible for writing to/resetting its respective timer clock variable). Externally generated signals inherently were consumer-exclusive, so their most recent value could be used each tick (no need for pre()).

Most significantly, this meant that VP and AP (paced) signals interacted with the system a tick after they were generated and emitted, while VS and AS (natural) signals could interact with the system in the same tick. This one-tick discrepancy would need to be treated carefully.

### 2.6. Draft 3 in SCCharts (SCCharts Rev3-3.5)

*24th August 2021 – 6 Hours*

Revision 3.5 of our SCCharts design was initially a testing branch from our master (Revision 3) SCCharts design. The chart was compiled and used in a testing configuration written in C++. Issues were discovered in the SCCharts design during development of the testing configuration and were resolved within this version. Design decisions made during this stage were iterative and reactive, and carried through to Revision 4.

Revision 3 used immediate transitions to mitigate the one-tick delay discrepancy between processing of paced (P) and natural (S) signals, but this led to errors we could not explain at the time. We incorrectly assigned the blame to odd interactions with immediate transitions when the actual cause was using current PVARP_Elapsed and VRP_Elapsed values, rather than pre() values. The pre(VP) signal was resetting the PVARP_Elapsed and VRP_Elapsed values early in the tick, so subsequent logic incorrectly treated P (paced) signals as refractory signals. This issue cropped up later in Revision 4 and was resolved properly then.

Revision 3.5 (Fig. 6) lacks proper compensation for the tick discrepancy between paced (P) and natural (S) signals for this reason. This was also resolved in Revision 4.

AEI also remained in the 'Elapsed' state once reached, so AEI_Elapsed was persistently high (leading to continuous output of AP). Revision 4 addressed this by entering a 'Finished' state a tick after the 'Elapsed' state, so that AEI_Elapsed could only be logic high for one tick.

Errors in the AEI and AVI logic inherited from Draft 3 were rediscovered during the testing but addressed in Revision 4.

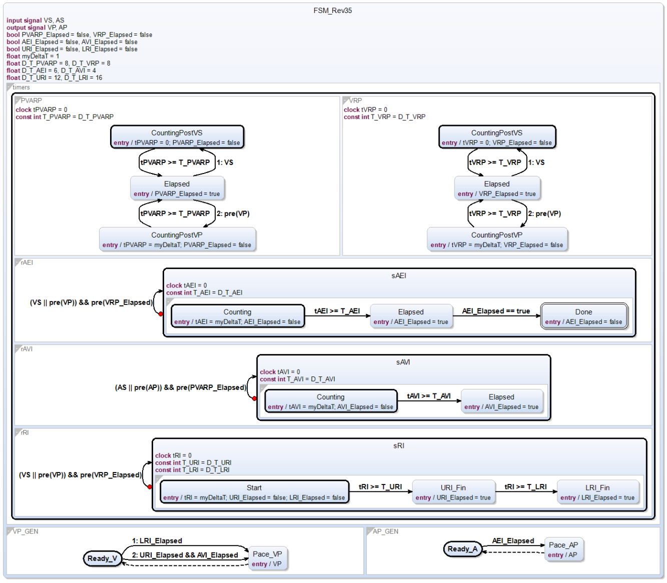

FSM_Rev35
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
float myDeltaT = 1
float D_T_PVARP = 8, D_T_VRP = 8
float D_T_AEI = 6, D_T_AVI = 4
float D_T_URI = 12, D_T_LRI = 16
timers
PVARP
clock tPVARP = 0
const int T_PVARP = D_T_PVARP
CountingPostVS
entry / tPVARP = 0; PVARP_Elapsed = false
tPVARP >= T_PVARP
1: VS
Elapsed
entry / PVARP_Elapsed = true
tPVARP >= T_PVARP
2: pre(VP)
CountingPostVP
entry / tPVARP = myDeltaT; PVARP_Elapsed = false
VRP
clock tVRP = 0
const int T_VRP = D_T_VRP
CountingPostVS
entry / tVRP = 0; VRP_Elapsed = false
tVRP >= T_VRP
1: VS
Elapsed
entry / VRP_Elapsed = true
tVRP >= T_VRP
2: pre(VP)
CountingPostVP
entry / tVRP = myDeltaT; VRP_Elapsed = false
rAEI
sAEI
(VS || pre(VP)) && pre(VRP_Elapsed)
clock tAEI = 0
const int T_AEI = D_T_AEI
Counting
entry / tAEI = myDeltaT; AEI_Elapsed = false
tAEI >= T_AEI
Elapsed
entry / AEI_Elapsed = true
AEI_Elapsed == true
Done
entry / AEI_Elapsed = false
rAVI
sAVI
(AS || pre(AP)) && pre(PVARP_Elapsed)
clock tAVI = 0
const int T_AVI = D_T_AVI
Counting
entry / tAVI = myDeltaT; AVI_Elapsed = false
tAVI >= T_AVI
Elapsed
entry / AVI_Elapsed = true
rRI
sRI
(VS || pre(VP)) && pre(VRP_Elapsed)
clock tRI = 0
const int T_URI = D_T_URI
const int T_LRI = D_T_LRI
Start
entry / tRI = myDeltaT; URI_Elapsed = false; LRI_Elapsed = false
tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP_GEN
Ready_V
1: LRI_Elapsed
2: URI_Elapsed && AVI_Elapsed
Pace_VP
entry / VP
AP_GEN
Ready_A
AEI_Elapsed
Pace_AP
entry / AP

*Figure 6:* Draft 3 in SCCharts (Rev 3.5). Appendix 5.

### 2.7. Draft 3 in SCCharts (SCCharts Rev4)

*1st-5th September 2021 – 12 hours*

Revision 4 (Fig. 7) of our SCChart design included all the iterative changes made in Revision 3.5, in addition to further tweaks and adjustments to timing. The design decisions were informed by the observed behaviours of the FSM when analysed by our simulation testing software, and test cases derived by formalisation and expansion of the timing conditions and parameters. These expansions are elaborated on in Section 3.1.

