Irfan Šehić, BSc

Development and Implementation of a Dive Computer

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass. Prof. Dipl. -Ing. Dr. techn. Steger Christian
Institute for Technical Informatics

Univ.-Prof. Dipl.-Ing. Dr. techn. Deutschmann Bernd
Institute of Electronics

Graz, December 2015

AFFIDAVIT

Kurzfassung

Ziel dieser Masterarbeit war die Entwicklung einer Platform, die man als Basis für einen modernen Tauchcomputer verwenden kann. Die erste Phase des Projekts umfasst die Gestaltung des Tauchcomputers, betrachtet von zwei Standpunkte: Hardware und Software. In der zweiten Phase hat man sich mit der Umsetzung von diesen beiden Aspekte beschäftigt. Es wurden industriekonformen Bauteile verwendet, inklusive ARM-Prozessor, einen Touchscreen der unter Wasser funktioniert, DDR3 Arbeitsspeicher, hochpräzise Sensoren, usw.
Außerdem wurde ein Linux-basiertes Betriebssystem in Betrieb genommen. Während der Umsetzungsphase dieser Arbeit wurden mehrere Proof of Concept Software-Module implementiert. Die Ergebnisse der Masterarbeit geben wertvolle Informationen, wie man die Tauchcomputer Modernisierung schaffen kann, indem man die Weiterentwicklung von anderen Technologien wie die Halbleiter-, Sensor-, Anzeige-, Software-, und Batterietechnologie ausnutzt.

Abstract

The purpose of this project was to design and build a platform which can be utilized for a modern era dive computer. The first phase of the project includes the design of the dive computer, from two standpoints: hardware and software. The second phase was the implementation and integration of the these two aspects. As for the components, industry compliant parts were used, ranging from an ARM based processor to the high precision sensors. On top of these parts, a Linux based operating system was used. During the implementation phase of this project, several proof of concept software modules were implemented. This research will provide valuable information on how we can modernize the dive computers, with the use of the progress made in other technologies like the semiconductor, sensor, display, software, and battery technology.

Danksagung

Diese Masterarbeit wurde im Studienjahr 2014/2015 an dem Institut für Technische Informatik und an dem Institut für Elektronik an der Technischen Universität Graz durchgeführt.

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herr Ass.Prof. Christian Steger, der meine Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Ebenfalls möchte ich mich bei Univ.-Prof. Deutschmann Bernd und Ass.Prof. Gunter Winkler bedanken, die mir mit viel Hilfsbereitschaft zur Seite standen. Bedanken mÖchte ich mich fr die zahlreichen Ideen, die dazu beigetragen haben, dass diese Masterarbeit in dieser Form vorliegt.

Außerdem gebührt mein Dank Herr Dr. Arne Sieber von der Firma Seabear Diving Technology GmbH aus Graz, der für die Material- und Bauteilkosten gesorgt hat.

Ich bedanke mich auch bei den Mitarbeitern des Institutes für Elektronik, die mir bei der praktischen Umsetzung einigen Aspekten dieser Arbeit geholfen haben.

Abschließend möchte ich mich bei meinen Eltern Vahid und Enisa bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für meine Sorgen hatten.

Graz, im November 2015 Irfan Sehic

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A dive computer and a decompression table share a common purpose. They both allow the diver to determine the decompression schedule for a certain dive profile and the breathing gas. The advantage of the dive computer lies in the fact that it performs a continuous calculation of the partial pressure of inert gases based on the actual time and depth profile of the dive [5], whereby a decompression table assumes a square dive.

There is an abundance of dive computers on the market, ranging from entry-level ones, to those used by the army. They differentiate in various features, like the maximum depth they operate at, battery life, gas mixtures and decompression algorithms supported, etc.

In this paper, we present a hardware and software platform, which can be used as a basis for a dive computer. From the hardware point of view, it features an AM335x microprocessor unit (MPU) with its required peripherals, such as a power management unit (PMIC), and necessary sensors including a compass and a barometer. It also includes a a 3.5″ LCD, a touchscreen, a wireless module which can be used to exchange data with the PC, and an SD card which holds the Linux kernel and the filesystem.

From the software standpoint, the dive computer features a Debian Linux distribution with an emphasis on a short boot time and reliability. On top of Linux, custom components are present, such as a proof of concept graphical user interface (GUI), drivers for the sensors, and several other software modules. The interestingness of the dive computer also lies in the fact that it features a touchscreen which can be operated under water, since it is filled with oil, which does not compress as much as air, when under pressure.

It is also noteworthy that an important concern during our design was the battery life of the dive computer, as the hardware components, if not properly optimized, can draw a lot of power, which can reduce the dive time.

As a starting point for the development of our system, an evaluation board called AM335x Starter Kit, by Texas Instruments was chosen. The AM335x Starter Kit provides an affordable platform for the evaluation of the Sitara ARM Cortex-A8 AM335x [6], which we decided to use in our design. Texas Instruments provided schematics and the board layout files, which were then used to develop a custom board for the dive computer.

For the development of the dive computer, the AM335x Starter Kit was stripped off the unnecessary features, such as the dual Gigabit Ethernet controller, audio codec, USB hub, etc. A different DDR3 chip with less capacity than the one used in the AM335x Starter Kit design was chosen in our design. Besides this, appropriate sensors for pressure and temperature, and a compass module were integrated in the system. A WiFi module, 3.5″ LCD and a touchscreen were used as output and input for the data.

As for the software aspect, we used Debian as our operating system, since it offered us plenty room for customizability. Several software modules were then implemented on top of the operating system, which showed the possibilities of the hardware. These modules include the GUI, reading out the sensor data, and accessing some of the boards features over the WiFi. Other software aspects, like customizing the bootloader, configuring the kernel for the hardware, were of no less importance either.

For such a device, power consumption is also of importance, since it runs on a limited power supply, i.e. a battery. For this reason, we also looked into the power saving capabilities of the hardware we used, specifically the AM335x and the DDR3 memory.

This paper is divided in several chapters. Chapter 2 gives an insight in the work related to our project. Chapter 3 gives a glimpse in the modern dive computers. Chapter 4 and 5 give information about the hardware and software design of the dive computer platform, respectively, and Chapters 6 and 7 show how the implementation of the ideas from the design phase went by. In Chapter x, we inform the reader on the eventual path this project might go.

In Chapter 9, we draw some conclusions from this work.

Chapter 2

Related Work

This chapter investigates the work related to the content of this paper, from the hardware and software points of view.

Albeit, finding scientific papers whose content directly deals with the same matter to ours, namely, developing a hardware and software platform for a dive computer, was not an easy task, as most of such works are done commercially, so their implementation is kept private.

For this reason, we turned to the resources which discuss more general topics which appear in our work. We found a lot of information concerning the software aspect of our work, such as the books about Linux for embedded systems and its optimization. We also found a lot of general information about designing printed circuit boards (PCB) and guidelines for various aspects such as a high-speed design, routing guidelines, etc. We first focus on the software resources which were useful to us during the course of this project.

Building Embedded Linux Systems [7], a book by Yaghmour, K. et al. offered an in-depth guide to understanding and building embedded systems based on Linux. This book was a good guide on how to configure and use the packages needed for compiling, configuring and testing Linux based operating systems. It provides a good documentation on basic concepts such as a processor architecture, cross compiling, a general overview of an embedded Linux system, and its real life application. Besides the general concepts, we got a deeper understanding of the importance of the Linux kernel, and what its role is. Since configuring and tailoring the kernel to our needs is a no trivial task, this book served as a good information source on what kernel versions are available, and how to pick a suitable version for our embedded device, and how to configure it properly. Besides the kernel, we also acquired a good understanding of the root filesystem, and what the purpose of the filesystem is. Basic concepts like the location of the binaries, bootloader files, configuration files, device and media mount points were also of importance to us. The book also gives a vague overview of the bootloader, and what role it plays.
Another good read about Linux for embedded systems was *Linux for Embedded and Real-time Applications* [8], by Abbott D. Similar to the first book, it gave us another good reference for general information on the operating system we have chosen, Linux. These two books were important for our understanding of how the operating system interacts with hardware, and how the filesystem interacts with Linux. This way, we can successfully integrate this important

building block in our project.

As for the hardware development, we found many resources concerning both the general references for PCB design, but also some more specific topics, like a high-speed design, which we will need when dealing with the processor and RAM, and design guidelines for certain interfaces like the Joint Test Action Group (JTAG) interface, or Inter-Integrated Circuit (I^2C) interface.
The Art Of Electronics, 2nd Edition [9], was a great reference book concerning electronics. It provides an in-depth information starting from the basics, to the more advanced topics. This book was used as a general reference to fill in the blanks in topics which were directly related to our work.
One of the most useful resources for the PCB design we had a chance to use were the lecture materials for the lecture *Development of Electronic Systems* at the Technical University of Graz, held by the professors of the Institute for Electronics. The main idea of the lecture was the development of market-ready electronic devices. Several topics were interesting for our work, including the development of hardware which respected the laws of electromagnetic compatibility (EMC), security and reliability. Other topics, such as component placement and board zoning, micro controllers, sensors and sensor interfaces were also interesting as an entry point for a deeper understanding of these broad topics. We go in depth about these materials in Chapter 6.1.
Another important resource for the PCB design we found useful is a collection of guidelines provided by Texas Instruments [10]. This relatively short read is a good resource which gives many practical guidelines and information which needs to be considered while designing a PCB. Topics such as the the causes of electromagnetic interference (EMI), board layout planning, information about electric signals were all covered by this work.

Deco for Divers [11] was a good informal read about diving and decompression, This book gives a comprehensive guide to the basics of decompression theory, and is targeted to an average diver. From this guide, we were able to extrapolate what information should be provided by a dive computer, and how this information is used by the computer and the diver.

Chapter 3

State of the Art

In this chapter, we briefly review the products on the market which are related to the one we are presenting in this paper. This includes several dive computers which are conceptually similar, i.e. have a large LCD, and offer an appealing user interface.

3.1 Modern Dive Computers

The first device that comes to mind in a discussion about modern dive computers is the Suunto Eon Steel. Besides this one, Seabear H3 is also a notable example of what a modern dive computer should offer. We will present these two devices in Chapter 3.1.1 and Chapter 3.1.2 respectively.

3.1.1 Suunto Eon Steel

Suunto Oy is a company with a long tradition, located in Vantaa, Finland, and it has more than 300 employees worldwide.
This company is known for its multi-functional electronic wristwatches, which provide a variety of sensors and functions, such as a compass, altitude, GPS location, etc. Suunto also makes dive computers, and has several of them in their portfolio. The one that is of most interest to us is the Eon Steel, visible in Figure 3.1. As we see in the Figure 3.1, this dive computer has a relatively large display with sharp colours. Besides the display, we also see three buttons on the front side. Looking through the manual gave us a good information on what this device can offer. It has two main views in surface and dive mode, and several other where some sensor data is shown, such as the compass. Switching between the surface and dive mode is done automatically if the device is deeper than 1.2 meters in water [12]. When the device is under water, the interface changes to the dive mode, shown in Figure 3.2. The interface we see uses neon colours which are well visible on a black surface. Here we see the actual depth, active gas, dive time, tank pressure, and some other important parameters. The bar on the side indicates the ascent rate.

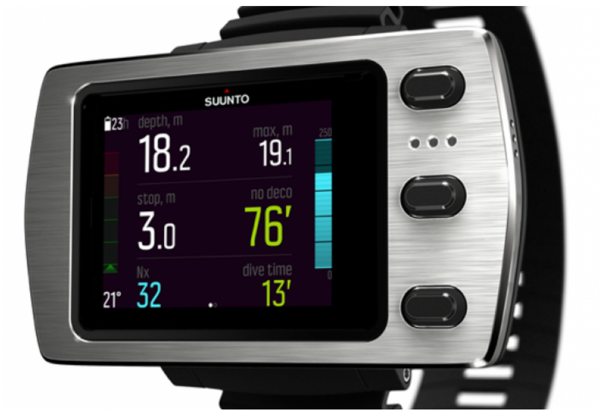


Figure 3.1: Suunto Eon Steel dive computer [12]

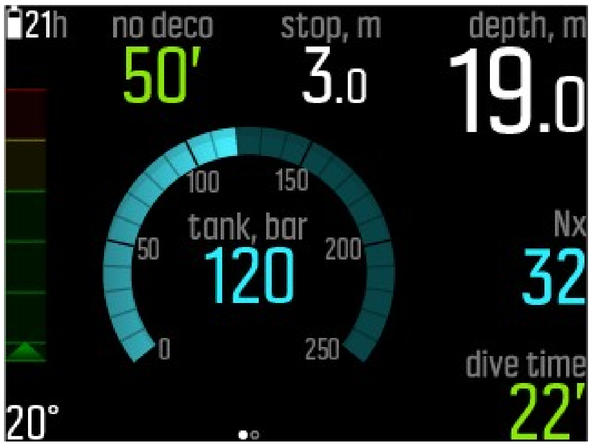


Figure 3.2: Suunto Eon Steel dive mode [12]

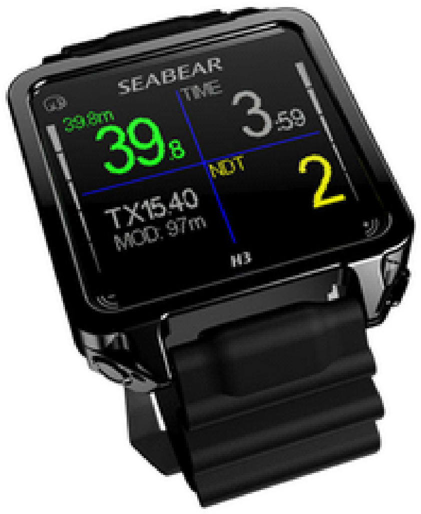


Figure 3.3: Seabear H3 dive computer [13]

This dive computer has colour coded alarms, warnings and notifications. If the diver breaks the decompression ceiling or the partial oxygen pressure is not in a safe margin, an alarm will be shown.

