NTNU
Norwegian University of
Science and Technology

# Real-Time System Implementation on Autonomous Lego-Robot

Konstantino Zuraris Helders

Master of Science in Cybernetics and Robotics
Submission date: July 2017
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

# Problem Statement

A working implementation of a robot system that will navigate and chart an unknown maze is given. The system is realized with an AVR microcontroller running FreeRTOS. The robot should be able to co-operate autonomously with other robots using a simultaneous localization and mapping algorithm, which is run from a central server.

This thesis describes the process of defining and building a common hardware abstraction layer which will be used in a team of autonomous robot systems along with the AVR. The hardware layer should define an interface used to connect the hardware to FreeRTOS.

The hardware used in this thesis is the Lego Mindstorms EV3 which has a Texas Instruments AM1808 MCU. When the hardware layer is complete, the drivers for sensors specific to the EV3 are to be implemented on top of FreeRTOS along with the SLAM application from the AVR.

Defining a common hardware abstraction layer will allow new devices to use the same interface in order to decrease the amount of work needed to port the final RTOS and application to new systems.

The work can be summarized as:

- Create a hardware abstraction layer API that will interface with FreeRTOS.
- Write IO drivers such that sensors and features specific to the EV3 system are accessible in FreeRTOS.
- Port the application running on the AVR to the EV3.
- Test the final system and decide if the resulting system performs satisfactorily.

# Abstract

A new hardware abstraction layer was created for the EV3. This includes drivers for UART, timers, interrupt controller, SPI and I2C peripherals. This was documented using Doxygen; an in-code documenting tool. Doxygen was used to create an off-line reference manual for the abstraction layer as a PDF (generated with $\LaTeX$).

Using the new drivers for the timer and interrupt controller, FreeRTOS was implemented on the EV3 using the timer interrupt as a RTOS tick interrupt for the FreeRTOS scheduler. A new port file for the RTOS was created that is compatible with the Texas Instruments AM1808 MCU in the EV3, and Doxygen was then configured to extract the FreeRTOS code structure such that the RTOS layer interface documentation was added to the reference manual.

The robot was built using a previously built AVR-based robot as reference. *EV3 Large Servo Motors* were used for wheel movement and to control the sensor tower. Given the remaining sensor ports on the EV3 only fits four sensors and there is no access to the GPIO pins in the AM1808 in the EV3 directly, there is not enough room for the remaining sensors to be implemented on the robot without an additional hardware card to interface with the sensors. The sensors needed for the SLAM application are four IR sensors, one gyro, one electronic compass and a BLE dongle. Plans were made to use a card developed for a similar problem on a Lego NXT robot, but due to time constraints, the card was not produced in this thesis.

Finally, the application was ported to the EV3 and with FreeRTOS running almost no changes to the code was needed to get implemented, only a few minor EV3 specific hardware calls to the motors needed to be changed.

# Sammendrag

Et nytt hardware abstraksjonsnivå ble laget til Lego EV3 roboten. Dette inkluderer drivere for UART, timere, interrupt kontrollere, SPI og I2C enhetene. Dette ble dokumentert ved å bruke Doxygen. Doxygen er et dokumentasjonsverktøy for kildekode hvor utvikleren kan skrive dokumentasjon direkte i koden som kommentarer med en egen Doxygen syntaks. Dette ble brukt til å generere en off-line referansemanual i PDF (formatert med $\LaTeX$).

De nye driverene for timer og interruptkontrolleren ble brukt til generere et interrupt ved en gitt frekvens på AM1808 mikrokontrolleren. Dette ble brukt som en tikk-interrupt kilde i FreeRTOS sin ressursplanlegger. En ny porteringsfil for FreeRTOS tilpasset AM1808 mikrokontrolleren ble skrevet slik at operativsystemet fungerer med EV3. Doxygen ble så konfigurert til å hente ut kodestrukturen til FreeRTOS slik at dette og ble dokumentert i den samme referansemanualen som hardware abstraksjonsnivået.

Roboten ble bygget opp ved å bruke en tidligere bygget AVR robot som referansepunkt. *EV3 Large Servo Motors* ble brukt til å rotere hjul for bevegelse, samt installert sammen med et sensortårn på toppen av roboten slik at denne kan justeres frem og tilbake. Siden EV3 roboten kun har 4 sensorporter og SLAM applikasjonen trenger å koble opp 4 avstandssensorer, 1 gyro, 1 kompass og 1 blåtannadapter så var det ikke nok plass til å installere disse direkte på EV3. Kretskortet på EV3 har ikke tilgjengelig GPIO pinnene til AM1808 mikrokontrolleren annet enn de som er tilgjengelig via portene så disse kunne ikke brukes direkte. Planer ble lagt for å bruke et ytterligere kretskort laget for NXT roboten til å få koblet på de resterende sensorene, men grunnet tidsfrister ble det ikke prioritert tid til å starte på den faktisk produksjonen av kortet.

Til slutt ble SLAM applikasjonen overført til EV3-prosjektet. Siden FreeRTOS nå kjører på EV3 var det svært lite endringer som trengtes å gjøres på applikasjonen, kun noen hardware kall som var spesifikke for EV3 roboten.

# Preface

This master's thesis builds on a project first started in 2004, and has been further developed in various specialization projects and master's theses by different students since then. The final goal of the project is to navigate and map an unknown environment using a team of co-operating, autonomous Lego-robots.

When starting work on this project back in the fall of 2016 I must admit I did not know what I was getting myself into. With a background from autonomous navigation and vessel control, I had very little experience with embedded software development. The project I understood to be development of the SLAM application for the robots turned out to be a very low level OS implementation when the RTOS needed for the application did not exist for the AM1808 microcontroller in the Lego EV3. Because of this a big part of my work has been to learn about low-level driver implementations, hardware abstractions and OS development.

Looking back at the work now I can say I am very glad it turned out this way. I have learned a lot of what really happens closer to the hardware than the code software developers usually write. I feel the knowledge and experience I have gained this semester has made me a better engineer, and I have to thank Tor Onshus, my supervisor through all of this, for guiding me through it when I had no idea where to go or what to do next.

I also want to thank Jo Arve Alfredsen, one of the professors at NTNU. As someone who has experience working on Texas Instruments microcontrollers in the past, his suggestions and advice proved valuable when developing the hardware layer of the AM1808 microcontroller.

Finally, a big thanks to Real Time Engineers Ltd and Jernej Kovačič for their guidance in porting FreeRTOS to the AM1808 microcontroller.

*Konstantino Z. Helders, July 2017, Trondheim*

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | | |
|---|---|---|
| NTNU | = | Norwegian University of Science and Technology |
| RTOS | = | Real-time operating system |
| SLAM | = | Simultaneous localization and mapping |
| MCU | = | Microcontroller unit |
| CCS | = | Code Composer Studio |
| TI | = | Texas Instruments |
| IDE | = | Integrated Development Environment |
| CPU | = | Central processing unit |
| ISR | = | Interrupt Service Routine |
| SWI | = | Software Interrupt |

# Chapter 1

# Introduction

## 1.1 Background and Motivation

A fundamental challenge within robotics is the cooperation of multiple robots to perform certain tasks. Just as robots have improved efficiency when performing individual assignments, enabling them to cooperate will further enhance the system as a whole.

In 2004 a project was started at the Norwegian University of Science and Technology where the goal was to build a Lego robot using the LEGO Mindstorms RXC system. By taking advantage of the hardware within the system brick and compatible sensors, a simultaneous localization and mapping algorithm was written to enable the robot to navigate an unknown maze. This project was the first iteration of what would later be a series of student projects and master theses at NTNU, each bringing improvements to the original system. The end-goal is to create a team of autonomous robots successfully navigating an unknown environment while cooperating using the same software. The system should be modular, well documented and easily transferred to new robots.

## 1.2 Previous Work

The RXC robot was initially built by Skjelten (2004) and later upgraded with an AVR microcontroller. A second robot using a LEGO Mindstorms NXT brick was introduced by

Bakken (2008). A third robot using an Arduino board was built by Ese (2016). The complete process from 2004 to 2014 is summarized by Homestad (2013) and Ese (2016).

The need of an RTOS was evaluated by Ese (2015) as everything required for the SLAM application can be programmed as non-OS specific, sequential tasks. However, Ese (2015) came to the conclusion that as the SLAM application continues to grow, the complexity of adding an RTOS will eventually be less than the complexity of writing a growing SLAM application as a non-OS specific, sequential task.

A real-time operating system, FreeRTOS, was implemented on the AVR and Arduino robot by Ese (2016) using a port file found online at feilipu (2011), and the SLAM application running on the AVR was re-written to take advantage of this.

The LEGO EV3 was first introduced by Steuper (2015) using an official MATLAB support package, but because of a problem with the IR sensors, the implementation was never finished.

The EV3 implementation was continued by Helders (2016), and FreeRTOS was ported from the AVR robot to run on the AM1808 MCU used by the EV3 using a virtual emulation software called Qemu. Helders (2016) discovered that the RTOS implementation contained a bug which will have to be found and fixed before implementing the rest of the system.

## 1.3 Goals

The AVR robot is currently running FreeRTOS with a real-time application which treats independent functions as separate threads; this enables the robot to multi-task more efficiently when running the SLAM application.

In order to get similar performance on the EV3 robot, FreeRTOS will have to be to be successfully ported to its AM1808 MCU. The FreeRTOS implementation on the EV3 has a bug which is yet to be solved. Since similar functionality to the AVR is wanted for the EV3 and the system should be as easy as possible to port to new robots, the following goals were outlined:

- Finish the FreeRTOS implementation on the EV3.
- Create a hardware abstraction layer API that will interface with FreeRTOS and document the source code.

- Write IO drivers such that sensors and features specific to the EV3 system are accessible in FreeRTOS and for the user.
- Port the application running on the AVR to the EV3.
- Test the final system and decide if the resulting system performs satisfactorily.

Completing the above goals should enable the EV3 to match the AVR in functionality and performance, as well as create a solid foundation for porting the software to additional systems.

## 1.4 Limitations

The work in this project limits to research, programming and physical alterations on the EV3 robot. A general document for a creating a hardware abstraction layer for other MCUs will be created. The software on the other robots or the software running on the central computer with the SLAM algorithm, will not be changed or improved here.

## 1.5 Contents of the Report

This report is written in LaTeX. All references, cross-references, figures and tables are hyperlinked. Formatting follows the convention given by Orakeltjenesten, NTNU IT (2013) for items not exclusive to Microsoft Word including, but not limited to, fonts, size and line spacing.

This thesis has 8 chapters, following the IMRaD style and format.

Chapter 2

# Inspection of Current System

## 2.1 Current hardware setup

The EV3 ”robot” currently consists of the MindStorms EV3 brick, a soldered-on JTAG connection with an ARM20 pin-out connected to a Texas Instruments XDS200 debugger for flashing and emulation, and a custom UART to USB serial cable built using the Silicon Labs CP210x USB to UART chip.

PICTURE

## 2.2 Current Development Process

Currently the development process for the EV3 system is:

1. Write source code using a text editor of choice.
2. Manually create and update a makefile for compilation.
3. Use a custom toolchain generated by Helders (2016) to cross-compile the source code for the target ARM9 architecture.
4. Transfer the binaries to the EV3 system through a serial connection.
5. Debug the code using prints in the code to the serial connection.

This is a slow process and, given the task of porting an operating system to new hardware, will take too long with the time available. Helders (2016) soldered on a JTAG connection and included a Texas Instruments XDS 200 debugger during the fall of 2016. It was suggested by Jo Arve Alfredsen, a professor at NTNU and course coordinator of the subject Embedded and Industrial Computer Systems Design to discard the current work process and try to find a complete integrated development environment with support for the AM1808 MCU in order to speed up development.

## 2.3 The EV3 JTAG Interface

Helders (2016) connected a JTAG debugger to the Texas Instruments AM1808 MCU. Since Lego has made it challenging to connect a debugger to the EV3 system, and the guide created by Helders (2016) was missing some information about the final header pins which might cause confusion, a complete guide has been added below:

1. Remove the battery cover of the EV3 and loosen and remove the 4 screws there as shown in figure 2.1.

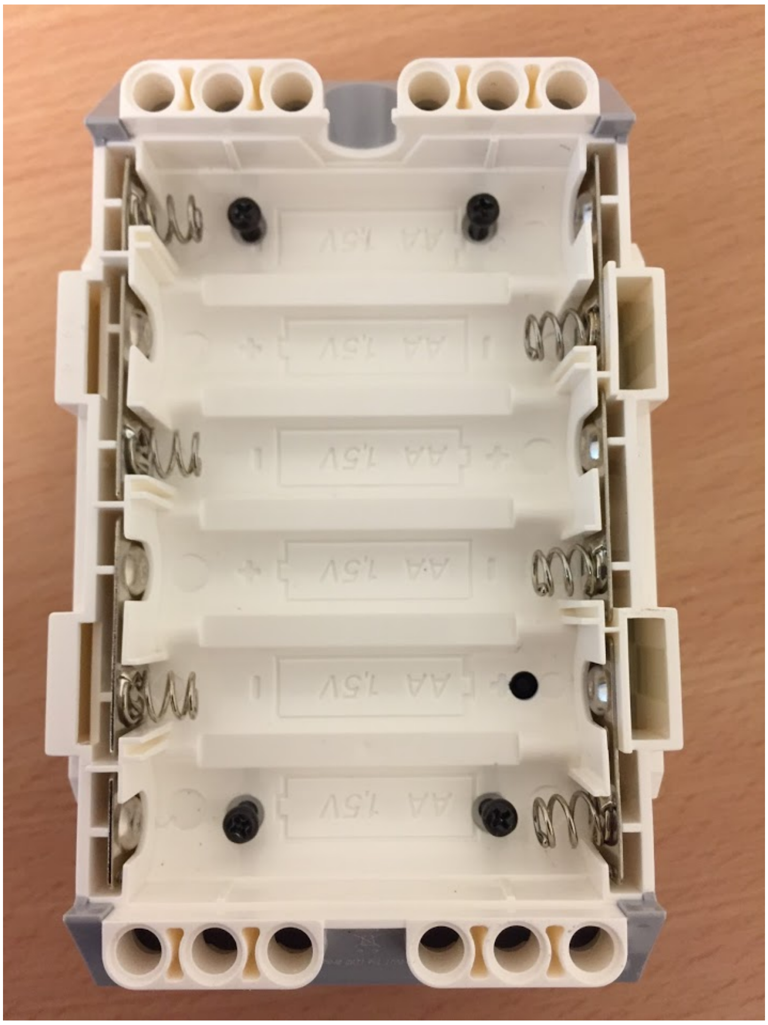


**Figure 2.1:** Removing the screws

2. Carefully open the EV3, taking care not to rip out the connector to the LCD screen. The plastic buttons will be loose and fall out so take care not to lose any of them.

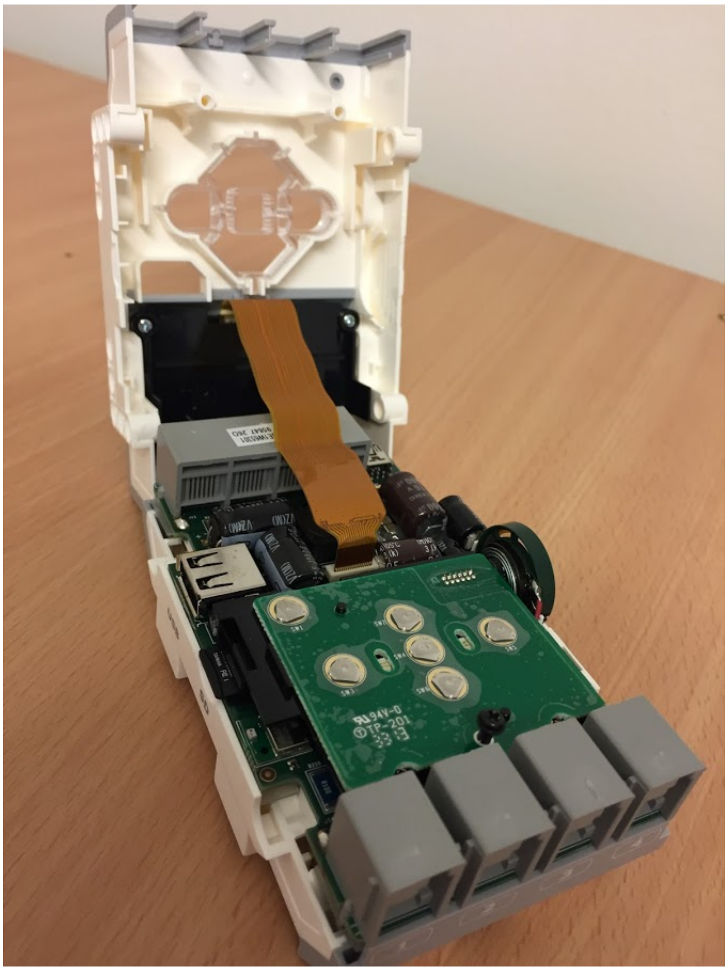


**Figure 2.2:** Opening the EV3.

3. The top plate visible is populated with pressure plates for the EV3 buttons. This is covering the circuitry beneath and will have to be removed. To remove it, simple loosen the screw sitting at the bottom of the plate (in figure 2.2 it can be seen already loosened) and carefully pull the plate out of its slots. This will give you access to the board directly as seen in figure 2.3

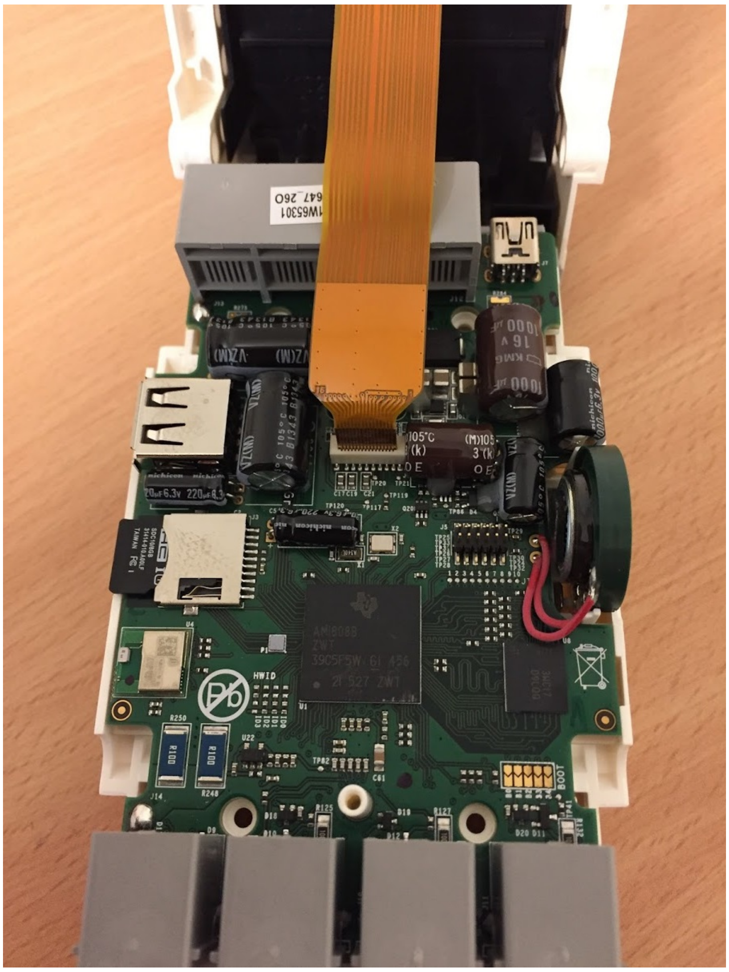


**Figure 2.3:** Pressure plate removed.

The AM1808 can be seen in the middle of the board. The connection pins in the upper right serve as connection for the pressure plate and buttons. Right below that is the point of interest as shown in figure 2.4.

