

UN

test

Hello



Title

Kent Odde, Stian Onarheim, Tarald Vestbøstad

October 23, 2020

Contents

1	Abstract	4
2	Introduction	4
3	Specifications	4
4	Technologies	6
4.1	The Ada Programming Language	6
4.2	Arduino Nano 33 BLE Sense	6
5	Implementation	6
5.1	Ada on the Arduino	6
5.1.1	Ada Drivers Library	6
5.1.2	Ravenscar Run-time Library	6
6	Maybe useful text	7
7	Ravenscar	9
8	Various problems we've encountered along the way	10
8.1	Pin layout	10
9	Maybe useful pictures	11
10	Conclusion	11
	References	12
11	Maybe appendix	13
11.1	Stacktrace	13
11.2	Arduino Nano 33 BLE sense PIN layout	14
11.3	Arduino Nano 33 BLE sense PIN layout	15

1 Abstract

This is our submission for the semester assignment in the course Real-Time Systems DSA3102, fall of 2020.

2 Introduction

The semester assignment as given, was to design and implement a real-time system, which would run on an Arduino Nano 33 BLE Sense, with the requirement that it would be programmed in Ada.

A broad definition of a real-time system, is a system that reacts to external stimuli, make calculations, and performs some output, within a given deadline. In other words, the challenge here is not only to write a logically correct program, but also that it finishes all tasks in time. Which naturally increases the difficulty.

This report will explain our project choice as well as the design. It will give a detailed description of the challenges, especially concerning running Ada on an Arduino Nano. Finally we will explain how the final solution works, and how we can prove that it is safe, and that the system will be able to meet all its deadlines.

3 Specifications

Very early on, we decided on building an autonomous car. We initially had a plan on using a hololens to interact with the car in some way, but we quickly moved away from this, in order to reduce the overall complexity.

The car should not only drive, but it would also have a bottle of antibac on top of it, with a sensor that would stop the car, and provide a person with a few drops of disinfecting liquid.

The initial design looked like this.

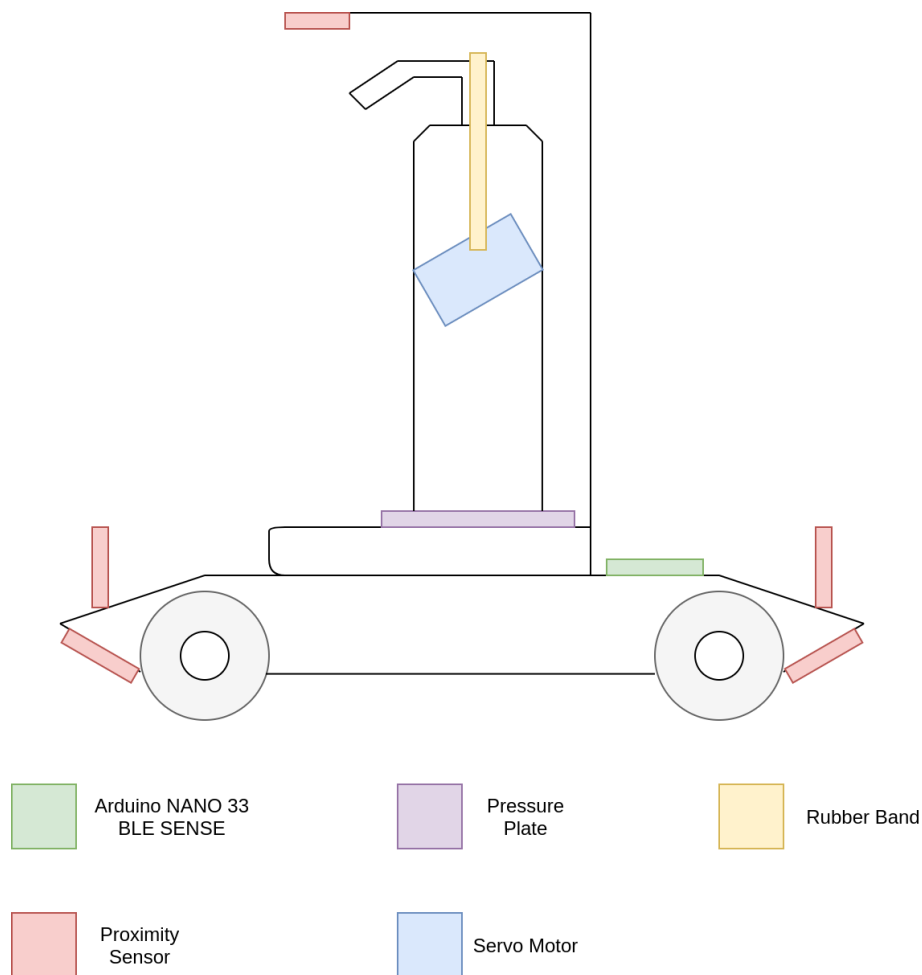


Figure 1: Prototype drawing of our vehicle

As the car would provide people with antibac, we naturally wanted the car to be able to drive on top of a table. That is why it had downpointing sensors, to be able to know when it would drive of an edge. However, after receiving the car from our professor, we quickly found, that the navigation system we would need to implement would be way to comprehensive, given the small timeframe of the assignment.