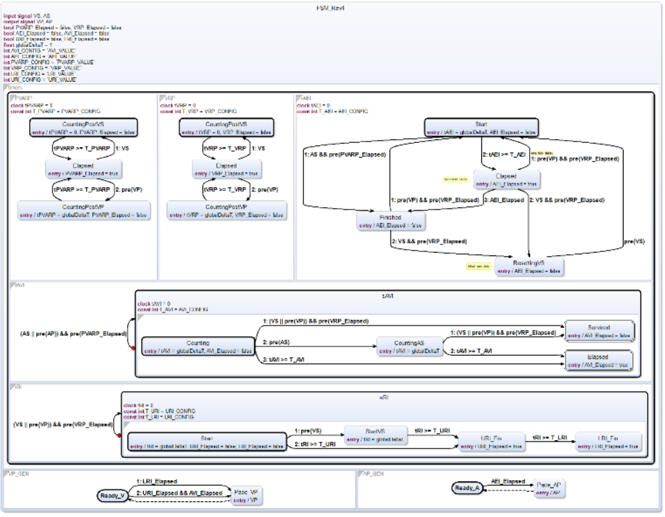

FSM_Rev4
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
float globalDeltaT = 1
int AVI_CONFIG = "AVI_VALUE"
int AEI_CONFIG = "AEI_VALUE"
int PVARP_CONFIG = "PVARP_VALUE"
int VRP_CONFIG = "VRP_VALUE"
int LRI_CONFIG = "LRI_VALUE"
int URI_CONFIG = "URI_VALUE"
timers
PVARP
clock tPVARP = 0
const int T_PVARP = PVARP_CONFIG
CountingPostVS
entry / tPVARP = 0; PVARP_Elapsed = false
tPVARP >= T_PVARP
1: VS
Elapsed
entry / PVARP_Elapsed = true
tPVARP >= T_PVARP
2: pre(VP)
CountingPostVP
entry / tPVARP = globalDeltaT; PVARP_Elapsed = false
VRP
clock tVRP = 0
const int T_VRP = VRP_CONFIG
CountingPostVS
entry / tVRP = 0; VRP_Elapsed = false
tVRP >= T_VRP
1: VS
Elapsed
entry / VRP_Elapsed = true
tVRP >= T_VRP
2: pre(VP)
CountingPostVP
entry / tVRP = globalDeltaT; VRP_Elapsed = false
rAEI
clock tAEI = 0
const int T_AEI = AEI_CONFIG
Start
entry / tAEI = globalDeltaT; AEI_Elapsed = false
1: AS && pre(PVARP_Elapsed)
2: tAEI >= T_AEI
1: pre(VP) && pre(VRP_Elapsed)
Elapsed
entry / AEI_Elapsed = true
1: pre(VP) && pre(VRP_Elapsed)
3: AEI_Elapsed
2: VS && pre(VRP_Elapsed)
Finished
entry / AEI_Elapsed = false
2: VS && pre(VRP_Elapsed)
pre(VS)
ResettingVS
entry / AEI_Elapsed = false
rAVI
sAVI
(AS || pre(AP)) && pre(PVARP_Elapsed)
clock tAVI = 0
const int T_AVI = AVI_CONFIG
1: (VS || pre(VP)) && pre(VRP_Elapsed)
Counting
entry / tAVI = globalDeltaT; AVI_Elapsed = false
2: pre(AS)
CountingAS
entry / tAVI = globalDeltaT
1: (VS || pre(VP)) && pre(VRP_Elapsed)
2: tAVI >= T_AVI
3: tAVI >= T_AVI
Survival
entry / AVI_Elapsed = false
Elapsed
entry / AVI_Elapsed = true
rRI
sRI
(VS || pre(VP)) && pre(VRP_Elapsed)
clock tRI = 0
const int T_URI = URI_CONFIG
const int T_LRI = LRI_CONFIG
Start
entry / tRI = globalDeltaT; URI_Elapsed = false; LRI_Elapsed = false
1: pre(VS)
StartVS
entry / tRI = globalDeltaT
tRI >= T_URI
2: tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP_GEN
Ready_V
1: LRI_Elapsed
2: URI_Elapsed && AVI_Elapsed
Pace_VP
entry / VP
AP_GEN
Ready_A
AEI_Elapsed
Pace_AP
entry / AP

*Figure 7:* Draft 3 in SCCharts (Rev 4). Appendix 6.

One of the major changes in this revision was the reworking of AEI logic, following the flaws in Draft 3. Key issues included:

1. The tick discrepancy between paced (P) and natural (S) signals for the use of resetting the timers.
2. AEI failing to respond to atrial signals, despite its purpose being to measure the maximum time between a ventricular event and its subsequent atrial event.
3. Situations in which the AEI_Elapsed value could emit logic high for more than one tick in a row.

Issue 1 was addressed by providing two reset states. Naturally triggered resets would occur within the same tick of the signal being measured, so the timers could be reset to zero. Resets triggered by paces would be processed the tick after they occurred due to the use of pre(), so the timers should be set to DeltaT to account for this tick delay. If the minimum timer period is at least two ticks in length this solution should present no errors. As DeltaT is a protected default SCCharts value we used a custom placeholder, MyDeltaT, which needs to be edited and assigned to the generated DeltaT value at the time of emulation.

In the case of AEI and AVI it was simpler to effectively reset the timer twice on naturally triggered signals, providing the same functionality as above. The minimum timer period would have to be three ticks to avoid errors.

Issue 2 was solved by (re)adding a transition to the 'Finished' state, the transition taken if AS is detected while the AEI timer has not yet elapsed. This ensured AEI_Elapsed could not trigger if AS occurred within AEI. We do not need to extend this for AP events as the AEI timer is the sole condition for generating AP events.

Issue 3 was resolved by leaving the 'Elapsed' state a tick after entering it, transitioning to a 'Finished' state that reset the AEI_Elapsed value to logic low.

However, in doing so we realised that we had created a new timing issue: if a resetting VS or pre(VP) condition occurred the tick the system was in the 'Elapsed' state, just before transitioning to the 'Finished' state (where all the reset functionality was currently implemented), the reset could potentially be missed. A strong abort was not suitable due to the ventricular signals not always being relevant. Thus, the simplest method was to duplicate the reset transitions in the 'Finished' state to the 'Elapsed' state and accept the increased complexity.

## 3. Testing configuration

### 3.1. Timing Validation

Absolute upper and lower bounds for the timing of VP and AP generation could be derived from the timing parameters. Table 3 and 4 show the lower bounds (conditions for which pacing is prohibited) and upper bounds (conditions for which a pace is required) of the atrial and ventricular events. Note how AP has no lower bounds, as in inhibited mode it is triggered solely by ventricular signals (and VP does have a lower bound: URI).

Table 5 and 6 expand these conditions to all viable substitutions, the 'Period Measured' column showing the sequence of events the condition applies to. For example, 'V -> A' is the period between a ventricular signal and its subsequent atrial signal. The '+' operator is the sum of the periods, assuming no overlap (i.e., the other period only begins counting once the prior has elapsed). As a result, Table 5 and 6 show the absolute timing conditions that should not be violated, but do not explore subtleties within these conditions.

These conditions were implemented in a MATLAB script to analyse logged data and ensure our design did not violate any of the timing bounds.