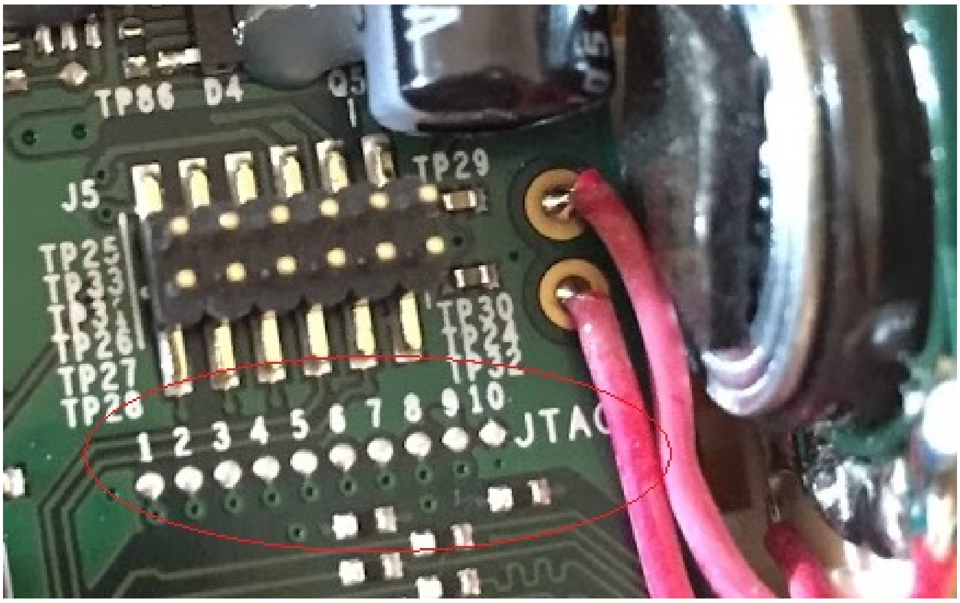


**Figure 2.4:** JTAG point of interest.

Consulting the The LEGO Group (2013) it can be seen from figure 2.5 which connection is connected to which port on the microcontroller.

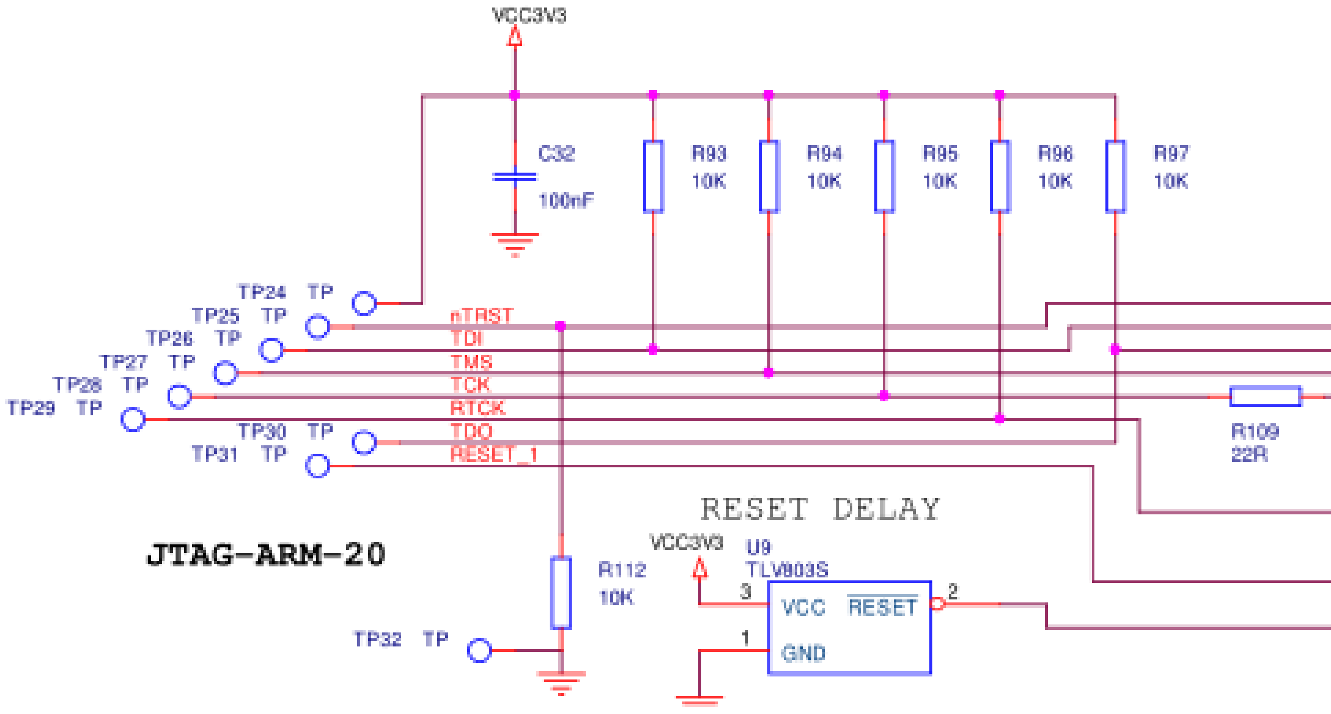


**Figure 2.5:** Hardware schematics showing JTAG interface.

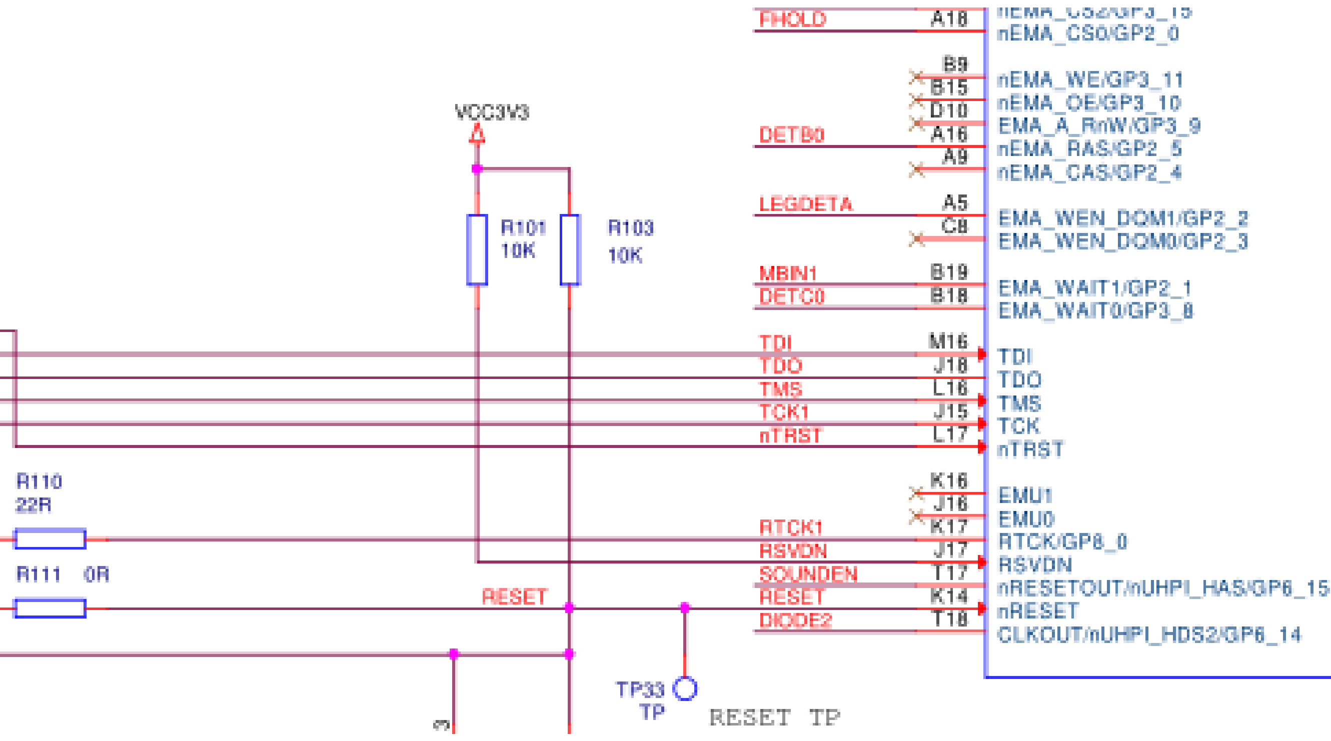


**Figure 2.5:** Continued from last page.

In order to connect a JTAG to this, a 20-pins header (according to the hardware schematic) will have to be soldered on to the soldering pads on the EV3 board, with multiple pins connected to ground. This was done using equipment available at NTNU. Given the small size of the soldering pads, the process was done carefully under a camera that can zoom in on the workplace to make the job easier as seen in figure 2.6.

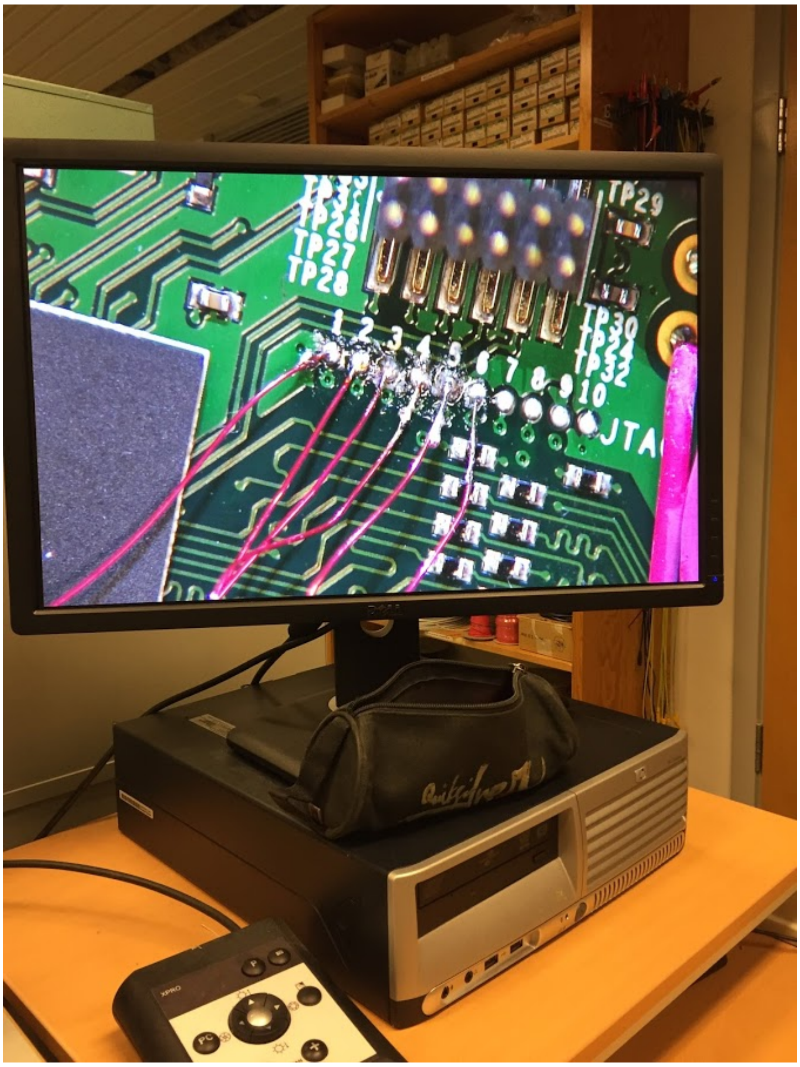


**Figure 2.6:** Soldering process.

The XDS 200 Quick Start Guide (Spectrum Digital, Inc., 2012) lists the possible connections for the included XDS200 JTAG adapters as seen in table 2.1. The EV3 hardware

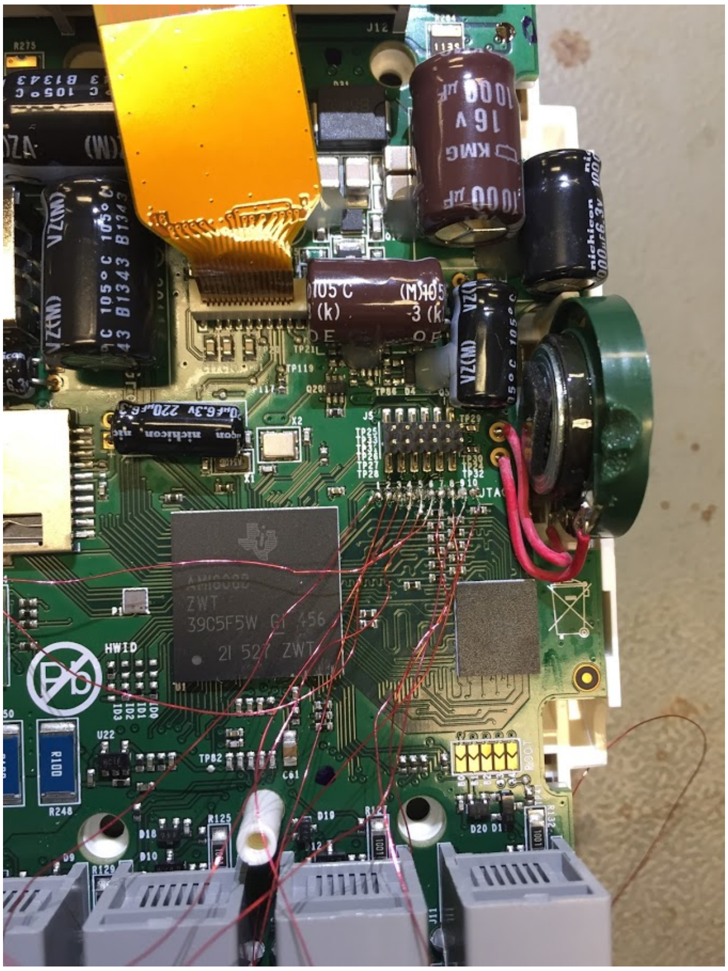


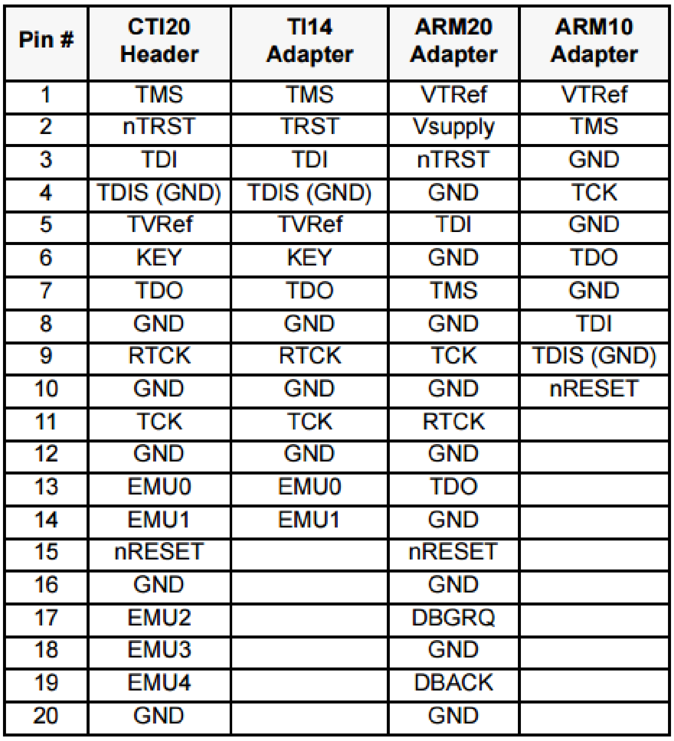
**Figure 2.7:** Soldering completed.

schematics clearly states JTAG-ARM-20, so the ARM 20 adapter was chosen and the wires were soldered to a 20-pin connector as in the schematic in figure 2.8

The XDS200 debug probe can then finally be connected to the system as in fig 2.9.

To test that the connection works, Texas Instruments' Code Composer Studio was installed and a new CCS project was created. A target configuration window will appear as seen in figure 2.10. Here the EV3's AM1808 MCU was entered as the target, along with XDS200 Debug Probe as the connection. Clicking 'Verify...' tests the connection. If everything has been done correctly done, the software will return a success as in figure 2.11.