Throughout the project, the system went through several iterations, as as we met challenges along the way, the antibac idea was abandoned. The final design would look like this:

*****Diagram eller bilde av sluttdesign*****

4 Technologies

4.1 The Ada Programming Language

4.2 Arduino Nano 33 BLE Sense

5 Implementation

5.1 Ada on the Arduino

5.1.1 Ada Drivers Library

5.1.2 Ravenscar Run-time Library

6 Maybe useful text

Thursday 20. August 2020, the group gathered and had a meeting with Steven Bos. We discussed our project idea and the potential use of the Microsoft HoloLens 2. It was in the group's best interest to put our resources into the microcontroller rather than the goggles, dismissing any development with the HoloLens.

On September 10. the group met to plan the project. After deciding on the initial design, we drew a sketch and wrote a short description. This was sent to our professor, along with a list of required parts as well as the associated budget. The initial design can be seen in section XX.

On the 16th of September, we ordered three Arduino Nano 33 BLE Sense w/headers from Arduino.cc. This came to 1080NOK. In addition to this the professor provided us with a car for the project. It is called Turnigy Trooper [5], and was originally a radio controlled car. However everything but the battery and servo motors for thrust and steering, have been stripped away.

The servo motors are controlled by a pulse-width modulated signal. This means in essence that we send discrete signals, where the signal will rise at a fixed frequency. The length of time the signal is high before going low will determine the behaviour of the servo motors. Luckily for us there is a standard regarding the behaviour generated by a specific pulse width.

We will need to write a library which will abstract this away in the code. Where the interface for steering will take an angle, and the motor interface will hopefully be able to take a velocity in m/s, where negative numbers will mean going backwards.

After Steven showed us a proof of concept that it will be quite doable to base our library for the Arduino Nano on the existing toolchain for NRF52840, we started developing on the 17th of September. However there quickly emerged a problem, as Gnat arm-elf initially only supports ZFP-runtime libraries for this card. This meant that it would only support barebone functionality, and we would never be able to make use of the Ada realtime library, or even multithreading the software. Luckily, there exists support for full raven-scar run-time libraries for the card. We followed the official guide at the bb-runtime repo [3], to generate and build them ourselves.

```
./build_rts.py --rts-src-descriptor ~/opt/GNAT/2020-arm-elf/arm-eabi/lib/  
gnat/rts-sources.json --output=temp nrf52840
```

```
gprbuild -P ~/opt/GNAT/2020-arm-elf/lib/gnat/arm-eabi/lib/gnat/raven-scar-  
full-nrf52840/raven-scar-build.gpr
```

On the 18th of September we improved our library build, by letting the `project_wizard` script in Ada drivers library generate the gpr-files for our board automatically. All we had to do was to provide some basic information about the microcontroller, and the script did the rest. We also started adapting the pin mappings from the nRF52832 to the nRF52840.

Provided that our efforts lead to a useful library, we will create a pull request into the Ada drivers library, so that others may benefit from our work.

```
./Ada_Drivers_Library/scripts/project_wizard.py
```

October 5.

We received the Segger J-Link debugger from our professor. In order to get this to work, we installed the firmware from segger.com. However we also had to install a pack for pyocd in order to find the debugger:

```
pyocd pack --install stm32l476VG
```

As all development within the group is done on Linux, we also had to add udev rules, so that we were able to access the usb ports without being root. These rule-sets were found on pyocd's git repository.

We also had to give the respective user access to the serial port by running the command:

```
usermod -a -G dialout MY\_USER\_NAME
```

Legge til utilization test og grafikk.

October 12.

We are able to flash and debug by manually holding cables to the debug connections on the Arduino. We used the diagram on the segger website [4] to find out which JLink pins to "connect" to the Arduino points.

AnalogWrite seems like the perfect choice to generate a pulse for both servo and sensor. In the Ada Drivers Library [1] there is a very good example for controlling a servo. It changes the analog signal period with **Set_Analog_Period** to generate a steady pulse.

However we were never able to make it work on the arduino. Seems to be a problem with incompatibility between the drivers and the Arduino or the nRF52840 chip.

We suspect it is tied to interrupts since the stacktrace from the debugger kept stopping on a new line each time.

Progress is slow because of the manual labour required for each flash and debugging session.

