# Development of a Control System for a Nuclear Submarine

Alex Barker - 40333139

SET10112 - Edinburgh Napier University

**Abstract.** This document is a specification for the design of a control system for a nuclear submarine including a report containing the specification definitions and descriptions. A control system was designed and implemented using Ada-SPARK to include the following subsystem controls: operational status, airlock doors, airlock locks, life support levels, nuclear reactor temperature levels, diving depth levels and torpedo storing, loading and firing. The control system meets the requirements specification of the nuclear submarine and achieved a SPARK level of Gold. The system outputs a series of tests and conditions to a console where control system statuses and warnings are displayed so the user can determine if the system is operating safely.

## 1 Introduction

This report describes the process of designing and implementing the control system for a nuclear submarine. The aim of the report is to provide definitions, descriptions and formal proofs to demonstrate the correctness of the system using pre and post-conditions. The nuclear submarine must operate under the following conditions:

- The submarine must have at least one airlock door closed at all times.
- The submarine must have at least one airlock door closed at all times.
- If the oxygen runs out, the submarine has to surface.
- If the reactor overheats, the submarine has to surface.
- If the oxygen runs low, a warning must be shown.
- The submarine can't dive beneath a certain depth.
- The submarine must be capable of storing, loading and ring torpedoes safely.

To ensure the nuclear submarine operates according to the above specification, an application using the Ada programming language was created following the SPARK method of formal proofing. The control system meets the requirements specification of the nuclear submarine and achieved a SPARK level of Gold. The system outputs a series of tests and conditions to a console where control system statuses and warnings are displayed so the user can determine if the system is operating safely.

## 2 Controller Structure

The system consists of a series of procedures that control all the required aspects of each subsystem in order to meet the specification. The main subsystems are described below.

1. Airlock Subsystem - the doors of a submarine that can be *Open, Closed, Locked or Unlocked*. Each of the statuses has procedures that verify the state of the door and change the status of the door. To prevent the subsystem from failing the system specifications, procedures for preventing opening both doors were implemented.
2. Weapon System Subsystem - the weapons system first verifies that the submarine is operational by using the `WeaponsSystemCheck` procedure. The number of torpedoes available is controlled by a range type and procedures for storing, loading and firing control the usage of these torpedoes.
3. Diving Depth Subsystem - the position of the submarine in the water is controlled by procedures that track the submarines depth range. The subsystem provides warnings if the submarine is exceeding maximum depth and can surface the submarine in emergency conditions.
4. Life Support Subsystem - the oxygen levels within the submarine are tracked using a range type where the tanks are full at 100 and empty at zero. There are procedures that provide *Danger* warnings if the oxygen level drops below 20 where other subsystems can return the submarine to the surface.
5. Reactor Temperature Subsystem - the temperature of the submarine is controlled by a range type between zero and 250. The status of the nuclear reactor is either *Optimal* or *Overheating* where other procedures can return the submarine to the surface if the temperature reaches the maximum.
6. Submarine - the submarine is controlled by a record type which contains all subsystems and an `Operational` check.

The submarine control system is implemented in three files; `trident.adb, trident.ads and main.adb`. Global variables, range types and records are used to store and communicate information about the subsystems. The `main` file is responsible for initialising subsystems and outputting test cases to the console.

## 3 Descriptions of Procedures and Functions

This section will describe in more detail the responsibilities and functions each subsystem have.

### 3.1 The Airlock Procedures

The airlock subsystem is responsible for allowing the submarine to be in an `Operational` state or not. This is due to the requirement of both doors needing to be *Closed* and *Locked* before the submarine can be operated. This is achieved by a number of Boolean variables that control whether the airlock door is in a *Open, Closed, Locked or Unlocked* state. `AirlockDoorOne` and `AirlockDoorTwo` have a Boolean for *Closed* and *Open*. `AirlockLockOne` and `AirlockLockTwo` have a Boolean for *Locked* and *Unlocked*. The other condition that must be correct is that one airlock door must be closed at all times, therefore, procedures were put in place to prevent two doors being open at the same time. For example, the precondition for CLOSING airlock one is, `CloseAirlockOne` is *Open* and then `CloseAirlockTwo` is *Closed* with the post-condition `CloseAirlockOne` is *Closed*, whereas the precondition for OPENING airlock one is `LockAirlockOne` is *Unlocked* and then `CloseAirlockOne` is *Closed* and then `CloseAirlockTwo` is *Closed* with post-condition `CloseAirlockOne` is *Open*. This allows doors to be closed with weak preconditions but opened with stronger preconditions. The unlocking procedures also have stronger pre and post-conditions as the Operational state of the submarine depends on it.

### 3.2 The Weapon System Procedures

The weapon system procedures control whether the system is *Available* or *Unavailable* via a Boolean. The weapons system requires the submarine to be operational with available torpedoes for loading and life support and reactor temperature at *Optimal* before becoming available. This is achieved by the precondition of, operating is *Yes* and then `loadedTorpedoes` is *Loaded* and then `lifeSupport` is not *Danger* and then `reactorTemperature` is not *Overheating* with post-condition `WeaponsAvailablity` is *Available*.