Besides these user interface elements which show the information, the Suunto Eon Steel offers a wide variety of options, such as using a customized GUI which can be customized on a PC, viewing diving history, setting gas mixtures and so on.

3.1.2 Seabear H3

Seabear GmbH is a young company situated in Graz, Austria. They specialize in dive computer research and development. The dive computer of a particular interest to us is the Seabear H3. It features a 1.7″ OLED screen protected by a sapphire window. The case is made of a 316L stainless steel.
We see a Seabear H3 unit in Figure 3.3. An interesting fact about the H3 is the fact that it has an NFC chip on-board, where the diver can store one of the dive logs on the H3, which can than be read out by an near field communication (NFC) equipped smart phone.
There are two sensors mounted on the H3 PCB, namely a pressure sensor and an compass. The hardware is powered by a 400 mAh Li-Ion battery. As for the software, this dive computer also offers several screens to choose from. When diving, one screen gives an overview of the most important parameters, like the depth, no-stop time, dive time and so on. Besides this screen, there are several others. One can e.g. choose the layout with the compass while underwater. As for the decompression algorithm, H3 uses a Buehlmann ZH-L16 algorithm with gradient factors [13].

Other options, like an automatic activation of the dive mode at a certain depth, a logbook, setting various gas mixtures, are similar to the Suunto Eon Steel, and other dive computers on the market.

Chapter 4

Hardware Design of a Dive Computer

This section covers the hardware considerations which were made during the design phase of the dive computer. Such considerations include a proper selection of an MPU, main memory, sensors, input methods, and other sub-modules which are necessary for the functionality of an embedded system.

There were several criteria during the selection of components, ranging from power consumption, availability, testability, costs, and software environment surrounding them. Also, it is clear that there are certain size and weight constraints bound to such a product. Since the display should be proportional in size to the dive computer, a small dive computer might not be usable in water, as the readings on the display would be intelligible. On the other hand, a large device would be bulky. It would also draw more power, and would affect the buoyancy of the diver.

An LCD, which were to be used, needed to satisfy a trade-off between power consumption and a good data representation.

To fulfil its task of determining the decompression schedule for a dive profile, the dive computer needs sensors which would make the reading of the environment data possible. These sensors include a compass and a pressure sensor. The sensors need to have a certain accuracy and respect the size and power constraints.

A storage medium is required, where the firmware, algorithms, dive logs, etc. would be permanently stored.

As an input method, a 3.5″ touchscreen which works under water will be used.

For the data transfer and communication to the users PC, a WiFi module will be used, which will then work in access point mode.

The core of the system is the MPU, which would, at a certain rate, read out the sensors, and present relevant data on the display. The MPU needed to support certain protocols, which make the communication to the sensors, LCD, and the storage medium possible.

It is also worth noting that all the components must be available on the market in the next 5-6 years, as changing either one of the components might be expensive.

This chapter is divided in several sub-chapters, each covering an important hardware aspect of the device. Firstly, we will introduce the requirements for a microprocessing unit, which are given in Chapter 4.0.3. Display considerations are given in Chapter 4.0.4. Chapter 4.0.8 gives more information on the module used for WiFi communication.
In Chapter 4.0.6, things concerning the permanent data storage are explained, and Chapter 4.0.7 gives information about the touchscreen. The main memory of the system is explained in Chapter 4.0.9. For testing purposes, a JTAG interface is also necessary, and its description is given in Chapter 4.0.12.

4.0.3 Microprocessing Unit

As previously noted, an MPU which supports certain communications interfaces including 24-bit RGB interface, serial peripheral interface (SPI), analogue-to-digital converter (ADC), for the touchscreen, and I^2C was needed. For this reason, the MPU was the first component which was chosen. Since ARM is one of the leading processor designers which offers a wide variety of processor designs, a decision was made to pick one of the ARM based processors.

Certain criteria like software and tool ecosystem, reference implementation flows and a well-understood and documented processor were applied.

Possible MPU candidates, picked from a wide variety of processors, which fulfilled our requirements are visible in Table 4.1.

Vendor	**MPU family**	**Processor core**	**Architecture**
Freescale	i.MX51	ARM Cortex-A8	ARMv7-A
Texas Instruments	Sitara	ARM Cortex-A8	ARMv7-A
Samsung	S3C6410	ARM11	ARM11

Table 4.1: Micro processing unit candidates for the dive computer

These chips are quite similar in terms of what they offer. They all target environments where a rich operating system, high performance and low-power footprint are needed.
The AM335x from Texas Instruments Sitara family was chosen for the reason of a good support in terms of software and reference documents. The application examples for the Sitara family state that it is targeted towards retail, industrial and medical application, and single board computing. Texas Instruments gives several concrete usages for the AM335x , including building and home automation, industrial communication, single board computers, data concentrator, etc. [14].

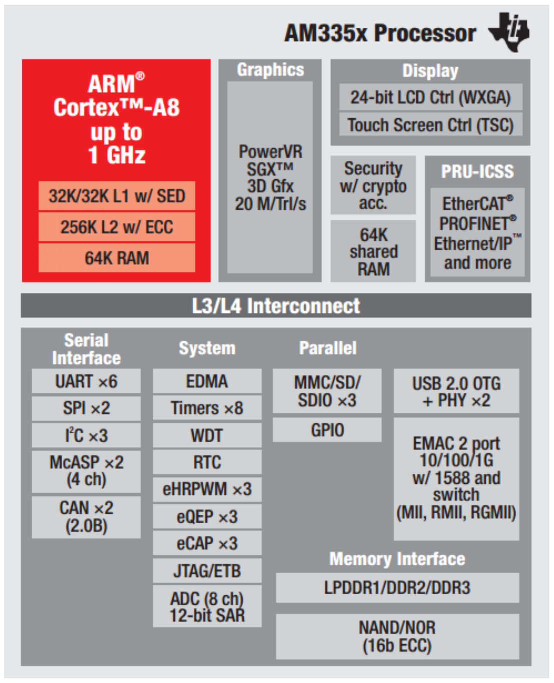


Figure 4.1: AM335x ARM processor block diagram [6]

A block diagram for the Cortex-A8 based AM335x MPU is given in Figure 4.1.
As we see in Figure 4.1, the AM335x comes with a lot of features. It supports a clock frequency of up to 800MHz, which is too much for us, but the lowest operating frequency is 275MHz, which will fit our needs in terms of speed requirements. The AM335x has a wide range of periphery it can support. For us relevant are the 24-bit RGB interface, a touchscreen controller, 2 SPI buses, 3 I^2C buses, and the memory interfaces for DDR2 and DDR3. Besides this, there is a support for MMC/SD cards and a NAND flash.

The AM335x comes in several configurations. They differ in the frequency range they are designed for, presence of the 3D graphics accelerator chip, and the I/O supply voltage. For example, the AM3352, which is on the lowest end of the portfolio, is capable of running at frequencies of 300 MHz, 600 MHz, 800 MHz and 1 GHz, while the highest end AM3359 runs at 800 MHz. Compared to AM3359, the AM3352 also lacks the 3D graphics accelerator chip.

We decided to use the AM3352 for several reasons, including the lower clock frequency, and the lack of unnecessary features like a 3D graphics accelerator. The lower frequency will help us reduce the power consumption.

As for the software and product support, Texas Instruments offers many resources, including a community forum, free MPU samples, power distribution switches, reference designs for various peripherals including DDR memory, power management ICs specifically designed for this processor, and so on. Several evaluation boards were also offered by TI for the AM335x family:

- BeagleBone Black

- AM335x Starter Kit (low cost starter kit)
- AM335x EVM (full featured AM335x EVM)

To test the capabilities of the processor, we used the AM335x Starter Kit.

4.0.4 Display

Next in line to choose is the display, where the information will be presented to the user.
When choosing the right display, several aspects must be considered. The first aspect are the optical properties of the display. The display which would have to have the right size, since the touchscreen which is used only comes in one size, namely 3.5″.
The contrast and the viewing angle would also need to be satisfying, as the application area is outdoors, underwater. The second aspect is a module with a well-known interface. Either a display with an SPI interface is used, or an LCD with a 24-bit RGB interface. The AM3352 supports both ways, but in practice, the industry dictates that a 24-bit RGB interface is used, since the other option with the SPI is mainly targeted to hobbyists, and the LCD modules are discontinued after a few years. Still, we decided to explore both options for the sake of research.

An open-source library by a Github member *notro* called *fbtft* offers a wide portfolio of the Linux framebuffer drivers for the most common TFT LCD controllers. There are a lot of displays offered which use one of these controllers, some of them even satisfying our criteria in terms of size and optical properties, but for the aforementioned reason of availability in the future, we decided not to explore this option any further.
The other option of using a display with a 24-bit RGB interface was picked. A display called *P320X-35ALWS*, by a company called Electronic Assembly was recommended to us by a company well versed in this matter, so a decision was made to use this particular display.
Main features of the display are [15]:

- 3.5″ diagonal screen size
- 320 * (R * G * B) * 240 dots
- Sunlight visible
- White LED backlight
- 24-bit RGB interface, HSYNC, VSYNC, SPI
- *Himax HX8238-D* controller
- conforms ROHS

The outline dimension of the LCD module is 76.9mm (W) * 63.9mm (L) * 3.2mm (H) with an active area of 70.08mm (W) * 52.56mm (L). The digital power supply voltage ranges from 3.0V to 3.6V. The backlight supports max. 21V, with a typical voltage of 19.2V, with a forward current of 20mA.

Since the 24-bit RGB interface is naturally supported by Linux, we only need to send the data over it at the correct clock speed, so the software main concern for this display is the initialization. For example, some displays must be initialized over SPI. Luckily, in our case, the *Himax HX8238-D* controller found in the LCD we will use includes a *Power on Reset* circuitry, so all internal circuitry will initialize when the power is applied to it. So, to use our display, we just need to match the signals from the LCD controller of AM335x and our LCD module.

As for powering the backlight, we will require a buck-boost converter with the above mentioned parameters.

4.0.5 Real Time Clock

A real time clock (RTC) is considered a default part of any MPU, but it can also be realised in a form of a separate integrated circuit (IC). It is present in devices which need to precisely track the passage of time.
When the device is powered off, the RTC still needs to continue tracking time. For this matter, it usually has its own backup battery, so it can continue working when the primary source of power is turned off. If a CPU switches the system to a lower power state because it is idle, or the battery is empty, the RTC can continue the task of tracking time.

We consider an RTC to be an important part of our design, since we can use it to wake up the board, or trigger periodical actions. An important task of an RTC IC is providing timed interrupts. The engineer can either define periodical interrupts, or alarms at a certain time. Provided it has enough power, it can track the passage of time for several years.

The AM335x has a real time clock subsystem (RTCSS), which enables it an easy tracking of time and date. Using RTCSS, it is also possible to generate real time interrupts, which can occur every second, minute, hour, or day, depending on how the user defined it. It can also be used to wake up the rest of the board from a power down state - a feature which might be useful for a dive computer.
For the clock reference, three options are offered. It can be sourced by an external crystal, an external 32.768kHz oscillator, or from a Peripheral Phase Locked Loop (PLL). If a crystal is used, RTC_XTALIN pin is used as the input to the on-chip oscillator, and RTC_XTALOUT is the output back to the crystal. RTC_OSC_REG register can be used to disable the oscillator. If an external 32.768kHz clock oscillator is used, only RTC_XTALIN pin is connected. The source of the RTC is selected by SEL_32KCLK_SRC bit in the OSC_CLK register [1].