October 19.

We recieved the clamps from Richard and Steven was able to put it together with the Arduino and a breadboard. We are finally able to easily flash and

debug.

We were able to create a steady pulse using digital I/O and tasks with a fixed priority. With a proper priority we were able to use three tasks for three separate servos and one task for three sensors.

We encountered some problems using a 9V battery power source, as the Arduino requires at least 5V and the peripherals need 5V.

October 21.

We wrote the code for the entire system.

October 22.

This was the first day we tested our servo code on the car. After a lot of trial and error on the steering we quickly moved on to the drive motor. We got the wheels to move fairly quickly after discovering that the motor needs a neutral (1500ms PWM) signal to initialize, before giving it signals which indicate torque. However, there quickly emerged problems. First of all the minimum speed was much too high, and secondly, it was impossible to get the car to switch between going forwards and backwards. After conversations with our professor, we learned that Joakim Bjork, had worked a lot with this car, and sat on a lot of documentation for the motors. This documentation would be made available for us on the next day.

October 23.

D-Day (Documentation day)

7 Ravenscar

The full raven-scar run-time library offers what is called the Ravenscar profile. This is a subset of the Ada language, where the features and available libraries are limited. The reason for this is to ensure robust software on realtime and safety critical systems.

The specifications of the raven-scar profile are the following: <https://www.ada-switzerland.ch/rm/RM-D-13.html>

There are a couple of limitations here that will affect the way we initially planned our software.

The maximum number of entries in tasks are set to 0. This means that we won't be able to have dynamic tasking, and all tasks will rather have to be defined at compile-time.

Another thing we planned was to specify which tasks would run on the respective cores of the CPU. This however, is not supported within the raven-scar profile, and we will need to do all the scheduling as if we only had a single-core processor.

8 Various problems we've encountered along the way

By the time you're reading this, some of these issues might have been fixed. Down below are some problems we have encountered during our development phase from 17.02.2020 - ???. Most of these issues have occurred using Linux, it is stated otherwise.

- Flashing to the Arduino board without a debugging probe might be impossible.
- After installing software for the Segger J-Link debugger from their official webpage, a new `.rules` file for the debugger appears in `/etc/udev/rules.d` containing the necessary information for the debugger to be found by `pyocd list`. This has only been successful with one of our computers. A workaround is to run `pyocd` with `sudo` privileges.
- Including unused libraries in the Ada source-code makes the arduino crash under run-time.
- When trying to flash to the board without having the cables soldered, errors in the list below have appeared as a consequence of unstable hands.
 - Unable to start CPU core.
 - Target System has no power.
 - Unspecified Error.
 - J-Link is already open.
- Including `Arduino_Nano_33_Ble_Sense.Time` makes the microcontroller crash in real time. –ref til Debug lstlisting i appendix–
- Only Digital I/O works. Possibly because Analog relies on interrupt/-timers which seem broken in our drivers or runtime.
- Generating a pulse accepted by the servos and ultrasonic sensor is more challenging than setting pins to low/high and using a delay.
- Using a copied `bb-runtime` library will not work correctly on other computers under debugging. They must be generated on every new computer.

8.1 Pin layout

11.10.2020

The nRF52840 has two ports, P0 has 32 pins while P1 has 16 pins.

We found the P1 base address for the nRF52840 seen on page 23 in the datasheet [8]. Without it we were only able to use the pins with P0 seen on figure 11.3. After adding the base address to the Ada code and the option to chose between 0 and 47 pin number we were able to make all the pins function.

