Modelling CANDO

Leo Freitas

Overture 11/06/2025, Aarhus

In collaboration with :

- Alistair Pollitt PhD Student
- Patrick Degenaar Brain pacemaker eng. (NCL) lead

Motivation

- Most medical device companies are small
- CE-marking safety-critical software is tricky:
  - Regulation and standards only provide guidelines
  - Engineers are not trained in dependable software techniques
- Myriad of regulatory documentation
  - Over 20 different standards to adhere to and address
  - Lack of clarity for a pathway to certification
  - Software is regulated / classified by IEC62304
- Case study selection
  - Safety-critical software for a medical device
  - Prior work done by Newcastle

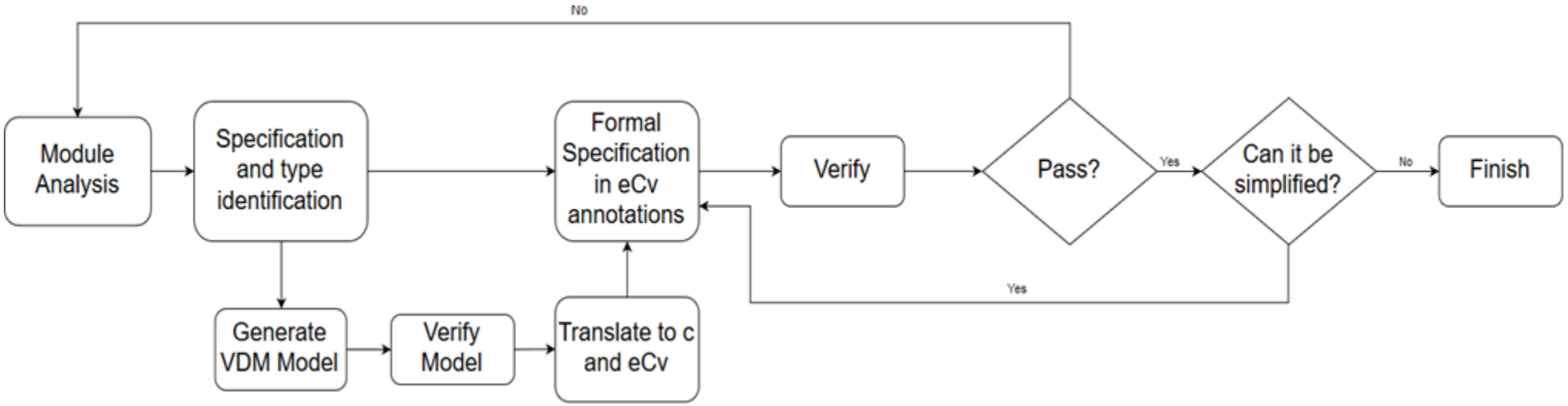
Methodology

- Application of model based techniques to medical device
  - Model-based formalisms for the design (i.e. FSM + properties)
  - Code-level verification for C/C++ (partial/total correctness)
  - **Link solutions to regulatory process is crucial**

- Have a direct link with risk/hazard profile
  - Ensures stakeholders and regulator care about the formal result
  - Enables clear demonstration of what the approach can achieve

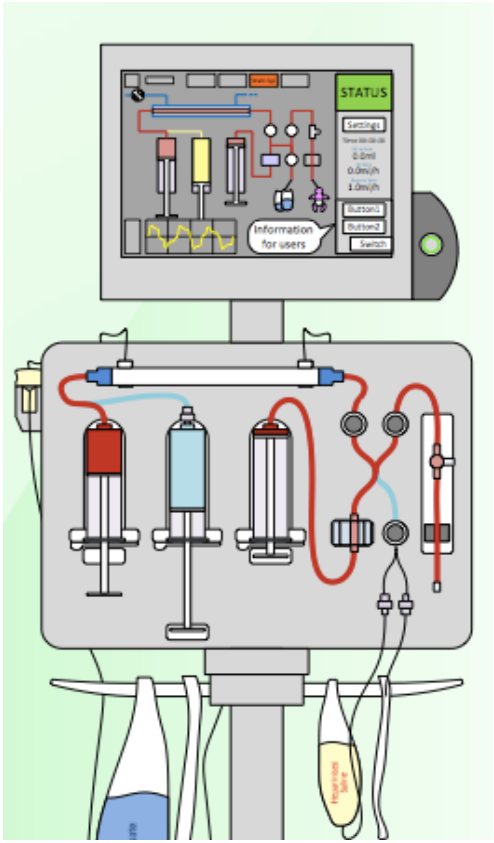
Design / Verification Tools - various

- Model-based verification languages and tools
  - CSP/FDR, MAL/NuSMV : risk assessment properties model checking
  - VDM/Overture, Isabelle : symbolic simulation and proof of specific properties
- Code-level verification languages and tools
  - Dafny/Z3, C/Frama-C : specification discovery
  - C and C++/eCv : MISRA compliance, auditable correctness proofs



