Modelling the CANDO3 Optrode Command Interface

Leo Freitas*, Alastair Pollitt* and Patrick Degenaar+

School of Computing* and School of Electronic Engineering+, Newcastle University, UK

Abstract. *Background*: implantable medical devices are safety-critical systems. This paper describes the formal model of a micro chip for an optogenetics brain pacemaker preventing epileptic seizures.
Methods: we use a combination of abstract modelling, theorem proving, model checking, and C verification. This paper describes the abstract modelling only.
Results: we mathematically document hidden assumptions, provided electronic engineers with an abstract symbolic simulator and generated minimal test cases. Three serious flaws were found and fixed that had not been previously detected.
Conclusions: this was a socio-technical experiment: an electronic-engineering student computing-MSc with no experience in formalisms. This convinced stakeholders of formalism usefulness with clearly defined costs in time, effort, and expertise. Results are part of the evidence for the ongoing certification process.

1 Introduction

"Controlling Abnormal Network Dynamics using Optogenetics" (CANDO, `cando.ac.uk`) is a clinically orientated project to develop a new form of brain implant. Its aim is to utilise a combination of gene therapy and optoelectronics to provide closed-loop therapies to aberrant neurological conditions. The first target condition is being developed for focal epilepsy, which affects millions of people worldwide [21].

The approach is radically different to current neuromodulation therapies, which act as either open-loop pacemakers or attempt to provide a single burst of electrical stimulus at the onset of a seizure. The objective is to continuously control the brain state of the location of brain tissue where the seizures begin (seizure focus) to prevent it from operating outside of a safe domain. CANDO utilises a gene therapy called optogenetics to make brain cells light sensitive using channelrhodopsin [9], which can be genetically inserted into cells to make them light sensitive. The great advantage of this technique is that it is possible for different types of nerve cells to be sensitive to different wavelengths of light. Furthermore, as the stimulus and recording modalities are different, operation can be achieved without, or with significantly reduced crosstalk[1], allowing for real time closed-loop control.

All new techniques create challenges. In this case, optical pulses have to be generated in the brain. That necessitates a radically different architecture. Neuro-modulator devices

[1] Crosstalk is the unwanted coupling between signal paths like electrical coupling between transmission media or capacitance imbalance between wire pairs, non-linear performance, voltage and/or capacitance coupling, which are all essential features for closed loop control.

have a central control unit in a hermetic metal cannister, which determines the therapeutic intervention. However, in addition to a central control unit, there is an intelligent brain unit with electronic communication between the two. The brain unit provides optoelectronic stimulation and electrical recording.

Importantly, the brain unit needs to safely acquire commands from the control unit and return recordings of brain function. Any stimulus must be limited to ensure heating is kept below the 2^{o}C degree regulatory limit [6] and photochemical damage to brain tissue is minimised [17]. It therefore, is important to ensure that there are no errors in the stimulus timing commands or the overall control of the brain unit. It should never be the case that a light emitting diode (LED) is left on after the therapeutic stimulus is complete. This control is achieved locally with a finite state machine (FSM), whose correct functionality needs to be determined.

This is a multidisciplinary project involving electronic, chemical and material engineering, physics, computing science, medicine and microbiology at multiple academic institutions. The task of formalising the complex electronics became and interesting and difficult challenge given limited documentation. We started participation just before the middle (2016) of the pre-clinical phase (2014–2021). Both rodent and non-human primate trials are underway. It is hoped that human trials can be expected in the early 2020's during a follow up clinical phase.

This paper is focused on formal modelling and verifying the CANDOv3 system microelectronics controlling the brain unit. Specifically, we intervened at the micro chip, which was the first generation with a fully-functional FSM allowing for stimulation and recording. The process discovered: potential problems for certain instructions; unreachable instructions; and error recovery and chip reset problems. The work presented here describes an abstract VDM model of the FSM. This was then used for: a proof of correctness using Isabelle/HOL and model checking of the control loop termination using SPIN [20]; and to aid program verification its device driver that communicates with and commands the cortical implant applying the optogenetic treatment [15].

1.1 CANDO System

The CANDO system (see Figure 1) is composed of two primary intelligent components: i) a control unit placed in the chest; and ii) a brain unit responsible for brain stimulation and recording, as well as performing self-diagnostics. The control unit is responsible for the overall operation, closed loop processing, and communication with the outside world (for monitoring and programming). It is sealed in a hermetic metal canister placed in the chest. The brain unit needs to implement precisely timed stimulus and recording and comprises both local electronics and an optoelectronic array of "optrodes" that penetrate the target brain tissue. The design and implementation of the implantable optrodes are discussed in [4,23]. The communication and power supply between the control unit in the chest and the brain will eventually by developed in the form of a 4-wire alternating voltage interface. However, in this generation [11], a DC supply and serial peripheral interface (SPI) is used, which is sufficient for animal-grade

testing. The area of the brain that the optoelectronic array will be implanted into will have been made sensitive to light via a vector-mediated gene therapy [9].