The torpedoes are initialised in a range type between 0 and 5 and are controlled by the `Store, Load and Fire` procedures via the `TorpedoesStored` (Stored, Notstored), `TorpedoesLoaded` (Loaded, Notloaded) and `TorpedoesFiring` (Firing, Waiting) Booleans. A pre-check for a loaded torpedo can occur by checking the status of the loaded Boolean.

Torpedoes can be stored up to five by having the conditions: `Store`

```
Global => (In_Out => TridentSubmarine),
Pre => TridentSubmarine.WeaponsAvailablity = Available
and then TridentSubmarine.torpedoes < TorpedoesCount'Last,
Post => TridentSubmarine.torpedoes = TridentSubmarine.torpedoes'Old + 1
and then TridentSubmarine.storedTorpedoes = Stored;
```

Where attempting to store more than five will be blocked and a warning sent to console.

Torpedoes can be loaded one at a time by having the conditions: `Load`

```
    Global => (In_Out => TridentSubmarine),
    Pre => TridentSubmarine.WeaponsAvailablity = Available
    and then TridentSubmarine.loaded = False
    and then TridentSubmarine.torpedoes > 0,
    Post => TridentSubmarine.loaded = True
    and then TridentSubmarine.loadedTorpedoes = Loaded
    and then TridentSubmarine.torpedoes = TridentSubmarine.torpedoes'Old;
```

Where confirmation message is sent to console if loaded or not. Not being loaded is triggered by running out of torpedoes and loading removes a torpedo from storage.

Torpedoes can be fired one at a time by having the conditions: `Fire`

```
    Global => (In_Out => TridentSubmarine),
    Pre => TridentSubmarine.WeaponsAvailablity = Available
    and then TridentSubmarine.loaded = True
    and then TridentSubmarine.torpedoes > 0,
    Post => TridentSubmarine.loaded = False
    and then TridentSubmarine.firingTorpedoes = Firing
    and then TridentSubmarine.torpedoes = TridentSubmarine.torpedoes'Old
```

Where confirmation message of firing is sent to console and a torpedo is removed from `loaded`.

### 3.3 The Diving Depth Procedures

The Diving procedure simply checks whether the submarine is ready to dive and is instructed to dive between a `depthRange` of 0 to 1000 meters all after checking if the submarine is still `Operational`. The diving procedure is controlled by a *Yes/No* Boolean and the `depthPositionCheck` Boolean provides warnings and confirmations based on whether it is at `Surface`, `OptimalDepth` or `MaximumDepth`. The `EmergencySurface` functions are triggered by other procedures where `depthRange` is simply set to zero by `lifeSupport` or `reactorTemperature` warnings. Either one of these conditions must be true for depth range to be set to zero.

The `DepthAtSurface` pre and post-conditions check that `OptimalDepth` and `MaximumDepth` are not true along with the operating status of the submarine before setting it to `Surface`.

```
    Global => (In_Out => TridentSubmarine),
    Pre => TridentSubmarine.operating = Yes and then
```

```
    TridentSubmarine.depthPositionCheck /= OptimalDepth and then
    TridentSubmarine.depthPositionCheck /= MaximumDepth,
    Post => TridentSubmarine.operating = Yes and then
    TridentSubmarine.depthPositionCheck = Surface;
```

The `DepthAtOptimal` pre and post-conditions check that `Surface` and `MaximumDepth` are not true along with the operating status of the submarine before setting it to `OptimalDepth`.

```
    Global => (In_Out => TridentSubmarine),
    Pre => TridentSubmarine.operating = Yes and then
    TridentSubmarine.depthPositionCheck /= Surface and then
    TridentSubmarine.depthPositionCheck /= MaximumDepth,
    Post => TridentSubmarine.operating = Yes and then
    TridentSubmarine.depthPositionCheck = OptimalDepth;
```

The `DepthAtMaximum` pre and post-conditions check that `Surface` and `OptimalDepth` are not true along with the operating status of the submarine before setting it to `MaximumDepth`.

```
   Global => (In_Out => TridentSubmarine),
    Pre => TridentSubmarine.operating = Yes and then
    TridentSubmarine.depthPositionCheck /= Surface and then
    TridentSubmarine.depthPositionCheck /= OptimalDepth,
    Post => TridentSubmarine.operating = Yes and then
    TridentSubmarine.depthPositionCheck = MaximumDepth;
```

### 3.4 The Life Support Procedures

The Life Support subsystem procedures monitor the percentage amount of oxygen, `oxygenRange`, remaining in the submarine from a range of 0 to 100. `OxygenLevel` is of type range and the warning Booleans for `LifeSupportWarning` are *Safe, Warning, and Danger*. The *Safe* range for `lifeSupport` is anything between 100 and 21. The *Warning* range for `lifeSupport` is anything from 20 to 1 where a console message will be displayed. If the `oxygenRange` reaches zero then the `lifeSupport` warning will be *Danger*, which triggers `EmergencySurface`, where the submarine will surface to a `depthRange` of zero. The oxygen level can only be changed in the source code with this implementation. The ideal scenario would be for the percentage of oxygen to tick down for every hour the submarine is not at *Surface*.