Socio-technical experiments

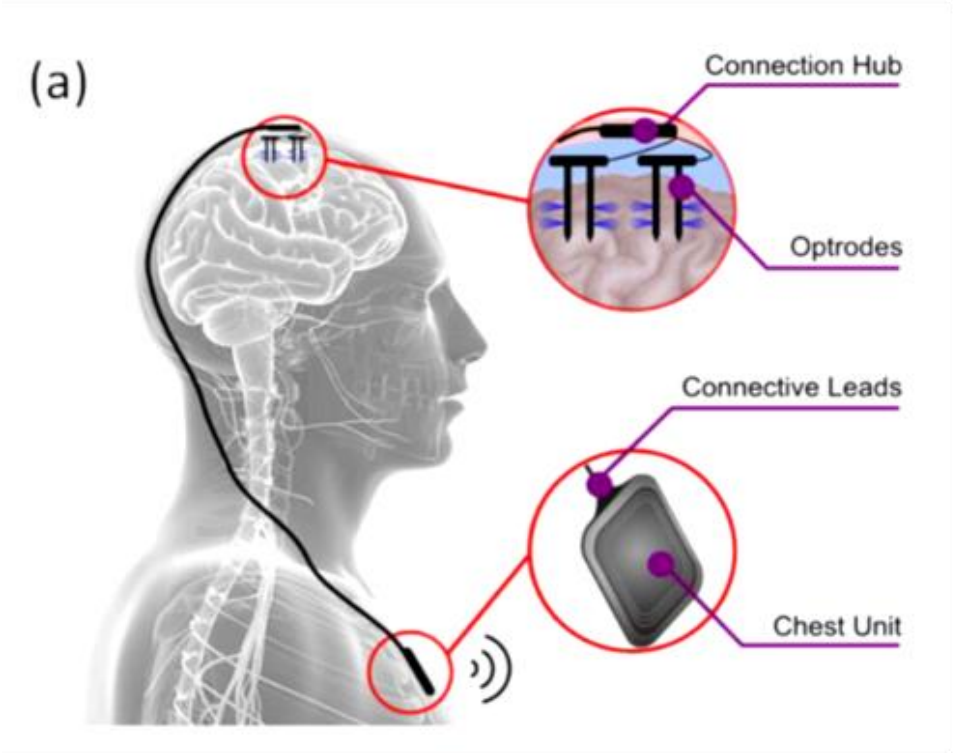
Infant Dialyser (NHS)
2008 – 2017 (2014)