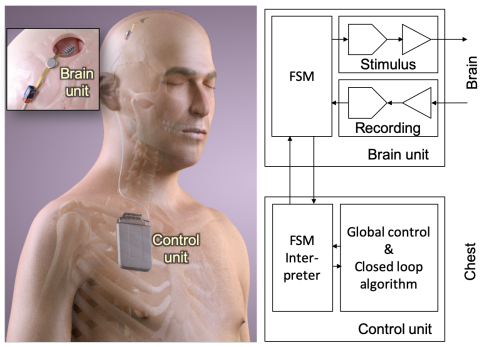

Fig. 1. The CANDO System

The brain unit consists of four optoelectronic arrays of optrodes that are assembled into a single package and implanted into the brain. The brain unit has three primary functions [4,22,23]: i) record neural activity from specific electrodes to amplify and filter that data and convert to a digital data stream to be sent to the control unit; ii) stimulate from specific light sources with defined intensity and pulse widths as determined by the control unit; and iii) perform diagnostic checks to ensure continued safe function of the brain unit with period data to be sent to the control unit.

Each optrode contains multiple electrodes and LEDs, all of which are controlled by a specially designed complementary metal oxide semiconductor (CMOS) chip with a 24-bit word bus and 17 optrode commands APIs (*e.g.* switching LEDs on/off, switching electrode recording sites on/off, diagnostics, *etc.*). The array, in turn, controls each of its optrodes (*e.g.* synchronicity between multiple electrode recording sites and LED response sites). Each control chip has a FSM, which determines the state of intervention (*e.g.* recording data, transmitting data and optical stimulation and subsets thereof). Global control and closed-loop processing occurs in the control unit situated in the chest (*e.g.* specific focal epilepsy treatment algorithms).

Treatment is delivered through algorithms in the chest unit, which distill to commands for the individual optrode (*e.g.* switch specific LEDs on for specified amounts of time and intensity; monitor/diagnose intended behaviour to ensure expected treatment is delivered, *etc.*), hence delivering the countermeasure to the focal seizure electric spike. For each optrode LED, a crucial safety property is that LEDs cannot stay on for long, as this would cause intolerable temperature differentials [6], and consequent brain function impairments [17]. Other properties exist within optrodes, within the optrodes in the array, and within the array and the chest unit.

As the CANDO project is implementing a medical device with complex control software in a safety critical application, verification is critical to ensure that the optrode

command interface is free from errors, operates as expected meeting its specification and ultimately ensuring the safety of patients. However, medical devices certification does not always guarantee safety. In [1], authors report recalls by the Food and Drug Administration (FDA) databases between 2006–2011 and found that 14.7% of recalls were software related. There were 1210 computer-related recalls that affected over 12 million devices. For these computer-related recalls, the FDA found that software failure was the single biggest cause of recall, making up 64.3% of the total. This is despite the fact that medical device software is heavily regulated [3,18].

Related Work

An excellent description of how regulatory bodies are influenced by evidence-based methods is in [19]. Application of formal reasoning to medical devices exist. For instance, Phillips Medical and Verum created a combination of BSDM and CSP for a number of complex devices [14]. This is highly commendable, yet necessitates adoption at the early stages of development, because of various tool dependencies are needed, yet require specific development-team configuration (*i.e.* a trained software engineer/-formalism expert) and demands investment (*i.e.* tools are not free).

There are high-quality open-source tools [10,2] enabling application of formalisms from capturing of requirements and risks, all the way to source code with data refinement and proof support. Yet, examples of their application to medicine are to a simplified dialysis machine, with unrealistic abstractions to how they work in practice. This tool chain also demands considerable investment to a number of "alien" languages to the non-expert. Moreover, these brilliant combinations of tools would face an uphill struggle over regulatory processes, which may not recognise efforts.