The functional block diagram of the AM335x RTC is seen in Figure 4.2. *timer_intr_pend* is a timer interrupt, generated periodically, and
alarm_intr_pend is an alarm interrupt. They are both generated by the RTC.

4.0.6 Permanent Data Storage

To permanently store the bootloader, Linux kernel and the filesystem, a non-volatile data storage is needed. Since our device is power and size constrained, viable options were either a flash storage or an SD card.

The AM335x has a NAND controller with 16-bit Error Code Correction (ECC), so a NAND flash of a considerate capacity sounds like a good option. Unfortunately, due to time and tool limitations, we decided to use the SD card as our non-volatile memory, since its implementation is much simpler, but at one small cost: a reduced speed of data transfer.

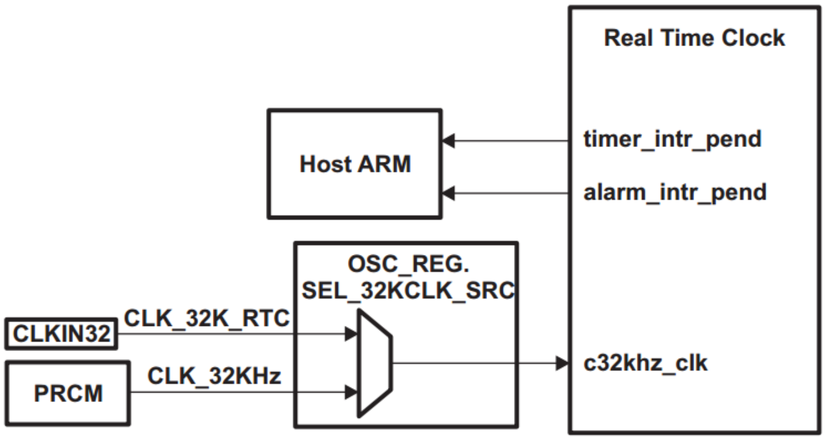


Figure 4.2: AM335x real time clock functional diagram [1]

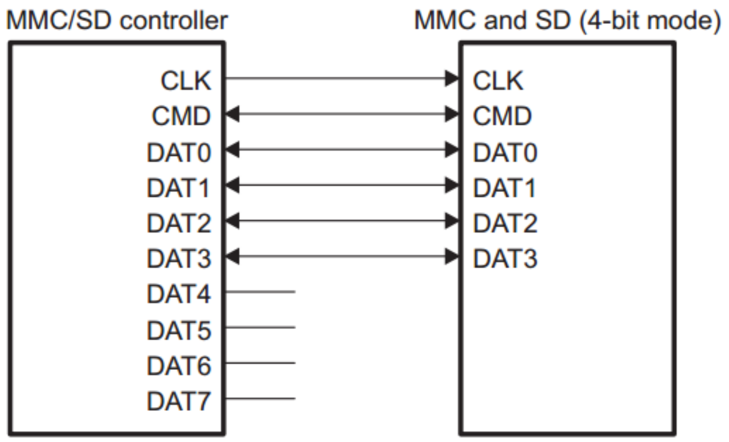


Figure 4.3: MMC/SD interfacing in 4-bit mode

The AM335x comes with three MMC/SD controllers [1]. The MMC/SD protocol performs the necessary communication between the card and the MMC/SD controller.
It can be configured to work as an MMC or SD controller, based on the type of card we use. There are several modes which can be used while interfacing this type of storage, depending on the number of data pins one uses. SD cards can use one, four or eight data lines. MMC cards support one and four data lines. The configuration we decided to use was an micro SD card with four data lines. The interfacing is then given in Figure 4.3. As visible in the Figure 4.3, there are four data lines, a clock signal used for a synchronized communication, and a command signal line, over which the commands are exchanged between the controller and the card. Besides these 6 signals, an SD card also requires a 3.3V voltage source and a ground pin.

The MMC/SD controller of AM335x has a built-in 1024 byte buffer for reading and writing, and supports up to 24Mbyte/sec transfer in 4-bit mode.

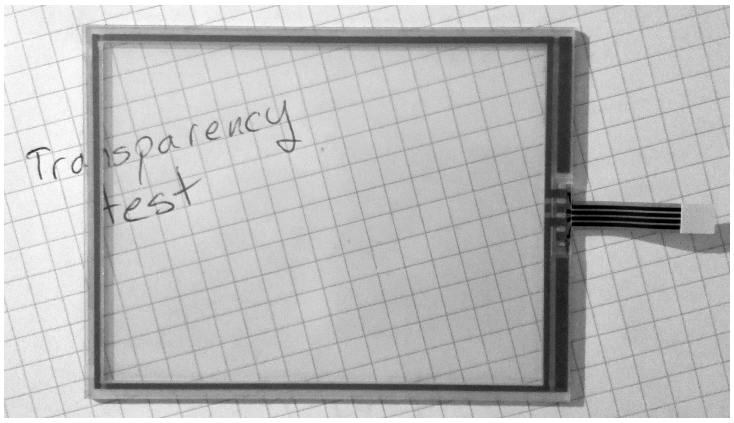


Figure 4.4: Touchscreen module and a transparency test

4.0.7 Touchscreen

A resistive touchscreen is an input device which is usually placed over an LCD. It is a glass panel whose touch area covers the viewable area of the display. This type of sensor is ubiquitous nowadays. If a product has an LCD and expects an input from the user, there is a high probability it also uses a touchscreen.
Several technologies exist which enable this technology, and in one way or another, they rely on the voltage change to determine the location of the touch. Most common touch sensor technology which is often encountered in embedded systems such as ATM's, restaurants, older smartphones, etc., are mostly resistive [16].
In our design, we will also use a resistive touchscreen. Instead of air between the layers, it is filled with oil, so it will work under pressure, and the layers will not short circuit.
The 3.5″ module we will use is shown in Figure 4.4, where we do a small transparency test.

4.0.8 Wireless module

Enabling a communication between a PC and the dive computer is also an important aspect we must consider. This way, we give the user a way to access their data (e.g. divelogs), make data backups, and do some basic configuration from their PC (like recalibrating the touchscreen or updating the software).
For this matter, a WiFi module will be integrated in our design, which will be used in access point mode. After some research, we found a module which suits our needs in terms of features, size and long-term availability.

We decided to use an OEM WiFi solution by *Acme Systems*, an Italian company which specializes in embedded hardware and software solutions. The module used is the *WiFi-2-IA*, and it can be seen in Figure 4.5. It has an on-board antenna, with a range of up to 3 meters [17], which perfectly suits our needs.
It uses a *RaLink RT5370* system on chip (SOC). It is a very popular SOC, and there are several vendors which offer a module where *RaLink RT5370* is used, one of them being the one we

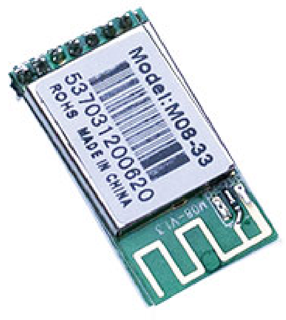


Figure 4.5: WiFi module by Acme Systems [17]

picked.
This module can be soldered directly to the PCB, and it has a 6 pins with a 2mm pitch. It uses a USB 2.0 as a means of communication, and requires a 3.3V power source.

4.0.9 Main Memory

Every computer needs a working memory where the kernel is loaded, and where the program code is stored, so a consideration was needed to choose a fitting solution in a form of a DDR memory chip.
The AM335x Starter Kit has an DDR3 SDRAM chip on-board.
DDR3 SDRAM uses a double data rate architecture to achieve high-speed operation. The double data rate architecture is an 8n-prefetch architecture with an interface designed to transfer two data words per clock cycle at the I/O pins. A single read or write operation for the DDR3 SDRAM effectively consists of a single 8n-bit-wide, four-clockcycle data transfer at the internal DRAM core and eight corresponding n-bit-wide, onehalf-clock-cycle data transfers at the I/O pins [3].
The AM335x External Memory Interface (EMIF) is used for interfacing various memory types, like mDDR, DDR2 and DDR3. It has a 16-bit wide datapath to the external SDRAM memory, and an addressability of 1Gb. It is heavily customizable, as it offers a flexible bank/row/column/chip-select address multiplexing schemes, various Column Access Strobe (CAS) latencies, page sizes of various sizes and so on.
In the AM335x Starter Kit design, a DDR3 chip by Micron is used. Due to our lack of experience with high-speed design, we decided to keep the DDR3 chip, and just swap the 2Gb chip mounted on AM335x Starter Kit, with a 1Gb chip with the same pin layout but a lower speed rate and price. We will also get a chance to understand the software aspect of interfacing a DDR3 chip, since we will have to reconfigure the software module which takes care of interfacing the main memory.
Let us take a look at 1Gb and 2Gb DDR3 solutions by Micron in Table 4.2, and how they are addressed. The 2Gb solution was used by on the AM335x Starter Kit. As one can see, The 1Gb option requires one less signal (The 13th signal for row addressing).

Parameter	**64 Meg x 16 (1Gb) DDR3 chip**	**128 Meg x 16 (2Gb) DDR3 chip**
Memory layout	8 Meg x 16 x 8 banks	16 Meg x 16 x 8 banks
Refresh count	8K	8K
Row addressing	8K (A[12:0])	16K (A[13:0])
Bank addressing	8 (BA[2:0])	8 (BA[2:0])
Column addressing	1K (A[9:0])	1K (A[9:0])
Page Size	2KB	2KB

Table 4.2: 1Gb and 2Gb DDR3 solutions by Micron and their addressing

As for the AM335x Starter Kit, the evaluation kit we use, a 2Gb chip by Micron is used on the board. Only 265Mb of this chip are used, as defined in the device tree of the EVM-SK. The reason for this is the fact that there is only one Chip Select (CS) pin, so only one DDR bank can be used, and the size of one bank of the chip is 256Mb.
DDR is high-speed, with a complex design and test flow. It also requires personnel with highly specialized skills and expensive simulation and testing tools. Implementing a DDR2 IC on our board would considerably reduce power consumption, but it would require simulations, evaluations, EMI compliance testing and so on, all with unavoidable iterations of such processes. This would extend the scope of this work tremendously. For this matter, we decided to stick with the working memory layout and chip used on AM335x Starter Kit. Thus, we can choose any of the Micron DDR3 chips, since they are share the same pin layout. We will use the *MT41J64M16JT-15E*, a slower chip with less capacity.

4.0.10 Sensors and Sensor Interfacing

A purpose of a dive computer is mainly to track the dive profile of the diver, by means of measuring time and pressure.
The surrounding pressure is used to track the partial pressure of gases in the human tissue. Besides the pressure, a dive computer measures the water temperature, to e.g. notify the user of the extreme temperatures, or to make a more accurate calculation of the partial pressure in the tissue. These measurements are, combined with the decompression algorithm, used to calculate the partial pressure of inert gasses in the divers tissue.
Besides the information needed for the decompression algorithm, a module necessary for navigation, such as a compass, is needed.
For this reason, several types of sensors will be mounted on the board: *MS5803*, a pressure and temperature sensor, and *LSM303D*, a 3D magnetometer and 3D accelerometer.

4.0.10.1 MS5803, Pressure and Temperature Sensor

A sensor module which will be employed on our board is the *MS5803* pressure and temperature sensor. It fulfils the requirements of the dive computer, as it offers a satisfiable precision, fast conversion and low power modes. It has an operating voltage in range of 1.8 to 3.6V, and an

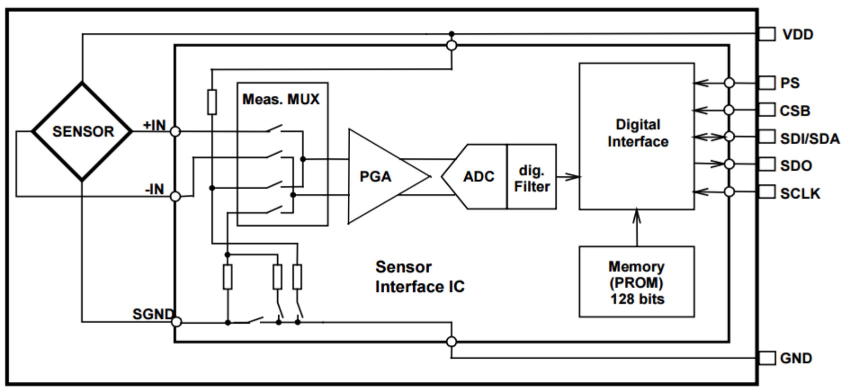


Figure 4.6: MS5803, pressure and temperature sensor block diagram [4]

operating range of 0 to 30bar for the pressure, and -40 to +85°C for the temperature sensor [4]. It is also hermetically sealable. On top of that, the communication protocol is simple, without the need to program internal registers in the device. The device can be interfaced with the I^2C and SPI.