**Table 2.1:** XDS200 CTI20 Header and Adapter Pin Outs.



| Pin # | CTI20 Header | TI14 Adapter | ARM20 Adapter | ARM10 Adapter |
|---|---|---|---|---|
| 1 | TMS | TMS | VTRef | VTRef |
| 2 | nTRST | TRST | Vsupply | TMS |
| 3 | TDI | TDI | nTRST | GND |
| 4 | TDIS (GND) | TDIS (GND) | GND | TCK |
| 5 | TVRef | TVRef | TDI | GND |
| 6 | KEY | KEY | GND | TDO |
| 7 | TDO | TDO | TMS | GND |
| 8 | GND | GND | GND | TDI |
| 9 | RTCK | RTCK | TCK | TDIS (GND) |
| 10 | GND | GND | GND | nRESET |
| 11 | TCK | TCK | RTCK | |
| 12 | GND | GND | GND | |
| 13 | EMU0 | EMU0 | TDO | |
| 14 | EMU1 | EMU1 | GND | |
| 15 | nRESET | | nRESET | |
| 16 | GND | | GND | |
| 17 | EMU2 | | DBGRQ | |
| 18 | EMU3 | | GND | |
| 19 | EMU4 | | DBACK | |
| 20 | GND | | GND | |

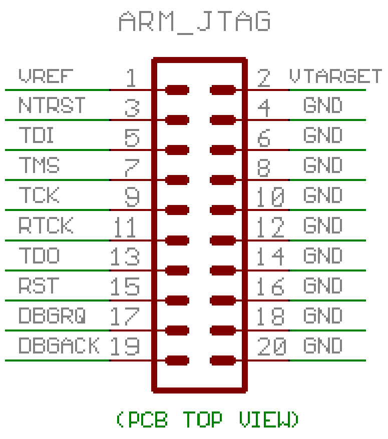


**Figure 2.8:** ARM 20 pin out.

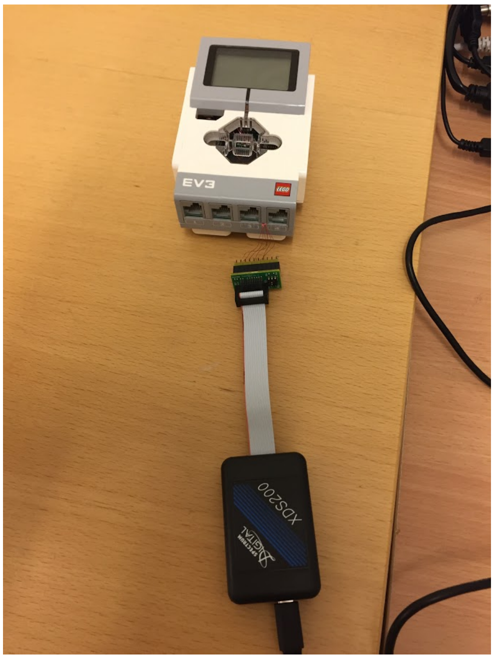
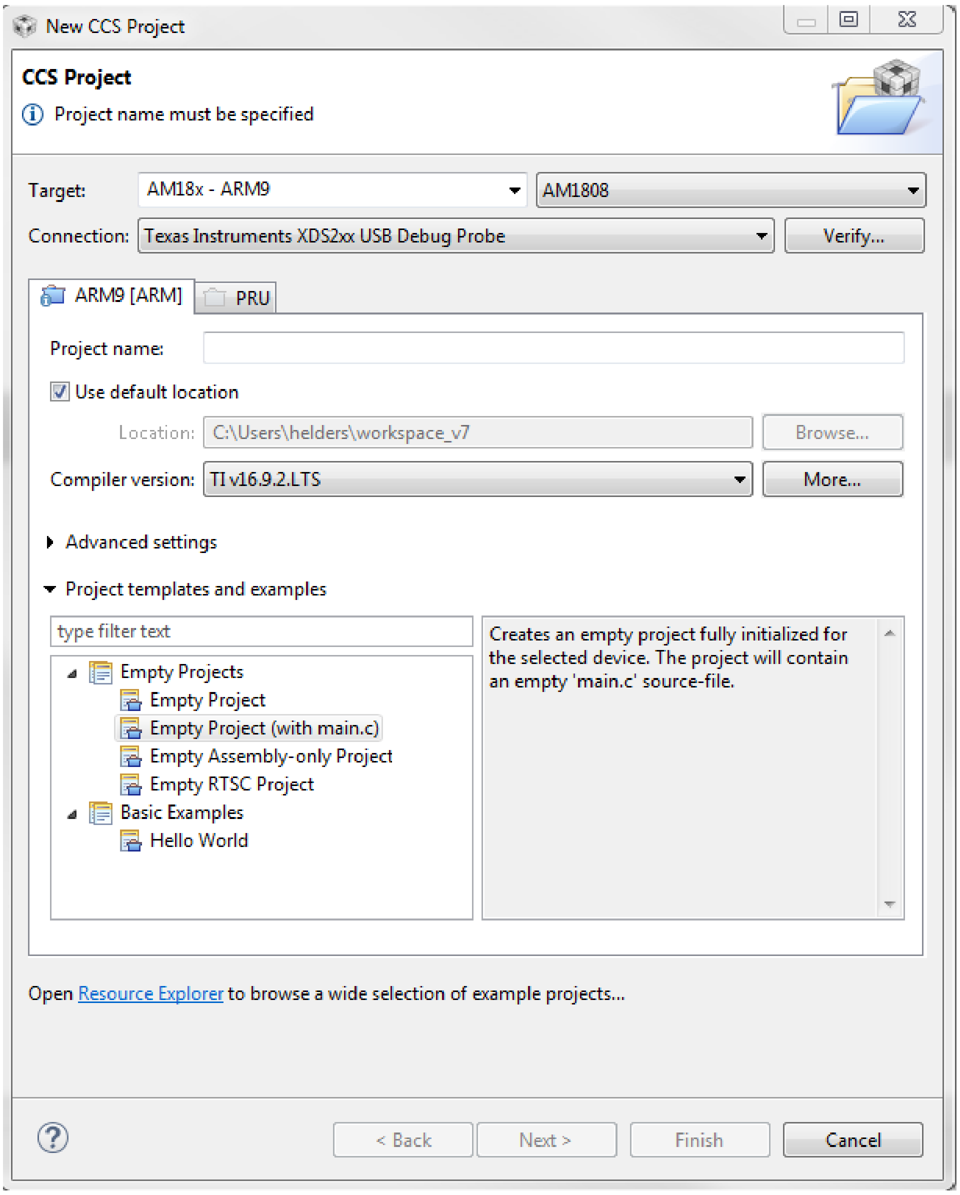


**Figure 2.9:** XDS200 connected to the Lego EV3.



New CCS Project

**CCS Project**

Project name must be specified

Target: AM18x - ARM9 | AM1808

Connection: Texas Instruments XDS2xx USB Debug Probe | Verify...

ARM9 [ARM] | PRU

Project name:

[✓] Use default location

Location: C:\Users\helders\workspace_v7 | Browse...

Compiler version: TI v16.9.2.LTS | More...

Advanced settings

Project templates and examples

type filter text

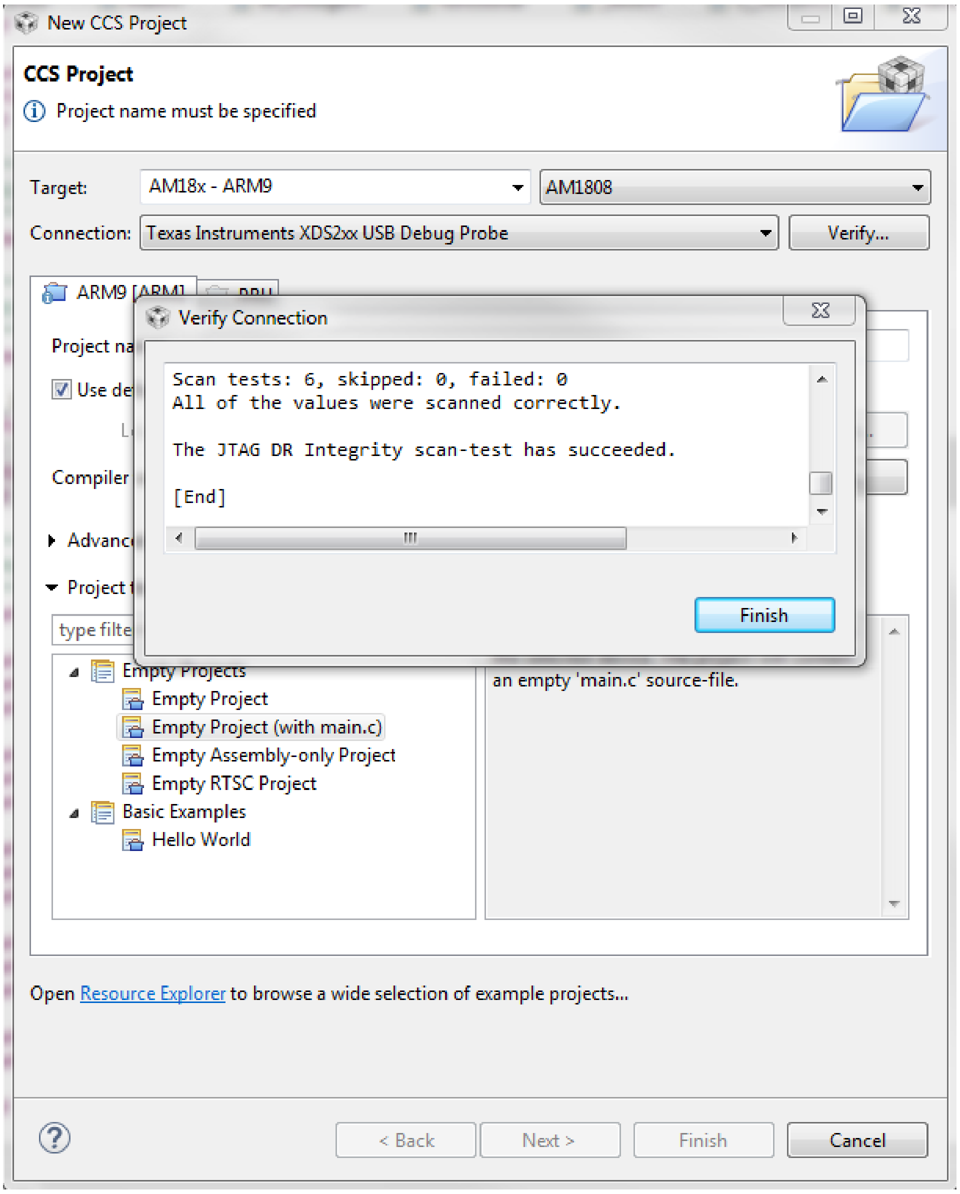
- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty RTSC Project
- Basic Examples
  - Hello World

Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

Open Resource Explorer to browse a wide selection of example projects...

< Back | Next > | Finish | Cancel

**Figure 2.10:** CCS new project target configuration window.



```
Scan tests: 6, skipped: 0, failed: 0
All of the values were scanned correctly.

The JTAG DR Integrity scan-test has succeeded.

[End]
```

Finish
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project
Empty RTSC Project
Basic Examples
Hello World
an empty 'main.c' source-file.
Open Resource Explorer to browse a wide selection of example projects...
< Back
Next >
Finish
Cancel

**Figure 2.11:** JTAG connection works.

## 2.4 The EV3 UART Interface

The EV3 provides an interactive serial console interface on the first sensor port. The AM1808 UART1 interface is wired to the first sensor port on the EV3 brick and can be used for communication with the MCU using a UART to USB cable. See Helders (2016) for more information.

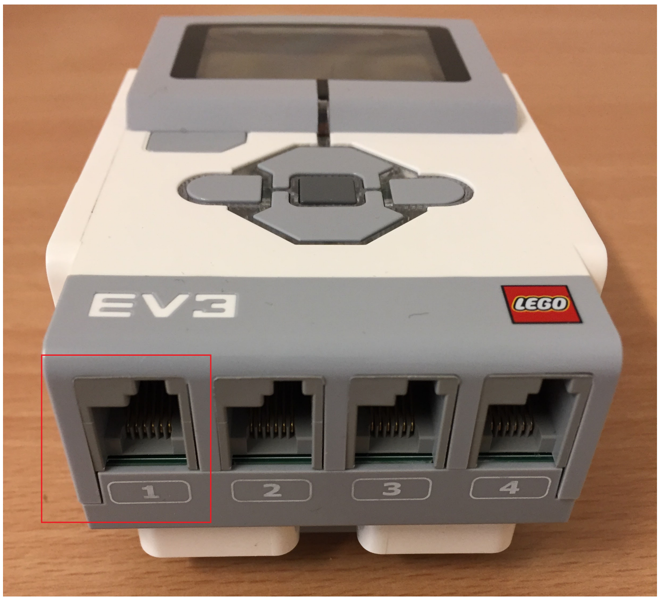


**Figure 2.12:** Port with interactive serial console interface

Chapter 3

# Software Tools

## 3.1 Choice of IDE

With the addition of a JTAG debugger it is now possible to work on the EV3 system completely within an integrated development environment. While Helders (2016) wrote the code in C, cross-compiled with a custom toolchain, and transferred and debugged the system through a serial connection, it is now possible to write the code, compile, debug and flash all within one software application. There are multiple IDEs that can be chosen so a comparison and evaluation of the different software tools is needed to make an informed choice.

### 3.1.1 Code Composer Studio

Code Composer Studio is made by Texas Instruments, the company that also makes the AM1808 MCU in the EV3 system and the XDS200 USB JTAG Emulator, and available with a free license. This means that the software has built-in support for both the AM1808 MCU and XDS200 JTAG Emulator, and can be utilized at no cost. Section 2.3 shows that CCS also has a self-test function in order to confirm that the debugger is properly connected and communicates with the MCU that can be used to verify that everything works correctly. This can save the user a lot of time when trying to discover the source of eventual bugs during communication.

Next FreeRTOS.org was checked to see if any of their ports or demos used the CCS compiler. According to Real Time Engineers ltd. (2017). The majority (if not all) the code that is specific to a single port is contained in a file called `FreeRTOS/source/portable/[compiler]/[microcontroller]/port.c` and an accompanying header file called `FreeRTOS/source/portable/[compiler]/[microcontroller]/portmacro.h`, where `[compiler]` is the name of the compiler being used, and `[microcontroller]` is the name of the microcontroller family being used.

For some compilers the `port.c` and `portmacro.h` files are all that is required. For others (those with less flexible features) an assembler file is also required. This will be called `portasm.s` or `portasm.asm`.

Sadly, the CCS is compiler is one of those with less flexible features and requires a `.asm` Since no CCS port to the ARM9 architecture exists, this will have to be written if CCS is used. To give the reader an idea of what this entails a small snip of the assembly instructions for another architecture port in CCS is included in appendix A.1.

Finally, while the support for TI devices is great, support for non-TI devices is very limited. Since the robots used by NTNU all use different hardware

**Pros:**

- Free.
- Fully supports AM1808 MCU.
- Fully supports XDS200 Debugger.

**Cons:**

- Limited feature set means part of the porting job of FreeRTOS will have to be done in assembly.
- Limited support for non-TI devices.

### 3.1.2 IAR Embedded Workbench

When researching FreeRTOS it quickly became clear that a popular choice of IDE was IAR Embedded Workbench. A lot of the FreeRTOS ports available used this IDE and, since it was so popular, the FreeRTOS forum contained a lot of discussion and resources on how utilize this IDE efficiently.

IAR Embedded Workbench is a licensed IDE made by IAR Systems who are based in Uppsala, Sweden. They support 11 619 devices at the time of writing, including full support for both the TI AM1808 MCU and TI XDS200 JTAG Emulator. In addition to this IAR Systems have a close relationship to Atmel and according to IAR Systems (2017) are the only tool supplier that offers a complete toolchain for all Atmel microcontrollers and microprocessors. Since the AVR, Arduino and NXT robots all use Atmel microcontrollers this means IAR would work well as an IDE for them too. The AVR, Arduino and NXT use the Atmel Studio IDE as seen in Ese (2015) and Ese (2016). Since IAR Embedded Workshop supports Atmel devices natively, less code will need to be adapted when porting the application from the other robots and to the EV3 should this IDE be chosen.

However, a license for IAR Embedded Systems is very expensive, and while they provide a discount for educational licenses, the price will still be in the range of a couple of thousand Norwegian kroner for a life-time license, and lower for a time-limited license.

They also provide a 30-day limited trial license with some restrictions. These include no inclusion of source code for runtime libraries, no support for MISRA C and the trial license must not be used for product development or any other kind of commercial use. A runtime library is a set of low-level routines used by a compiler to invoke some of the behaviors of a runtime environment, by inserting calls to the runtime library into compiled executable binary. This means that a set of execution startup routines (often written in assembly) which performs initialization of the CPU and MCU required before calling the program's main function is not included and the user will have to write this himself. Implementations can however be found online if the user researches "crt0". Finally, the trial is time-limited to 30 days, but can be renewed for another 30 days at the end of the trial period.

### Pros:

- Fully supports AM1808 MCU.
- Fully supports XDS200 Debugger.
- A lot of resources for FreeRTOS porting available.
- Contains the same support and libraries as Atmel Studio which is used on the AVR, Arduino and NXT robots.

- Trial available that enables the IDE to be used for non-commercial work.
- FreeRTOS porting can be done completely in C, parts in assembly are not needed.

**Cons:**

- Expensive if a trial license is not used.
- Trial has some limitations, including having to be renewed every 30 days.

### 3.1.3 Conclusion

Given the likeness in support for Atmel devices between IAR Embedded Workshop and Atmel Studio used by the other robots as well as being the more flexible IDE compared to CCS (no assembly file for FreeRTOS porting required by the compiler), it was decided to use a free, time-limited license of IAR to do development on the EV3.

The general idea for this project is to keep the work required when moving the application from one robot to the others to a minimum. Ideally one would use the same IDE for all robots, but since Atmel Studio has no support for Texas Instruments devices, choosing an IDE with comparable device support is better than porting the application to a totally new IDE that does not support previous devices. If this is done it would most likely require significant changes to existing code.

IAR Embedded Workbench was downloaded and installed. During installation support for the TI XDS200 debugger was an option that could be included in the IDE and so this was installed as well.

Chapter 4

# FreeRTOS Development Guide

The general recommendation when implementing FreeRTOS on new hardware is to always start with a working demo project available on FreeRTOS.org and then adapting this to your project. Doing this ensures the new project includes all the necessary source and header files, and installs the necessary interrupt service routines, with no effort on the part of the project's creator. If a port exists for the hardware and IDE being used, it is as simple as downloading the pre-configured demo, and the hardware and OS layers should compile and run directly so the developer can immediately focus on developing his or her application. If there does not exist a FreeRTOS port for the MCU development becomes a bit more challenging, and a lot of the lower-end parts of the code will have to be written from scratch.

Helders (2016) researched the possibility of using the EV3 built-in bootloader ”U-Boot” to load pre-compiled binaries transferred through a serial connection to the EV3. Since this process was replaced with a full IDE and JTAG connectivity, help was sought out on the FreeRTOS forums on how to go about porting FreeRTOS to a new MCU using IAR Embedded Workshop. The user ”rtel” which is the official representative of Real Time Engineers Ltd.[1] on the forums recommended a step-by-step approach:

1. Get a simple bare metal project running. Just a main() function that does something to make sure you can compile, download and debug the part first.

2. Next select a timer you are going to use to generate the tick interrupt and write a

[1]Real Time Engineers Ltd. are the owners and developers of FreeRTOS.

driver to configure and use it to ensure you can get it generating interrupts at the frequency you want.

3. Add in the FreeRTOS code and rewrite the port files to use the drivers previously written so the scheduler behaves as needed.
4. Set up functions for your application as separate tasks and define priorities so that scheduler knows how to switch between tasks.

In a typical embedded development project it is common to separate source code into layers organized by what the code does. The most frequently defined layers are HARDWARE, OPERATING SYSTEM and APPLICATION. The hardware layer will interface directly with the hardware, this means drivers that make some piece of hardware work is written here. Next, the operating system is the layer that consists of the system scheduler, this decides what gets to happen next, or what device/hardware component gets access to the CPU for work at a given time. Finally, the application can be created by the software developer to realize whatever functionality is wanted by the system.