An ambitious and successful attempt at applying formalisms that inspired our earlier efforts was the formal analysis of the Boston Scientific cardiac pacemaker [12]. This was done within the McMaster's mock certification centre initiative (*i.e.* development was done as if under regulatory approval for Canada). To our knowledge, this was the first attempt to tackle the combination of applying formal techniques realistically for industrial-scale certified medical applications. A crucial difference to our efforts, however, is that it was a post-hoc exercise, rather than during actual development. Despite Boston Scientific's involvement throughout the exercise, a challenging (yet common) socio-technical issue emerged: what happens if an error is found? Pacemaker recalls are complicated, and serious perception/financial/legal damage would follow. To solve the conundrum, an earlier version was used in the exercise, where later/current versions were adjusted to take the use of findings through formal techniques into account. CANDO is similar given its embedded in-vivo nature as a brain implant.

We have successfully applied our approach to other medical devices. In [5], the identification, adaptation, and application of cost-effective industry standard formal techniques with acceptable learning requirements is described for a novel heamodialyser. Many of these results featured in the final CE-marked product (`allmedgroup.com`). Patent protected IP exists, and this machine is now in use within the UK national health ser-

vice. Finally, in [8], we describe the social-technical issues in using formal modelling and tools within three different certified medical devices.

2 Verification Method

The micro chip architecture (*i.e.* hardware commands, and data layouts) and embedded device driver control software is written in C. The chip fabrication automatically generates 115KLOC of C types, structs, *etc*. The user written (3.1KLOC) code is the actual controller that we modelled and verified. It comprises of three entities:

1. CMOS FSM governing chip's behaviour;
2. CMOS command APIs, assembling instructions packets for each optrode;
3. CMOS main loop, gluing together the FSM and serial communication.

A significant challenge arose from the nature and completeness of the documentation. The CANDOv3 CMOS design was documented via a mixture of circuit diagrams and VHDL code. As an academic project, it was difficult to get better documentation at this stage. As such we had no choice but to start from the C directly.

Understanding the interaction of modules and their dependencies allows the order of verification to be determined, where a bottom-up approach from the C was inevitable. We used VDM as an intermediate language to help hunt for formal specification before verifying the C code [15], which was guided by the workflow in Figure 2. This enabled the capture of various invariants implicitly expected from the FSM and the C code, but that had never been explicitly stated anywhere.

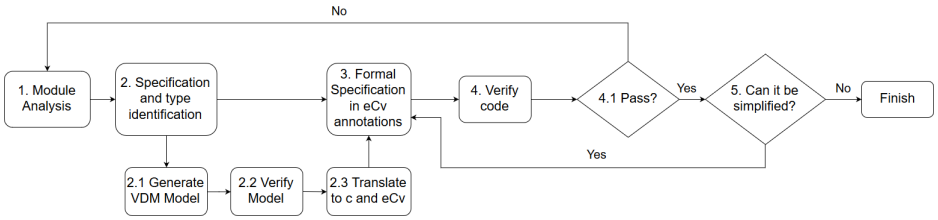

Fig. 2. Verification workflow

3 CANDO Optrode VDM Model

The CANDO FSM (Figure 3) has 34 states (nodes) and 21 events (edges), which are manipulated through 17 control commands (*e.g.* switch LED_k on/off). In the C program, these states and events are C `#define` binary constants with bit-vector representation closely associated to the hardware; in VDM, these are enumerated types `State`, `Event`, and `Command`. We identified a series of sets of `State` of particular relevance

in the way they were used. These were non-empty sets such as: states that sent or received data; states that created payloads for the top-level (17) instructions; error states, *etc.* Packets contain optrode addresses (*e.g.* i^{th}optrode within the array), what instruction to perform (*e.g* LED on/off), and data parameters (*e.g.* intensity, time, *etc.*).