The resolution of the data is 24 bits. This high resolution offers us a chance to use the pressure sensor to measure the depth, with a promised water depth resolution of 2.5cm.

In Figure 4.6 a block diagram for the *MS5803* is visible. In the block diagram, a simple nature of the sensor is shown. A piezo-resistive sensor provides the analogue data which is converted to 24-bit digital value.

We will use I^2C to interface this module.

4.0.10.2 LSM303D, Compass Module

Another sensor which which will be used in the design is an eCompass module. The technical info about the module stems from the datasheet.

The *LSM303D* is a chip developed by *STMicroelectronics*, and it is a solution which offers a temperature sensor, a 3D digital linear acceleration sensor and a 3D digital magnetic sensor. There were several reasons to choose this chip. It provides all the necessary sensors for a digital compass, has compact size (3x3x1mm), it offers several power profiles, and has a fairly high data resolution. The communication between the MPU and the *LSM303D* is done over the I^2C or the SPI interface [2].

In Figure 4.7 a block diagram for the *LSM303D* is shown. The block diagram shows the intuitive nature of the sensor. There are two inputs for the accelerometer and the magnetometer. This analogue input is then converted by an ADC. The data is then processed by the control logic which forwards it to the MPU over the I^2C or the SPI interface.

In Figure 4.8 a pin description, and the direction of detectable orientations and magnetic fields of the chip are shown.

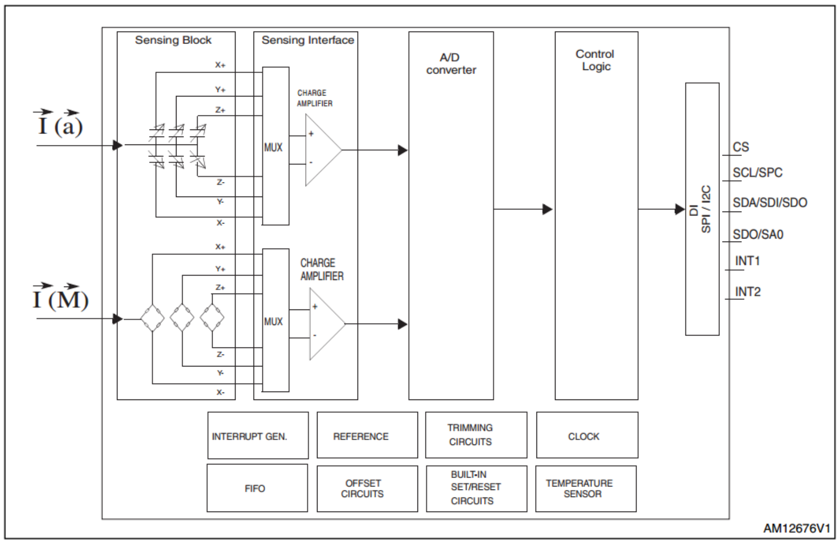


Figure 4.7: *LSM303D* eCompass module block diagram [2]

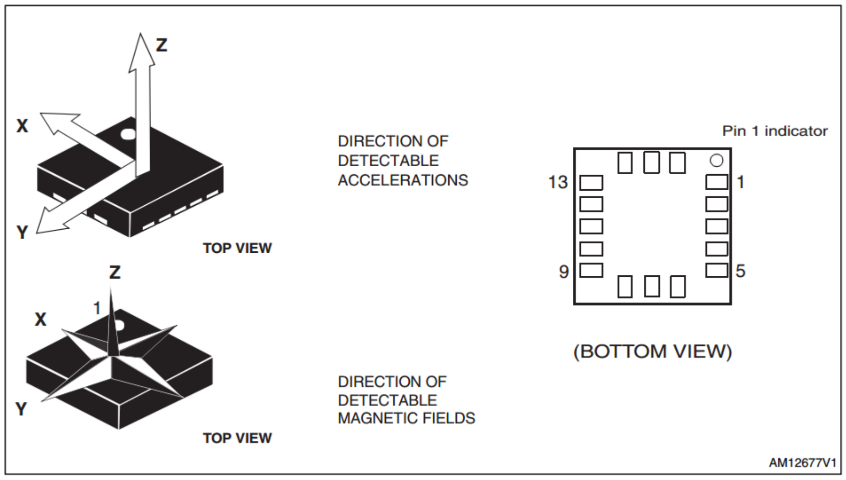

Z
X
Y
TOP VIEW
DIRECTION OF DETECTABLE ACCELERATIONS
Pin 1 indicator
13
1
9
5
(BOTTOM VIEW)
DIRECTION OF DETECTABLE MAGNETIC FIELDS
AM12677V1

Figure 4.8: *LSM303D*, eCompass module pins and detection directions [2]

4.0.11 Powering and PMIC

In this chapter, the powering requirements for the MPU and other components we introduced earlier will be discussed. We start of by giving information about the power supply rails of the MPU, which is given in Table 4.3.

Signal	**Description**	**Voltage**
VDD_CORE	Core domain	0.95V-1.1V
VDD_MPU	MPU domain	0.95V-1.325V
VDDS_RTC	RTC domain	1.8V
VDDS_DDR	DDR IO domain (DDR2 / DDR3)	1.8V/1.5V
VDDS	Dual voltage IO domains	1.8V
VDDS_SRAM_CORE_BG	Core SRAM LDOs, Analogue	1.8V
VDDS_SRAM_MPU_BB	MPU SRAM LDOs, Analogue	1.8V
VDDS_PLL_DDR	DPLL DDR, Analogue	1.8V
VDDS_PLL_CORE_LCD	DPLL Core and LCD, Analogue	1.8V
VDDS_PLL_MPU	DPLL MPU, Analogue	1.8V
VDDS_OSC	System oscillator IOs, Analogue	1.8V
VDDA1P8V_USB0/1	USB PHY, Analogue, 1.8V	1.8V
VDDA3P3V_USB0/1	USB PHY, Analogue, 3.3V	3.3V
VDDA_ADC	ADC, Analogue	1.8V
VDDSHVx	Dual Voltage IO domain	1.8/3.3V

Table 4.3: AM335x power supply rails [1]

We see that the power rails required for the MPU are either 1.8V or 3.3V. The CORE and MPU domain require a variable voltage range between 0.95V and 1.1V and 0.95V and 1.325V respectively, depending of the clock frequency. These voltages will be provided by a PMIC.

The *TPS65910x* IC is the one used for AM335x Starter Kit. It is designed specifically for AM335x, and can provide all of the power rails listed in Table 4.3, from both voltage and amperage perspective. It is designed for applications powered by one Li-ion or Li-ion polymer batteries, or a 5V input. It has three step-down converters, one step-up converter, and eight low-dropout regulators. These regulators are controllable by an I^2C interface. The PMIC has an embedded power controller to manage the power sequencing of the OMAP systems such as the AM335x.

The DDR subsystem will also be powered by the PMIC with dedicated power rails for it.

The eCompass module has a supply voltage range ranging from 2.16V to 3.6V, so we will be able to supply it by one of the VDDSHVx power domains. One of these domains will also be used for the sensors.
The digital domain of the LCD module has a typical supply voltage of 3.3V, and will also be powered by one of the VDDSHVx domains.

On the other side, the display backlight typically needs 19.2V, so we will need to use a buck-boost converter rated to this voltage and current to power it properly.

This PMIC also has an RTC subsystem, but for our purpose, it is not precise enough, so it will not be considered in our design.

4.0.12 JTAG and Testing

As it can be quite tedious to find an error in a complex system where both hardware and software are developed at the same time, a way to test the PCB and the low-level code running on it is needed.

For this matter, we will use JTAG which is the de facto standard to debug integrated circuits. It is covered by the IEEE 1149.1 standard. Practically, JTAG allows engineers to perform single stepping, setting hardware and software breakpoints, viewing memory content and register values of the MPU [18], and so on.

Designing for testability is a major concept which must not be overlooked. To test the system, we must put it in a known state, and supply it known test data. We can then observe the output data, and see if the system performs as we designed it. This is the only way to know if the system does what it should. Mostly all processors implement this standard, and AM335x is no exception.

Texas Instruments provides comprehensive guides on how to use the JTAG interface and what guidelines need to be followed while designing the circuit. Luckily, the schematics and the guidelines for the connection between the JTAG header and the AM335x was given by TI, so implementing a JTAG interface should not be a difficult task.

This concludes the section of the work which deals with the hardware design of our dive computer. In the next section, we will discuss the software design, both on low level, dealing with hardware, and the applications and utilities available to the user.

Chapter 5

Software Design of a Dive Computer

In this chapter, the software requirements for the dive computer are presented. These requirements were then used as a guideline and a reference during the implementation of the software modules for the dive computer.
Chapter 5.1 gives an overview of the requirements, Chapter 5.1.1 introduces Linux based Debian as our operating system.
Chapter 5.2 gives more information about the software aspect of the sensors implementation, and Chapter 5.3.1 will introduce with software utilities which will be implemented, and the simple GUI where the sensor data, is shown is introduced. Be advised that in this section we only introduce the conceptual idea of the software modules we plan on implementing. More details and the implementation can then be obtained from Chapter 7.

5.1 Software Requirements

One of the reasons to use such a feature rich processor like AM335x was the fact that there was a good software environment surrounding it.
Several operating systems were already ported to it. Notable are Windows Embedded CE, Android, and Linux [6].
For our work, Linux as an operating system was a good option, as it is freely available, and it can be easily modified and adapted to the needs of our board.
There is a GNU ARM toolchain for this platform too. Since we will use Linux, an integration of the drivers for the sensors, display and other periphery is easier than working on bare silicon, where there is no abstraction layer such as the kernel.

A bootloader was also required, as it loads the Linux kernel and the filesystem from the storage medium to the working memory.
A bootloader is the first piece of software loaded to the (internal) RAM by BIOS. It then prepares the environment so the kernel can be loaded to the main memory, and get started. Preparing the environment includes initializing certain submodules such as the external RAM, power management unit and so on. Since most modern processors include some sort of multiplexing and

configuration of I/O pins, the bootloader also takes care of the pin setup.

Information about the board is contained in a so called device tree file. This file is a data structure which describes the hardware on the board. So, instead of hard coding this information in the OS or the bootloader, this data is passed to bootloader during boot time [19].

After getting a short overview of the operating system and the boot process, we will now give a more thorough information about some important aspects of Linux, which directly concern our work.

5.1.1 Linux as Operating System

The initial release of MINIX, a Unix-like operating system happened around 30 years ago. It was created by Andrew S. Tanenbaum for an educational purpose, which, with a release of MINIX 2005, shifted to the creation of a reliable microkernel OS which would be used not just in academia, but also in the industry.
This open source OS inspired Linus Torvalds who initially released Linux in 1991. When introduced, it was specifically targeted to desktop PCs with an Intel 80x86 architecture. Torvalds said ″Linux will never run on anything but a PC with an IDE hard disk since that is all that I have.″. Since then, this quote proved to be wrong, as Linux quickly evolved from a one-man project to a world-wide project which involved thousands of developers.
Nowadays, Linux based operating systems are prevalent in the embedded world, ranging from consumer electronics to medical devices. Its open source nature allowed it to be ported to a lot of platforms, like ARM, AVR32, x86 and so on.
Another reasons for a high popularity of Linux is the fact that it requires no royalties or fees, as long as the licensing technicalities are respected. The reason for this lies in the fact that Linux, in its early days, adopted the General Public Licence (GPL) which protected the Linux kernel from the commercial exploitation.

5.1.1.1 Overview of Linux

As Tanenbaum [20] says, a Linux system is regarded as a pyramid.
At the bottom of the pyramid is the hardware (CPU, memory, I/O devices).
On top of the hardware, the operating system is running, whose function is to manage the hardware resources and to provide system calls for the programs. The system calls then enable creation and management of files, processes and other resources.
These layers of the Linux system are presented in Figure 5.2. As we see in the Figure 5.2, the user applications are executed at the top, where the user (i.e. the application) resides.
Worth noting is also the *glibc* library, a GNU implementation of the C standard library, which also nowadays supports the C++. This library defines the system calls and other functionalities like *open*, *malloc*, *printf*, *exit*, etc.
At the top of the Linux kernel is the system call interface. Under this layer, an architecture-independent kernel code resides. Under the kernel code lies the Board Support Package (BSP), which is the architecture dependent code.