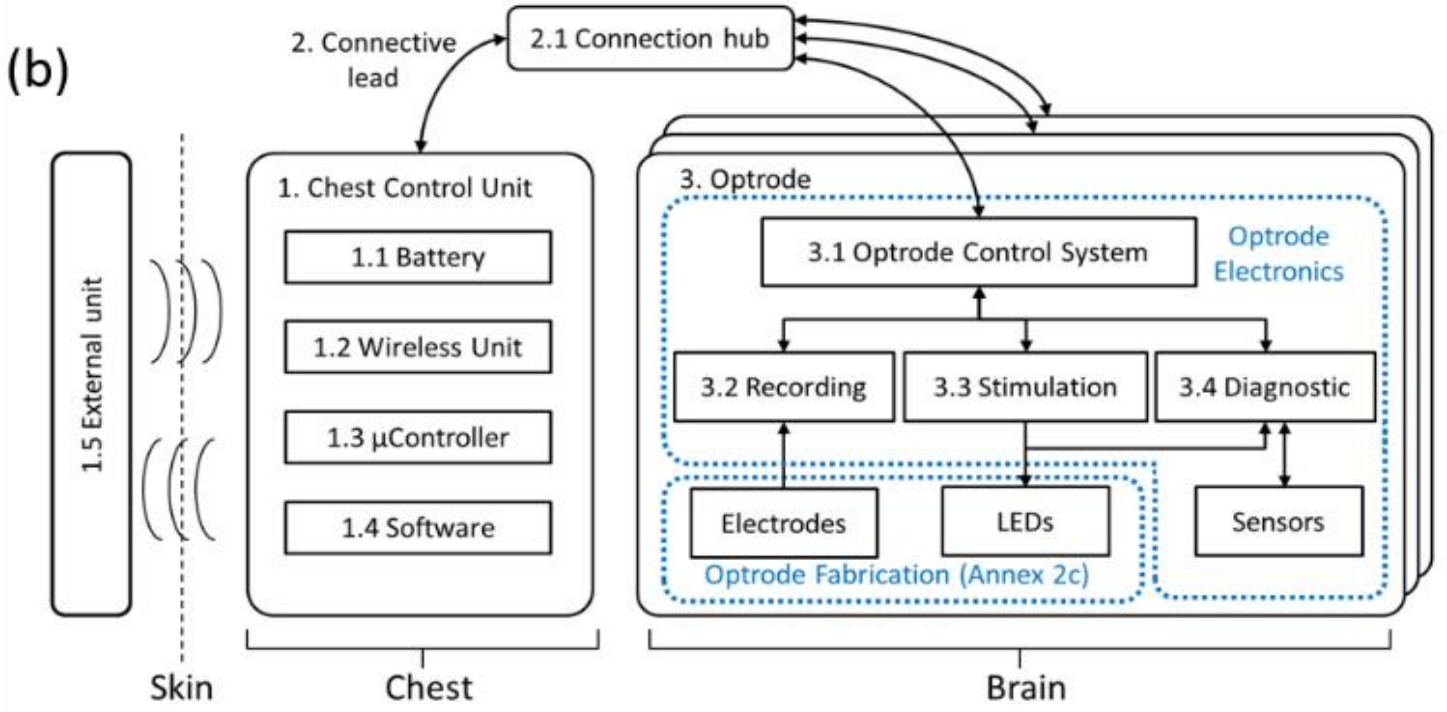


Brain Pacemaker (EPSRC/CANDO)
2014 – 2022 (2016)

Brain Pacemaker (mid point adoption)

- Optogenetic stimulation (i.e. electrical signal input; light signal output)
- Embedded software within the controller unit for closed-loop stimulation
- External software for wireless download diagnostic information and upload control parameters





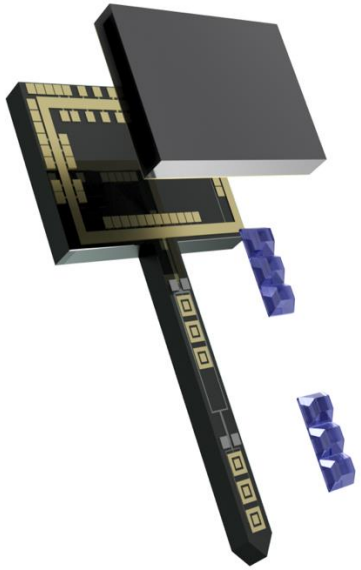
What's been verified?

- Verifying the optrode CMOS chip command interface API:
  - Optrodes communication over SPI (Serial Peripheral Interface)
  - FSM controlling low-level optrode functionality

- FSM control software implemented in C (~3KLOC) for a Freescale chip

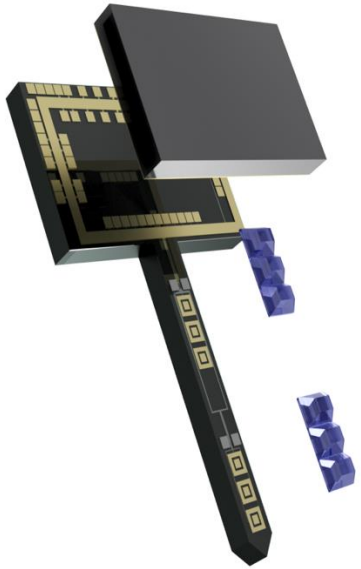
- Verification boundaries
  - Not verifying generated device-driver code (~5KLOC)
  - Not verifying dependant libraries for printed hardware (~110KLOC)