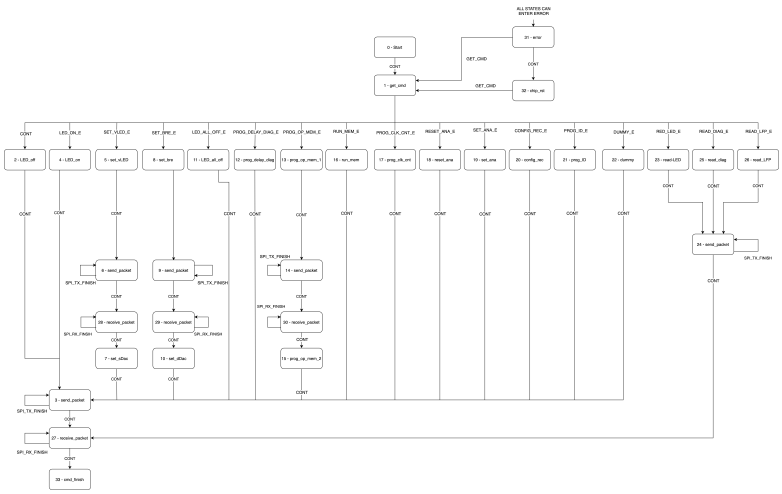

Fig. 3. Visualisation of the Optrode Command Interface Finite State Machine

The FSM is defined in C as a matrix of integers representing optrode API entry points indexed by states as rows and events as columns to states as cells, where only numbers are used (*i.e.* no named abstractions). That is, from an initial state and specific events to corresponding transitioned states (Listing 1). In VDM, this matrix is represented as maps. We chose to have a map from `Event` to a map from `State` to `State`: for each `Event` that can occur, the FSM gives the destination `State` from the original `State` where it occurs. This is because the corresponding invariants for the original C matrix are unnecessarily more involved. Our VDM matrix library also benefits from this homomorphic set up with the transposed matrix.

We define the matrix columns as a map from `State` to `State` with 8 invariants. The first five limit reachability of different states: 1) nothing reaches the starting state; and 2–5) start, error, command finish and chip reset states can only reach specific states. Next, 6) enforces that all known states that create data packets must map to send or error states. Similarly, 7) known states that receive data packets must map to states receiving data (*e.g.* command finish, error and multiple-stage packet construction for larger data payloads). Lastly, 8) only initial packet creation states are available as CMOS chip instructions (*i.e.* state `get_cmd` can only be mapped to command APIs).

```
1 StateMap = map State to State
2 inv s ==
3 not <start>  in set rng s and                                    --1
```

```
  (<start>   in set dom s => s(<start>)   in set {<get_cmd>,<error_>}) and      --2
  (<error_>  in set dom s =>s(<error_>) in set {<get_cmd>,<chip_rst>,<error_>})and--3
  (<chip_rst>in set dom s => s(<chip_rst>)in set {<get_cmd>,<error_>}) and      --4
  (<cmd_fin> in set dom s => s(<cmd_fin>)  = <error_>) and                      --5
  (forall  p in set dom s inter packet_creator_states &                        --6
        s(p) in set ({<error_>} union send_states)) and
  (forall  r in set dom s inter receive_states &                               --7
        s(r) in set {<cmd_fin>, r, <error_>} union
                multiple_stage_packet_creator_states) and
  (<get_cmd> in set dom s => s(<get_cmd>) in set                              -- 8
    ({<error_>} union stage_one_packet_creator_states) \
    (send_states union receive_states union multiple_stage_packet_creator_states));
```

Listing 1. C FSM columns represented as a VDM map

Next, we define invariants about 8 sub maps within the FSM columns. A `TXMap` maps send states to receive states only, whereas an `IdMap` and `ErrorMap` map states to themselves or error, respectively. A `PacketMap` maps packet creator states (*i.e.* CMOS chip instructions) to send states, whereas a `ReceiveMap` maps receive states to multiple-stage packet states or signals that transmission has finished.

```
TXMap = StateMap
inv  m == dom m subset send_states and rng m subset receive_states;

IdMap = StateMap
inv  m == forall s in set dom m & m(s) = s;

ErrorMap = StateMap
inv em == forall s in set dom em & em(s)=<error_>;

PacketMap = StateMap
inv pm == dom pm subset packet_creator_states and rng pm subset send_states;

ReceiveMap = StateMap
inv rm == dom rm subset receive_states and
          rng rm subset multiple_stage_packet_creator_states union {<cmd_fin>};
```

The actual FSM is a total map on all known events and states, given C arrays are dense, which entails that all possible transitions have to be defined. Finally, we map all events to total map over states. The partial map `FSM` is useful when initialising the system state: we can construct the total map by parts, depending on the kind of CMOS instruction, which then when finished, should be to a total `TFSM` map.

```
TStateMap = StateMap             inv sm == dom sm = ALL_STATES;
FSM       = map Event to StateMap;
TFSM      = map Event to TStateMap inv fsm == dom fsm = ALL_EVENTS;
```

The CANDO FSM has 12 invariants filtering allowed state sub maps per event. All maps are total: no domain checks are necessary for map applications. The state map from the `CONT` event has to be filtered (`<:`) in 3 cases: 1) send states has to be a transmission map as all send states must map to receive states on the `CONT` event; 2) packet creation states has to be a packet map as all packet creator states must map to send states; and