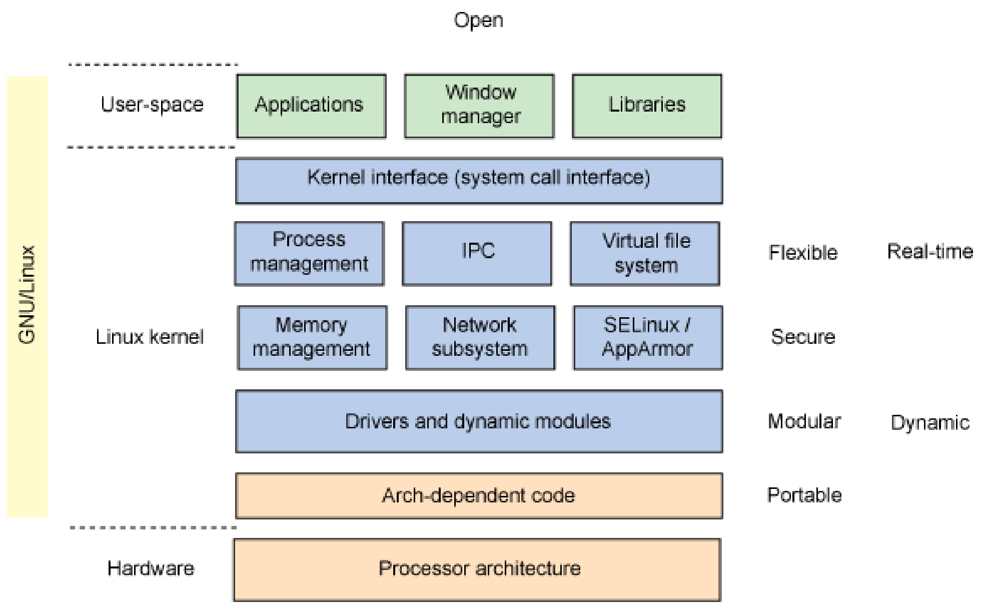


Figure 5.1: Linux system layers [20]

The three pillars of the kernel are the I/O component, the memory management component and the process management component. At its lowest level, the kernel contains interrupt handlers, which are a primary way of interfacing with devices. When an interrupt occurs, a dispatching mechanism is triggered.

The I/O component contains the kernel code which is used for interaction with devices and performs network and storage I/O operations. For example, at the top level, an I/O interaction happens with a file, and on the lowest level, the the I/O operations pass through a device driver.

Memory management takes care of the virtual to physical memory mapping, taking care of the cache of recently accessed pages, and an employment of a swapping algorithm.

The far right pillar, the process management, intuitively takes care of the processes, their creation and termination. A part of this component is also the scheduler which schedules the executable entities (processes and threads).

On top of these components is the system call interface, which provides a transition from the user space to protected kernel mode, and passes the control of the application execution to one of the above mentioned components.
A nice visualisation of these components is provided in Figure 5.2.

5.1.2 Kernel Modules and Drivers

A module is a file which contains compiled code which extends the functionality of the kernel. They are used to support new hardware, or to add a new pack of system calls.
They can also be statically bound to the kernel, or dynamical loaded by the *modprobe* utility in

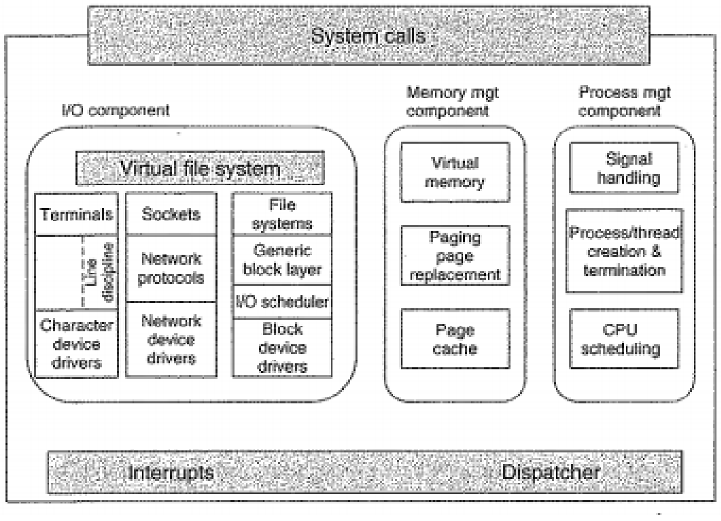


Figure 5.2: Linux kernel structure [20]

userspace. Both of these approaches have their advantages and disadvantages.
The advantage of a loadable module is the fact that not all possible modules have to be included in the OS, and thus loaded to the main memory where they would waste resources. On the other side, for the embedded systems, where devices and their drivers are known a priori, binding the modules statically might prove beneficial, as we do not have to bother with dynamic kernel modules [21].

Device drivers, as a part of standard practice, are built as kernel modules. It is said that ″Device drivers take on a special role in the Linux kernel. They are distinct black boxes that make a particular piece of hardware respond to a well-defined internal programming interface; they hide completely the details of how the device works. User activities are performed by means of a set of standardized calls that are independent of the specific driver; mapping those calls to device-specific operations that act on real hardware is then the role of the device driver. This programming interface is such that drivers can be built separately from the rest of the kernel and plugged in at runtime when needed. This modularity makes Linux drivers easy to write, to the point that there are now hundreds of them available.″ [22].

5.1.3 Booting

In this part, we explain in more detail how an embedded system using Linux boots, i.e. initializes the system.

Low-level booting is architecture specific. For example, the AM335x has 176kb of internal boot ROM, where the first stage bootloader is stored. It is also referred to as the Initial Program Loader (IPL). The purpose of the bootloader is to set up the basic peripherals, such as the I^2C controller. This way, the communication between the AM335x and its I^2C peripherals, such as the PMIC is enabled.
Other than that, peripherals from which the second bootloader will be loaded are also initialized.

This can be the SD card, NAND flash and so on. The boot source is set by the SYSBOOT[15:0] bits.
After the IPL has done the basic initialization, it is time to load the second boot loader. In our case, this is the U-Boot boatloader.

During the first stage bootloader, SPI is used for the communication with the SD card. The reason is the fact that the MMC/SD interface has not yet been initialized.
Out of the 128kb RAM AM335x has, 18kb are used by the IPL. The rest can be used by U-Boot, but since its binary is bigger than 110kB, it cannot be fully loaded to this memory. For this reason, a solution was devised by the TI engineers, and their version of U-Boot has a two stage design, i.e. two binaries are generated after the bootloader is compiled.
The first binary is the Minimal Bootloader (MLO), which, among other things, initializes the DDR chip. This way, there is practically unlimited space for the second binary, the U-Boot, to get loaded to, and get executed.

After U-Boot is loaded, it prepares the rest of the hardware for the Linux kernel. This includes loading of the device tree, where the platform information is present, and copying Linux binary from the boot source (SD card) to the DDR3 chip. It then gives the control to the kernel.

5.1.4 Graphics

Since we will be using an LCD module to display the data, we must have an idea how graphics in Linux kernel work, For this reason, we briefly introduce the framebuffer concept.
The kernel documentation [23] states that framebuffer is a device which abstracts the graphics hardware (like an LCD panel), and allows the software to access this hardware through a well-defined interface. This way, the application software (e.g. an X11 server) does not need to know anything about the hardware details. The framebuffer is accessed through a device node, usually located in the /dev directory, i.e. /dev/fb*.
It is a memory device - and there is a portion of memory in the RAM reserved for the framebuffer, which is then written to the hardware by the LCD controller by using Direct Memory Access (DMA), thus not putting a big load on the CPU, since it only needs to fill the memory designated for the framebuffer data.
The main parts of the kernel responsible for the framebuffer are given in Figure 5.3. We see here the components which are necessary for a functioning system. In some cases, a low-level LCD driver is needed, e.g. if the panel requires a special initialization sequence over SPI or I^{2}C. Most of the panels which offer a parallel LCD interface, which we also use, do not require any special initialization procedure, and other interfaces are only used to change settings like brightness, gamma correction, and so on.

5.1.5 Kernel Build System

In order to suit the Linux kernel to our board, we will need to undertake a certain amount of customization.
Configuring the Linux kernel is rather simple, if we know what we want to do. For example, executing ″`make ARCH=arm menuconfig`″, a series of options, which are grouped in individual

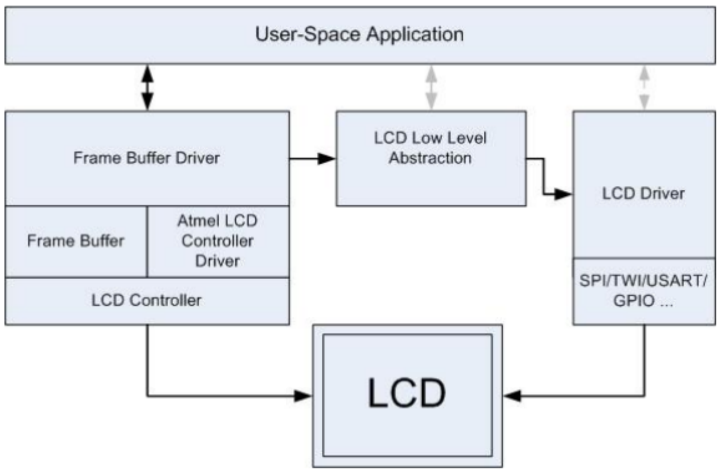


Figure 5.3: Linux framebuffer and LCD controller overview [24]

menus and submenus, are shown, and depending of the architecture selected, like in our case, ARM, relevant hardware support and kernel features are offered.
For each platform for which Linux kernel was ported to, a default configuration is given. We can edit this configuration to suit our purpose. For example, we will have to include the WiFi module driver in the kernel.

5.1.6 Real Time Operating System Properties and Requirements

There are certain time constraints bound to an operation of a dive computer.
Data collection must be done at a predefined interval, to ensure a correct calculation of the dive profile. Software alarms which warn the user that something is wrong, such as a fast ascent rate or low tank pressure, can also be triggered. All this requires a guarantee that the time dilatation between an event and the execution of a relevant piece of code will be smaller than a certain value. Usually these values are in a microsecond range.

Linux is not a real-time operating system (RTOS), and has no requirement to meet a certain deadline for a task scheduling after an interrupt which happened.
Since we are talking about the microsecond delay between an interrupt and the execution, this is not a problem, as the deadlines we deal with in our application are not that strict. A certain amount of time dilatation can be tolerated when doing data collection, and as for the alarms, even a few seconds are acceptable, since the algorithms which calculate the values which trigger these alarms already have some sort of tolerance and inaccuracy to them.

For this reason, we deem that Linux can satisfy our time constraints. When designing such a system, it is important to create a code which is without any critical bugs, which might cause the dive computer to become unresponsive, by e.g. crashing the kernel.

5.2 Sensors and Sensor Interfacing

The main purpose of a dive computer is to track the dive profile, by means of measuring time and pressure.
The environment pressure is used to track the partial pressure of gases in the human tissue. Besides the pressure, a dive computer measures the water temperature, e.g., to notify the user of the extreme temperatures, or to make a more accurate calculation of the partial pressure in the tissue. These measurements are, coupled with the decompression algorithm, used to calculate the partial pressure of inert gasses in the diver's tissue.
Besides the information needed for the decompression algorithm, information necessary for navigation such as compass is needed.
For this reason, several types of sensors will be mounted on the board. The first in line is the $MS5803$, a pressure and temperature sensor, and $LSM303D$, a 3D magnetometer/ 3D accelerometer.

5.2.1 MS5803, Pressure and Temperature Sensor

We now briefly introduce the software design concepts concerning the previously introduced pressure and temperature sensor. Here, the interfacing, the mathematics behind the sensors, and interfacing the modules will be explained.

In the previous section of the work, where we showed the hardware aspect of the modules, we saw that it supports SPI and I^{2}C. The interface used by us is the I^{2}C. This interface is chosen by pulling the Protocol Select (PS) pin of the sensor to 3.3V. The sensor can have two addresses, depending of the CSB pin. If it is pulled high, the address is 0x76. If it is low, the address is 0x77. The reason for this feature is the fact that we can have two of these devices on the same bus.
Basic five operations offered over the I^{2}C interface are [4]:

- reset
- read calibration words (16-bit per word)
- D1 conversion (pressure)
- D2 conversion (temperature)
- read ADC result (24-bit pressure / temperature)

Size of each command is 8 bits. They are given in Table 5.1.