## 4.1 The Hardware Layer

*The next section consists of development in bare metal arm; the topic is huge and only a minor part needed to explain what was done for this project is included here. If the reader would like to modify or adapt the code written it is recommended to also look up information on linker files, 'c' startup and CPU initialization to really understand how the hardware architecture works and what happens in the lowest levels of an operating system. Since this can get fairly technical and is not really needed to understand what was done it was decided to not include it in this chapter.*

Following Real Time Engineers Ltd.'s advice a new project was made in IAR Embedded Workshop. In order to create a simple bare metal project with a main() function for the AM1808, the CPU will have to be initialized, and a linker script included. The linker script will define the startup files for the compiler, the entry point of your program and format on the output among other things.[2] The CPU initialization is the startup file that contains the assembly code which will initiate the CPU stack before the linker jumps to your main() function for execution. The C startup file and linker script are generally processor and compiler specific, and according to Real Time Engineers Ltd. it is never recommended to

[2]Please research 'linker files' for additional information on the subject.

try to create these files from scratch (Real Time Engineers ltd., 2017), so it was decided to try to find some implementation online.

Searching online uncovered example projects for the TI AM1808 MCU called TI StarterWare (Texas Instruments, 2015). StarterWare is a free software development package that provides no-OS platform support for ARM and DSP TI processors and was also used by Helders (2016). StarterWare includes libraries and example applications that demonstrate the capabilities of the peripherals on the TI processors. Even better, example projects and guides using StarterWare for the TI AM1808 were available through the IAR Embedded Workshop IDE with a configured linker for the project, so it was decided to use these as a reference when writing the code for this project. This can be found under ”Example projects” in IAR Embedded Workshop.

StarterWare version 01.00.03.03 was downloaded and unpacked. Included in the package were example implementations of the various hardware peripherals on the AM1808 like UART, I2C, SPI, GPIO and many others. Most importantly it includes a file called `init.S`. This is the assembly code routine that will take care of CPU initialization. It defines an entry point for the code, so when the IAR compiles and downloads the code on the microcontroller the assembly initialization will run before jumping to the main() function and perform the code there. A printout of the code is included in appendix A.2.

### 4.1.1 A Simple UART Driver

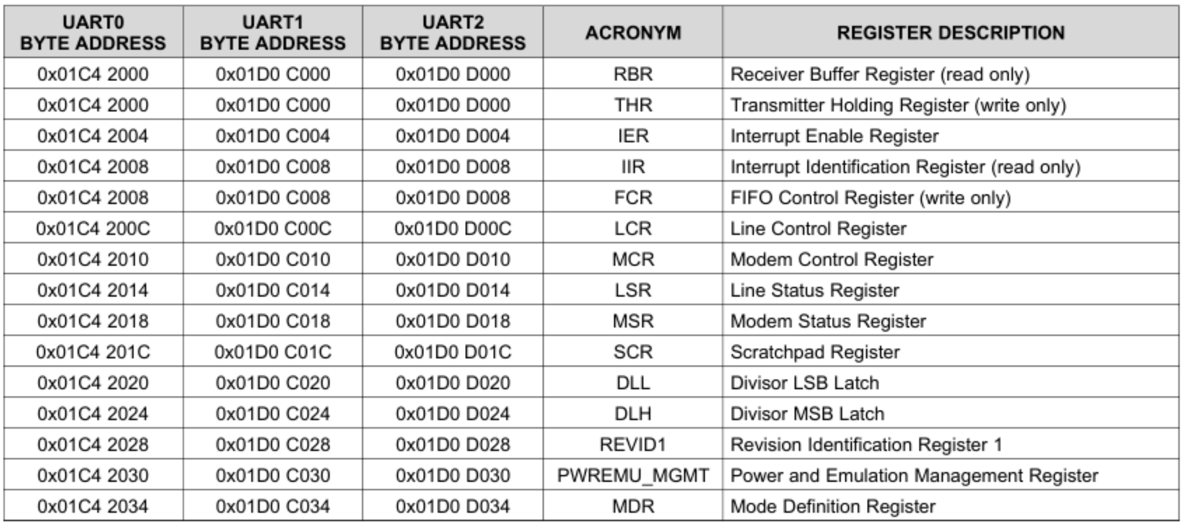
To make sure the setup to compile, download and debug on the EV3 worked, a peripheral needs to be configured so that it is possible to see that the code actually runs from somewhere other than in the IDE. Usually a small program to turn on and off a LED is written, but since Helders (2016) created a console cable for use with the EV3's UART controller, it was decided to create a program that would output something on the serial connection.

Helders (2016) noted that AM1808 contains three different UART modules that can be configured. StarterWare's UART implementation includes a driver example for the UART2 module. As mentioned in section 2.4 the EV3's AM1808 UART1 module is connected to the first sensor port on the EV3. Using the console cable created by Helders (2016) it is possible to rewrite this driver to use the UART1 module instead and test that the program compiles, downloads and runs.

Using the AM1808 datasheet (Texas Instruments, 2014) as reference, the addresses of the three UART modules was found as shown in table 4.1. The UART2 driver provided by

StarterWare was modified to use the byte addresses of the UART1 module instead. This gave no compiler errors when the code was rebuilt.

**Table 4.1:** UART register overview in the AM1808 datasheet.



| UART0 BYTE ADDRESS | UART1 BYTE ADDRESS | UART2 BYTE ADDRESS | ACRONYM | REGISTER DESCRIPTION |
|---|---|---|---|---|
| 0x01C4 2000 | 0x01D0 C000 | 0x01D0 D000 | RBR | Receiver Buffer Register (read only) |
| 0x01C4 2000 | 0x01D0 C000 | 0x01D0 D000 | THR | Transmitter Holding Register (write only) |
| 0x01C4 2004 | 0x01D0 C004 | 0x01D0 D004 | IER | Interrupt Enable Register |
| 0x01C4 2008 | 0x01D0 C008 | 0x01D0 D008 | IIR | Interrupt Identification Register (read only) |
| 0x01C4 2008 | 0x01D0 C008 | 0x01D0 D008 | FCR | FIFO Control Register (write only) |
| 0x01C4 200C | 0x01D0 C00C | 0x01D0 D00C | LCR | Line Control Register |
| 0x01C4 2010 | 0x01D0 C010 | 0x01D0 D010 | MCR | Modem Control Register |
| 0x01C4 2014 | 0x01D0 C014 | 0x01D0 D014 | LSR | Line Status Register |
| 0x01C4 2018 | 0x01D0 C018 | 0x01D0 D018 | MSR | Modem Status Register |
| 0x01C4 201C | 0x01D0 C01C | 0x01D0 D01C | SCR | Scratchpad Register |
| 0x01C4 2020 | 0x01D0 C020 | 0x01D0 D020 | DLL | Divisor LSB Latch |
| 0x01C4 2024 | 0x01D0 C024 | 0x01D0 D024 | DLH | Divisor MSB Latch |
| 0x01C4 2028 | 0x01D0 C028 | 0x01D0 D028 | REVID1 | Revision Identification Register 1 |
| 0x01C4 2030 | 0x01D0 C030 | 0x01D0 D030 | PWREMU_MGMT | Power and Emulation Management Register |
| 0x01C4 2034 | 0x01D0 C034 | 0x01D0 D034 | MDR | Mode Definition Register |

Next, the XDS200 was connected to the EV3 using the interface created in section 2.3, and the UART to USB cable was connected to the first sensor port and back to the computer as seen in figure 4.1. Using a telnet client like PuTTy, find the COM port the console cable is connected to and configure the serial connection with

- 115200 baud
- 8 databits
- No parity
- 1 stop bit

In IAR, open $Project \rightarrow Options... \rightarrow Debugger$ and make sure the debugger is set to TI XDS. Include the macro file `AM1808.mac` which is included in the StartWare package for IAR. Also check off ”Run to” and specify ”main”, this will ensure that when the code is downloaded and the debugger starts it will run the code until the beginning of main before letting the user step through the code, this way the assembly init won’t have to be stepped through every time the code is run for testing. Finally, click $General\ Options$ and make sure AM1808 is chosen as the target device.

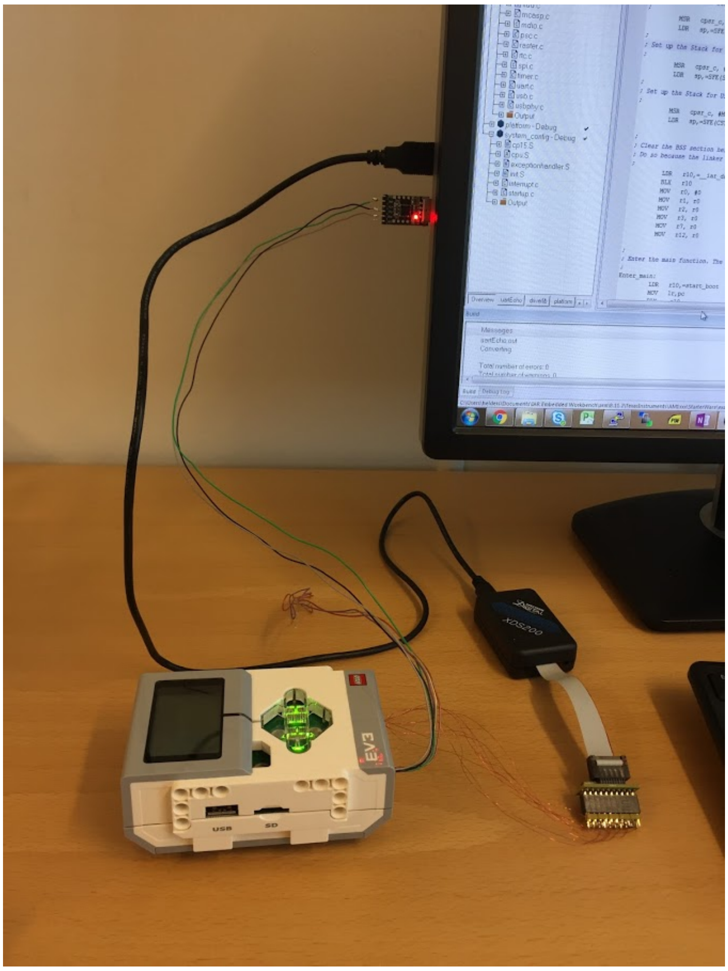
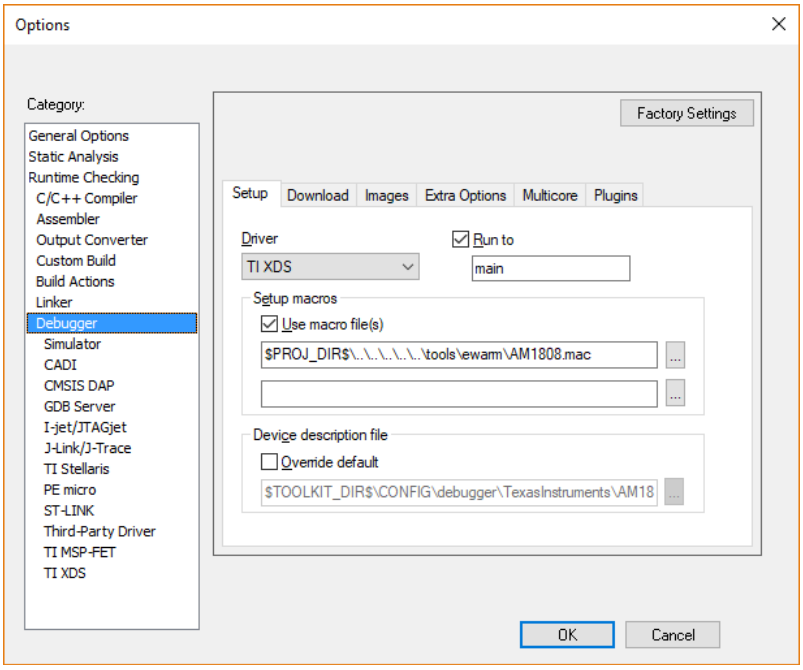


**Figure 4.1:** EV3 connected with JTAG and console cable.



Driver: TI XDS

☑ Run to: main

Setup macros

☑ Use macro file(s)

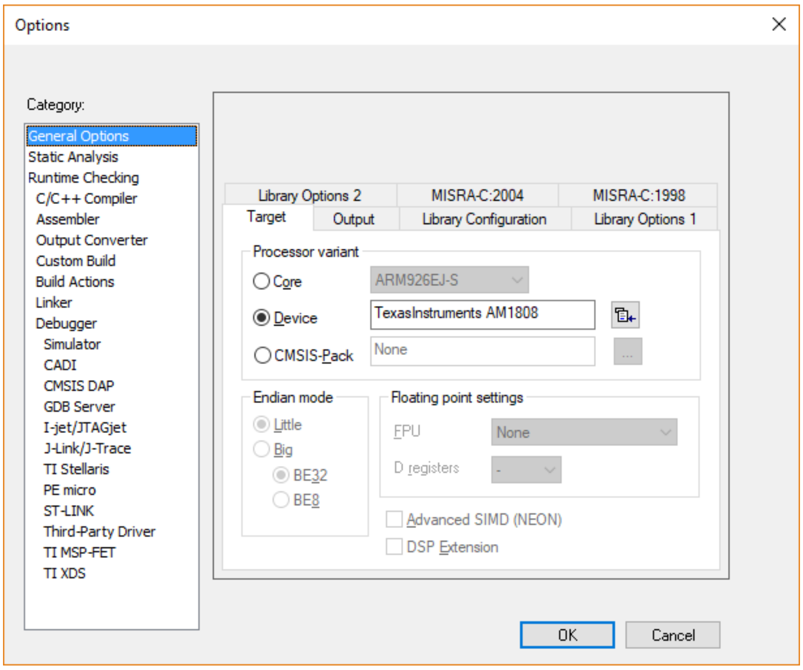
$PROJ_DIR$\..\..\..\..\..\tools\ewarm\AM1808.mac

Device description file

☐ Override default

$TOOLKIT_DIR$\CONFIG\debugger\TexasInstruments\AM18

**Figure 4.2:** IAR debugger settings.



Processor variant

○ Core: ARM926EJ-S

● Device: TexasInstruments AM1808

○ CMSIS-Pack: None

Endian mode

● Little

○ Big

● BE32

○ BE8

Floating point settings

FPU: None

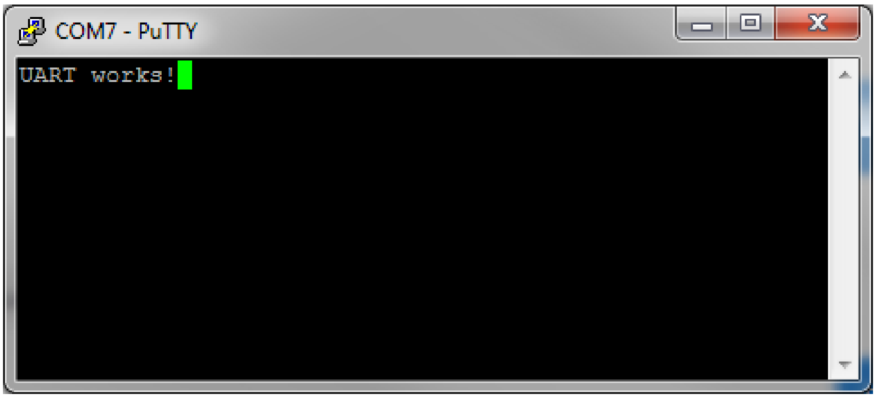
D registers: -

☐ Advanced SIMD (NEON)

☐ DSP Extension

**Figure 4.3:** IAR target settings.

To check that the application runs on target, ensure that the EV3 is powered on and click $Project \rightarrow Download\ and\ Debug$ (or hit CTRL+D). If the application is successful the user should be able to step through the main function and whatever specified by the UART-Puts() function will be output on the serial connection and readable in PuTTy.



```
UART works!
```

**Figure 4.4:** UART output in PuTTy.

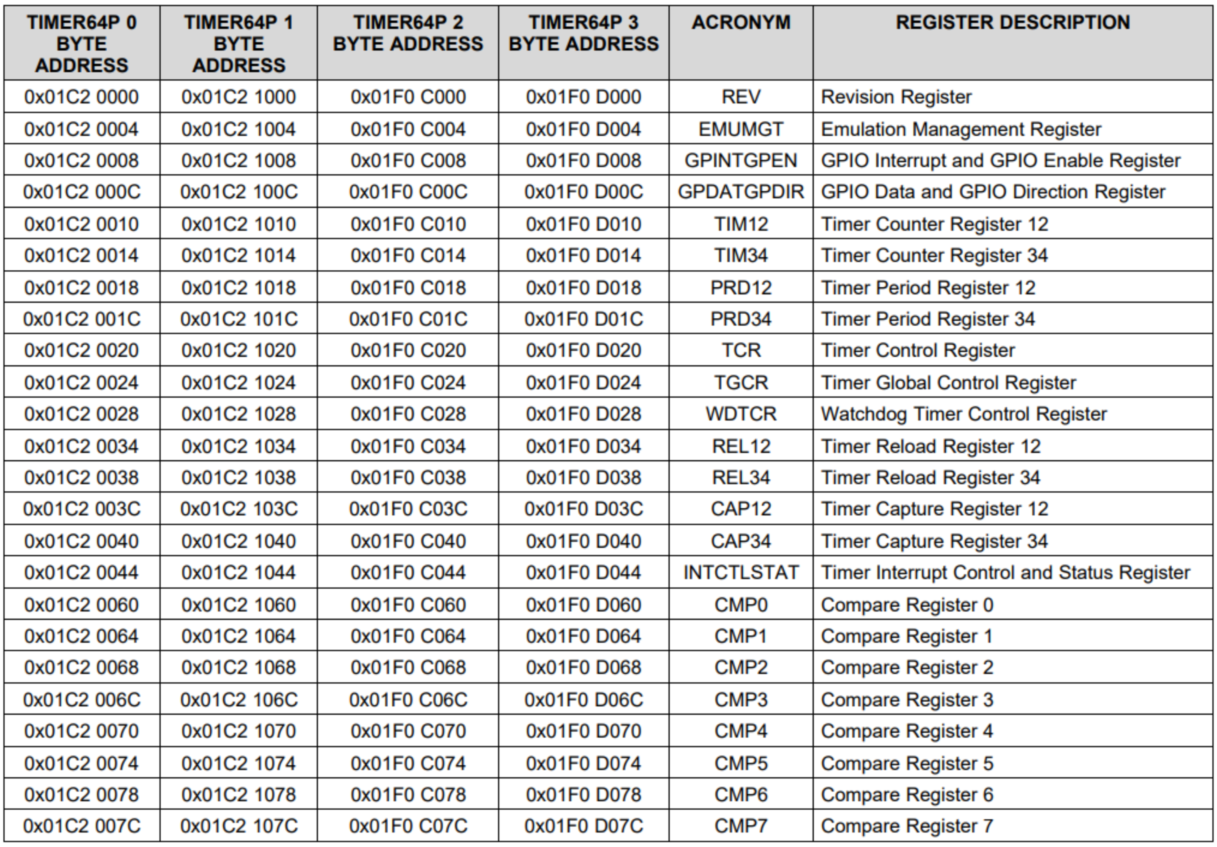
## 4.2 Configuring Timers and Tick Interrupt Source

Once a working project that compiled, downloaded and debugged correctly was created, the next part was to find a suitable tick interrupt source for the FreeRTOS scheduler. The AM1808 MCU has an ARM9 core with four external peripheral timers as shown in table 4.2. Helders (2016) created a driver for the TIMER64P2 module using the StarterWare package as a reference, and it was decided to re-use this code to initialize the timer and interrupt controller on the AM1808. This will be the main interrupt source for FreeRTOS.