3) receive states after a CONT event must map to command finish or a stage-two packet states (*i.e.* be a ReceiveMap).

```
CandoFSM = TFSM
inv fsm ==
       is_TXMap(send_states           <: fsm(<CONT>)) and                        --1
   is_PacketMap(packet_creator_states <: fsm(<CONT>)) and                        --2
  is_ReceiveMap(receive_states        <: fsm(<CONT>)) and                        --3
       is_IdMap(send_states           <: fsm(<SPI_TX_FIN>)) and                  --4
       is_IdMap(receive_states        <: fsm(<SPI_RX_FIN>)) and                  --5
    is_ErrorMap(error_states <: (packet_creator_states<-:fsm(<CONT>))) and      --6
  fsm(<CONT>)(<start>)         = <get_cmd>  and                                 --7
  fsm(<CONT>)(<error_>)        = <chip_rst> and                                 --8
  fsm(<GET_CMD_E>)(<error_>)   = <get_cmd>  and                                 --9
  fsm(<GET_CMD_E>)(<chip_rst>) = <get_cmd>  and                                --10
 (forall x in set dom({<CONT>}<-:fsm) & fsm(x)(<start>) = <error_>) and        --11
(forall x in set dom({<CONT>,<GET_CMD_E>}<-:fsm) & fsm(x)(<error_>)=<error_>);--12
```

Listing 2. Complete C FSM as a VDM map

The serial peripheral interface (SPI) send/receive packets via the SPI_TX/RX_FIN events. That is, 4-5) whilst these events are taking place, the FSM sub state map for send/receive states loops/waits (*e.g.* all send states must map to themselves whilst the SPI_TX_FIN event happens). Whenever a CONT event happens beyond the CMOS instruction commands (*e.g.* ALL_STATES less packet creator states), error states are unrecoverable (*i.e.* if they have been reached, the FSM cannot transit anywhere but to an error state). This is imposed by (6) the domain filtering and anti-filtering (<-:) operators over the FSM on the CONT event.

There are four cases where specific mappings have to be explicit: 7–8) the start and error_ states always leads to get_cmd and chip_rst states on the CONT event, respectively; and 9–10) GET_CMD event can recover an error_ or chip_rst state to a get_cmd state. Finally, we have two cases of error handling: 11) the start state always leads to error on every event but CONT; and 12) it is not possible to recover from an error_ state, unless a CONT or GET_CMD event happens.

The practical use of this type is within the state of the system (*e.g.* C program global variables), which is defined next. Given the hardware states expectation is fixed regardless of the current state, we have the peculiar invariant checking that the expectations between states hold: all known states form a partition between states that do handle packets, that do not, and that do not care. Partition is the mathematical notion that the distributed union of given states equal ALL_STATES, and that each set is pairwise disjoint (*e.g.* has no intersection between themselves). This makes initialisation easy (*i.e* multiple specific choices hold), except for the fsm total map, which is built with the auxiliary function fsm2tfsm below. The optrode_TX/RX_fin, bytes_rcvd/sent, and tx_cnt variables are important to encode the staged send/receive of packet, which occur in multiple calls to the send/receive_packet APIs.

```
functions
  disjoint [@elem]: seq of (set of @elem) +> bool
  disjoint(s) ==
    (len s > 1)    =>                          -- seq that are trivially disjoint
```

```
    len s = card { i | i in set inds s &.      -- seq of sets are pairwise disjoint
                       forall j in set inds s & j > i =>s(i) inter s(j) = {} };

  partition [@elem]: seq of (set of @elem) * set of @elem +> bool
  partition(s, p) == disjoint[@elem](s) and p = dunion(elems s);

values
  S_PACKET_STATES_INVARIANT: bool = partition[State](
    [NON_NIL_PACKET_STATES,NIL_PACKET_STATES,EITHER_PACKET_STATES], ALL_STATES);

state FSM3 of
  fsm       : CandoFSM    cmd_fin_flag  : bool   bytes_rcvd : Bytes
  currSt    : State       optrode_TX_fin: bool   bytes_sent : Bytes
  currEvt   : Event       optrode_RX_fin: bool   tx_cnt     : Count
  currentCmd: Command     s_packet      : [Packet]
inv  mk_FSM3(-,-,-,-,-,-,-,-,-,-,-,-) == S_PACKET_STATES_INVARIANT
init FSM3 == FSM3 = mk_FSM3(fsm2tfsm(recommended_fsm), false,
      <start>, <CONT>, <LED_ON_C>, false, true, true, nil, 3, 0, 0) end
```

The auxiliary functions complete the maps to the error state for all states and events based on an initial partial map (`recommended_fsm`) constant containing all explicit declared events and state relationships from the C code. This was useful to discover that not all error scenarios were being accounted for.

```
sm2tsm: StateMap -> TStateMap
sm2tsm(sm)=={s|->if s in set dom sm then sm(s) else <error_> | s in set ALL_STATES};

fsm2tfsm: FSM -> TFSM
fsm2tfsm(fsm) == { e |-> if e in set dom fsm then sm2tsm(fsm(e))
                         else sm2tsm({|->}) | e in set ALL_EVENTS };
```