*Table 3:* Timing conditions after a ventricular event.

| Next Event | Upper Bounds | Lower Bound |
|---|---|---|
| Atrial | AEI | PVARP |
| Ventricle | LRI | VRP \|\| URI |

*Table 4:* Timing conditions after an atrial event.

| Next Event | Upper Bounds | Lower Bound |
|---|---|---|
| Ventricle | AVI | 0 |
| Atrial | (AVI + AEI) | 0 |

*Table 5:* Expanded conditions for VP.

| Condition | Period Measured |
|---|---|
| There should be NO VP if any of the following are true: | |
| tV ≤ URI | V -> V |
| | |
| There should be VP if any of the following are true: | |
| tV > (AEI + AVI) | V -> V |
| tV > LRI | V -> V |
| tV > (tA + AVI + AEI + AVI) | A -> V |

*Table 6:* Expanded conditions for AP.

| Condition | Period Measured |
|---|---|
| There should be NO AP if any of the following are true: | |
| tA < 0 | A -> A |
| | |
| There should be AP if any of the following are true: | |
| tA > (AVI + AEI) | A -> A |
| tV > (tV + LRI + AEI) | V -> A |

### 3.2. Software simulation in C++

KIELER SCCharts' built-in simulator environment was not suitable for visualizing pacemaker functionality. A C++ testing program was developed to run the compiled design over a period (with simulated inputs), and log information on the timing counter values, states, and inputs and outputs. The logged information was analysed in Excel as a table and series of graphs, as seen in Fig. 8. This testing program data was later supplemented with Nathan Allen's provided simulator.

### 3.3. Edge Cases

In our testing we discovered that for certain timings of AS and AVI, in a situation where VS is arrested (but not AS) and the pacemaker is operating in inhibited mode the naturally generated AS signal may fall within PVARP of VP events. As a result, the AS signal will be incorrectly interpreted as a refractory AR signal. The ramifications of this can, for example, lead AVI to not reset on some AS signals. This is pictured in Fig. 8.

We argue that due to the nature of PVARP this operation is technically correct, if undesirable. Future work could be done to create edge case handling for situations where VP triggers a reset of PVARP and VRP with timing such that it obfuscates VS and AS signals.

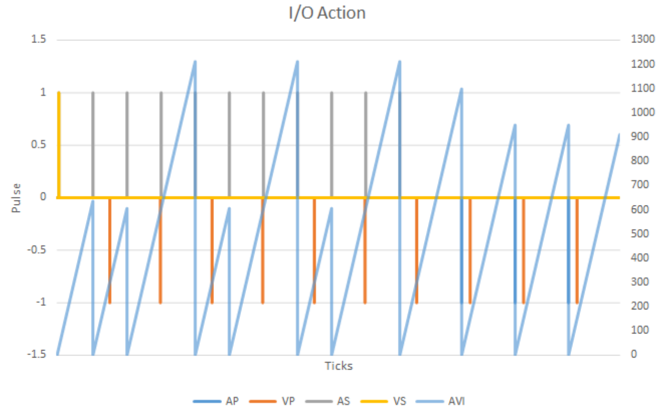


*Figure 8:* Visualisation of AVI failing to reset on AS.

## 4. Nios II Software Development

*Work distributed over the course of project – 20 Hours*

The compiled pacemaker FSM was emulated on an Altera DE2-115 Development and Education FPGA Board.

Software development on the Altera DE2-115 primarily interacted with the HAL interface, but some legacy functions such as 'priv/alt_legacy_irq.h' library were also used for interrupts. The greatest issues in the development originated from deprecated documentation and occupied the majority of the time spent developing the software.

Each peripheral was encapsulated in its own modular file; buttons, LEDs, switches, timers, and the UART, in addition to the FSM and main().

### 4.1. Mode 1

In Mode 1 the pacemaker expects manual inputs of AS and VS via the push keys and will display the I/O signals via LED displays. Each AS, VS, AP, VP signal is represented by one LED each.

The pulse logic of LEDs is separated from the key interactions. A dedicated handler for the FSM processes the input information from the board and the output information from the FSM. As a result, the code is portable and can be used for other modes.

### 4.2. Mode 2

In Mode 2 the pacemaker is expected to communicate with a PC application via UART. A provided virtual heart simulator transmits to the pacemaker the signals VS and AS, and measures AP and VP emitted by the pacemaker. Due to the unpredictable nature of incoming sense signals the UART is configured to operate in non-blocking mode.

### 4.3. Challenges

Handling the I/O information of the physical system and the FSM system was found to be challenging. While this project involves developing an FSM system that runs concurrently, the physical implementation on the development board is sequential and on a single process. There were three timing environments we needed to consider: the real world, the program, and the FSM in the program. The FSM is expected to operate at the same time as the real world, which is achieved by syncing the FSM's tick() function to a timer. Meanwhile, the program ran on the system's clock and relied on handlers to regularly service changes in the I/O produced by the FSM and the board peripherals.

The software development was suspended at its final stages as the software is tested for bugs. Yet to be addressed issues include the accumulation of meaningless information in the RX register the board receives from the virtual heart simulator in mode 1, where the information is neither dealt with nor consumed.

## 5. Future Work

The edge cases in Section 3.3 are yet to be addressed in a satisfactory manner, and we have not tested our SCCharts code to a standard suitable for safety critical software. Future work could be done to improve our current design, especially regarding AEI and AVI timer logic. Automated testing and test cases extended for edge-case situations are also another area for improvement.

Use of dynamic ticks would require extensive testing and potential adjustments due to our use of DeltaT, but could offer improved power usage optimisation, something especially important for implanted pacemakers.

Our Nios II implementation could also be revisited, further optimised and the portability improved for use on other devices (such as the target pacemaker embedded hardware).

## 6. Conclusions

We have used a sequentially constructed approach to concurrency to successfully design the logic of a DDD mode pacemaker, using the KIELER SCCharts framework for modelling and compiling our finite state machine.

Our design process included deliberate planning of control logic architecture, and iterative adjustments for edge-cases and unanticipated issues. Three separate control logic diagrams were created, and over four different SCCharts finite state machines.

Validation and testing were performed with custom C++ simulation software, post-simulation analysis MATLAB scripts, and the provided online simulation program. Test cases were derived from timing conditions and provided examples.

The final design was compiled to C for embedded target hardware and implemented in Altera Nios II on the DE2-115 FPGA board.

As outlined in Section 5, Future Work, while results have been promising a lot of progress remains to be done to ensure that our design (and the SCCharts compilation of it) meets critical timing parameters. This is of special concern given the safety-critical nature of a pacemaker.