9 Maybe useful pictures

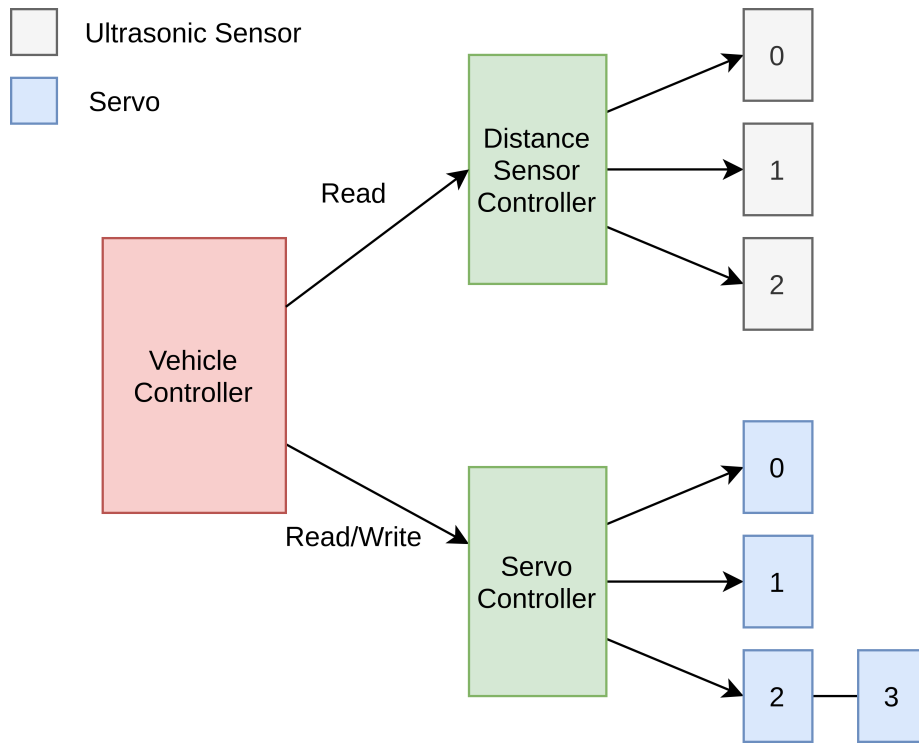


Figure 2: Block diagram of our system architecture

10 Conclusion

The purpose of this assignment was to try and implement the theory relating to real-time systems in practice, in order to better understand it. In spite of the challenges along the way, and how we had to downscale the project several times, we would still say that this goal was accomplished.

One can read about the difficulty of making a system meet all its deadlines, how scheduling works and what happens when a thread is starving, nothing can beat the feeling of actually experiencing it. As mentioned, we had several tasks that didn't deliver the output we expected, and many times this was due to the task not getting any time on the CPU. That is a powerful experience.

Learning to program in Ada along the way, as well as building the the raven-scar run-time library and the library for the NRF52840, was great, is guaranteed to prove very useful knowledge in the future.

References

- [1] Ada_Drivers_Library, github.com. https://github.com/AdaCore/Ada_Drivers_Library. Accessed: October-20.
- [2] Arduino Nano 33 BLE Sense, arduino.cc. <https://store.arduino.cc/arduino-nano-33-ble-sense-with-headers>. Accessed: October-20.
- [3] bb-runtimes, github.com. <https://github.com/AdaCore/bb-runtimes>. Accessed: October-20.
- [4] JLink Arduino Nano setup, segger.com. https://wiki.segger.com/Arduino_Nano_33_IOT. Accessed: October-20.
- [5] turnigy trooper, hobbyking.com. https://hobbyking.com/en_us/turnigy-trooper-sct-4x4-1-10-brushless-short-course-truck-arr.html?__store=en_u. Accessed: October-20.
- [6] Alan Burns and Andy Wellins. *Analysable Real-Time Systems, Programmed in Ada*. Createspace Independent Publishing Platform, 2009.
- [7] Henning Gundersen. DSA3102 slides, canvas. <https://usn.instructure.com/courses/22239/files>. Accessed: October-20.
- [8] Nordic Semiconductors. *nRF52840*, 2 2009. v1.1.

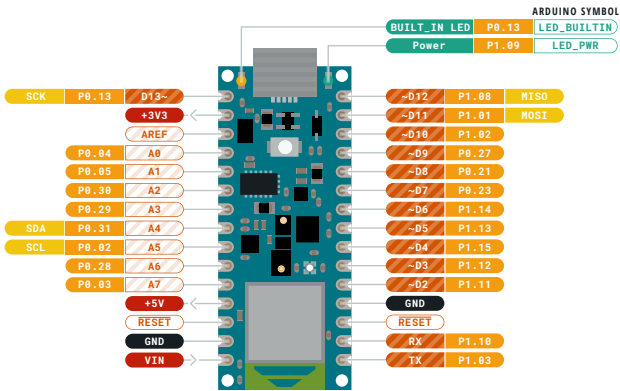
11 Maybe appendix

11.1 Stacktrace

11.2 Arduino Nano 33 BLE sense PIN layout



ARDUINO
NANO 33 BLE SENSE
STORE.ARDUINO.CC/NANO-33-BLE-SENSE



- Ground
- Power
- LED
- Internal Pin
- SWD Pin
- Digital Pin
- Analog Pin
- Other Pin
- Microcontroller's Port
- Default

- MAXIMUM** output current per pin is 15mA
- MAXIMUM** input current per pin is 5mA
- MAXIMUM** external current is 25mA for the sum of all GPIO currents and the current being drawn from VDD
- VIN** 5-21 V input to the board.

ARDUINO . CC
Last update: 16/09/2020
CC BY SA
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1868, Mountain View, CA 94042, USA.

11.3 Arduino Nano 33 BLE sense PIN layout



ARDUINO
NANO 33 BLE SENSE
STORE.ARDUINO.CC/NANO-33-BLE-SENSE