Command	**Hexadecimal value**
Reset	0x1E
Convert D1 (OSR=256)	0x40
Convert D1 (OSR=512)	0x42
Convert D1 (OSR=1024)	0x44
Convert D1 (OSR=2048)	0x46
Convert D1 (OSR=4096)	0x48
Convert D2 (OSR=256)	0x50
Convert D2 (OSR=512)	0x52
Convert D2 (OSR=1024)	0x54
Convert D2 (OSR=2048)	0x56
Convert D2 (OSR=4096)	0x58
ADC Read	0x00
PROM Read	0xA0 to 0xAE

Table 5.1: MS5803 command structure

Here, D1 represents the raw pressure value, and D2 is the temperature. OSR is the sample rate for these values.

During the production of sensors, certain production errors need to be considered, i.e. every sensor will provide a slightly different value in the same environment setup. For this reason, every sensor is individually calibrated at two temperatures and two pressures. The result of these calibrations are six coefficients (W1 - W6) which compensate the aforementioned production errors. These values are stored in a 128-bit PROM. The micro-controller then reads out these values then uses them in the conversion process for the pressure and temperature values. More information about the values is given in Table 5.2 [4].

Variable	**Description**	**Size in bits**	**Min. value**	**Max. value**
W1	Pressure sensitivity (SENS)	16, unsigned	0	65535
W2	Pressure offset (OFF)	16, unsigned	0	65535
W3	Temperature coefficient of pressure sensitivity (TCS)	16, unsigned	0	65535
W4	Temperature coefficient of pressure offset (TCO)	16, unsigned	0	65535
W5	Reference temperature (TREF)	16, unsigned	0	65535
W6	Temperature coefficient of the temperature (TEMPSENS)	16, unsigned	0	65535

Table 5.2: MS5803 calibration data

It is clear that we will use an unsigned integer of 16-bit width for these values.
Table 5.3 shows us more information about the data which is read after the conversion.

Variable	Description	Size in bits	Min. value	Max. value
D1	Pressure value	24, unsigned	0	16777216
D2	Temperature value	24, unsigned	0	16777216

Table 5.3: MS5803 pressure and temperature data after conversion

Here, an unsigned integer of 32-bit size will be used.
These values are used for conversion to actual usable information like in the flowchart shown in Table 5.4.

Variable	Description	Size in bits	Min. value	Max. value
dT	Difference between actual and reference temperature, dT = D2 - TREF	25, signed	-16776960	16777216
TEMP	Actual temperature TEMP = 20°C+ dT * TEMPSENS	41, signed	-4000	8500
OFF	Offset at actual temp. OFF = OFFT1 + TCO * dT	41, signed	-8589672450	12884705280
SENS	Sensitivity at actual temp. SENS = SENST1 +TCS * dT	41, signed	-4294836225	6442352640
P	Temp. compens. pressure P = D1 * SENS - OFF	58, signed	0	300000

Table 5.4: MS5803 pressure and temperature calculation

The values for temperature and pressure are stored in the variables TEMP and P.

5.2.2 LSM303D, Compass Module and its Implementation

Besides the driver for the pressure and temperature sensors, we will also need a driver which interfaces the *LSM303D* eCompass module.
An ultra-compact module offered by *STMicroelectronics* is feature rich, and offers for us relevant things like [2]:

- 3 channels for the magnetic field
- 3 acceleration channels
- SPI and I^{2}C interface
- power-down mode / low-power mode

This module can be used in various scenarios, including tilt-compensated compasses. The readings from the accelerometer provide pitch and roll angles, which are then used to correct the magnetometer data. This allows an accurate calculation of heading and yaw angles when the module is not held parallel to the ground.
Now that we explained what this sensor can offer, we explain how we will interface it.
As for the interface, I^{2}C will be used, and the module will be connected to the same bus as the *MS5803*.
The slave address of the module is 0b00111xx. The xx bits are either 01. if the SDO pin is connected to the voltage supply, and 10 if the pin is grounded. This way, we can use two same modules on the same bus. This one address is used for both the accelerometer and the magnetometer.

The *LSM303D* has plenty of configuration and output registers, each with an 8-bit width.
There are several configuration registers which we will have to configure. Things like enabling the sensors, setting the data-rate, anti-alias filter bandwidth and such are done by writing proper values to these registers.

In Table 5.5, we show the register address, the bit position, the description and the effect of the settings we will have to undertake.

Register	**Bit field**	**Value**	**Result**
CTRL1	ADDR[3:0]	0b0101	Accelerometer 50Hz ODR
CTRL1	AZEN	0b1	Acceleration Z-axis enabled
CTRL1	AYEN	0b1	Acceleration Y-axis enabled
CTRL1	AXEN	0b1	Acceleration X-axis enabled
CTRL2	FS[2:0]	0b01	+/- 4g accelerometer full scale
CTRL5	M_RES[1:0]	0b11	Magnetometer high resolution
CTRL5	M_ODR[2:0]	0b100	Magnetometer 50hz ODR
CTRL6	MFS[2:0]	0b00	+/- 2 gauss
CTRL7	MLP[2:0]	0b0	Magnetic data low power-mode off

Table 5.5: LSM303D configuration register fields and values [2]

After the sensor has been initiated, the configuration values which we acquired will be written to the corresponding registers. After this, we are ready to read out the values prepared by the accelerometer and the magnetometer.
For each magnetometer and accelerometer channel, there are two registers where the values are stored, one for the low byte and one for the high byte. This leaves us with 12 registers shown in Table 5.6.

Register	**Address**	**Description**
OUT_X_L_A	0x28	Accelerometer X channel, low byte
OUT_X_H_A	0x29	Accelerometer X channel, high byte
OUT_Y_L_A	0x2A	Accelerometer Y channel, low byte
OUT_Y_H_A	0x2B	Accelerometer Y channel, high byte
OUT_Z_L_A	0x2C	Accelerometer Z channel, low byte
OUT_Z_H_A	0x2D	Accelerometer Z channel, high byte
OUT_X_H_M	0x09	Magnetometer X channel, low byte
OUT_X_L_M	0x08	Magnetometer X channel, high byte
OUT_Y_H_M	0x0B	Magnetometer Y channel, low byte
OUT_Y_L_M	0x0A	Magnetometer Y channel, high byte
OUT_Z_H_M	0x0D	Magnetometer Z channel, low byte
OUT_Z_L_M	0x0C	Magnetometer Z channel, high byte

Table 5.6: LSM303D accelerometer and magnetometer value registers

Reading out the data is straightforward.
To read out six accelerometer values, we transmit the address of the lower byte of its X channel. Same goes for the magnetometer values, to read out six magnetometer values, we transmit the address of the lower byte of its X channel. The module then in both cases transmits six values, from low byte to high byte, first for X, then for Y, and then for the Z channel.

All this leads us to a conclusion that our code will first do an initialization of the module, a do proper configuration. Then we will have a simple interface to read out the values and store then in an array.

5.3 Software Modules Available to the User

In this part, we will introduce the reader to the software modules we plan on implementing during the course of this work. These software modules will show the capacity our system has, and can be considered as a proof of concept for what a modern dive computer should be able to offer.

5.3.1 Graphical User Interface for Data Representation

Previously, we mentioned the sensors which will be used, and how we plan to implement them in our system.
Although, the data which is acquired by the means of these sensors is important, it still needs to be put in a context, and presented to the user in a meaningful way. The reason for this approach is the fact that diving is a mentally challenging task, and requires focus at all times, and by making a good GUI, we will reduce the amount of focus needed to understand the GUI, and give the diver a chance to focus on the actual diving.

For this matter, a proof of concept GUI will be developed, to show the possibilities of Linux kernel and the Qt, framework we used.

A simple Qt application will be developed, merely to show the possibilities of the Qt for embedded Linux, and the potential it has to offer in the industrial setting for which a product like this one would be used.
The GUI area will be divided in three logical parts. One part will show the info about the system, including the battery status and the state of certain modules, like WiFi. The second part will show the sensor data, including pressure, temperature, alarms, and the compass information.
The last part will offer some sort of functionality. By clicking a button, the user can toggle the state of the WiFi module, recalibrate the touchscreen, and so on.
In the background, bash scripts will be implemented, which implement this functionality. They will then be executed by the Qt app.

5.3.1.1 Qt Framework

The framework for the GUI, as mentioned before, will be Qt.
″The Qt toolkit is a C++ class library and a set of tools for building multiplatform GUI programs using a *write once, compile anywhere* approach. Qt lets programmers use a single source tree for applications that will run on Windows 95 to XP, Mac OS X, Linux, Solaris, HP-UX, and many other versions of Unix with X11. A version of Qt is also available for Embedded Linux, with the same API.″ [25].
Qt framework offers a broad API, and its availability for all the common platforms is what made it popular.
Qt also features a language construct called *Signals and Slots*. It allows the communication between the objects (GUI elements, classes, ...), which makes it easy to utilize the observer pattern in an easy way. For example, If a GUI button was pressed, a signal will trigger a function to be executed.

The most important feature of the Qt is the fact that it does not require a windows manager to be running, like X11 windowing system. It can write directly to the framebuffer [26] we introduced before.

5.3.2 Various Helping Utilities

Besides showing information about the environment, we also plan on implementing some features which might come in handy for the potential user. For example, flashing the new firmware over WiFi, changing settings, or reading out data like dive profiles. This will show how several technologies, like a webserver, bash scripts and executables interact with each other, and how this can be used.

Chapter 6

Hardware Implementation

This chapter gives an insight in how the hardware aspect of the dive computer was implemented. Before we started the implementation, we got some information about the guidelines one should follow while designing our PCB. Our findings about this topic are given in Chapter 6.1, In Chapter 6.2, we briefly introduce the AM335x Starter Kit by Texas Instruments, the evaluation module for the AM335x processor, which was used as a reference design for our board. Chapter 6.3 then thoroughly explains every aspect of the hardware implementation which was relevant to the board.

6.1 Hardware Design Guidelines and Precautions

In this chapter, a short overview of the PCB design guidelines and precautions which were taken during the customization of the AM335x Starter Kit board, is given.

The focus of these guidelines was reducing the electromagnetic interference (EMI) caused by the board. Chapter 6.1.1 will explain what causes EMI, and will also present common pitfalls concerning EMI during a PCB design which were avoided during the design stage of the project.

6.1.1 Electromagnetic Interference and How to Prevent it

To prevent EMI, it is required to know what it is, and what causes it in the first place.
EMI is defined as a disturbance of operation of an electronic device caused by an electromagnetic field in the radio frequency (RF) spectrum that is caused by another electronic device or a natural source. In a typical EMI setup, three major components are present [27]:

- source
- receptor

- coupling path

Source is the emitter of the noise, receptor is the device which receives the noise/interference, and the coupling path transmits the interference signal from the source to the receptor.

The coupling path can either be conductive, inductive or capacitive.
Conductive coupling, or direct coupling is about direct way of transferring the signal via a conductive medium. Inductive coupling, or magnetic coupling path is associated with a region where the magnetic field is dominant. In this case, the source and receptor are less than a wavelength separated. In capacitive, or electric coupling, the source and receptor are less than a wavelength apart, but the dominant part of the EM field is the electric field.
The goal was a custom board whose design respected the three EMC criteria:

- it does not interfere with the operations of other systems
- it is immune from the emissions of other systems
- it does not interfere with its own operation

To achieve this goal, we took a look in to what causes electromagnetic emission (EME). These sources for EME are:

- integrated circuits
- trace loops on the PCB
- attached cables

An IC is complex system, with many loops inside it, which act as antennas when current passes through them. The sane concept can be applied to the trace loops on the PCB. They are considered good antennas, as they create a much bigger surface then the loops in an IC, so we must consider them in our design.
The attached cables also act as antennas. There are two antenna types which we consider, which radiate an EM field [28]:

- loop, differential-mode radiation
- dipole, common-mode radiation

Since differential-mode currents flow in opposing directions, their radiated electric fields subtract, producing a small electric field because the traces are not perfectly parallel.
The common-mode current, on the other hand, flows in the same direction through two traces, so the fields they generate adds up. This leaves to a conclusion that common-mode radiation poses a bigger problem for EMC.
When it comes to avoiding differential-mode radiation, certain guidelines are to be followed,

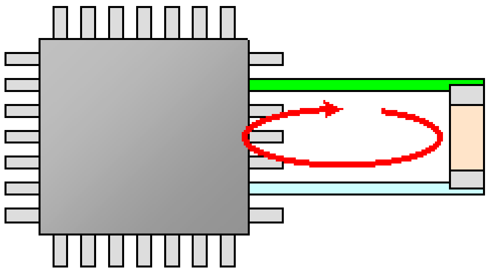


Figure 6.1: Loop area between IC and a decoupling capacitor [28]

because above a certain frequency, the current takes the path of the least impedance (smallest area between the forward and return paths).
For this matter, we must take a look into the current return path. The loop area between the signal trace and the trace between the grounds of the two devices which share the signal trace must be minimized, if EMC is desired. This leaves us with a smaller loop area and a weaker EM field which is generated.