### 6.1. Overall estimated timing breakdown

| **Project Aspects** | **Time** |
|---|---|
| FSM design | 5 hours |
| SCCharts design | 6 hours |
| Validation (creating tests software and test cases) | 12 hours |
| SCCharts validation and fixes | 16 hours |
| Nios II implementation | 20 hours |
| Documentation and report | 16 hours |
| **Total (shared and individual time)** | **75 hours** |

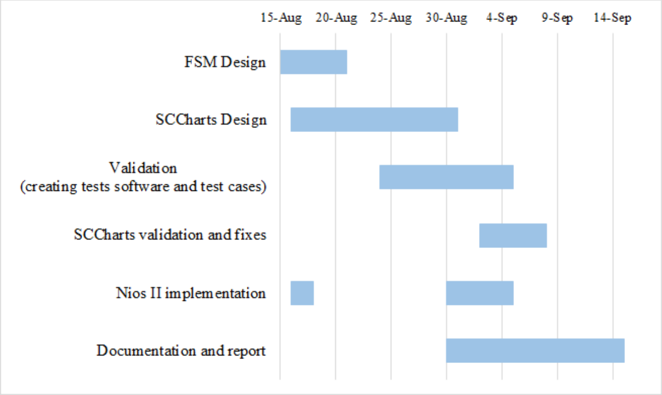


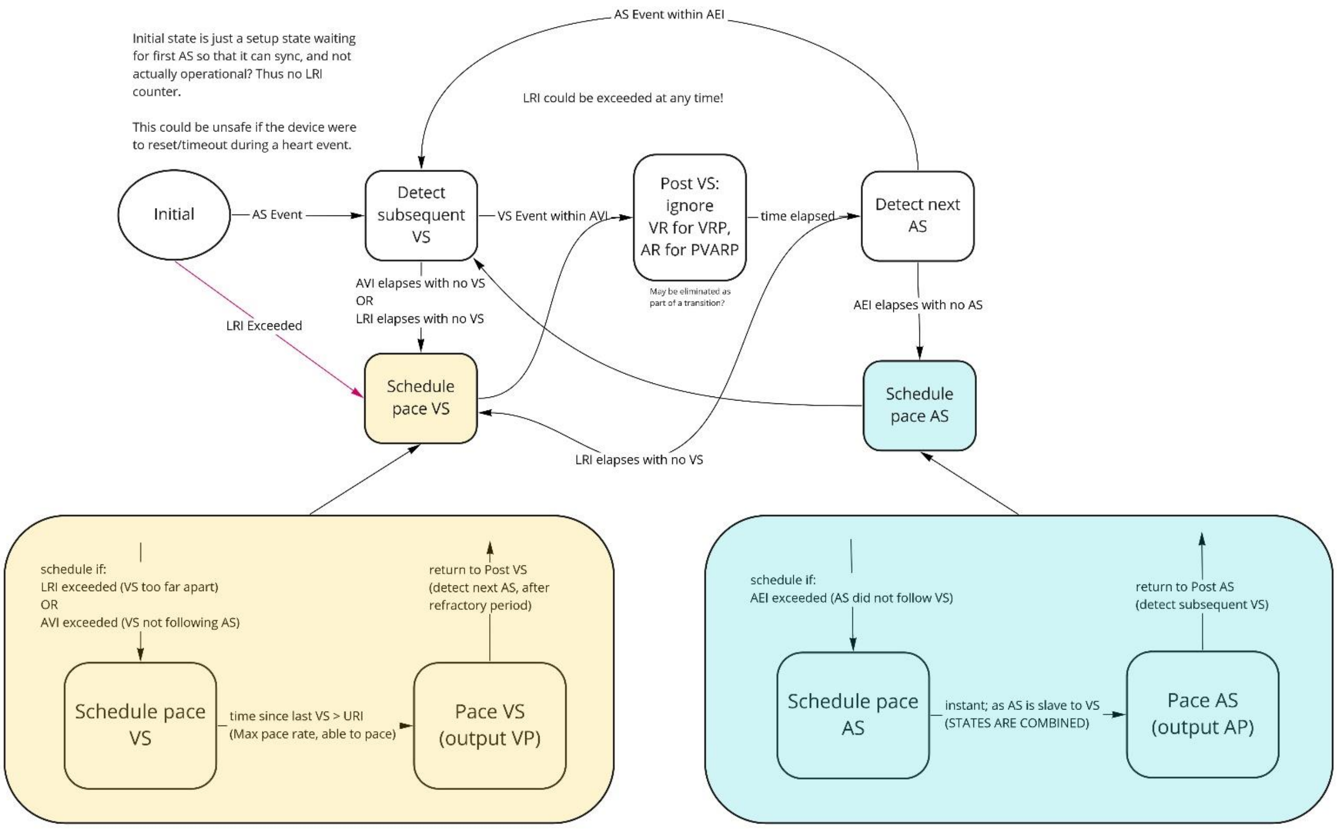
*Figure 9:* Gantt schedule of the project.

## 7. References

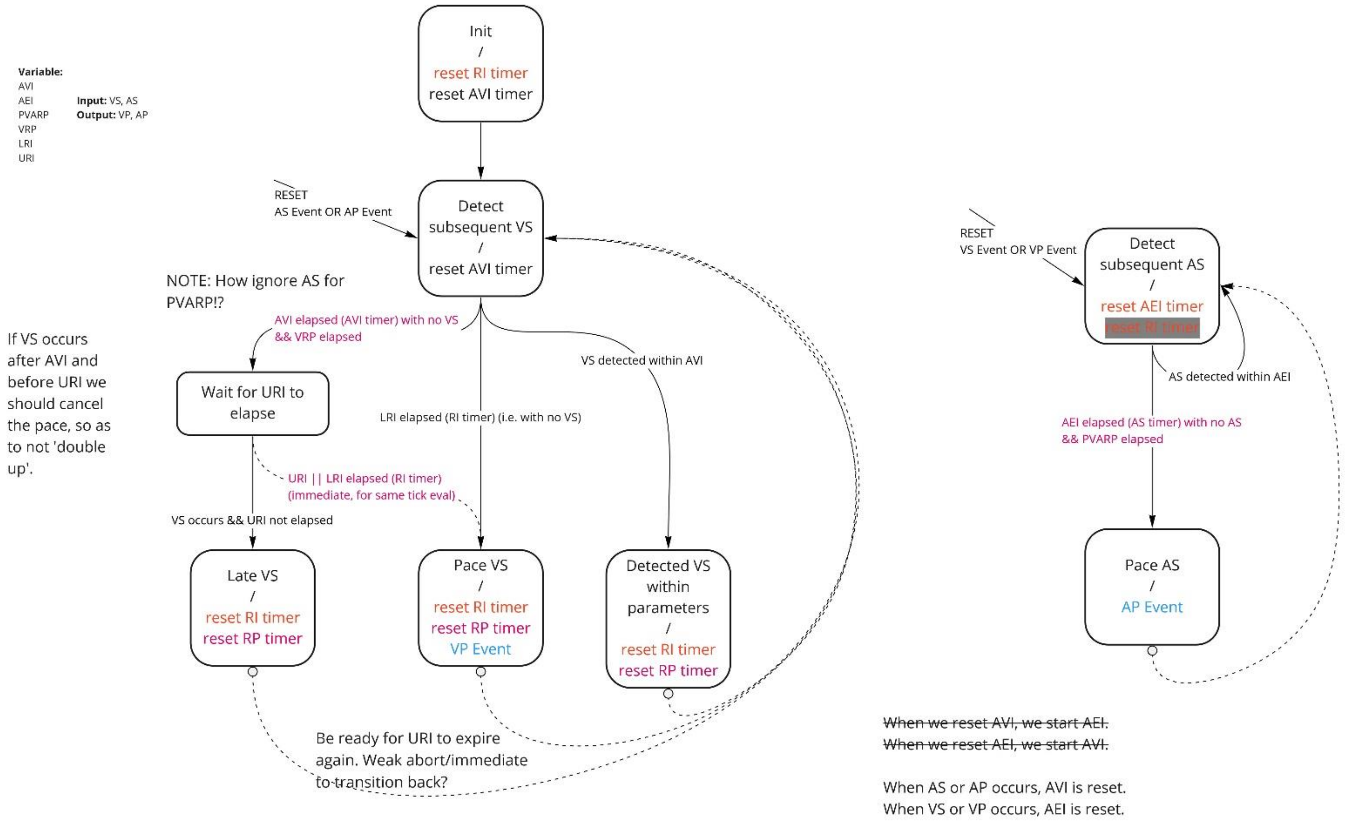
[1] Srinivas Pinisetty, Partha S Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, Reinhard Von Hanxleden. *Runtime Enforcement of Cyber-Physical Systems*. The ACM SIGBED International Conference on Embedded Software (EMSOFT), Seoul, South Korea, October, 2017.