For all 34 states one optrode API is defined. Of those, most (20) operations are for packet creator states: they assemble a binary packet containing a specific optrode address, command, and parameters. Their specification is straightforward. For the LED on command, the VDM frame insists on what part of the state can change (*e.g.* `ext` reads and writes clauses) and the corresponding C code (Listing 3). This is a concrete example where the VDM and C verification differ: in [**?**], there is considerable complexity in bit-vector expressions, which here can be completely abstracted given we are only interest in the FSM's behaviour.

```
LED_on() == (currEvt := <CONT>; s_packet := mk_Packet(<Opt_addr>,<LED_ON>,<LED_addr>))
ext rd currSt wr currEvt, s_packet
pre currSt = <LED_on> post currEvt = <CONT> and s_packet <> nil;
```

```
void LED_on(void) { event = CONT;
  packet = ((((Optrode0.Optr_addr & BITS_6) << 18) | ((LED_ON & BITS_6) << 12)
            | (Optrode0.LED_addr & BITS_5)) & BITS_24); }
```

Listing 3. FSM API in VDM and in C

The other operations are for (5) send and (4) receive states, which may loop (*e.g.* 8 bits at a time up to 24 bits), depending on the packet size and command. The are (2) operations for error management and (3) for initialisation, selection and termination states, respectively. The main loop is defined in Listing 4. It ensures that specific states

must always have a valid (non-`nil`) data packet. As packet lengths can be bigger than the hardware-bus size, they require multiple send/receive API calls. This is controlled by the postcondition: if the original state was a stage-two package creator state, we have to reset the `optrode_TX/RX_fin` flags; whereas if we are to start a send/receive operation, the flags must be true if bytes sent/received are within their size limit.

```
execute() ==
  (cases currSt:
      <start>   -> start(),
      <get_cmd> -> get_cmd(currentCmd),
      <LED_on>  -> LED_on(), ...                          -- one case per FSM state
   end)
ext rd currSt, currentCmd, s_packet
post
  (not currSt in set EITHER_PACKET_STATES =>
      (currSt in set NON_NIL_PACKET_STATES <=> s_packet <> nil) and
      (currSt in set NIL_PACKET_STATES     <=> s_packet = nil)) and
  (currSt in set stage_two_packet_creator_states => optrode_TX_fin)  and
  (currSt in set send_states and bytes_sent < PACKET_LEN => optrode_TX_fin) and
  (currSt in set receive_states and bytes_rcvd < PACKET_LEN => optrode_RX_fin);

manual()==
  (while(not command_finish_flag) do (currSt := fsm(currEvt)(currSt); execute()))
ext rd fsm, currEvt, currSt, command_finish_flag;
```

Listing 4. VDM access point to all FSM APIs

We checked that the recommended FSM differed from the original C version only at the points where identified design errors occurred. Finally, we wrote trace specifications to ensure that the the model had 100% coverage. This entail all parts of the specification were reachable (*i.e.* no "dead" specification). This was achieved through direct calls for all 17 APIs, as well as error states.

```
   (({<SPI_TX_FINISH>,<PROG_OP_MEM_E>}<-:recommended_fsm) =
    ({<SPI_TX_FINISH>,<PROG_OP_MEM_E>}<-:original_C_fsm));
```

4 Results and Discussion

The CANDOv3 VDM model, its proof of correctness in Isabelle/HOL [20] (*i.e.* satisfiability proofs over type invariants, state, and function and operations pre/post conditions specification) and the C device driver program verification [15] are now part of the certification submission, as well as being used to inform the CANDOv4 design. The main outcomes from our analysis associated with this paper are: i) VDM model and 100% coverage of FSM invariants; and ii) VDM simulator of CMOS APIs. After corrections in understanding, three serious scenarios were uncovered in the chip design. First, a key state related to programming the optrode memory was unreachable due to an earlier copy-paste error in the hardware design file, hence all its corresponding sub states were unreachable. Second, packet data were being sent to a mistaken state due to a wiring problem discovered through an API proof failure. Finally, the chip reset command led to an unrecoverable state. The first case was a coding mistake not observed during hardware testing, despite its potentially serious consequences in practice. The second case

was a misunderstanding by the CMOS engineers and device driver programmers, which would entail potential loss of a diagnostic signal, where consequences are yet unknown. The final case was a known incomplete part of the design on what was to happen under the specific reset conditions.