A common source of loop areas can be vias which are serialized on the PCB, so they create a slot in the layer. This causes the (return) current to make a detour, and a bigger loop area is created.
This leaves us with a conclusion that precautions must be taken so no slot antennas appear on the PCB. Potential areas where this can happen is the LCD connector, or any place where a lot of vias in a dense area are present.

Precaution must also be taken when using decoupling capacitors. This case is visible in Figure 6.1. The same principle applies here, the loop area must be minimized.
Besides these precautions, some rules of thumb will be followed while designing our board, such as

- good floorplanning, i.e. placement of various subsystems in their own zones before routing
- ground and power planes next to each other, for an extra capacitor
- make the current return path as short as possible
- avoid slots in planes
- signal traces of an interface must be matched in length
- high speed traces routed first and with minimal number of vias

This way, we hope to design a board with a low EMI.

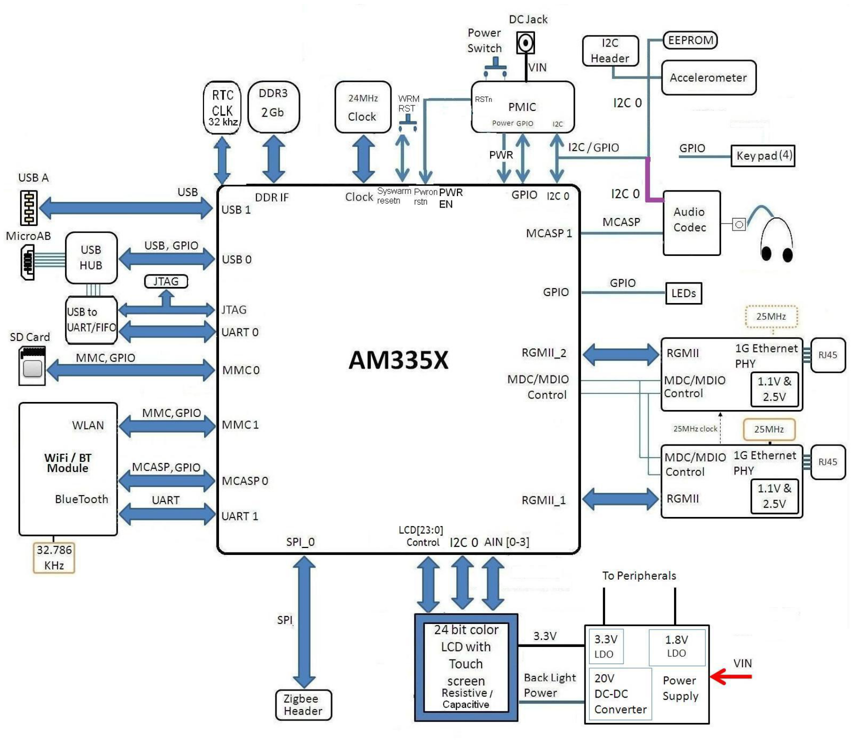


Figure 6.2: AM335x Starter Kit block diagram

6.2 AM335x Starter Kit as a Reference Design

Now that we have covered the guidelines we will follow during our design, it is time to introduce our starting point for our hardware design, the AM335x Starter Kit. The AM335x Starter Kit is a product offered by Texas Instruments. It provides an affordable platform for the evaluation of Sitara ARM Cortex-A8 AM335x processors (AM3352, AM3354, AM3356, AM3358). It features ready-for-production hardware, and is well supported from the software standpoint, which makes it ideal for our design.
The starter kit was used as the starting point for the PCB development because the schematics, bill of materials, and the PCB layout were provided free of charge, and can be used for academic and commercial projects.

This section will shortly cover the details of the EVM-SK and the included components.
A functional block diagram of the AM335x Starter Kit is shown in the Figure 6.2. Here, we see a feature rich system, with a useful periphery whose job is to show the capability of the AM335x. There are in total two USB interfaces, a WLAN/Bluetooth module, an LCD and a touchscreen module.
Besides this, the board has an Ethernet controller and an audio codec, and an SD card interface. Besides this periphery, a PMIC which we will also integrate in our design is present on the board. The schematics of our board rely on the schematics provided for the AM335x Starter Kit. The AM335x layout and its pinout, DDR3 and AM335x interfacing, the powering subsystem and the LCD backlight controller were used as a reference throughout our design. As for the PCB

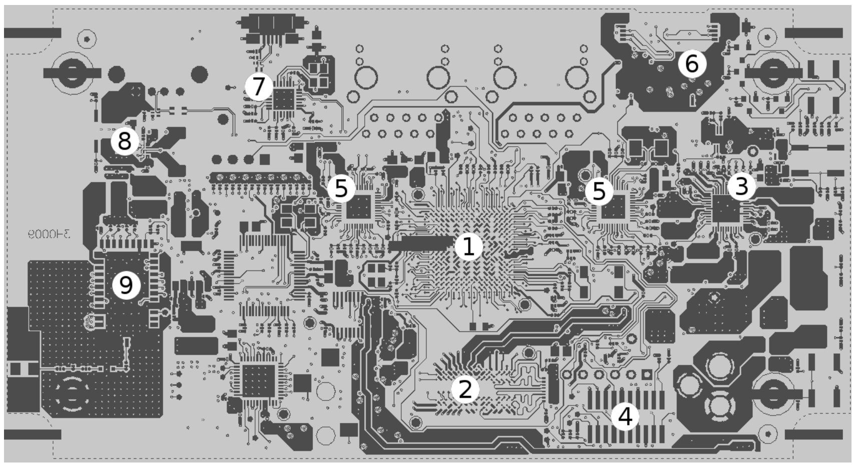


Figure 6.3: AM335x Starter Kit bottom layer

layout, the traces between the DDR3 IC and AM335x were copied to full extent, because of the fragile nature of the high speed signals.
In Figure 6.3, we see the bottom layer of the AM335x Starter Kit where we labelled some important submodules. Label 1 shows the position of the AM335x, whereby label 2 shows where the DDR3 chip is. Between these two ICs, there are signal traces on the signal layers. Number 3 shows the position of the PMIC. Number 4 represents the JTAG interface, and numbers 5 show where two Ethernet controllers are placed. Label 6 shows the position of the SD card socket, and number 7 designates the position of the USB controller. Number 8 is the voltage source for the WLAN/Bluetooth module labelled by 9.

6.2.1 EVM-SK Power Consumption

As one of several evaluation steps we performed on the AM335x Starter Kit, power consumption was one of them. The measurement method was straightforward. We used a power supply to power the unit with 5V, and we were able to read out the power consumption of the AM335x Starter Kitwhen the processor was clocked at 275Mhz. 0.32A were drawn with display attached, and 0.2A without the display.

6.3 Implementation and Integration of the Dive Computer Subsystems

In this section, implementation and integration of every subsystem, like the MPU, DDR3 memory, LCD module, and other periphery, is given. Information included here are the schematics, explanation of the schematics, and other important information for the particular subsysten.

6.3.1 Sitara ARM Cortex-A8 AM335x Processor

The microprocessor unit used for the AM335x Starter Kit is the AM3358ZCZ.
The ZCZ suffix means a 15x15mm package is used, with a 0.8mm clearance between the ball grids. In total, there are 18x18 balls (pins).

The MPU requires a 24Mhz crystal to derive the main clock, which is mounted close.

A substantial amount of pinout configuration of the AM335x can be defined in software, which is then parsed by the bootloader. Pins which cannot be configured are the pins responsible for the powering of the AM335x submodules. As for the PCB layout,four layers are used to access all the pins of the MPU: two signal layers, the top layer and the bottom layer.

A good starting point in explaining how the layout was done for the MPU, we will start by explaining the power domains of AM335x used in the design. These are listed and explained in Table 6.1

Power domain	**Supply voltage for**	**Min. (V)**	**Max. (V)**
VDD_CORE	core domain	-0.5	1.5
VDD_MPU	MPU domain	-0.5	1.5
VDDS_DDR	DDR IO	-0.5	2.1
VDDSHVx	dual-voltage IO domain	-0.5	3.8
VDAC	dual-voltage IO domain	-0.5	2.1
VDDS_RTC	RTC domain	-0.5	2.1
VDDS_OSC	system oscillator	-0.5	2.1
VDDA_ADC	ADC domain	-0.5	2.1
VDDA1P8V_USBx	USBPHY	-0.5	2.1
VDDA3P3V_USBx	USBPHY	-0.5	4
DDR_REF	DDR SSTL and HSTL reference voltage	-0.3	1.1

Table 6.1: AM335x power domains

Also noteworthy are the operating performance points (OPP) of the MPU. AM335x has several OPPs. The operating conditions for VDD_CORE are visible in Table 6.2 and in Table 6.3 for VDD_MPU, for the ZCZ package.
The frequencies designate the maximum operating performance for the given OPP.

OPP	**VDD_CORE Voltage**	**DDR3 (MHz)**	**L3 and L4**
OPP100	Min.: 1.056V Nom.: 1.1V Max.: 1.144V	400Mhz	200MHz and 100Mhz
OPP50	Min.: 0.912V Nom.: 0.950V Max.: 0.988V	-	100MHz and 50Mhz

Table 6.2: VDD_CORE operating points [1]

OPP	**VDD_MPU Voltage**	**ARM (A8)**
Turbo	Min.: 1.210V Nom.: 1.260V Max.: 1.326V	720MHz
OPP120	Min.: 1.152V Nom.: 1.200V Max.: 1.248V	600MHz
$OPP100_1$	Min.: 1.056V Nom.: 1.1V Max.: 1.144V	500MHz
$OPP100_2$	Min.: 1.056V Nom.: 1.100V Max.: 1.144V	275MHz

Table 6.3: VDD_MPU operating points [1]

For VDD_MPU, $OPP100_2$ is only supported on some devices capable at running at 275MHz. A valid combination of these OPPs is seen in Table 6.4.

VDD_CORE	**V**DD_MPU
OPP50	$OPP100_2$
OPP100	$OPP100_1$
OPP100	OPP120
OPP100	Turbo

Table 6.4: Valid combinations for operating points [1]

After this introductory part where important concepts like power domains and operating points were introduced, we will now talk about the actual implementation of the MPU on the board.
In Figure 6.4, wee see how the pinout of the pads was achieved, on three layers: the bottom layer and the two signal layers.
The area on top layer close to AM335x is used for the decoupling of various power domains of the processor. The capacitors used are in the microfarad range. A snapshot of the top layer and