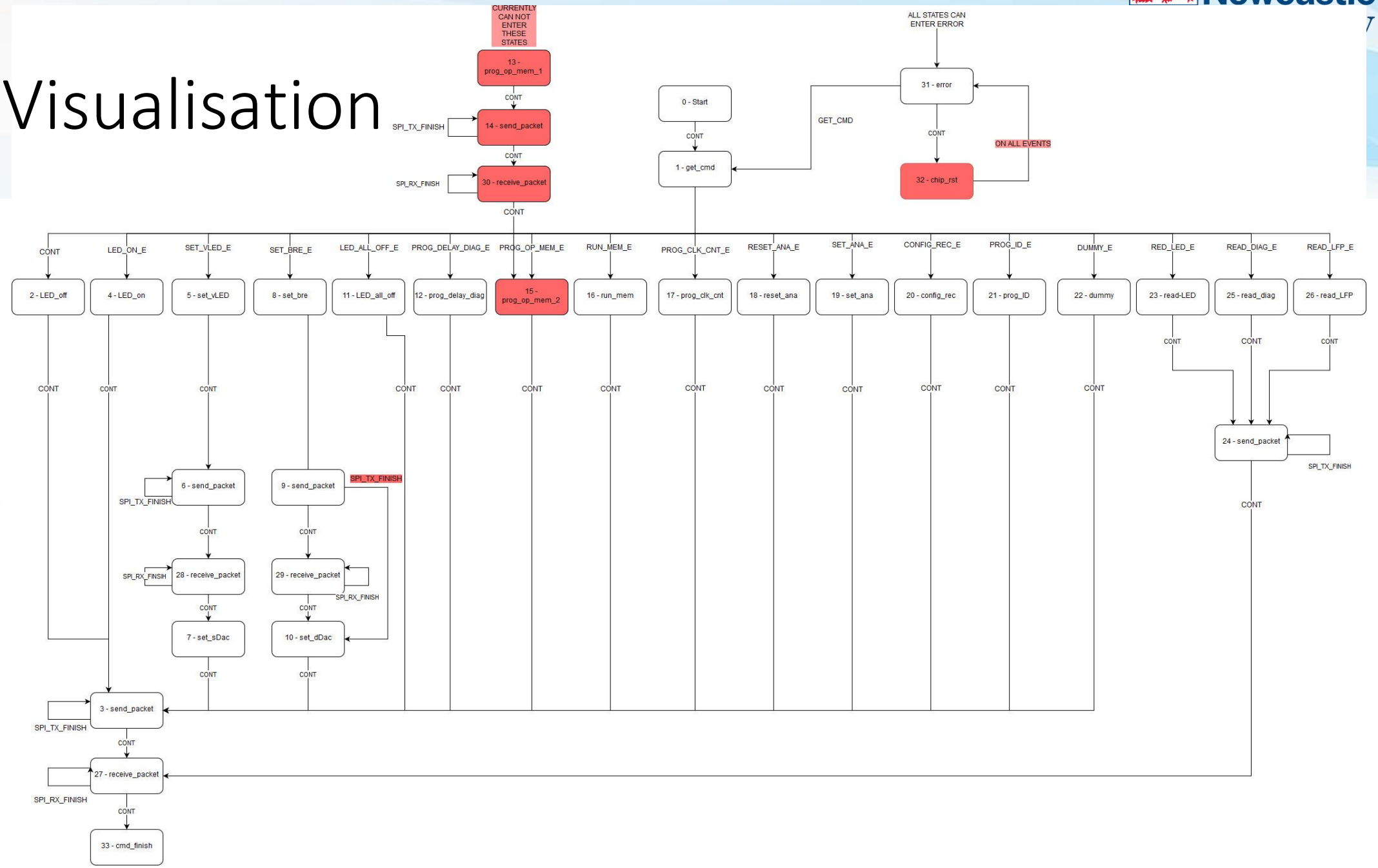
FSM Visualisation



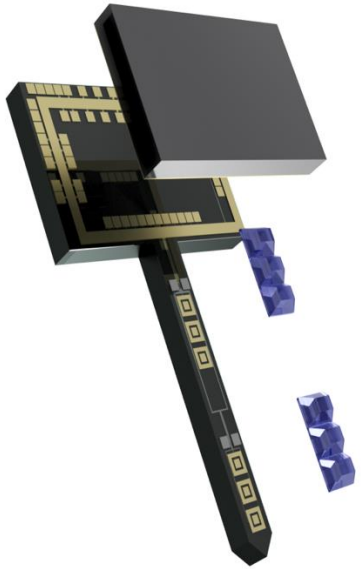
```
const unsigned char                 /* State transition table */
MANUAL_STATE_TABLE[OPTRODE_CMD_STATES][OPTRODE_CMD_EVENTS] =
        {//  TOUT  CONT    ERROR   SPI_TX_FINISH   SPI_RX_FINISH   LED_ON_E    SET_VLED_E  SET_BRE_E   LED_ALL_OFF_E   PROG_DELAY_DIAG_E
/* STATE 0  */  1,  1,      32,         0,              0,              0,          0,          0,          0,              0,          READ_DIAG   READ_LFP    GET_CMD
/* STATE 1  */  1,  2,      32,         1,              1,              1,          1,          1,          1,              1,          0,          0,          0,  // start
/* STATE 2  */  2,  3,      32,         2,              2,              5,          6,          9,          12,             13,         1,          1,          1,  // idle
/* STATE 3  */  3,  4,      32,         3,              3,              3,          3,          3,          3,              3,          26,         27,         2,  // get_cmd
/* STATE 4  */  4,  4,      32,         28,             4,              4,          4,          4,          4,              4,          3,          3,          2,  // LED_off
/* STATE 5  */  5,  4,      32,         5,              5,              5,          5,          5,          5,              5,          4,          4,          2,  // send_packet
/* STATE 6  */  6,  7,      32,         6,              6,              6,          6,          6,          6,              6,          5,          5,          2,  // LED_on
/* STATE 7  */  7,  7,      32,         29,             7,              7,          7,          7,          7,              7,          6,          6,          2,  // set_vLED
/* STATE 8  */  8,  4,      32,         8,              8,              8,          8,          8,          8,              8,          7,          7,          2,  // send_packet
/* STATE 9  */  9,  7,      32,         9,              9,              9,          9,          9,          9,              9,          8,          8,          2,  // set_sDAC
/* STATE 10 */  10, 10,     32,         30,             10,             10,         10,         10,         10,             10,         9,          9,          2,  // set_bre
/* STATE 11 */  11, 4,      32,         11,             11,             11,         11,         11,         11,             11,         10,         10,         2,  // send_packet
/* STATE 12 */  12, 4,      32,         12,             12,             12,         12,         12,         12,             12,         11,         11,         2,  // set_dDAC
/* STATE 13 */  13, 4,      32,         13,             13,             13,         13,         13,         13,             13,         12,         12,         2,  // LED_all_off
/* STATE 14 */  14, 14,     32,         14,             14,             14,         14,         14,         14,             14,         13,         13,         2,  // prog_delay_diag
/* STATE 15 */  15, 15,     32,         31,             15,             15,         15,         15,         15,             15,         14,         14,         2,  // prog_op_mem_1
/* STATE 16 */  16, 4,      32,         16,             16,             16,         16,         16,         16,             16,         15,         15,         2,  // send_packet
/* STATE 17 */  17, 4,      32,         17,             17,             17,         17,         17,         17,             17,         16,         16,         2,  // prog_op_mem_2