CMOS engineers valued the possibility of FSM simulation without the complicated and *fiddly* instrumentation (*e.g.* debugging device drivers lead to unexpected outcomes if lag/delays are introduced), as well as the precise documentation of underlying assumptions. Device driver engineers appreciated the outcomes in terms of helping them identify issues, as well as to ensure that potential (error-prone) device driver encoding mistakes were caught as early as possible. There has been significant insight gained through the process with regards provision of suitable documentation for non-microelectronics experts. Furthermore, whereas test engineers normally test the chip in simulation, in this case a grudging acceptance developed that formal methods can provide useful additional insight. All became convinced of its future value for regulatory submission. Finally, the regulatory submission process itself looks at the history of development. This work on CANDOv3 microelectronics will be part of the regulatory submission.

The VDM model has become a valuable resource for constructing future CANDO FSMs. The VDM model is being updated for CANDO v4. The CMOS instruction set FSM model in VDM is small (*i.e.*1040LOC with documentation) and captured 28 implicit invariants/assumptions not clearly stated anywhere. It was written with (symbolic) executability in mind: we can simulate the FSM behaviours and play with any variations of test scenarios. This enabled simulation of optrode treatment algorithms and to calculate FSM coverage (*e.g.* whether the FSM had been completely traversable).

The exercise was crucial to help understand what was the specification for the various FSM commands encoded in the C device driver, given that eliciting such invariants directly from C was very difficult and error prone (*e.g.* C details were mixed within the key hidden abstractions we were hunting for). The VDM model enabled translation to Isabelle/HOL and its proof of satisfiability reported in [**?**,20]. For instance, the initial (weaker) postcondition of the `execute` operation stated that `s_packet` had to be valid or not for specific states only (Listing 4). The failed satisfiability proof in Isabelle highlighted the subtle nature of when that was the case, which entailed the extra check that the current state could not be in `EITHER_PACKET_STATES` (*i.e.* `error_, cmd_fin`, or any receive state), given they allowed for both `nil` and non-`nil` data packets. Furthermore, this proof failure also led to the discovery of the last three conjuncts of the `execute` postcondition, which then increased the VDM symbolic execution coverage from 85% to 100%. This demonstrates the importance of proof within formal development.

The process of transforming the 1040LOC from VDM to Isabelle was manual and at first based on [7], then on an automated process [16]. The transformation of the VDM model to C was also novel [**?**]: VDM partial maps and their corresponding C arrays have been proved using Isabelle/HOL, and the C code templates corresponding to the VDM were verified for functional correctness (*e.g.* satisfiability of type invariants, state, pre/post conditions of functions/operations, *etc*.) with a C program verifier [**?**].

Socio-technical Challenges. We hope to improve on incorporating formal verification into the design process of a multidisciplinary team on a project with funding and time-scale constraints. Pre-clinical projects first need to demonstrate efficacy and that is where the predominant scientific effort needs to be placed. Nevertheless, scalability to long-term safe operation is of significant interest. So in balancing these needs, the key perceived issue centred on the opportunity cost: within a defined budget any time not spent by the development team on the primary objective (demonstrating effective operation), could lead to greater project risk. The ideal situation would be for a verification team exploring the secondary objective (demonstrating safe operation).

A key challenge was that the development team, mostly researchers and PhD students in electronic engineering, point blank refused any extra work/learning given the intense milestone delivery timelines. This first round of development from a semi-experienced team would be code from exemplar functions provided by the Microcontroller company. This makes adding MISRA compliance [13] during the development phase a significant extra challenge. It was through convincing the team leader about the research potential (*i.e.* represent the FSM and the assessed risks mathematically in order to enable multiple kinds of analysis), that work was possible. Device driver programmers became curious, and in conjunction with their patience to explain various unclear/undocumented decisions, they engaged with the process. Once that happened, we could see a direct improvement in how the next stages of the C device driver code got implemented: MISRA-compliance, a key characteristic of safety-critical C programs was achievable and the verification process has since been completed. Eventually, the CMOS engineers also got onboard once they "saw" how mistakes could be prevented.

Introducing the process of formal verification to the development team and working with them during the application of techniques to help develop their understanding is crucial. This knowledge transfer is galvanised by regularly attending engineering meetings for discussion and presenting the progress of the work, as well as our understanding of the medical device software regulatory process.

5 Conclusions

This paper summarises the VDM model of a micro chip finite state machine (FSM) controlling a novel closed-loop optogenetic brain neuromodulator for epilepsy. The VDM model helped discover and fix errors and documents previously unknown assumptions. It has been used to identify and maintain the chip's FSM correctness (Isabelle/HOL proof [20]), and to inform the C device driver invariants [15] ensuring safety properties of a safety-critical medical device that is about to start primate and human trials. The work is now part of the CE-marking certification process. This demonstrates how formalisms can be effectively applied for realistic novel medical devices in practice.

