Software for a Total Artificial Heart

Nils Brynedal Ignell
R&D engineer
+46 (0)70 730 65 33
nils.brynedal@realheart.se
Background

- Today, heart disease is one of the leading causes of death in the western world.

- About 50,000 patients worldwide are placed on a heart transplantation list annually.

- Shortage of donor hearts is causing many patients to die before transplantation.
Background

- Scandinavian Real Heart is developing a Total Artificial Heart (TAH)
- A Total Artificial Heart *replaces* the natural heart, unlike Ventricular assist devices (VADs)
- Initially intended for bridge-to-transplant therapy
- Currently in a *preclinical* stage, meaning:
  - Research and development
  - No formal safety requirements for the development process (for cost reasons)
  - No testing in humans, mostly lab testing, but animal testing for verification
Design features

- As realistic as possible
- Pulsating outflow
- Heart rate and stroke volume can be changed within realistic ranges
- Two halves that pump synchronously, each having:
  - a BLDC motor, controlled by
  - a STM32F4 ARM microcontroller
  - two valves (just as the human heart)
  - an atrium and a ventricle (just as the human heart)
Design overview

![Diagram of heart device with labels for Mechanical heart, Battery A, Battery B, and Control box on the front, and Front and Back views on the right side.]
Blood flow
The need for verification of the software

- You don’t want your heart to crash or hang
- There is no Ctrl + Alt + Del on the heart
- You’d rather not try the typical “turning it off and on again” trick

- You’d rather want it to work all the time...
Software verification via proof

- All software in written in Ada/SPARK
- Some properties have been proved statically using GNATprove, for example:
  - Correct data initialization and data flow
  - Correct program flow
  - No array access out-of-bounds
  - No error-prone features such as pointers
  - Settings (e.g. heart rate) guaranteed to be within bounds
- Stack usage verified with GNATstack
Examples of proven properties

- Specific type used for indexing an array type

```ada
type Phase_Type is (PhaseA, PhaseB, PhaseC);
for Phase_Type use (PhaseA => 0,
    PhaseB => 1,
    PhaseC => 2);

type Hall_Sensor_Values is array (Phase_Type) of Boolean;
```

- This allows SPARK to *prove* that every array access will be within bounds
  - This use of types can be done in Ada, it does not require SPARK
Examples of proven properties

```
procedure Update (This : in out PID_Controller_Type;
    Error : Float;
    Delta_Time : Time_Sec; -- Seconds since last update
    Output : out Float;
    P_Out : out Float;
    I_Out : out Float;
    D_Out : out Float) with

Global => null,
Depends => ((This, I_Out, D_Out, Output) => (Error, Delta_Time, This),
            P_Out => (Error, This)),
Pre  => (Delta_Time > 0.0);
```

- Verification of program flow:
  - This procedure will not depend on, nor affect, any globals
  - The output parameters This, I_Out, D_Out and Output shall depend on Error, Delta_Time and This
  - P_Out shall only depend on Error and This
- These properties will be proven by GNATprove
Examples of proven properties

- A type used to define heart rate is given a limited range
  - This use of types can be done in Ada, does not require SPARK

```ada
Max_Hearth_Rate : constant Unsigned_8 := 170;
Min_Hearth_Rate : constant Unsigned_8 := 15;

type Heart_Rate_Type is range Min_Hearth_Rate .. Max_Hearth_Rate;
for Heart_Rate_Type'Size use 8;
```

- Maximum or minimum heart rate is easy to change
Examples of proven properties

- Precondition: The buffer can’t be full before calling the procedure
- Postcondition: The buffer can’t be empty after calling the procedure

```ada
procedure Put (This : in out CAN_Buffer;
              Message : in  CAN_Message_Type) with
Pre  => not Full (This),
Post => not Empty (This);
```

- Pre and post conditions are *checked* in Ada,
In SPARK they are *proven* by GNATprove (if possible)
But what if the buffer is full?

- Handle it!
- Or don’t!

```vhd
procedure Push_To_Receive_Buffer (Message : CAN_Message_Type) is
    Full : constant Boolean := Receive_Buffer.Full;
begin
    -- Please note that nothing will be done if the Buffer is full,
    -- the message will be lost!
    if not Full then
        Receive_Buffer.Put (Message);
    end if;
end Push_To_Receive_Buffer;
```
Pros of software verification via proof

• Lesser need for software testing (properties are proved rather than tested)
• Lesser need for code review
• Errors detected earlier (during compiling or verification)
• Fewer errors later during system testing
• No “once in a blue moon” errors
• Only logical errors
  (i.e. it’s really *my fault* if the code doesn’t work... 😞)
Cons of software verification via proof
(Yes, there are some...)

• Risk of over-reliance on proof (not testing enough)
• Remember, no matter how good your tools are, you still need to do a good job!

• It’s harder to google your problems (the Ada/SPARK world is small)

• Steep up-front learning curve (you’re probably used to C/C++...)
• Initially time consuming to write contracts
• It is really hard to write contracts (other than trivial ones)
Advice for future users

• Learn to use your tools
  • Start with a small test project and play around with it, try all Ada/SPARK features with on it

• Begin with the end in mind
  • Start with a good software architecture from the beginning
  • Add contracts from the beginning

• Ask for help, don’t bang your head against the wall...
  • ...you can find better use for your head
  • ...and for the wall
Questions?

Nils Brynedal Ignell
R&D engineer
+46 (0)70 730 65 33
nils.brynedal@realheart.se

www.realheart.se/en
info@realheart.se