/* STATE 18 */  18, 4,      32,         18,             18,             18,         18,         18,         18,             18,         17,         17,         2,  // run_mem
/* STATE 19 */  19, 4,      32,         19,             19,             19,         19,         19,         19,             19,         18,         18,         2,  // prog_clk_cnt
/* STATE 20 */  20, 4,      32,         20,             20,             20,         20,         20,         20,             20,         19,         19,         2,  // reset_ana
/* STATE 21 */  21, 4,      32,         21,             21,             21,         21,         21,         21,             21,         20,         20,         2,  // set_ana
/* STATE 22 */  22, 4,      32,         22,             22,             22,         22,         22,         22,             22,         21,         21,         2,  // config_rec
/* STATE 23 */  23, 4,      32,         23,             23,             23,         23,         23,         23,             23,         22,         22,         2,  // prog_ID
/* STATE 24 */  24, 25,     32,         24,             24,             24,         24,         24,         24,             24,         23,         23,         2,  // dummy
/* STATE 25 */  25, 25,     32,         28,             25,             25,         25,         25,         25,             25,         24,         24,         2,  // read_LED
/* STATE 26 */  26, 25,     32,         26,             26,             26,         26,         26,         26,             26,         25,         25,         2,  // send_packet
/* STATE 27 */  27, 25,     32,         27,             27,             27,         27,         27,         27,             27,         26,         26,         2,  // read_diag
/* STATE 28 */  28, 32,     32,         28,             28,             28,         28,         28,         28,             28,         27,         27,         2,  // read_LFP
/* STATE 29 */  29, 8,      32,         29,             29,             29,         29,         29,         29,             29,         28,         28,         2,  // receive_packet
/* STATE 30 */  30, 11,     32,         30,             30,             30,         30,         30,         30,             30,         29,         29,         2,  // receive_packet
/* STATE 31 */  31, 16,     32,         31,             31,             31,         31,         31,         31,             31,         30,         30,         2,  // receive_packet
/* STATE 32 */  1,  1,      32,         32,             32,             32,         32,         32,         32,             32,         31,         31,         2,  // receive_packet
        };                                                                                                                              32,         32,         32  // cmd_finish

 void (*MANUAL_STATE_ADD[OPTRODE_CMD_STATES])(void) =
        {
/*0-3  */   &start,             &idle,              &get_cmd,           &LED_off,
/*4-7  */   &send_packet,       &LED_on,            &set_vLED,          &send_packet,
/*8-11 */   &set_sDAC,          &set_bre,           &send_packet,       &set_dDAC,
/*12-15*/   &LED_all_off,       &prog_delay_diag,   &prog_op_mem_1,     &send_packet,
/*16-19*/   &prog_op_mem_2,     &run_mem,           &prog_clk_cnt,      &reset_ana,
/*20-23*/   &set_ana,           &config_rec,        &prog_ID,           &dummy,
/*24-27*/   &read_LED,          &send_packet,       &read_diag,         &read_LFP,
/*28-31*/   &receive_packet,    &receive_packet,    &receive_packet,    &receive_packet,
/*32-35*/   &cmd_finish
        };
```