Acknowledgements. We are thankful to Ben Wooding, Dimitrios Firfilionis and Ahmed Soltan from CANDO (Wellcome: 096975/Z/11/Z; EPSRC: 102037/Z/13/Z) for their patience and interest. To STRATA (EPSRC EP/N023641/1) for financial support. Finally, thanks to Nina for inspiration to work with medical devices dependability.

References

1. H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, 2013.
2. P. Arcaini, S. Bonfanti, A. Gargantini, A. Mashkoor, and E. Riccobene. Integrating formal methods into medical software development: The ASM approach. *Science of Computer Porgramming*, 158:148–167, 2018.
3. BSI. Medical device software — software life-cycle processes. Technical Report BS EN 62304, British Standards (BSi), May 2011.
4. F. Dehkhoda, A. Soltan, R. Ramezani, H. Zhao, Y. Liu, T. Constandinou, and P. Degenaar. Smart optrode for neural stimulation and sensing. In *SENSORS*, pages 1–4. IEEE, 2015.
5. M. D.Harrison, L. Freitas, M. D. snd Jose C. Campos, P. Masci, C. di Maria, and M. Whitaker. Formal Techniques in the Safety Analysis of Software Components of a new Dialysis Machine. *Science of Computer Porgramming*, 175:17–34, Feb 2019.
6. N. Dong et al. Opto-electro-thermal optimisation of optoelectronic probes for optogenetic neural stimulation. *Journal of Biophotonics*, 11(10), March 2018.
7. L. Freitas, C. B. Jones, A. Velykis, and I. Whiteside. How to say why. Technical Report CS-TR-1398, Newcastle University, www.ai4fm.org/tr, November 2013.
8. L. Freitas, B. Scott, and P. Degenaar. Medicine-by-wire: formal techniques for dependable medical systems automation. *Science of Computer Porgramming*, under-corrections, 2020.
9. P. Hegemann and G. Nagel. From channelrhodopsins to optogenetics. *EMBO Molecular Medicine*, 5(2):173–176, Feb 2013.
10. T. S. Hoang, C. Snook, A. Salehi, M. Butler, and L. Ladenberger. Validating and verifying the requirements and design of a haemodialysis machine using the rodin toolset. *Science of Computer Programming*, 158:122 – 147, 2018.
11. J. Luo et al. The neural engine: A reprogrammable low power platform for closed-loop optogenetics. In *IEEE TBCAS*, October 2018.
12. Macedo H.D. et al. Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM. In *International Symposium on Formal Methods*, volume 5014 of *LNCS*, pages 181–197. Springer, 2008.
13. MISRA Ltd. MISRA-C:2012 Guidelines for the use of the C language in critical systems. Technical Report MISRAC:2012, MISRA Ltd, March 2013.
14. A. Osaiweran, M. Schuts, and J. Hooman. Experiences with incorporating formal techniques into industrial practice. *Empirical Software Engineering*, 19(4):1169–1194, 2014.
15. A. Pollitt. Verifying the CANDO Project Optrode Command Interface in eCv. Master's thesis, School of Computing Science, Newcastle University UK, August 2018.
16. J. Simm. Creating a Tool for Translating VDM to Isabelle/HOL. Master's thesis, School of Computing Science, Newcastle University UK, July 2019.
17. A. Soltan et al. A head mounted device stimulator for optogenetic retinal prosthesis. *Journal of Neural Engineering*, 15(6), August 2018.
18. U.S. Department of Health and Human Services. General principles of software validation; final guidance for industry and fda staff. Technical Report UCM085281, FDA, Jan 2002.
19. D. Vogel. *Medical Device Software Verification, Validation, and Compliance*. Artech House, 2010.
20. B. Wooding. Using Formal Methods and Proof to Verify a CANDO Epilepsy Medical Device. Master's thesis, School of Computing Science, Newcastle University UK, June 2019.
21. World-Health-Organization. Epilepsy, Feb 2018. Accessed 08-4-2018.
22. H. Zhao, F. Dehkhoda, R. Ramezani, D. Sokolov, P. Degenaar, Y. Liu, and T. Constandinou. A cmos-based neural implantable optrode for optogenetic stimulation and electrical recording. In *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE*, pages 1–4. IEEE, 2015.

23. H. Zhao, A. Soltan, P. Maaskant, N. Dong, X. Sun, and P. Degenaar. A scalable optoelectronic neural probe architecture with self-diagnostic capability. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2018.