To test that the hardware drivers were working, the timer and interrupt controller were initialized and a counter function was written using the UART implementation. The idea is to initialize UART, timer and interrupts, then output a number on UART. When the timer counts down and generates an interrupt, the counter function would subtract the number outputted on UART by one. This would continue until 0 was reached and the program stops.

The system was again connected through JTAG and serial to test. The function was downloaded and outputted on UART. Using PuTTy to show the UART output, the counter was started and successfully counted down to 0 before terminating.

**Table 4.2:** Timer register overview in the AM1808 datasheet.



| TIMER64P 0 BYTE ADDRESS | TIMER64P 1 BYTE ADDRESS | TIMER64P 2 BYTE ADDRESS | TIMER64P 3 BYTE ADDRESS | ACRONYM | REGISTER DESCRIPTION |
|---|---|---|---|---|---|
| 0x01C2 0000 | 0x01C2 1000 | 0x01F0 C000 | 0x01F0 D000 | REV | Revision Register |
| 0x01C2 0004 | 0x01C2 1004 | 0x01F0 C004 | 0x01F0 D004 | EMUMGT | Emulation Management Register |
| 0x01C2 0008 | 0x01C2 1008 | 0x01F0 C008 | 0x01F0 D008 | GPINTGPEN | GPIO Interrupt and GPIO Enable Register |
| 0x01C2 000C | 0x01C2 100C | 0x01F0 C00C | 0x01F0 D00C | GPDATGPDIR | GPIO Data and GPIO Direction Register |
| 0x01C2 0010 | 0x01C2 1010 | 0x01F0 C010 | 0x01F0 D010 | TIM12 | Timer Counter Register 12 |
| 0x01C2 0014 | 0x01C2 1014 | 0x01F0 C014 | 0x01F0 D014 | TIM34 | Timer Counter Register 34 |
| 0x01C2 0018 | 0x01C2 1018 | 0x01F0 C018 | 0x01F0 D018 | PRD12 | Timer Period Register 12 |
| 0x01C2 001C | 0x01C2 101C | 0x01F0 C01C | 0x01F0 D01C | PRD34 | Timer Period Register 34 |
| 0x01C2 0020 | 0x01C2 1020 | 0x01F0 C020 | 0x01F0 D020 | TCR | Timer Control Register |
| 0x01C2 0024 | 0x01C2 1024 | 0x01F0 C024 | 0x01F0 D024 | TGCR | Timer Global Control Register |
| 0x01C2 0028 | 0x01C2 1028 | 0x01F0 C028 | 0x01F0 D028 | WDTCR | Watchdog Timer Control Register |
| 0x01C2 0034 | 0x01C2 1034 | 0x01F0 C034 | 0x01F0 D034 | REL12 | Timer Reload Register 12 |
| 0x01C2 0038 | 0x01C2 1038 | 0x01F0 C038 | 0x01F0 D038 | REL34 | Timer Reload Register 34 |
| 0x01C2 003C | 0x01C2 103C | 0x01F0 C03C | 0x01F0 D03C | CAP12 | Timer Capture Register 12 |
| 0x01C2 0040 | 0x01C2 1040 | 0x01F0 C040 | 0x01F0 D040 | CAP34 | Timer Capture Register 34 |
| 0x01C2 0044 | 0x01C2 1044 | 0x01F0 C044 | 0x01F0 D044 | INTCTLSTAT | Timer Interrupt Control and Status Register |
| 0x01C2 0060 | 0x01C2 1060 | 0x01F0 C060 | 0x01F0 D060 | CMP0 | Compare Register 0 |
| 0x01C2 0064 | 0x01C2 1064 | 0x01F0 C064 | 0x01F0 D064 | CMP1 | Compare Register 1 |
| 0x01C2 0068 | 0x01C2 1068 | 0x01F0 C068 | 0x01F0 D068 | CMP2 | Compare Register 2 |
| 0x01C2 006C | 0x01C2 106C | 0x01F0 C06C | 0x01F0 D06C | CMP3 | Compare Register 3 |
| 0x01C2 0070 | 0x01C2 1070 | 0x01F0 C070 | 0x01F0 D070 | CMP4 | Compare Register 4 |
| 0x01C2 0074 | 0x01C2 1074 | 0x01F0 C074 | 0x01F0 D074 | CMP5 | Compare Register 5 |
| 0x01C2 0078 | 0x01C2 1078 | 0x01F0 C078 | 0x01F0 D078 | CMP6 | Compare Register 6 |
| 0x01C2 007C | 0x01C2 107C | 0x01F0 C07C | 0x01F0 D07C | CMP7 | Compare Register 7 |

## 4.3 FreeRTOS Implementation

Finally it was time to add in the FreeRTOS source code. FreeRTOS was downloaded from FreeRTOS.org. The structure of the FreeRTOS/Source directory is shown below as described at FreeRTOS.org.

```
FreeRTOS

+-Source        The core FreeRTOS kernel files

    +-include   The core FreeRTOS kernel header files

    +-Portable  Processor specific code.

        +-Compiler x  All the ports supported for compiler x
        +-Compiler y  All the ports supported for compiler y
        +-MemMang     The sample heap implementations
```

The core RTOS code is contained in three files, which are called called `tasks.c`, `queue.c` and `list.c`. These three files are in the `FreeRTOS/Source` directory. The same directory contains two optional files called `timers.c` and `croutine.c` which implement software timer and co-routine functionality respectively. Co-routine is only necessary for very memory limited systems (not needed for the EV3).

Each supported processor architecture requires a small amount of architecture specific RTOS code. This is the RTOS portable layer, and it is located in the `FreeRTOS/Source/Portable/[compiler]/[architecture]` sub directories, where `[compiler]` and `[architecture]` are the compiler used to create the port, and the architecture on which the port runs, respectively.

All the source code files located in `Source` and `Source/Include` were imported directly into the timerCounter project created in section 4.2 without any changes. In `Source/Portable` multiple port implementations were available, and the IAR Atmel SAM9 was chosen as a basis for the porting job as Atmel's MCU has the same ARM9 core as the AM1808 and was using the same IAR IDE as used in this project. This means any compiler and CPU specific code can be re-used. The port files were imported, but since the port is not compatible with AM1808, all functions in `port.c` were commented out.

A new file was created, `freeRTOSconfig.h` and added to the root folder of the project. This is a configuration file used to customize FreeRTOS. The file tailors the RTOS kernel to the application being built. It is therefore specific to the application, not the RTOS, and should be located in an application directory, not in one of the RTOS kernel source code directories (Real-time Engineers Ltd., 2016). The configuration file was set-up as described by Real-time Engineers Ltd. (2016) using specifications from the AM1808 datasheet (Texas Instruments, 2014).

Once all necessary files were included in the project it was necessary to add them to the pre-processor include path so the compiler can find them. In IAR this is done by doing the following:

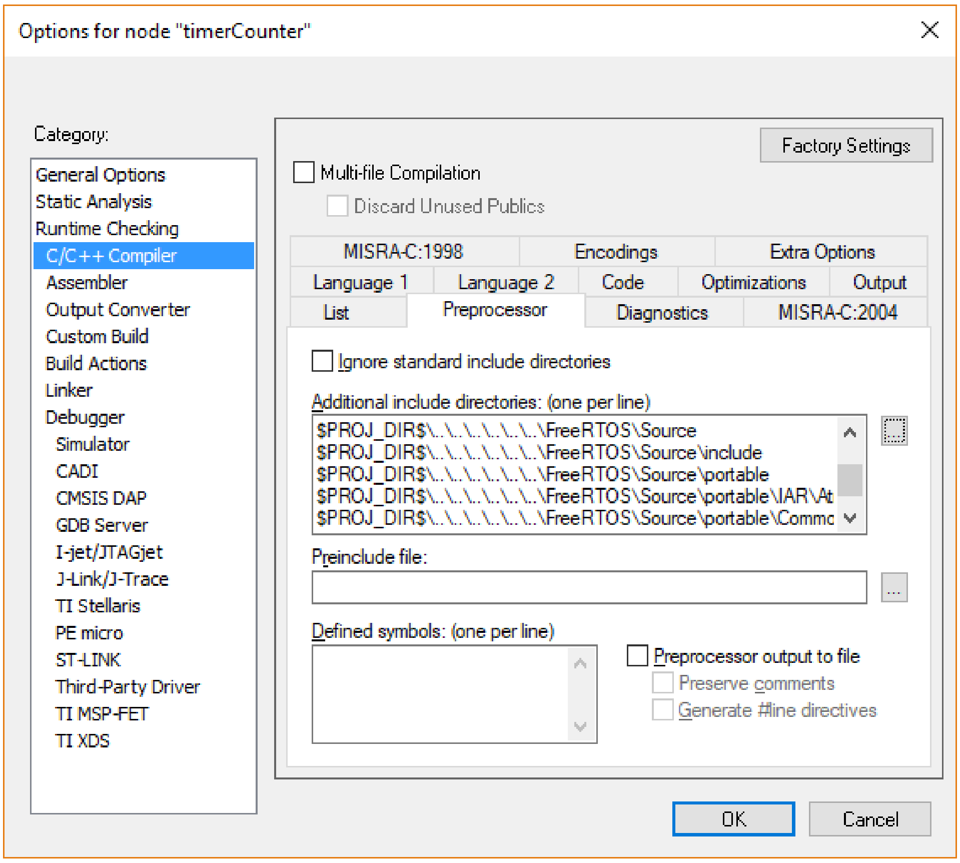
1. Open $Project \rightarrow Options... \rightarrow C/C++\ Compiler \rightarrow Preprocessor$
2. Under "Additional include directories: (one per line)" click the ellipsis button.
3. Add the file locations of every FreeRTOS source code file including the configuration file. Make sure the files are actually located in the project folder to make version control and back-up easier.
4. After adding the file locations, the absolute path of the files will be added to the pre-processor. It is recommended to click the small arrow pointing down to the right of the path to change this to the relative path of the the project (path starting with $PROJ_DIR$ so the project can be moved to different locations/computers without having to set-up the pre-processor paths again).

With the source code in the port file mostly commented out the project was now be able to compile once includes to files specific to the SAM9 MCU were removed (they are not needed since relevant parts in the port file are commented out).

When the project compiled, it was time to start rewriting the port file. The main kernel port, as far as context saving and restoring, are identical on all ARM9 based MCUs so here the SAM9 configuration could be kept. However, a new tick interrupt source needed to be implemented. The peripheral timer configured in 4.2 was used to implement a tick interrupt source. This was done by updating `prvSetupTimerInterrupt()` to configure the timer, and configuring `vPortTickISR()` as the timer interrupt service routine.

Once this was done, it was ensured that the interrupt vectors[3] were set correctly to service

[3] An "interrupt vector table" is a data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of interrupt vectors. An entry in the interrupt vector is the address of the interrupt handler. While the concept is common across processor architectures, each IVT may be implemented in an architecture-specific fashion.


Options for node "timerCounter"

Category:

General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
CADI
CMSIS DAP
GDB Server
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
PE micro
ST-LINK
Third-Party Driver
TI MSP-FET
TI XDS

Factory Settings

☐ Multi-file Compilation
☐ Discard Unused Publics

MISRA-C:1998 | Encodings | Extra Options
Language 1 | Language 2 | Code | Optimizations | Output
List | Preprocessor | Diagnostics | MISRA-C:2004