# 8. Appendices

## 8.1. Appendix 1 – Draft 1

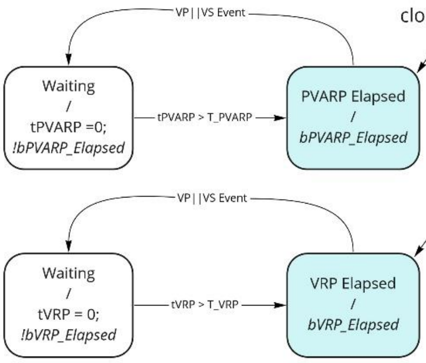

Initial state is just a setup state waiting for first AS so that it can sync, and not actually operational? Thus no LRI counter.
This could be unsafe if the device were to reset/timeout during a heart event.
AS Event within AEI
LRI could be exceeded at any time!
Initial
AS Event
Detect subsequent VS
VS Event within AVI
Post VS: ignore VR for VRP, AR for PVARP
May be eliminated as part of a transition?
time elapsed
Detect next AS
AVI elapses with no VS
OR
LRI elapses with no VS
LRI Exceeded
AEI elapses with no AS
Schedule pace VS
Schedule pace AS
LRI elapses with no VS
schedule if:
LRI exceeded (VS too far apart)
OR
AVI exceeded (VS not following AS)
return to Post VS
(detect next AS, after refractory period)
Schedule pace VS
time since last VS > URI
(Max pace rate, able to pace)
Pace VS
(output VP)
schedule if:
AEI exceeded (AS did not follow VS)
return to Post AS
(detect subsequent VS)
Schedule pace AS
instant; as AS is slave to VS
(STATES ARE COMBINED)
Pace AS
(output AP)

## 8.2. Appendix 2 – Draft 2


Variable:
AVI
AEI
PVARP
VRP
LRI
URI
Input: VS, AS
Output: VP, AP
Init
/
reset RI timer
reset AVI timer
RESET
AS Event OR AP Event
Detect subsequent VS
/
reset AVI timer
NOTE: How ignore AS for PVARP!?
If VS occurs after AVI and before URI we should cancel the pace, so as to not 'double up'.
AVI elapsed (AVI timer) with no VS
&& VRP elapsed
VS detected within AVI
Wait for URI to elapse
LRI elapsed (RI timer) (i.e. with no VS)
URI || LRI elapsed (RI timer)
(immediate, for same tick eval)
VS occurs && URI not elapsed
Late VS
/
reset RI timer
reset RP timer
Pace VS
/
reset RI timer
reset RP timer
VP Event
Detected VS within parameters
/
reset RI timer
reset RP timer
Be ready for URI to expire again. Weak abort/immediate to transition back?
RESET
VS Event OR VP Event
Detect subsequent AS
/
reset AEI timer
reset RI timer
AS detected within AEI
AEI elapsed (AS timer) with no AS
&& PVARP elapsed
Pace AS
/
AP Event
~~When we reset AVI, we start AEI.~~
~~When we reset AEI, we start AVI.~~
When AS or AP occurs, AVI is reset.
When VS or VP occurs, AEI is reset.

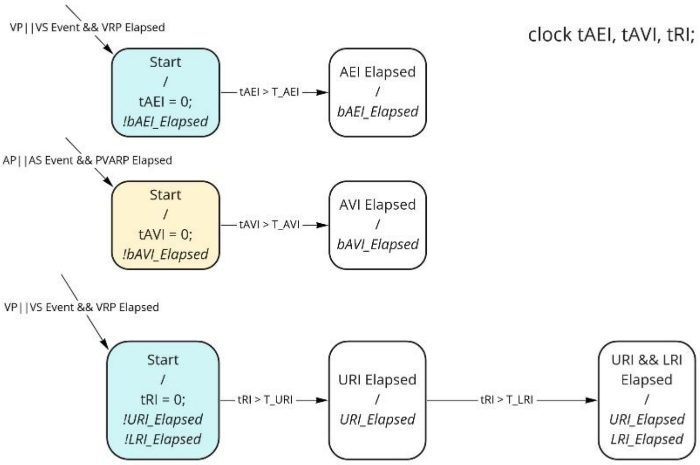
## 8.3. Appendix 3 – Draft 3

AS and VS Events are externally triggered signals.
AP and VP are internally triggered signals.



clock tPVARP, tVRP

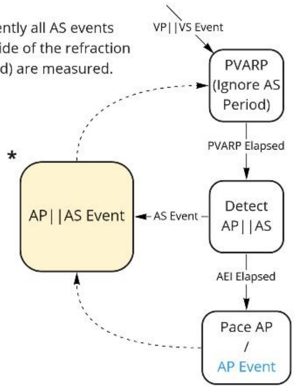
Only reset on signals outside of the refraction period.