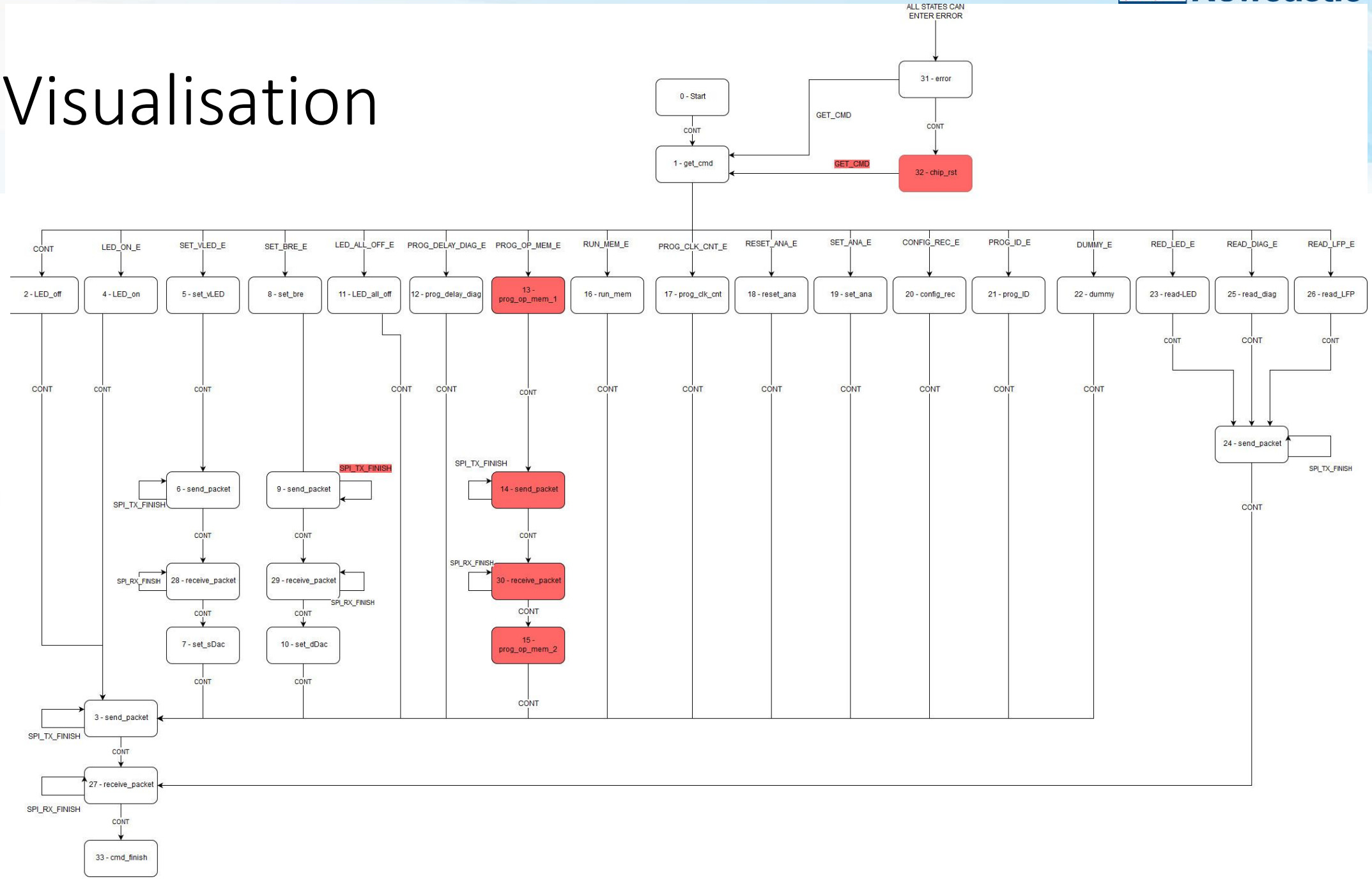
FSM Visualisation





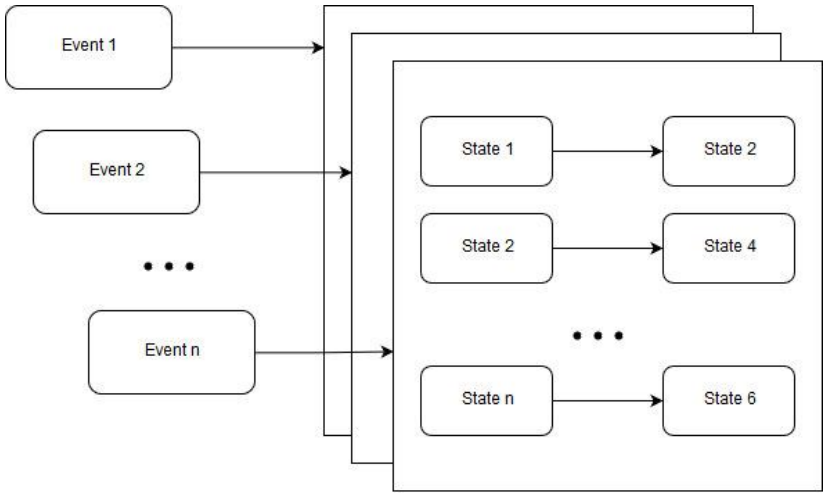
FSM Visualisation





FSM Verification

- Modelled the FSM a map from event to state to state
  - Abstracted away from the C implementation details
  - Invariants are not as fiddly as in C code annotations
  - Translated back to eCv annotations to verify the C code



- Modelling done in VDM
  - Symbolic simulation, 98% coverage, QC, Isabelle proof
  - 1053LOC VDM, 45-50 FSM invariants, 272POs
  - Original effort had 136 POs (2019 POG), 10 QC failed!

PO Type	Total (272)
Provable	93 (34%)
Maybe	147 (54%)
Unchecked	32 (12%)
Failed	0

- Implemented optrode CMOS chip command-set APIs
  - The model can simulate executions of the FSM loop
  - Independent of hardware, which is fiddly and hard to debug
  - Whole API simulation only one (main) command path proof.