☐ Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\..\..\..\..\FreeRTOS\Source
$PROJ_DIR$\..\..\..\..\..\..\FreeRTOS\Source\include
$PROJ_DIR$\..\..\..\..\..\..\FreeRTOS\Source\portable
$PROJ_DIR$\..\..\..\..\..\..\FreeRTOS\Source\portable\IAR\At
$PROJ_DIR$\..\..\..\..\..\..\FreeRTOS\Source\portable\Commo
```

Preinclude file:

Defined symbols: (one per line)

☐ Preprocessor output to file
☐ Preserve comments
☐ Generate #line directives

OK Cancel

**Figure 4.5:** Preprocessor include paths.

IRQ and SWI interrupts. StartWare has an example implementation for this, but since the vector table will be the same for all ARM9 based devices, it was instead decided to copy the vector table from an existing ARM9 FreeRTOS port.

Finally, the rest of the port file was gone through and all calls to SAM9 specific drivers were changed to ones written for AM1808. The interface (*.h files) were preserved to keep the FreeRTOS layer as close to the original as possible, only changing the actual code in the implementation.

To test it was decided to create two tasks outputting different strings to the UART peripheral on the EV3. One task would output ”String 1” and the second would output ”String 2”, both in infinite loops. The program would start one task and, with priorities set, the scheduler would interrupt the first task and start performing the other before interrupting again and resuming the first.

Testing showed that FreeRTOS was successfully interrupting task 1 and outputting ”String 2” on UART at defined intervals.

Chapter 5

# Hardware Abstraction

## 5.1 The Need for a Common Abstraction Layer

Because the SLAM application developed at NTNU makes use of multiple robots running different hardware, any small addition or improvement to the software on one robot might turn into a lot of work when porting this feature or improvement over to the other robots.

To avoid this a common abstraction layer for each part of the software has to be implemented. The lowest layer of software should interface directly with the hardware and only communicate directly with the operating system on top of it. The SLAM application will interact with FreeRTOS and should make no calls to the hardware abstraction layer directly. This way, a common interface can be designed which all the robots will use. Any changes to the software on the higher levels of code will be independent of the content of the lower layers, and the application should be able to be ported between the robots with minimum extra work required.

## 5.2 Doxygen

At the time this thesis was written there existed no central documentation for the NTNU Lego robot SLAM project. Any new developer wanting to read about what had been done previously would have to go through multiple reports and theses to discover what

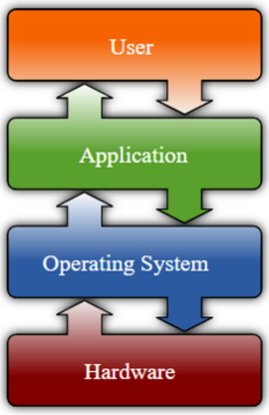


**Figure 5.1:** Abstraction layers.

had been done and by whom, often finding that the previous student had done little to no documentation of the code itself. This left the student with going over the source code directly, prodding and poking every function and class to discover exactly what each part of the code did before being able to start contributing himself. The previous student himself would often have moved on from the university, and getting help from him was not possible.

The root of the problem is that few students are motivated to write a separate document and keep this updated in addition to their master thesis or project report. To remedy this, research was done on source code documentation using comments directly in the code. Students often write (in varying quality) small comments hidden away with the functions themselves that contain information on how things were implemented, what the functions did if it was not easily observable, and why certain decisions in the code were made. If it was possible to extract these comments automatically to a list in a PDF or something similar, it might motivate students to spend the extra time to write a bit more thorough comments as these can then be used as a PDF documentation of the code later.

After some research Doxygen was found to fit this description. According to Heesch (2017) Doxygen is the de facto standard tool for generating documentation from annotated C++ sources. Doxygen also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D. Doxygen can be found at:

`http://www.stack.nl/~dimitri/doxygen/index.html`.

Doxygen can help the user in three ways:

1. It can generate an on-line documentation browser (in HTML) and an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. Doxygen can be configured to extract the code structure from undocumented source files. This is very useful for navigating large source distributions quickly. Doxygen can also visualize the relations between the various elements using dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. The user can also use Doxygen for creating normal documentation (as is done for the Doxygen user manual and web-site).

To generate dependency graphs, Doxygen can use the ”dot” tool from graphviz to generate more advanced diagrams and graphs. Graphviz is an open-source, cross-platform graph drawing toolkit and can be found at:

`http://www.graphviz.org/.`

These tools were downloaded and installed. A short guide for configuring Doxygen and the Graphviz together is included below:

1. Download and install both Doxygen and Graphviz in your preferred location. Take note of the installation folder of Graphviz as you’ll need the folder location later.

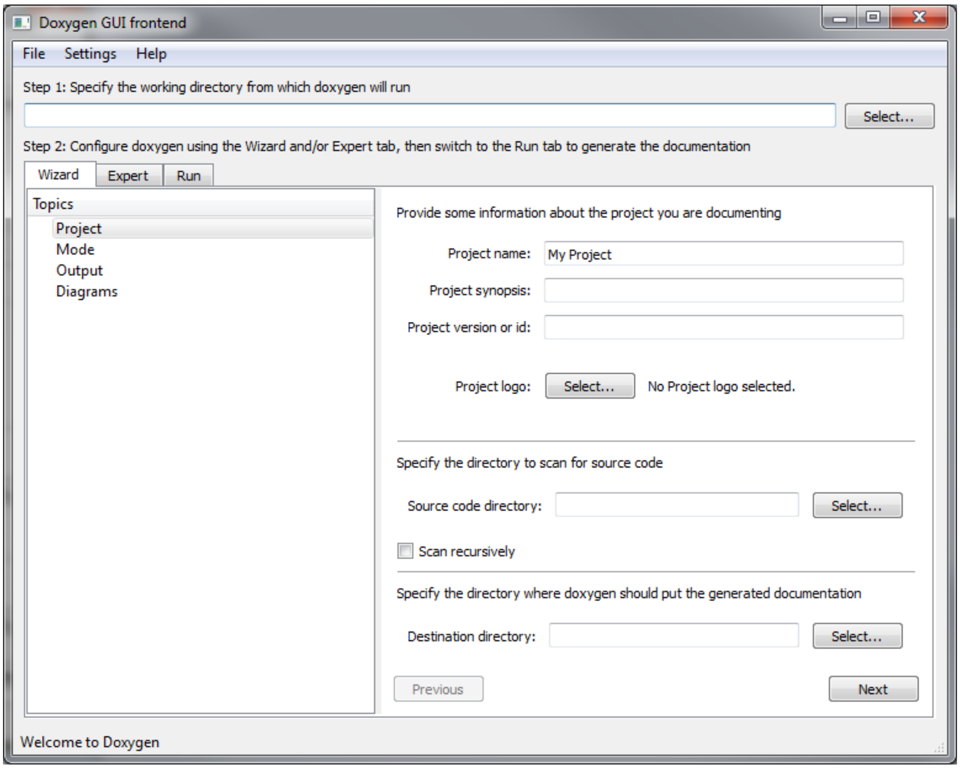
2. Search for and start doxywizard, this is a GUI show in figure 5.2 that lets the user configure and run doxygen.

3. First specify the working directory for doxygen, this is a folder used for temporary files when doxygen runs, as well as the default save location for configuration files. You should create a separate folder for this.

4. Go to step 2 to configure doxygen and choose a name for your project, project synopsis and project version/ID as needed.

5. Specify the source code directery and check ”Scan recursively”.

6. Set the destination directory. This is the folder where doxygen will output the generated documentation for your source code.



Step 1: Specify the working directory from which doxygen will run

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Provide some information about the project you are documenting

Project name: My Project

Project synopsis:

Project version or id:

Project logo: No Project logo selected.

Specify the directory to scan for source code

Source code directory:

☐ Scan recursively

Specify the directory where doxygen should put the generated documentation

Destination directory:

Welcome to Doxygen

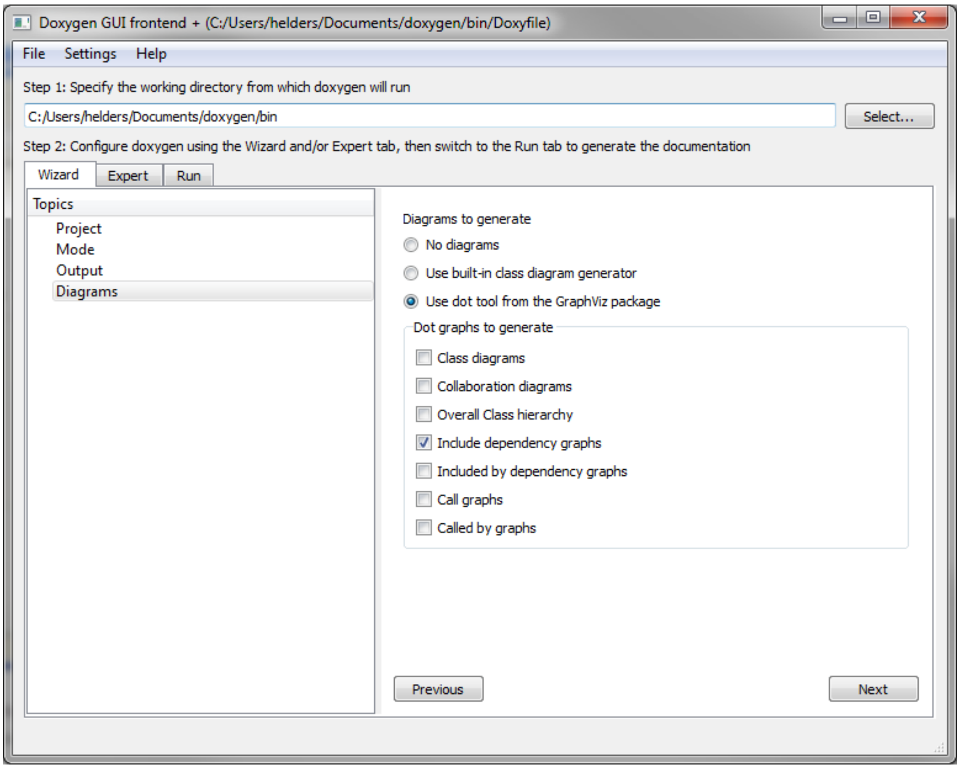
**Figure 5.2:** Doxygen GUI.

7. Click on ”Mode” in the left panel, choose ”All entities” and pick your source code language from the list.

8. Next select ”Output” and choose your preferred documentation type(s). HTML will create a website-like documentation which the user can navigate. Picking LaTeX will generate LaTeX files which can then be directly compiled to a PDF using a LaTeX compiler or `https://www.sharelatex.com/`. If LaTeX is picked it is recommended to choose ”as intermediate format for hyperlinked PDF”.

9. Then select ”Diagrams”. Choose ”Use dot tool from the GraphViz package” and select the diagrams and graphs wanted, in this example only dependency graphs are chosen as shown in figure 5.3.

10. Switch to the ”Expert” tab and scroll the left panel down to ”Dot”. Here the user can make more advanced configurations compared to the previous list, but the only setting needing to be changed is DOT_PATH. Point this to the installation folder



Doxygen GUI frontend + (C:/Users/helders/Documents/doxygen/bin/Doxyfile)

File Settings Help

Step 1: Specify the working directory from which doxygen will run

C:/Users/helders/Documents/doxygen/bin

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Wizard | Expert | Run

Topics
- Project
- Mode
- Output
- Diagrams

Diagrams to generate
- No diagrams
- Use built-in class diagram generator
- [✓ Use dot tool from the GraphViz package]

Dot graphs to generate
- Class diagrams
- Collaboration diagrams
- Overall Class hierarchy
- [✓ Include dependency graphs]
- Included by dependency graphs
- Call graphs
- Called by graphs

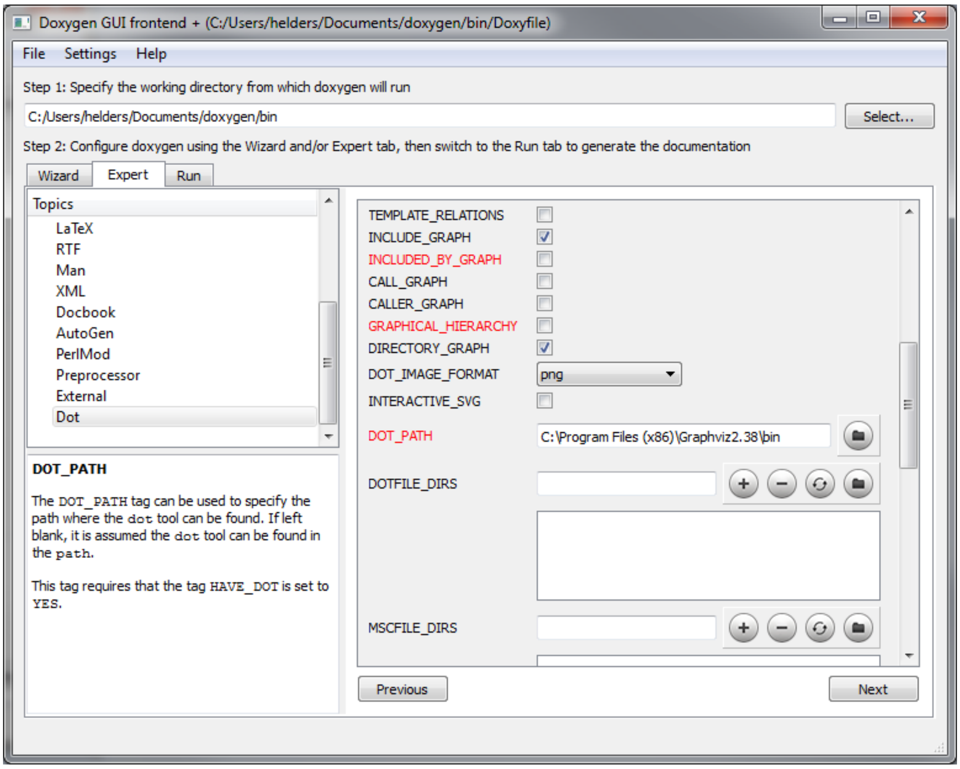
**Figure 5.3:** Doxygen - Include dependency graphs.

chosen during the installation of GraphViz as pointed out in step 1. The tool itself is found in the bin folder so this is where doxywizard needs to look as seen in figure 5.4.

11. Finally, switch to the run tab and click the button to run doxygen. It will go through all source code files looking for comments in the doxygen syntax, as well as all functions, classes and files dependencies amongst other things. Unless something went wrong, doxygen should return ”Doxygen has finished” when it is done.

The documentation generated by Doxygen lets the user look through all functions with parameters and returns, data structures and a complete file list with dependencies. These are illustrated with graphs and diagrams if the dot tool is used. For functions doxygen lets the user create explanations for what a functions takes as parameter inputs, what it will return and example of usage directly in comments in the source code.

The elements in the graphs generated by the dot tool have the following meaning:


Doxygen GUI frontend + (C:/Users/helders/Documents/doxygen/bin/Doxyfile)

Step 1: Specify the working directory from which doxygen will run

C:/Users/helders/Documents/doxygen/bin

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

**DOT_PATH**

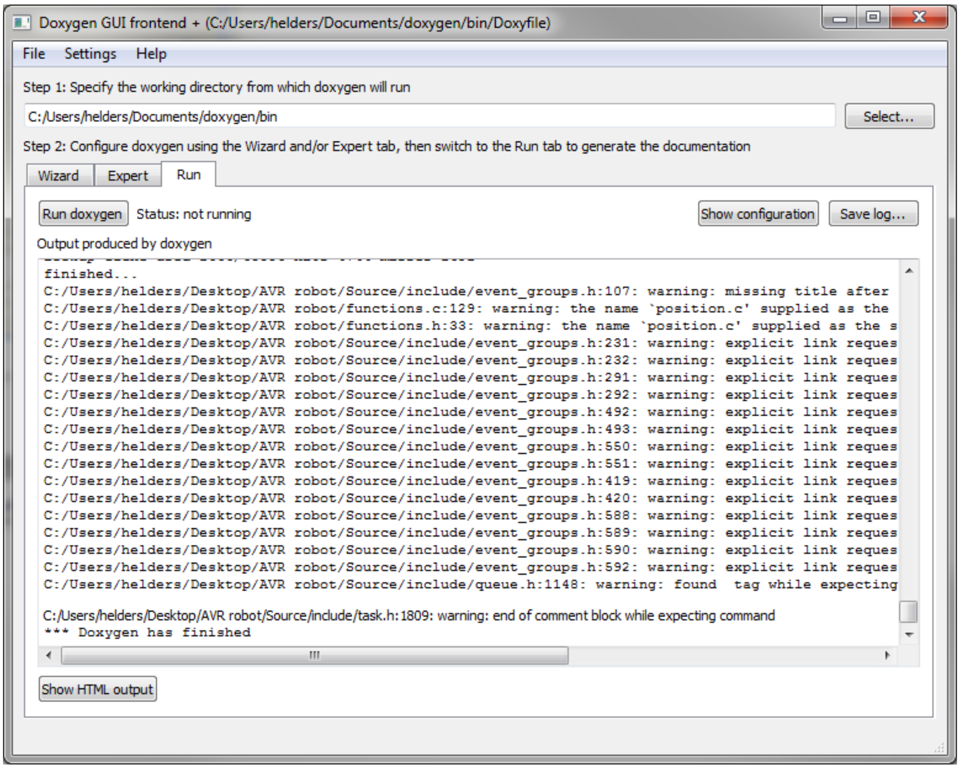
The `DOT_PATH` tag can be used to specify the path where the `dot` tool can be found. If left blank, it is assumed the `dot` tool can be found in the `path`.

This tag requires that the tag `HAVE_DOT` is set to `YES`.

| | |
|---|---|
| TEMPLATE_RELATIONS | ☐ |
| INCLUDE_GRAPH | ✓ |
| INCLUDED_BY_GRAPH | ☐ |
| CALL_GRAPH | ☐ |
| CALLER_GRAPH | ☐ |
| GRAPHICAL_HIERARCHY | ☐ |
| DIRECTORY_GRAPH | ✓ |
| DOT_IMAGE_FORMAT | png |
| INTERACTIVE_SVG | ☐ |
| DOT_PATH | C:\Program Files (x86)\Graphviz2.38\bin |
| DOTFILE_DIRS | |
| MSCFILE_DIRS | |

**Figure 5.4:** Doxygen - Point to the dot tool.

- A white box indicates a class or struct or file.
- A box with a red border indicates a node that has more arrows than are shown. In other words: the graph is truncated with respect to this node. The reason why a graph is sometimes truncated is to prevent images from becoming too large. For the graphs generated with dot doxygen tries to limit the width of the resulting image to 1024 pixels.
- A black box indicates that the class' documentation is currently shown.
- A dark blue arrow indicates an include relation (for the include dependency graph) or public inheritance (for the other graphs).
- A dark green arrow indicates protected inheritance.
- A dark red arrow indicates private inheritance.
- A purple dashed arrow indicated a "usage" relation, the edge of the arrow is labeled



Step 1: Specify the working directory from which doxygen will run

C:/Users/helders/Documents/doxygen/bin

Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Status: not running

Output produced by doxygen

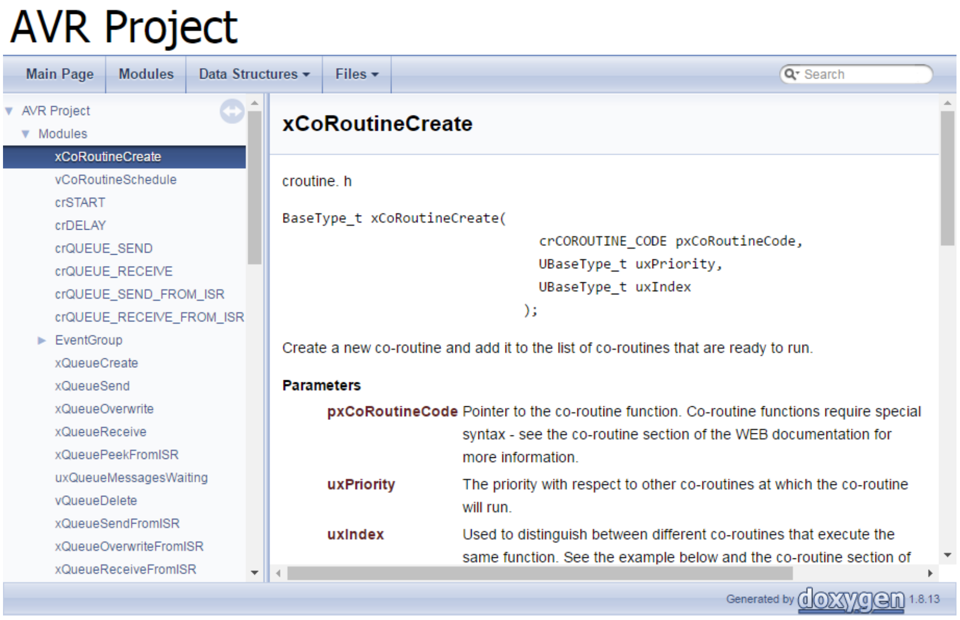
```
finished...
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:107: warning: missing title after
C:/Users/helders/Desktop/AVR robot/functions.c:129: warning: the name `position.c' supplied as the
C:/Users/helders/Desktop/AVR robot/functions.h:33: warning: the name `position.c' supplied as the s
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:231: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:232: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:291: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:292: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:492: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:493: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:550: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:551: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:419: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:420: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:588: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:589: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:590: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/event_groups.h:592: warning: explicit link reques
C:/Users/helders/Desktop/AVR robot/Source/include/queue.h:1148: warning: found  tag while expecting

C:/Users/helders/Desktop/AVR robot/Source/include/task.h:1809: warning: end of comment block while expecting command
*** Doxygen has finished
```

**Figure 5.5:** Doxygen - Doxywizard has finished generating the documentation.

with the variable(s) responsible for the relation. Class A uses class B, if class A has a member variable m of type C, where B is a subtype of C (e.g. C could be B, B*, T* ).

More information about doxygen usage and its syntax can be found at `https://www.stack.nl/~dimitri/doxygen/manual/index.html`


AVR Project

Main Page | Modules | Data Structures | Files | Search

- AVR Project
  - Modules
    - xCoRoutineCreate
    - vCoRoutineSchedule
    - crSTART
    - crDELAY
    - crQUEUE_SEND
    - crQUEUE_RECEIVE
    - crQUEUE_SEND_FROM_ISR
    - crQUEUE_RECEIVE_FROM_ISR
    - EventGroup
    - xQueueCreate
    - xQueueSend
    - xQueueOverwrite
    - xQueueReceive
    - xQueuePeekFromISR
    - uxQueueMessagesWaiting
    - vQueueDelete
    - xQueueSendFromISR
    - xQueueOverwriteFromISR
    - xQueueReceiveFromISR

**xCoRoutineCreate**

croutine. h

```
BaseType_t xCoRoutineCreate(
                              crCOROUTINE_CODE pxCoRoutineCode,
                              UBaseType_t uxPriority,
                              UBaseType_t uxIndex
                            );
```

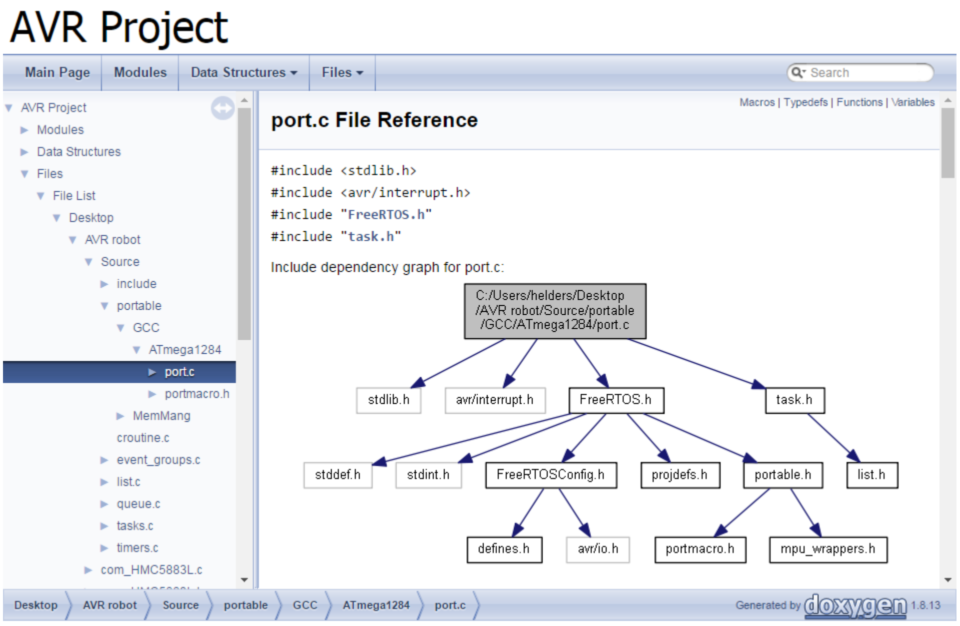
Create a new co-routine and add it to the list of co-routines that are ready to run.

**Parameters**

| | |
|---|---|
| **pxCoRoutineCode** | Pointer to the co-routine function. Co-routine functions require special syntax - see the co-routine section of the WEB documentation for more information. |
| **uxPriority** | The priority with respect to other co-routines at which the co-routine will run. |
| **uxIndex** | Used to distinguish between different co-routines that execute the same function. See the example below and the co-routine section of |

Generated by doxygen 1.8.13

**Figure 5.6:** Doxygen - Example of outputted documentation when doxygen is run on FreeRTOS.


AVR Project

Main Page | Modules | Data Structures | Files | Search

- AVR Project
  - Modules
  - Data Structures
  - Files
    - File List
      - Desktop
        - AVR robot
          - Source
            - include
            - portable
              - GCC
                - ATmega1284
                  - port.c
                  - portmacro.h
            - MemMang
            - croutine.c
            - event_groups.c
            - list.c
            - queue.c
            - tasks.c
            - timers.c
          - com_HMC5883L.c

Macros | Typedefs | Functions | Variables

**port.c File Reference**