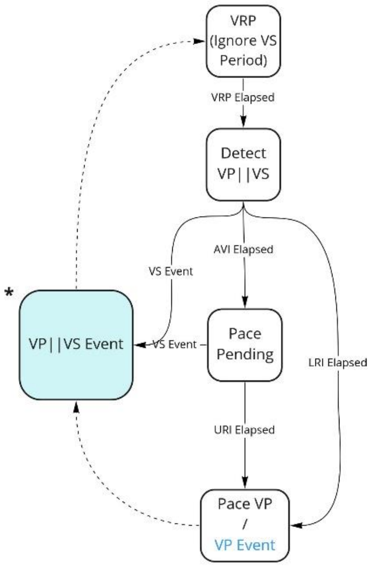


clock tAEI, tAVI, tRI;

Must reset on VP||VS, otherwise will jump to 'Detect' state (as PVARP only reset on VP||VS).

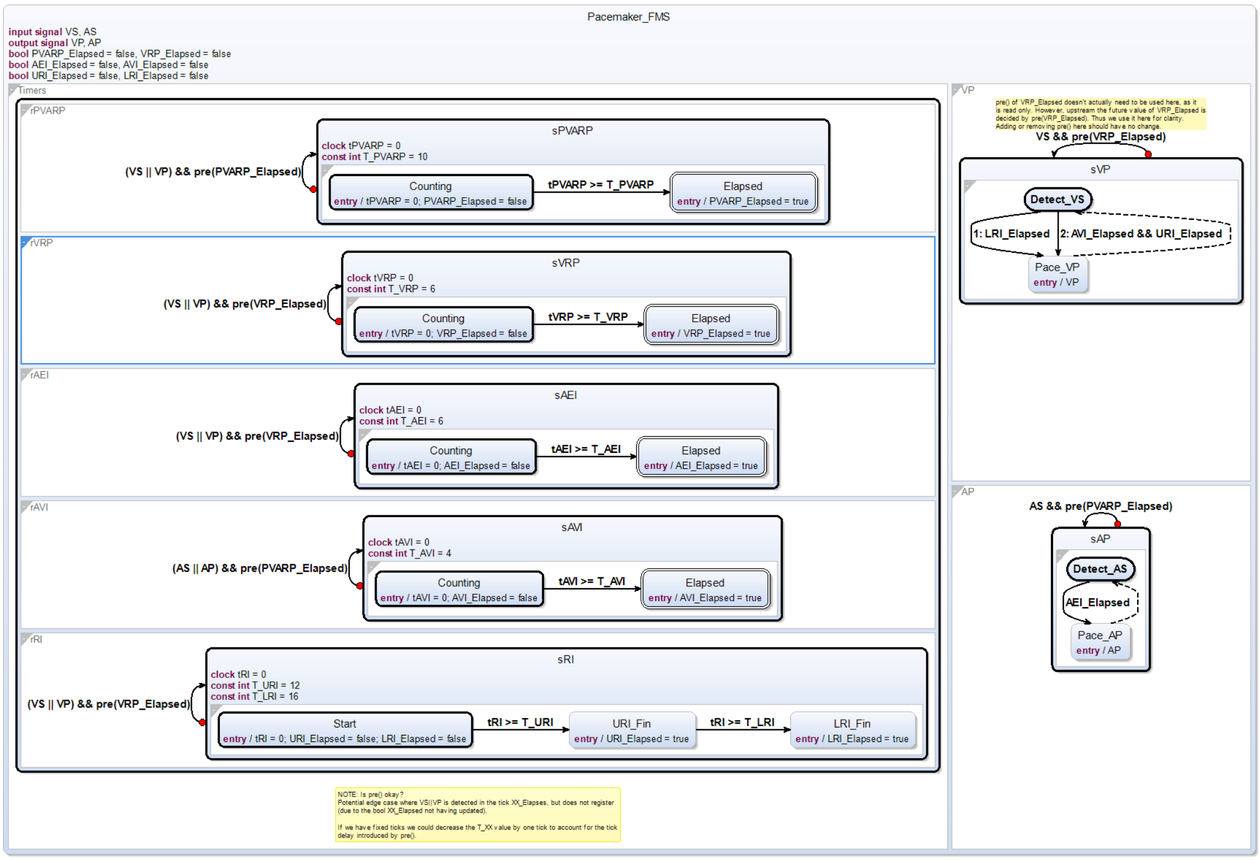
Currently all AS events (outside of the refraction period) are measured.





* Note that these states are representative and transient, and for display purposes only. In practice they should be 'short-circuited', the inbound transition connections attached to the sole outbound transition destination state.