Finite State Machine (FSM) Analysis

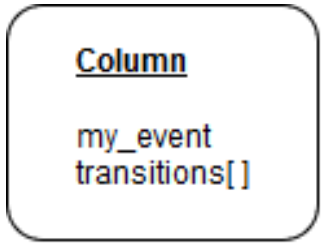
- FSM invariants led to rearrangement of VDM operations from C.
- Identified unreachable states in C + incorrect/inadequate transitions.
- Examples of identified invariants:
  - Send and receive states must transition to themselves on transmission finish events – ensure all bytes of the packets are processed
  - Packet creation states must go to a send state on a continue event – if they don't, the packet can be overwritten by another packet creation state.
  - Receive packet states must go to the command finished or a stage 2 packet creation state – command is either finished or a new packet must be created
- Simulated system satisfied these invariants.

eCv Verification of C FSM

Translated VDM to eCv annotations to verify the C code

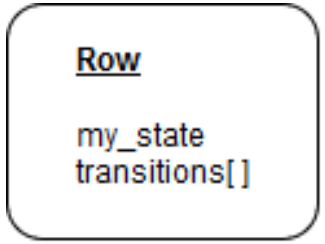
Verifying the FSM in C (2 versions):

- *Column struct*
  - invariants in terms of events
  - event -> (state -> state)




```
(value.my_event == SPI_TX_FINISH =>

        (forall x in all_states :-
                ((x in send_states_eCvset) =>
                        (value.transitions[x] == x))
                &&
                (!(x in send_states_eCvset) =>
                        (value.transitions[x] == error_s))
        )
)
```

- *Row struct*
  - invariants in terms of states
  - state -> (event -> state)




```
(check_if_send_state(value.my_state) =>

        ((check_if_receive_state(value.transitions[CONT]))
        &&
        (value.transitions[SPI_TX_FINISH] == value.my_state))
        &&
        (forall x in value.transitions.indices :-

                                (x != CONT && x != SPI_TX_FINISH) =>

                                        (value.transitions[x] == error_s )
        )
```

- FSM Table is then constructed as an array of columns or rows

FSM Data Packet Construction

- Several progressive iterations and steps to discharge C POs:
  - Remove errors produced during verification
  - Discharge further verification conditions
  - Simplify the verification process

Communication Packet - 24 bits		
Address: 6 bits	Command: 6 bits	Data: 12 bits

- Original package assembly contained own bit-vector expression, e.g.:

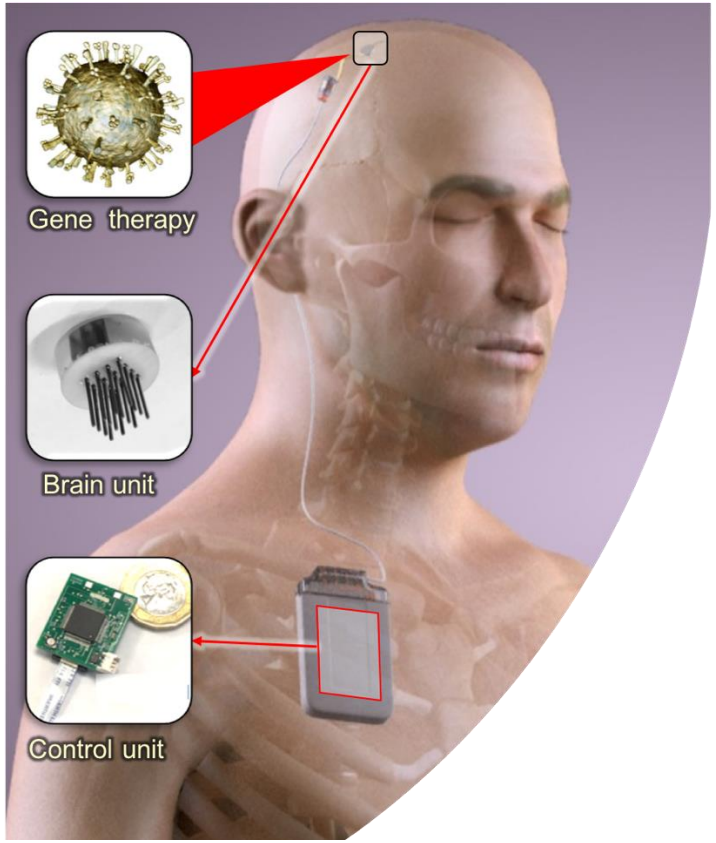
```
Packet = ((((addr & BITS_6) << 18) | ((cmd & BITS_6) << 12) | (data & MASK)) & BITS_24);
```

```
// Hardware instruction: binary package construction
inline packet_t assemble_packet(const bits_6_t addr, const bits_6_t cmd, const bits_12_t data)
                                ghost(const packet_data_t data_length_ghost)
pre  (range_check(DATA_LOW, data_length_ghost, data))
post ( result == (((addr << ADDR_SHIFT) | (cmd << CMD_SHIFT) | data ) & BITS_24))
{
        unsigned long packet_assembly =
                        (assemble_packet_addr(addr) |
                         assemble_packet_cmd(cmd)   |
                         assemble_packet_data(data) ghost(data_length_ghost));
        return (packet_t) (packet_assembly & BITS_24);
}
inline packet_t assemble_packet_addr(const bits_6_t addr) const
pre  (range_check(ADDR_LOW, ADDR_HIGH, (addr << ADDR_SHIFT)))
post (range_check(ADDR_LOW, ADDR_HIGH, result))
post (result == old addr << ADDR_SHIFT)
{       return ((packet_t)addr << ADDR_SHIFT);  }
```