```
#include <stdlib.h>
#include <avr/interrupt.h>
#include "FreeRTOS.h"
#include "task.h"
```

Include dependency graph for port.c:

Desktop › AVR robot › Source › portable › GCC › ATmega1284 › port.c — Generated by doxygen 1.8.13

**Figure 5.7:** Doxygen - Example of outputted dependency graph when doxygen is run on FreeRTOS.

## 5.3 Current Project Structure on the AVR

Ese (2016) implemented FreeRTOS on the AVR and rewrote the application to take advantage of this; this version was then implemented on the Arduino robot as well. Since this is the latest major overhaul of the SLAM system, it will be used as a starting point for designing the abstraction layers in the code.

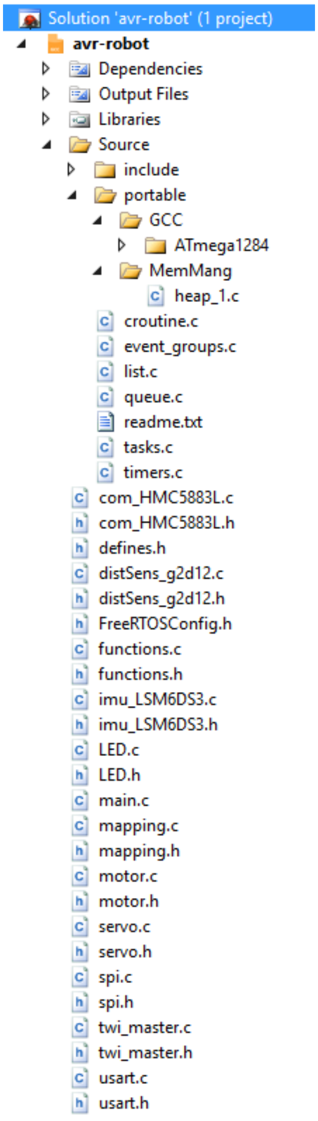
To inspect the current project structure, Atmel Studio 7 was used to set up and compile the AVR source code. Because no updated instructions to do this was available for Atmel Studio 7, this is given below:

1. Open Atmel Studio 7 and choose File $\rightarrow$ New $\rightarrow$ Project.
2. Choose GCC C Executable Project and give it the name ”main”. Set the solution name to something recognizable of your choosing.
3. Choose ATmega1284P under the ATmega device family and click OK.
4. The project should be created, and a file overview should be visible in the Solution Explorer (if Solution Explorer is not visible click View in the toolbar and choose Solution Explorer).
5. Right-click ”main.c” in the solution explorer and open file location.
6. Copy all .c and .h files from the AVR source code into this folder using the same folder structure as the source code (in the MemMang folder only `heap_1.c` is needed).
7. Reload the main file as requested by Atmel Studio.
8. In the ”Solution Explorer” toolbar click ”Show all files” and mark all files and folders imported from the source code, right click $\rightarrow$ Include in Project. The file overview should now look like figure 5.9.
9. To make sure the IDE’s internal makefile knows where to look for your files, select any source file in your project and click Project $\rightarrow$ Properties (ALT+F7) in the toolbar.
10. Include the path to all source folders under AVR/GNU C Compiler $\rightarrow$ Directories as seen in figure 5.9.
11. The project should now compile by clicking Build $\rightarrow$ Build Solution (F7) in the toolbar.

To evaluate the current code structure on the AVR, it was decided to generate dependency graphs using Doxygen.

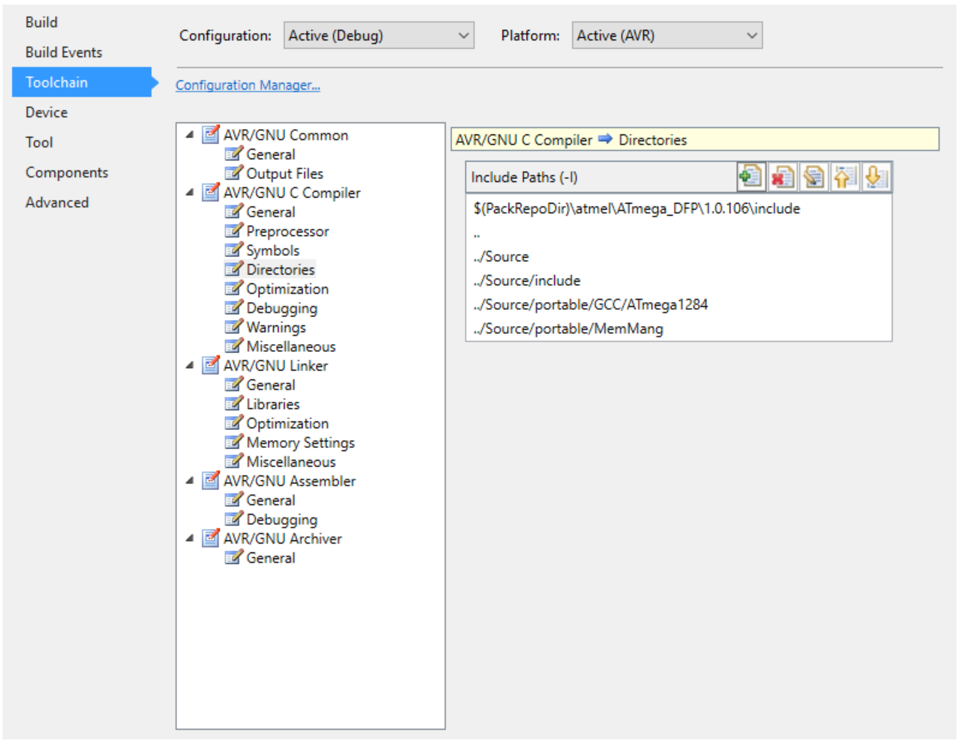
Using doxywizard, a manual for the AVR source code was created. Here it became clear that the people behind the FreeRTOS source actually use Doxygen-syntax in there source code comments. This is lucky as it means a lot of work documenting FreeRTOS for this project will already be done if Doxygen is chosen as an in-code documenting tool.

It is also possible to extract dependency graphs for all files in the project, this makes it easier for developers to create abstraction layers and making sure all dependencies follow the rules set the abstraction layers.



```
Solution 'avr-robot' (1 project)
  avr-robot
    Dependencies
    Output Files
    Libraries
    Source
      include
      portable
        GCC
          ATmega1284
        MemMang
          heap_1.c
      croutine.c
      event_groups.c
      list.c
      queue.c
      readme.txt
      tasks.c
      timers.c
    com_HMC5883L.c
    com_HMC5883L.h
    defines.h
    distSens_g2d12.c
    distSens_g2d12.h
    FreeRTOSConfig.h
    functions.c
    functions.h
    imu_LSM6DS3.c
    imu_LSM6DS3.h
    LED.c
    LED.h
    main.c
    mapping.c
    mapping.h
    motor.c
    motor.h
    servo.c
    servo.h
    spi.c
    spi.h
    twi_master.c
    twi_master.h
    usart.c
    usart.h
```

**Figure 5.8:** Solution explorer code structure.



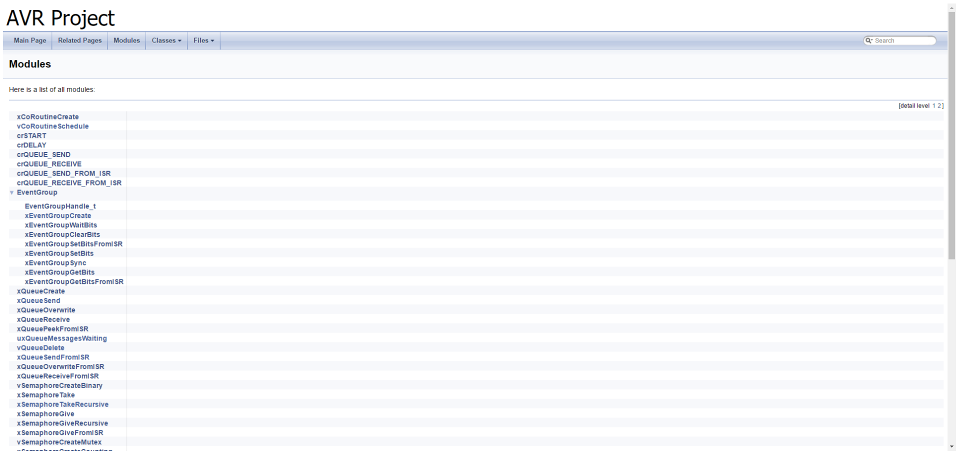
Configuration: Active (Debug)    Platform: Active (AVR)

AVR/GNU C Compiler ➡ Directories

Include Paths (-I)

```
$(PackRepoDir)\atmel\ATmega_DFP\1.0.106\include
..
../Source
../Source/include
../Source/portable/GCC/ATmega1284
../Source/portable/MemMang
```

**Figure 5.9:** Include all source folders so the compiler knows where to look.



**Modules**

Here is a list of all modules:

[detail level 1 2]