## 8.4. Appendix 4 – Revision 2

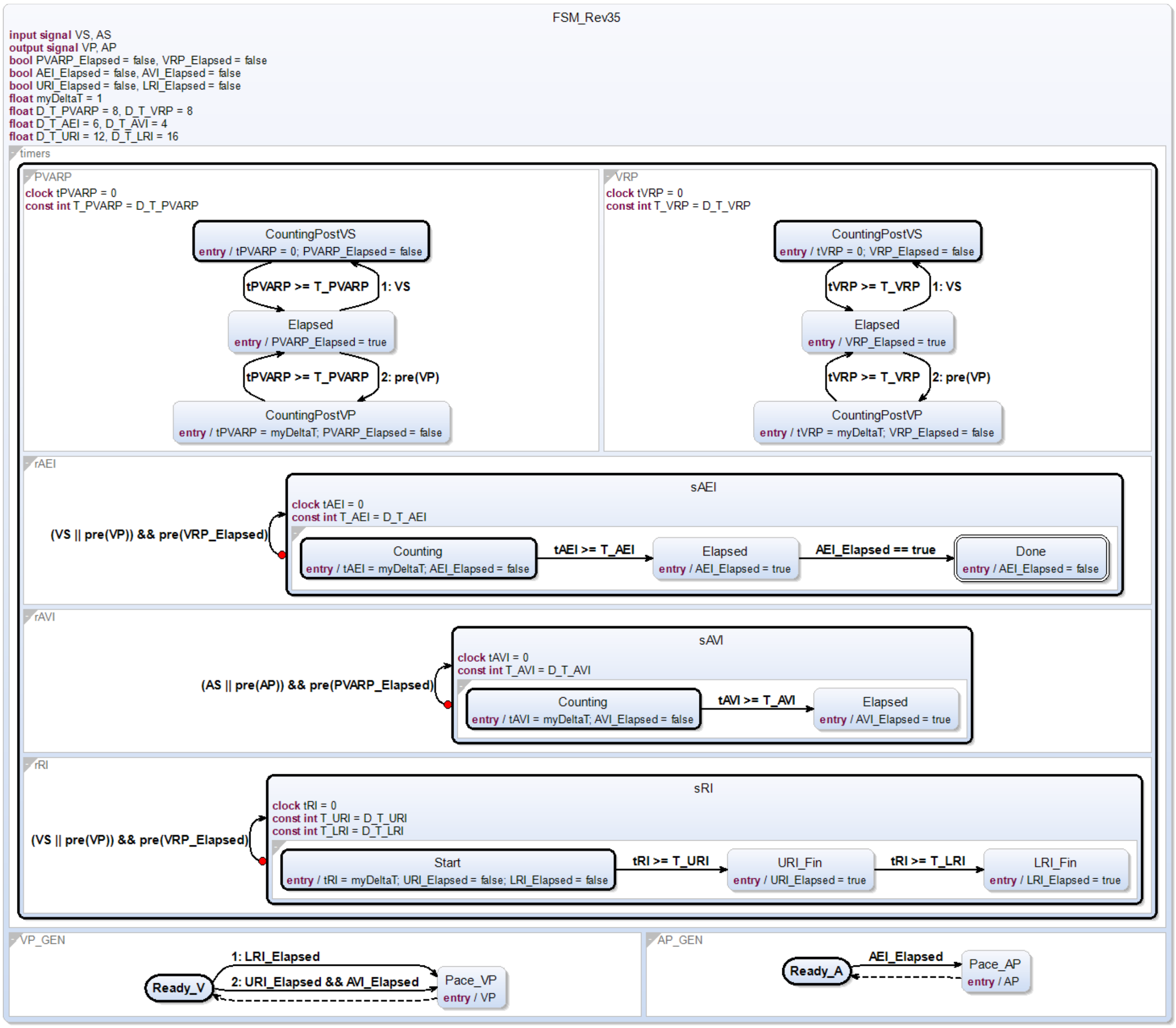

Pacemaker_FMS
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
Timers
rPVARP
sPVARP
clock tPVARP = 0
const int T_PVARP = 10
(VS || VP) && pre(PVARP_Elapsed)
Counting
entry / tPVARP = 0; PVARP_Elapsed = false
tPVARP >= T_PVARP
Elapsed
entry / PVARP_Elapsed = true
rVRP
sVRP
clock tVRP = 0
const int T_VRP = 6
(VS || VP) && pre(VRP_Elapsed)
Counting
entry / tVRP = 0; VRP_Elapsed = false
tVRP >= T_VRP
Elapsed
entry / VRP_Elapsed = true
rAEI
sAEI
clock tAEI = 0
const int T_AEI = 6
(VS || VP) && pre(VRP_Elapsed)
Counting
entry / tAEI = 0; AEI_Elapsed = false
tAEI >= T_AEI
Elapsed
entry / AEI_Elapsed = true
rAVI
sAVI
clock tAVI = 0
const int T_AVI = 4
(AS || AP) && pre(PVARP_Elapsed)
Counting
entry / tAVI = 0; AVI_Elapsed = false
tAVI >= T_AVI
Elapsed
entry / AVI_Elapsed = true
rRI
sRI
clock tRI = 0
const int T_URI = 12
const int T_LRI = 16
(VS || VP) && pre(VRP_Elapsed)
Start
entry / tRI = 0; URI_Elapsed = false; LRI_Elapsed = false
tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP
VS && pre(VRP_Elapsed)
sVP
Detect_VS
1: LRI_Elapsed
2: AVI_Elapsed && URI_Elapsed
Pace_VP
entry / VP
AP
AS && pre(PVARP_Elapsed)
sAP
Detect_AS
AEI_Elapsed
Pace_AP
entry / AP

pre() of VRP_Elapsed doesn't actually need to be used here, as it is read only. However, upstream the future value of VRP_Elapsed is decided by pre(VRP_Elapsed). Thus we use it here for clarity. Adding or removing pre() here should have no change.

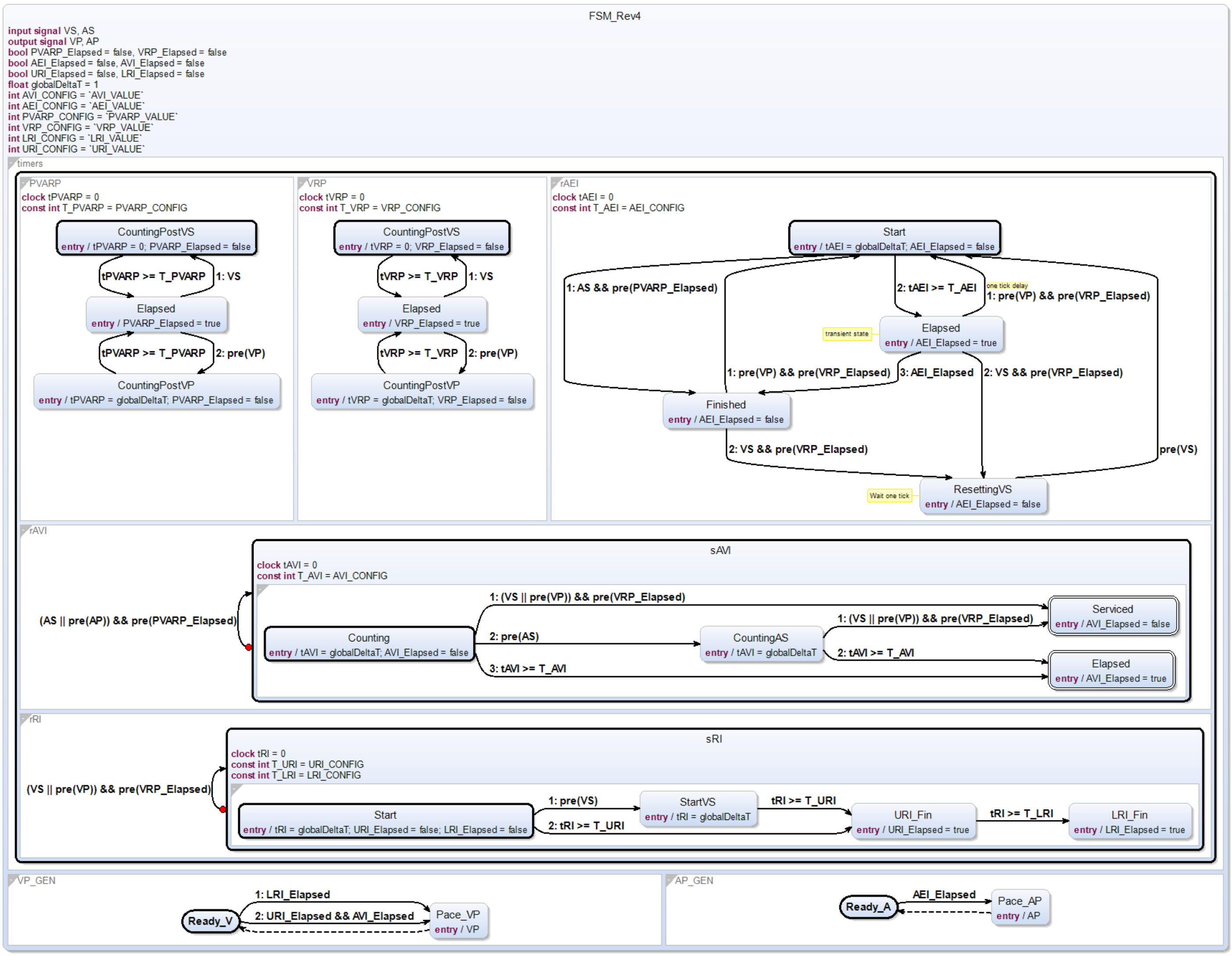
NOTE: Is pre() okay?
Potential edge case where VS||VP is detected in the tick XX_Elapses, but does not register (due to the bool XX_Elapsed not having updated).