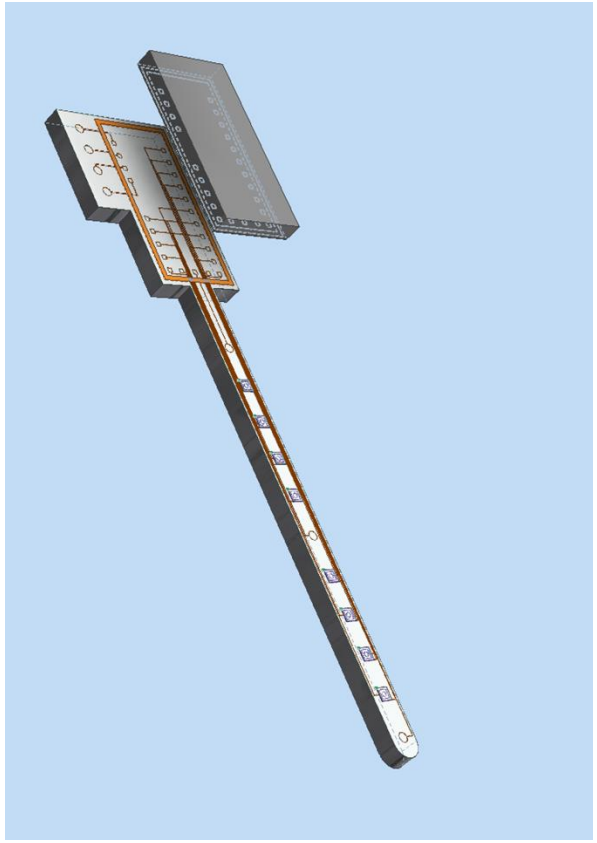
- Led to reduction from 64 to 8 C POs per packet construction
- Verification effort decrease (e.g. 6K to 460 C POs)
- Less POs due to lesser up/down casts

Timeline

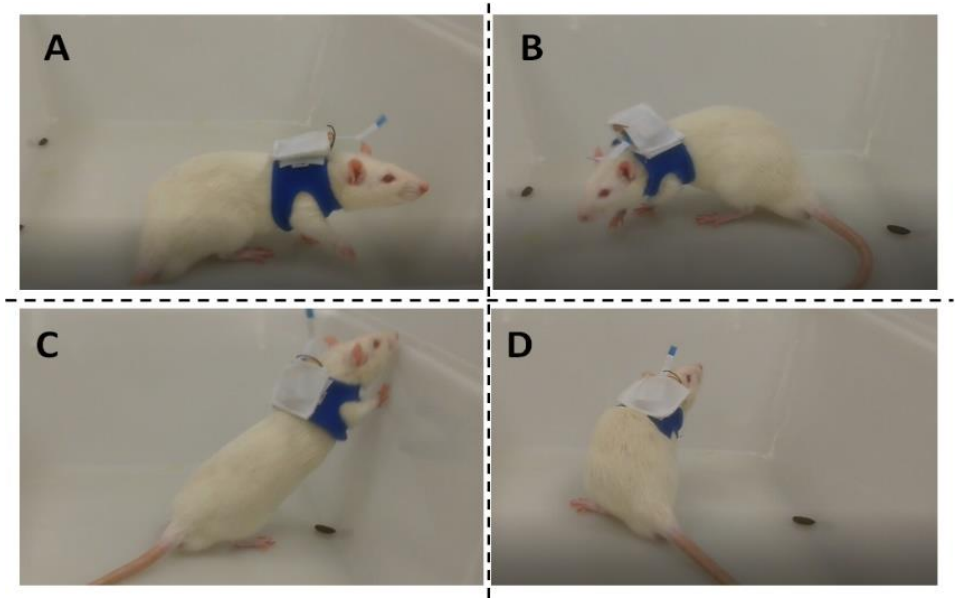
Inception
(2014-2015)



Design
(2014-2018)



Rodent Trials
(2017-2018)



Primate Trials
(2019)

Human Trials
(2020)

Conclusion

- Need to be careful not to be lost in translation
  - Model-based formalisms for the design (i.e. FSM + properties)
  - Code-level verification for C/C++ (partial/total correctness)
  - **VDM was crucial to identify hidden invariants from the C code**
  - **“Killer” C errors discovered early (using VDM and Isabelle)**

- Socio technical experiments
  - Early adoption brings more benefits
  - Important to have regulator/notified body on-side
  - **Link solutions to regulatory process is crucial**