### 3.5 The Nuclear Reactor Procedures

The Nuclear Reactor Temperature procedures monitor the changes in temperature in Celsius using a range type variable `reactorTemperature` between 0 and 250. The submarine will *Surface* if the `reactorTemperature` reaches 250 as the `ReactorWarnings` Boolean will become *Overheating* which is one of the conditions for `EmergencySurface`. The `reactorTemperature` will remain at *Optimal* between temperatures 0 and 249. The reactor temperatures can only be changed in the source code with this implementation. The ideal scenario would be for the temperature to increase as `depthRange` increases but remain constant when not diving up or down.

### 3.6 The Submarine Procedures

The Submarine is of record type and holds all the conditions of all subsystem procedures required to operate and make use of features. It is stored as follows:

```
 type Submarine is record
     operating : Operational;
     WeaponsAvailablity : WeaponsSystemAvailable;
     CloseAirlockOne : AirlockDoorOne;
     CloseAirlockTwo : AirlockDoorTwo;
     LockAirlockOne : AirlockLockOne;
     LockAirlockTwo : AirlockLockTwo;
     torpedoes : TorpedoesCount;
     loaded : Boolean;
     storedTorpedoes : TorpedoesStored;
     loadedTorpedoes : TorpedoesLoaded;
     firingTorpedoes : TorpedoesFiring;
     depthRange : Depth;
     depthPositionCheck : DepthPosition;
     oxygenRange : OxygenLevel;
     lifeSupport : LifeSupportWarning;
     reactorTemperature : ReactorTemp;
     temperatureWarnings : ReactorWarnings;
  end record;
```

And is initiated with the following values:

```
TridentSubmarine : Submarine := (operating => No, CloseAirlockOne => Open,
                CloseAirlockTwo => Closed, LockAirlockOne => Unlocked,
                LockAirlockTwo => Unlocked, WeaponsAvailablity => Unavailable,
```

```
                loaded => False, torpedoes => 0, storedTorpedoes => Notstored,
                loadedTorpedoes => NotLoaded, firingTorpedoes => Waiting,
                depthRange => 0, depthPositionCheck => Surface,
                oxygenRange => 0, lifeSupport => Safe,
                reactorTemperature => 100, temperatureWarnings => Optimal);
```

### 3.7 The Console Output

The Console Output is where tests can be carried out such as:

Here is the console output:

```
---------------------------------------------------
Is Nuclear Submarine Operational?: YES
Is Weapons System Available?: AVAILABLE
---------------------------------------------------
Is Weapons System Ready to Fire?: FALSE
Attempting to Store Torpedoes...
STORED Torpedo:  1
STORED Torpedo:  2
STORED Torpedo:  3
STORED Torpedo:  4
STORED Torpedo:  5

       ***WARNING - Attempted Torpedoe Storing Failure!***

NOTSTORED Torpedo:  5
---------------------------------------------------
Is Weapons System Ready to Fire?: NOTLOADED
Number of Torpedoes Stored:  5
---------------------------------------------------
Attempting to Load Torpedo...
Loading Torpedo: TRUE
Attempting to Fire Torpedo...
Is Weapons System Ready to Fire?: LOADED
FIRING Torpedo! Remaining  4
---------------------------------------------------
Attempting to Load Torpedo...
Loading Torpedo: TRUE
Attempting to Fire Torpedo...
Is Weapons System Ready to Fire?: LOADED
FIRING Torpedo! Remaining  3
---------------------------------------------------
```

**Fig. 1.** Console Output

## 4 Proof of Consistency

Storing, loading and firing of torpedoes is part of the verified Weapons System where the number of torpedoes available for storing is capped at the maximum range of `torpeadoCount`. While storing torpedoes it can be expressed as:

$$torpedo_{new} - torpedo_{old} = (torpedo + 1) - torpedo = 1$$

and when loading and firing a torpedo:

$$torpedo_{old} - torpedo_{new} = (torpedo - 1) - torpedo = -1$$

## 5 Conclusion

The specifications set out in Section 1 were all implemented in a rather sequential and long-winded manner. The more desirable software engineering method would be to split each subsystem into their own package/class. More formal methods could have been used to structure the flow of data such as invariants and arrays to simplify the code and prevent repeated code. Using SPARK to prove the implemented procedures produced *PutLine* warnings all the way from Gold to Stone, however, no other warnings or errors appear at any SPARK level. There was insufficient time to work out these warnings so the proposed SPARK level for this control system is Gold.

Future work can be to make use of object-orientated programming to split the subsystems into manageable packages to provide more modularity and reusability. This would allow functions to be implemented that can more readily accept user input that modifies that behaviour of the submarine.

A graphical user interface could be implemented to display the status and behaviour of the submarine using graphical representations of each subsystem. For example, a temperature dial with a needle that represents the current temperature within a colour range graphic where red denotes overheating, meaning the submarine must surface.
Another control system could involve time. Where the amount of oxygen left should limit the depth that the submarine can go based on how long it takes to dive or rise from certain depths within the maximum range of the nuclear submarine.