If we have fixed ticks we could decrease the T_XX value by one tick to account for the tick delay introduced by pre().

## 8.5. Appendix 5 – Revision 3.5


FSM_Rev35
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
float myDeltaT = 1
float D_T_PVARP = 8, D_T_VRP = 8
float D_T_AEI = 6, D_T_AVI = 4
float D_T_URI = 12, D_T_LRI = 16
timers
PVARP
clock tPVARP = 0
const int T_PVARP = D_T_PVARP
CountingPostVS
entry / tPVARP = 0; PVARP_Elapsed = false
tPVARP >= T_PVARP
1: VS
Elapsed
entry / PVARP_Elapsed = true
tPVARP >= T_PVARP
2: pre(VP)
CountingPostVP
entry / tPVARP = myDeltaT; PVARP_Elapsed = false
VRP
clock tVRP = 0
const int T_VRP = D_T_VRP
CountingPostVS
entry / tVRP = 0; VRP_Elapsed = false
tVRP >= T_VRP
1: VS
Elapsed
entry / VRP_Elapsed = true
tVRP >= T_VRP
2: pre(VP)
CountingPostVP
entry / tVRP = myDeltaT; VRP_Elapsed = false
rAEI
sAEI
clock tAEI = 0
const int T_AEI = D_T_AEI
(VS || pre(VP)) && pre(VRP_Elapsed)
Counting
entry / tAEI = myDeltaT; AEI_Elapsed = false
tAEI >= T_AEI
Elapsed
entry / AEI_Elapsed = true
AEI_Elapsed == true
Done
entry / AEI_Elapsed = false
rAVI
sAVI
clock tAVI = 0
const int T_AVI = D_T_AVI
(AS || pre(AP)) && pre(PVARP_Elapsed)
Counting
entry / tAVI = myDeltaT; AVI_Elapsed = false
tAVI >= T_AVI
Elapsed
entry / AVI_Elapsed = true
rRI
sRI
clock tRI = 0
const int T_URI = D_T_URI
const int T_LRI = D_T_LRI
(VS || pre(VP)) && pre(VRP_Elapsed)
Start
entry / tRI = myDeltaT; URI_Elapsed = false; LRI_Elapsed = false
tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP_GEN
1: LRI_Elapsed
Ready_V
2: URI_Elapsed && AVI_Elapsed
Pace_VP
entry / VP
AP_GEN
Ready_A
AEI_Elapsed
Pace_AP
entry / AP

### 8.6. Appendix 6 – Revision 4


FSM_Rev4
input signal VS, AS
output signal VP, AP
bool PVARP_Elapsed = false, VRP_Elapsed = false
bool AEI_Elapsed = false, AVI_Elapsed = false
bool URI_Elapsed = false, LRI_Elapsed = false
float globalDeltaT = 1
int AVI_CONFIG = `AVI_VALUE`
int AEI_CONFIG = `AEI_VALUE`
int PVARP_CONFIG = `PVARP_VALUE`
int VRP_CONFIG = `VRP_VALUE`
int LRI_CONFIG = `LRI_VALUE`
int URI_CONFIG = `URI_VALUE`
timers
PVARP
clock tPVARP = 0
const int T_PVARP = PVARP_CONFIG
CountingPostVS
entry / tPVARP = 0; PVARP_Elapsed = false
tPVARP >= T_PVARP
1: VS
Elapsed
entry / PVARP_Elapsed = true
tPVARP >= T_PVARP
2: pre(VP)
CountingPostVP
entry / tPVARP = globalDeltaT; PVARP_Elapsed = false
VRP
clock tVRP = 0
const int T_VRP = VRP_CONFIG
CountingPostVS
entry / tVRP = 0; VRP_Elapsed = false
tVRP >= T_VRP
1: VS
Elapsed
entry / VRP_Elapsed = true
tVRP >= T_VRP
2: pre(VP)
CountingPostVP
entry / tVRP = globalDeltaT; VRP_Elapsed = false
rAEI
clock tAEI = 0
const int T_AEI = AEI_CONFIG
Start
entry / tAEI = globalDeltaT; AEI_Elapsed = false
1: AS && pre(PVARP_Elapsed)
2: tAEI >= T_AEI
one tick delay
1: pre(VP) && pre(VRP_Elapsed)
transient state
Elapsed
entry / AEI_Elapsed = true
1: pre(VP) && pre(VRP_Elapsed)
3: AEI_Elapsed
2: VS && pre(VRP_Elapsed)
Finished
entry / AEI_Elapsed = false
2: VS && pre(VRP_Elapsed)
pre(VS)
Wait one tick
ResettingVS
entry / AEI_Elapsed = false
rAVI
(AS || pre(AP)) && pre(PVARP_Elapsed)
sAVI
clock tAVI = 0
const int T_AVI = AVI_CONFIG
1: (VS || pre(VP)) && pre(VRP_Elapsed)
Counting
entry / tAVI = globalDeltaT; AVI_Elapsed = false
2: pre(AS)
CountingAS
entry / tAVI = globalDeltaT
1: (VS || pre(VP)) && pre(VRP_Elapsed)
Serviced
entry / AVI_Elapsed = false
2: tAVI >= T_AVI
3: tAVI >= T_AVI
Elapsed
entry / AVI_Elapsed = true
rRI
(VS || pre(VP)) && pre(VRP_Elapsed)
sRI
clock tRI = 0
const int T_URI = URI_CONFIG
const int T_LRI = LRI_CONFIG
Start
entry / tRI = globalDeltaT; URI_Elapsed = false; LRI_Elapsed = false
1: pre(VS)
StartVS
entry / tRI = globalDeltaT
tRI >= T_URI
2: tRI >= T_URI
URI_Fin
entry / URI_Elapsed = true
tRI >= T_LRI
LRI_Fin
entry / LRI_Elapsed = true
VP_GEN
Ready_V
1: LRI_Elapsed
2: URI_Elapsed && AVI_Elapsed
Pace_VP
entry / VP
AP_GEN
Ready_A
AEI_Elapsed
Pace_AP
entry / AP