```
xCoRoutineCreate
vCoRoutineSchedule
crSTART
crDELAY
crQUEUE_SEND
crQUEUE_RECEIVE
crQUEUE_SEND_FROM_ISR
crQUEUE_RECEIVE_FROM_ISR
▼ EventGroup
    EventGroupHandle_t
    xEventGroupCreate
    xEventGroupWaitBits
    xEventGroupClearBits
    xEventGroupSetBitsFromISR
    xEventGroupSetBits
    xEventGroupSync
    xEventGroupGetBits
    xEventGroupGetBitsFromISR
xQueueCreate
xQueueSend
xQueueOverwrite
xQueueReceive
xQueuePeekFromISR
uxQueueMessagesWaiting
vQueueDelete
xQueueSendFromISR
xQueueOverwriteFromISR
xQueueReceiveFromISR
vSemaphoreCreateBinary
xSemaphoreTake
xSemaphoreTakeRecursive
xSemaphoreGive
xSemaphoreGiveRecursive
xSemaphoreGiveFromISR
vSemaphoreCreateMutex
```

**Figure 5.10:** Documented FreeRTOS modules appeared after running Doxygen.

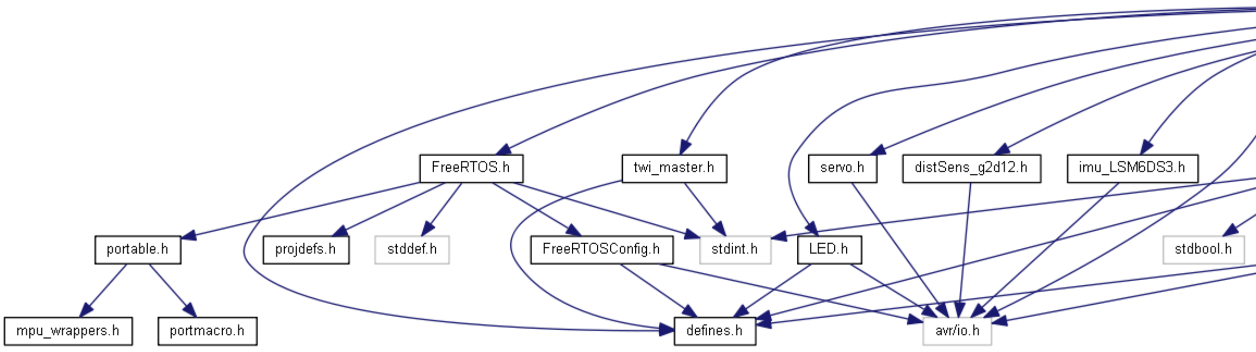


**Figure 5.11:** Main.c dependencies.

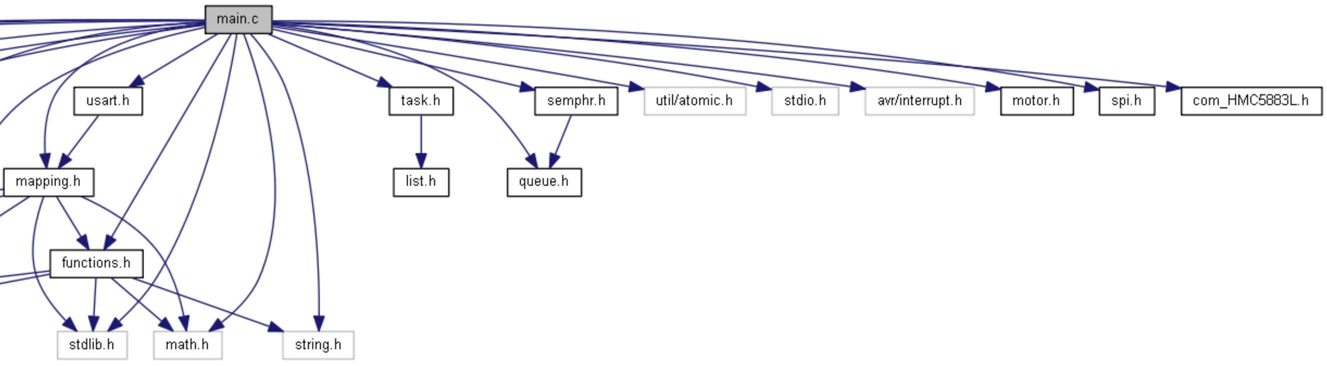


**Figure 5.11:** Continued from last page.

## 5.4 Evaluation of Doxygen

Doxygen was introduced to the other Lego robot developers at a monthly meeting. The pros and cons were discussed and summarized below.

### Pros:

- Finally a central place to find documentation of the project and source code.
- No need to look through multiple older theses to find the answer for what a certain part of the code does.
- Able to look up functions directly instead of having to iterate through the code with pen and paper to understand complex functions.
- Since Doxygen works by generatic documentation from comments in the code there is no need to keep a seperate document updated, simply udpate your comment when making change to a function and Doxygen will update the documentation.
- FreeRTOS is the largest part of the source code and is already fully documented with Doxygen.
- Leaves you with a search-able PDF in which to find documentation of functions and classes fast.

### Cons:

- Developers will have to learn Doxygen syntax in order to write comments in such a way that Doxygen compiles it.
- Might make writing code a bit more time-consuming since comments have to be written for each function so that that the function gets documented.

Going through this list, it was agreed on by all developers and the supervisor present that using Doxygen for documentation was a good idea.

Chapter 6

# Component Drivers and Application

## 6.1 Components

To enable the robot to sense and move around its surroundings for the SLAM algorithm multiple components must be connected and installed to the EV3 robot. Since the AM1808 running FreeRTOS has no built-in way to automatically talk with these components, drivers must be written for them.

The following components will need drivers for the AM1808 running FreeRTOS:

- LSM6DS3 - 6DOF IMU, Gyro and accelerometer
- nRF51 BLE dongle
- 4x GP2Y0A21YK Sharp IR sensors
- HMC5883L electronic compass
- 3x EV3 Large Servo Motors

These components will be connected to the EV3 brick. The circuit board used for the AM1808 MCU does not have any GPIO pins available for access by the user other than the ones going out to the four sensor ports. Since four IR sensors are required for measuring distance in all directions for the application, as well as three motors (one for each wheel, one for rotating the IR sensor tower), this means there are not enough ports for all the sensors and motors to be connected. This same problem was encountered by Lien (2017) on the NXT robot. To solve this Lien created an IO circuit card for interfacing all the sensors through one NXT sensor port. Since the EV3 sensor ports are backwards compatible with all NXT peripherals, this means the same card should work with the EV3.

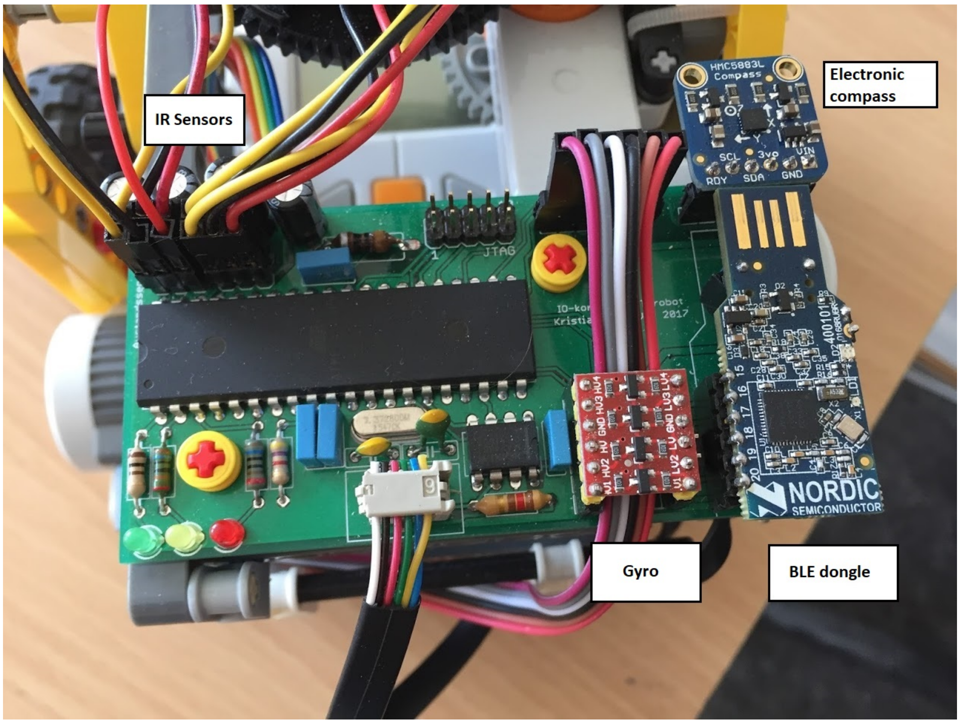


**Figure 6.1:** NXT IO card with sensors connected.

Since creating a new card for the EV3 is time-consuming it was decided to create another NXT card and fit this to the EV3 using the software created by Lien (2017). Because of time-limit concerns it was decided after conferring with supervisor Tor Onshus to postpone the creation of this card to further work in this thesis. Once the card is created, the same software to interface the sensors with the onboard MCU on the NXT should be implementable on the EV3 with minimum tweaking. Since both devices now run the same OS, and ports connections and wiring are the same, most likely only structs and the registers' bitmasks will have to be tweaked from the NXT to EV3 for communication with the sensors. The information and source code required for this is given by Lien (2017) and

Texas Instruments (2014).

## 6.2 Application

With the hardware and OS layer done, the application was moved from the AVR and to the EV3. Using the same tasks created for the AVR, all calls to FreeRTOS completes successfully as the FreeRTOS and hardware implementation on the EV3 does not change the interface seen from the application layer in any way. This means the application can be moved between the other robots running FreeRTOS at will. However, since the sensors are not implemented without an IO card on the EV3, direct calls from the application to the sensors on the EV3 have not been implemented. This will have to be configured once the IO card is made.

Chapter 7

# Results

With the hardware layer and FreeRTOS implemented, the application could be transferred to the EV3 with minimal work required.

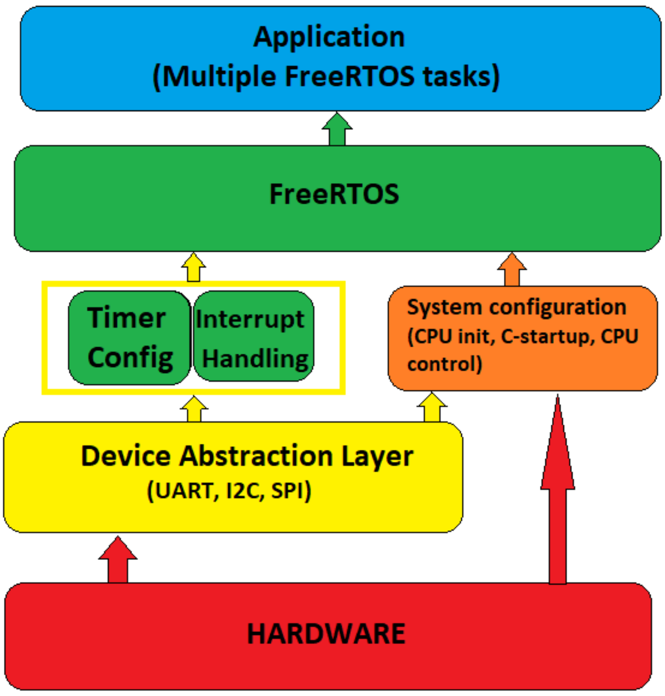
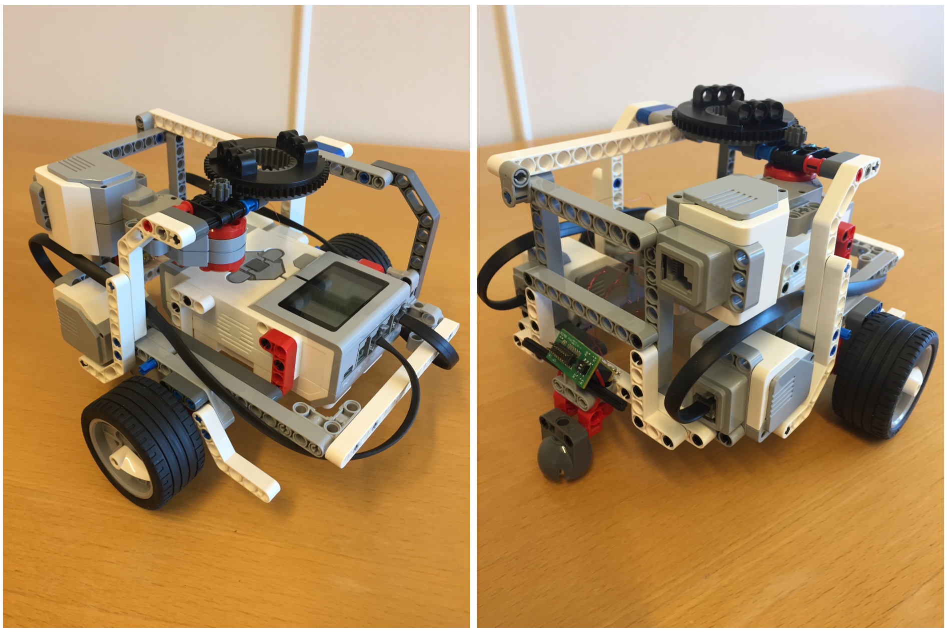


**Figure 7.1:** Finished abstraction layers.

The hardware interface hides all structs and register bitmasks behind an abstraction layer, so all peripherals on the EV3 can be called by the developer using the new driver functions and documentation without looking up peripheral data in the AM1808 datasheet (though it is always recommended to keep the datasheet on hand if deeper understanding is required).

Using this abstraction layer, FreeRTOS was ported to the AM1808 with a new port file written for the ARM9 based CPU. Care was taken to keep the FreeRTOS interface as close to original as possible to ensure the application would not notice a difference with the FreeRTOS implementation on the EV3 compared to the other robots.

With the missing IO card it was not possible to connect all the sensors required by the application for the EV3 to function fully as a SLAM device, still the EV3 was built using the AVR as reference. The JTAG was connected to the Lego body's backside, and the motors were used for the wheel movement and sensor tower. Space for the IO card was reserved.



**(a)** Front. **(b)** Back

**Figure 7.2:** Lego EV3 robot body.

# Chapter 8

# Discussion

## 8.1 Status of the Robots

### 8.1.1 EV3

The work on the EV3 robot has been focused on creating a new software base for the application to be added to. Since the EV3 is the only robot which does not have an available port for FreeRTOS used by the other systems, this had to be ported to the EV3 which was very time-consuming. Since the EV3 natively runs Linux, a new hardware layer where all AM1808 microcontroller peripherals were available to the end-user had to be created.

In addition to this, a new documenting tool, Doxygen, was used to document the EV3's source code. The software's features and performance were evaluated by all developers working on the SLAM robots and decided it was a good idea to use.

### 8.1.2 NXT

The work on the NXT this semester has revolved around implementing a new IO card in order to enable all the sensors required for the SLAM application to be connected to the EV3. With the new IO card, more sensors could be connected to the NXT and a communication protocol between the MCU on the IO card and the NXT MCU was created for communication.

NXT was also running nxtOSEK which was replaced with FreeRTOS using an available FreeRTOS port. Device drivers for the NXT written for nxtOSEK could be re-used in FreeRTOS ensuring LCD and RS-485 compatibility.

Finally, a new communication protocol with the server was implemented based on the OSI-model.

### 8.1.3 AVR

The work done by Lars Marius Strande's master's thesis on the AVR robot was not available at the time of writing this thesis. The current status of the AVR is unknown.

### 8.1.4 Arduino

The work on the Arduino robot has revolved around improving the SLAM algorithm. This was done by checking all the sensors for faults ant testing their accuracy. The gyro sensor was found to have a slight bias resulting in increasingly wrong measurements, this was corrected by implementing a poll for mean value method instead that improved the accuracy. The compass was evaluated and found to be give very noisy data. In the end it was decided to remove the compass from the system for better performance until a better implementation can be found. Finally, the accelerometer was found to give valuable, but very noisy data. Improvements on the sensor were unsuccessful, but it is suggested to look more into this given the value of the data.

One of the wheels of the Arduino was also found to be faulty and was repaired to avoid slipping and false data of the wheel positions.

Finally, the application instead was changed from using a distance and heading model, to specific coordinates. This was done by converting the heading and distance commands the server sends into specific coordinates, then use a similar controller to minimize the distance to the target. This improved the navigation.

## 8.2 Further Work

Suggestions for further work on the EV3 is listed below:

- Produce and implement an IO card like the one for the NXT on the EV3 to ensure all sensors can be connected.
- Write a driver for communication with the IO card (most likely the code on the NXT can be re-used, only changing the EV3 specific addresses).
- Write a driver for the LCD screen on the EV3 to post status messages and debug information (could be worth taking a look at the the EV3DEV OS source code for the EV3 for inspiration on how to do this).
- Write a driver for the USB connector on the EV3 in order to be able to connect the BLE dongle here instead (though this will require the EV3's code for the IO to differ more from the NXT, pros and cons should be evaluated).

## 8.3 Final Thoughts

Right now there are four robots in the SLAM team. The NXT and EV3 are both implementations on the Lego Mindstorms series of robots. As can be seen by multiple theses and projects working on these robots, a lot of time is spent working around the limitations set by Lego on developing the hardware. More specific this means lacking access to GPIO pins directly, difficulty to connect a JTAG interface (especially on EV3) and little resources available online on bare-metal programming for these devices because of the previously mentioned reasons (most implementations online are adaptations of the system already running on the Lego bricks). The goal is of course to end up with a system that should be easily implementable on multiple EV3s/NXTs, but with the amount of work that is required to make a Lego Mindstorms brick ”developer friendly” and adding custom operating systems, IO cards and hardware, it should be evaluated if future robots should be based on more open systems like the Raspberry PI to give the developer more freedom and creativity since any new EV3 or NXT will require a not insignificant amount of work before being ready to run the application.

It can also be argued that going with more open systems would enable the student to just use finished implementations found online and learn very little. Since the actual goal of this project is to learn and become a better engineer, a combination of the two approaches might be the best bet.

# Bibliography

Bakken, 2008. Bygge og programmere ny legorobot. Tech. rep., Norwegian University of Science and Technology.

Ese, E., 2015. Fjernstyring av legorobot. Tech. rep., Norwegian University of Science and Technology.

Ese, E., 2016. Sanntidsprogrammering på samarbeidande mobil-robotar. Master's thesis, Norwegian University of Science and Technology.
URL `http://hdl.handle.net/11250/2403570`

feilipu, 2011. FreeRTOS and libraries for AVR ATmega with Eclipse IDE.
URL `https://feilipu.me/2011/09/22/freertos-and-libraries-for-avr-atmega/`

Heesch, D. v., 2017. Doxygen.
URL `http://www.stack.nl/~dimitri/doxygen/`

Helders, K. Z., 2016. Remote Control of Lego Robots. Tech. rep., Norwegian University of Science and Technology.

Homestad, T. K., 2013. Fjernstyring av legorobot. Master's thesis, Norwegian University of Science and Technology.
URL `https://brage.bibsys.no/xmlui/handle/11250/260903`

IAR Systems, 2017. Iar systems and atmel partnership.
URL `https://www.iar.com/iar-embedded-workbench/partners/atmel/`

Lien, K., 2017. Embedded utvikling på en fjernstyrt kartleggingsrobot. Master's thesis, Norwegian University of Science and Technology.

Orakeltjenesten, NTNU IT, 2013. Formatting your Master's thesis in Microsoft Word. URL `https://innsida.ntnu.no/documents/10157/124399535/Formatting+your+master%27s+thesis+in+Microsoft+Word/03c81406-f69f-4caf-8cbd-cfe3bc9c6300`

Real-time Engineers Ltd., 2016. FreeRTOS Customisation. URL `http://www.freertos.org/a00110.html`

Real Time Engineers ltd., 2017. Modifying a FreeRTOS Demo. URL `http://www.freertos.org/porting-a-freertos-demo-to-different-hardware.html`

Skjelten, H., 2004. Fjernnavigasjon av lego-robot. Tech. rep., Norwegian University of Science and Technology.

Spectrum Digital, Inc., 2012. Xds200 quick start guide. URL `http://emulators.spectrumdigital.com/files/XDS200_QSG.pdf`

Steuper, 2015. Bygge og programmere ny legorobot. Tech. rep., Norwegian University of Science and Technology.

Texas Instruments, 2014. AM1808 ARM Microprocessor Datasheet. URL `http://www.ti.com/lit/ds/symlink/am1808.pdf`

Texas Instruments, 2015. StarterWare. URL `http://processors.wiki.ti.com/index.php/StarterWare`

The LEGO Group, 2013. EV3 Main Hardware Schematics.

# Appendices

Appendix A

# Source Code Samples

## A.1 CCS Assembly Code for FreeRTOS Port

```
        .thumb

        .ref  pxCurrentTCB
        .ref  vTaskSwitchContext
        .ref  ulMaxSyscallInterruptPriority

        .def  xPortPendSVHandler
        .def  ulPortGetIPSR
        .def  vPortSVCHandler
        .def  vPortStartFirstTask
        .def  vPortEnableVFP

NVICOffsetConst:                                      .word   0xE000ED08
CPACRConst:                                                 .word   0xE000ED88
pxCurrentTCBConst:                                    .word   pxCurrentTCB
ulMaxSyscallInterruptPriorityConst:  .word  ulMaxSyscallInterruptPriority

; ---------------------------------------------------------------------

        .align  4
ulPortGetIPSR:  .asmfunc
        mrs r0, ipsr
        bx  r14
        .endasmfunc
```

```
 ; -----------------------------------------------------------

        .align 4
vPortSetInterruptMask: .asmfunc
        push {r0}
        ldr r0, ulMaxSyscallInterruptPriorityConst
        msr basepri, r0
        pop {r0}
        bx r14
        .endasmfunc
; -----------------------------------------------------------

        .align 4
xPortPendSVHandler: .asmfunc
        mrs r0, psp
        isb

        ;/* Get the location of the current TCB. */
        ldr     r3, pxCurrentTCBConst
        ldr     r2, [r3]

        ;/* Is the task using the FPU context?  If so, push high vfp registers. */
        tst r14, #0x10
        it eq
        vstmdbeq r0!, {s16-s31}

        ;/* Save the core registers. */
        stmdb r0!, {r4-r11, r14}

        ;/* Save the new top of stack into the first member of the TCB. */
        str r0, [r2]

        stmdb sp!, {r3}
        ldr r0, ulMaxSyscallInterruptPriorityConst
        ldr r1, [r0]
        msr basepri, r1
        dsb
        isb
        bl vTaskSwitchContext
        mov r0, #0
        msr basepri, r0
        ldmia sp!, {r3}

        ;/* The first item in pxCurrentTCB is the task top of stack. */
        ldr r1, [r3]
        ldr r0, [r1]
        ;/* Pop the core registers. */
        ldmia r0!, {r4-r11, r14}

        ;/* Is the task using the FPU context?  If so, pop the high vfp registers
        ;too. */
        tst r14, #0x10
        it eq
        vldmiaeq r0!, {s16-s31}

        msr psp, r0
        isb
        bx r14
        .endasmfunc

; ------------------------------------------------------------------

        .align 4
vPortSVCHandler: .asmfunc
        ;/* Get the location of the current TCB. */
        ldr     r3, pxCurrentTCBConst
        ldr r1, [r3]
        ldr r0, [r1]
        ;/* Pop the core registers. */
        ldmia r0!, {r4-r11, r14}
        msr psp, r0
        isb
        mov r0, #0
        msr     basepri, r0
        bx r14
        .endasmfunc

; ------------------------------------------------------------------

        .align 4
vPortStartFirstTask: .asmfunc
        ;/* Use the NVIC offset register to locate the stack. */
        ldr r0, NVICOffsetConst
        ldr r0, [r0]
        ldr r0, [r0]
        ;/* Set the msp back to the start of the stack. */
        msr msp, r0
        ;/* Call SVC to start the first task. */
        cpsie i
        cpsie f
        dsb
        isb
        svc #0
        .endasmfunc

; ----------------------------------------------------------------

        .align 4
vPortEnableVFP: .asmfunc
        ;/* The FPU enable bits are in the CPACR. */
        ldr.w r0, CPACRConst
        ldr     r1, [r0]

        ;/* Enable CP10 and CP11 coprocessors, then save back. */
        orr     r1, r1, #( 0xf << 20 )
        str r1, [r0]
        bx      r14
        .endasmfunc

        .end

; ----------------------------------------------------------------
```

## A.2 AM1808 CPU Initialization

```
;******************************************************************************
;
; init.S - Init code routines
;
; Copyright (C) 2010 Texas Instruments Incorporated - http://www.ti.com/
; All rights reserved.

; Modified for use with EV3 by Konstantino Z. Helders, Spring 2017, NTNU
;
;******************************************************************************

        MODULE EXCEPTIONS
;**************************** Global Symbols *********************************
        PUBLIC Entry
        IMPORT start_boot
        IMPORT __iar_data_init3

        SECTION IRQ_STACK:DATA:NOROOT(3)
        SECTION FIQ_STACK:DATA:NOROOT(3)
        SECTION SVC_STACK:DATA:NOROOT(3)
        SECTION ABT_STACK:DATA:NOROOT(3)
        SECTION UND_STACK:DATA:NOROOT(3)
        SECTION CSTACK:DATA:NOROOT(3)
        SECTION SYSTEMSTART:CODE (4)
;************************ Internal Definitions ******************************

;
; to set the mode bits in CPSR for different modes
;
MODE_USR DEFINE 0x10
MODE_FIQ DEFINE 0x11
MODE_IRQ DEFINE 0x12
MODE_SVC DEFINE 0x13
MODE_ABT DEFINE 0x17
MODE_UND DEFINE 0x1B
MODE_SYS DEFINE 0x1F

I_F_BIT DEFINE 0xC0

;***************************************************************************
; This source file is assembled for ARM instructions
        CODE32
;***************************************************************************
;
; The reset handler sets up the stack pointers for all the modes. The FIQ and
; IRQ shall be disabled during this. Then, clearthe BSS sections, switch to the
; main() function.
;
Entry:
;
; Set up the Stack for Undefined mode
;
         MSR    cpsr_c, #MODE_UND|I_F_BIT        ; switch to undef  mode
         LDR    sp,=SFE(UND_STACK)              ; load the stack address
;
; Set up the Stack for abort mode
;
         MSR    cpsr_c, #MODE_ABT|I_F_BIT        ; Change to abort mode
         LDR    sp,=SFE(ABT_STACK)               ; load the stack address
;
; Set up the Stack for FIQ mode
;
         MSR    cpsr_c, #MODE_FIQ|I_F_BIT        ; change to FIQ mode
         LDR    sp,=SFE(FIQ_STACK)               ; load the stack address
;
; Set up the Stack for IRQ mode
;
```

```
        MSR    cpsr_c, #MODE_IRQ|I_F_BIT          ; change to IRQ mode
        LDR    sp,=SFE(IRQ_STACK)                 ; load the stack address
;
; Set up the Stack for SVC mode
;
        MSR    cpsr_c, #MODE_SVC|I_F_BIT          ; change to SVC mode
        LDR    sp,=SFE(SVC_STACK)                 ; load the stack address
;
; Set up the Stack for USer/System mode
;
        MSR    cpsr_c, #MODE_SYS|I_F_BIT          ; change to system mode
        LDR    sp,=SFE(CSTACK)                    ; load the stack address

;
; Clear the BSS section here. Use IAR library functions to init data sections.
; Do so because the linker script is not friendly enough for access this section
;
        LDR    r10,=__iar_data_init3
        BLX    r10
        MOV    r0, #0                             ; no arguments to main
        MOV    r1, r0                             ; zero registers r0-r3 and fp
        MOV    r2, r0
        MOV    r3, r0
        MOV    r7, r0
        MOV    r12, r0

;
; Enter the main function. The execution still happens in system mode
;
Enter_main:
        LDR    r10,=start_boot                    ; Get the address of main
        MOV    lr,pc                              ; Dummy return to main
        BLX     r10                                ; Branch to main
        SUB    pc, pc, #0x08                      ; looping

;
; End of the file
;
        END
```

Appendix B

# Description of Digital Attachements

Below is a list of digital attachements to this thesis:

1. Videos featuring testing of working modules.
2. EV3 source code.
3. Doxygen reference manual.
4. Datasheets and relevant technical data.
5. Previously handed in reports and theses.