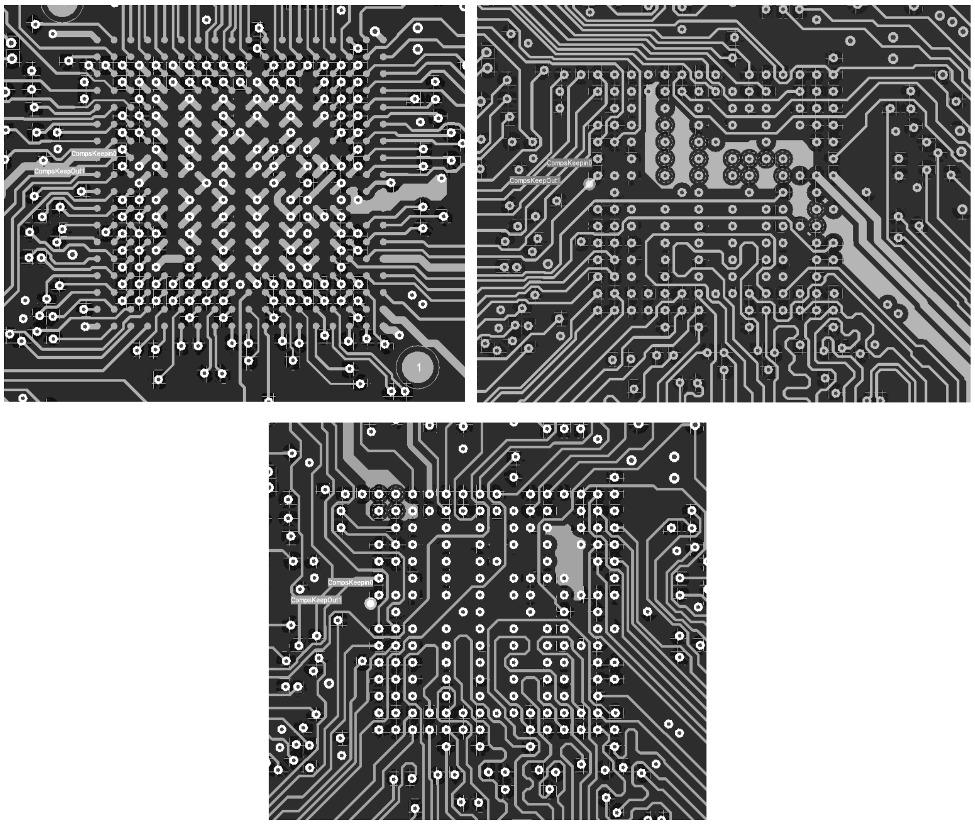


Figure 6.4: AM335x pinout on bottom, SIG1 and SIG2 layers

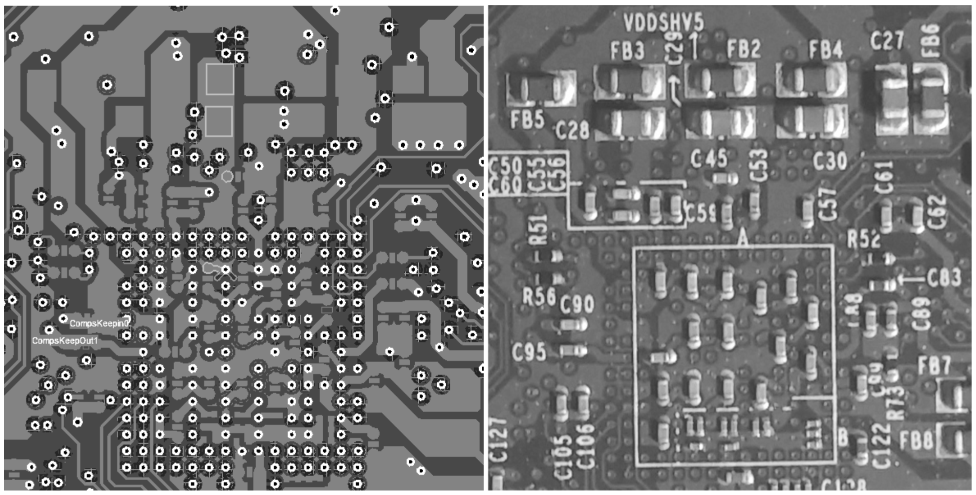


Figure 6.5: AM335x decoupling capacitors

the capacitors used are visible in Figure 6.5. The capacitors for power domains where currents at a higher frequency caused by clocking are present, the capacitors are of a small package, and the loop area is kept as small as possible. The larger capacitors on the upper part of the figure which are used to decouple the power domains such as one used for pullup for I^{2}C bus.
The capacitors used to decouple the AM335x power domains are listed in Table 6.5

Power domain	**0.01μF**	**10μF**
VDD_CORE	8	1
VDD_MPU	5	1
VDDS_DDR	19	2
VDDSHV(1-5)	2	1
VDDSHV6	6	1
VDAC	4	1

Table 6.5: AM335x decoupling capacitors

Another interesting concept are the crystals used for the clock generation in AM335x. Their schematics are depicted in Figure 6.6. Y6 is the main crystal with a resonant frequency of 24Mhz. Also, the crystal loading capacitors are visible in the schematics. Their dimension is 15pF . The second crystal oscillates at 32.768KHz, and its capacitors have a capacitance of 22pF .
The AM335X_OSC0_IN and AM335X_OSC1_IN are the oscillator frequency inputs and AM335X_OSC0_OUT and AM335X_OSC1_OUT are the oscillator frequency outputs.
Since all subsystems are in one way or another connected to the MPU, we will explain the interface for the subsystem in question in its own chapter.

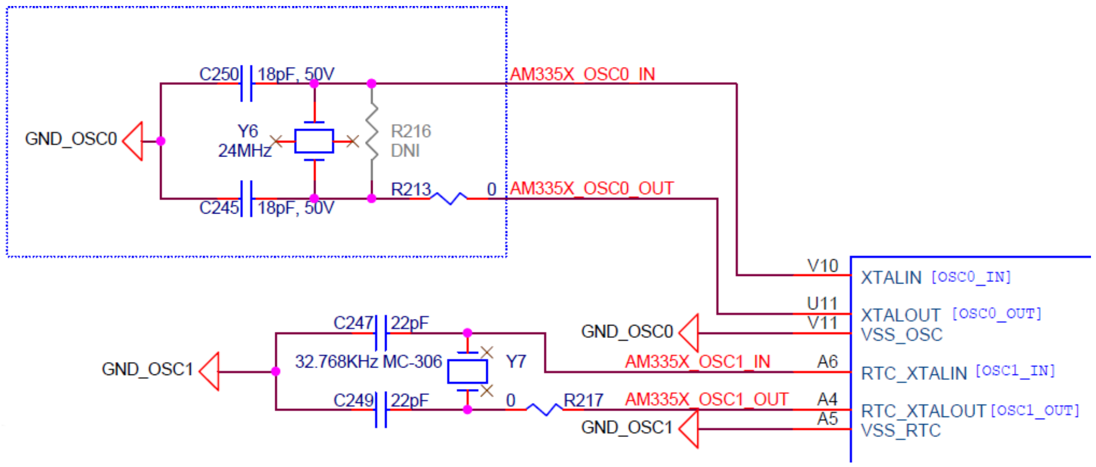


Figure 6.6: AM335x crystals schematics

6.3.2 DDR3 SDRAM

Another component used for AM335x Starter Kit which is of a particular interest is the *MT41J128M16JT-125* chip by Micron, a 2Gb DDR3 SDRAM chip. We will shortly explain the functionality of this chip, as we will use a similar one in our design. The differences will be given in the appropriate chapter.

Internally, it is configured as 16Mb x 16 x 8 banks [3]. It needs 14 bits for the row addressing, 3 bits for bank addressing, and 10 bits for the column addressing. Its page size is 2 KB, and the chip features a data rate of 1600 MT/s.

Generally, DDR3 SDRAM uses a double data rate architecture to achieve high-speed operation. The DDR3 SDRAM operates from a differential clock (CK and CK#). When CK goes HIGH and CK# goes LOW, this is referred to as a positive clock edge. Control, command, and address signals are registered at every positive edge of CK. Input data is registered on the first rising edge of differential data strobe (DQS) after the WRITE command, and output data is referenced on the first rising edge of DQS after the READ command.
An overview of the pins of the Micron chip are visible in Table 6.6.

Pin name	**Type**	**Description**
D[15:0]	I/O	Data I/O
A11, A10/AP, A[9:0]		address and auto precharge bit (A10) for READ/WRITE commands
BA[2:0]	Input	Bank address inputs
CK, CK#	Input	Clock: CK and CK# are differential clock input
CKE	Input	Clock enable
CS#	Input	Chip select
DM	Input	Input data mask
ODT	Input	On-die termination
RAS#, CAS#, WE#	Input	Command inputs
RESET#	Input	Reset: RESET# is an active LOW CMOS input.
DQ[3:0]	I/O	Data input/output: Bidirectional data bus for the x4 configuration.
DQ[7:0]	I/O	Data input/output: Bidirectional data bus for the x8 configuration.
DQS, DQS#	I/O	Data strobe
TDQS, TDQS#	Output	Termination data strobe
V_DD	Supply	Power supply: 1.5V ± 0.075V.
V_DDQ	Supply	DQ power supply
V_REFDQ	Supply	Reference voltage for data
V_SS	Supply	Ground.
V_SSQ	Supply	DQ ground
ZQ	Reference	External reference ball for output drive calibration
NC	-	No connect
NF	-	No function

Table 6.6: Micron DDR3 SDRAM pin assignment [3]

The hardware layout for the signal traces was probably done by an autorouter, as the trace lengths for the signals are almost the same for the data and clocks. Figure 6.7 shows the relative position of the chip and AM335x, and their connection. Here, number 1 represents the position of the DDR3 chip on the bottom layer. Number 2 is where the MPU is located.
Numbers 3 and 4 show where the signal termination of the data signals takes place. For high-frequency signals such as data and clock signals, where the propagation delay is relatively large compared to the rise time of the signal, distortions of the signal occur. To reduce these distortions, ringing and reflections, we need some sort of impedance continuity throughout the signal line. We do this by, e.g. introducing an equivalent amount of impedance at the point of discontinuity. Lower termination resistors provide better signal stability, at the cost of more power consumption. For the termination of the DDR3 chip, 33Ω resistors are used. *TPS51200* by Texas Instruments is used as the termination regulator, and VAUX33 is used for its power supply.

Input reference voltage is VDDS_DDR divided by 2, and output reference voltage is the DDR_VREF.
DDR_VTT_EN is connected to the enable pin, and is controlled by AM335x.
Figure 6.8 shows the physical appearance of the chip on the board. Here, we see the chip, and the termination resistor arrays.
The clock signals CK and CK# are coupled with a capacitor, to improve signal quality.
The AM335x has an EMIF which supports 16 bit data path to an external SDRAM memory including mDDR, DDR2 and DDR3.
DDR_CKE is pulled to DGND to enable self refresh.
The signals which are directly connected between the MPU and the DDR3 SDRAM are:

- DDR_RESETn, DDR_CLK, DDR_CLKn, DDR_CSN0, DDR_RASN,
- DDR_CASN, DDR_WEn, DDR_DQS0, DDR_DQSN0, DDR_DQS1,

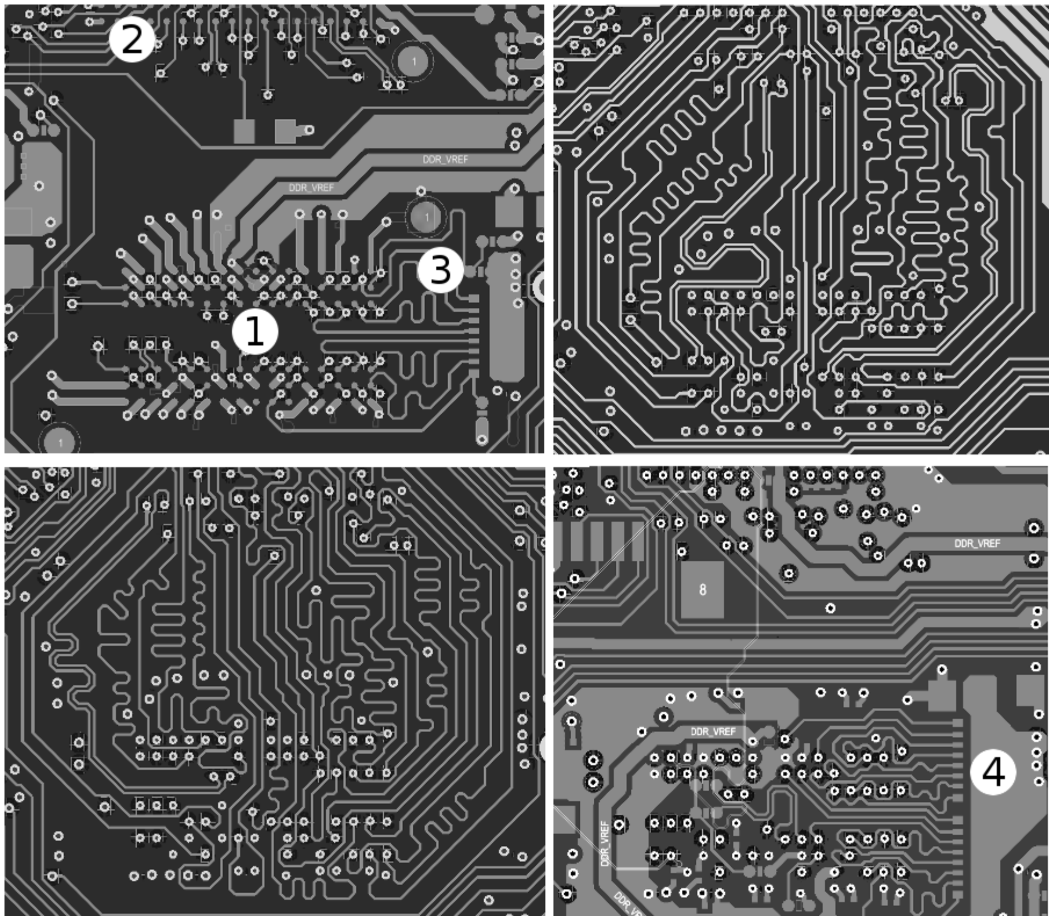


Figure 6.7: Hardware layout the AM335x and the DDR3 SDRAM

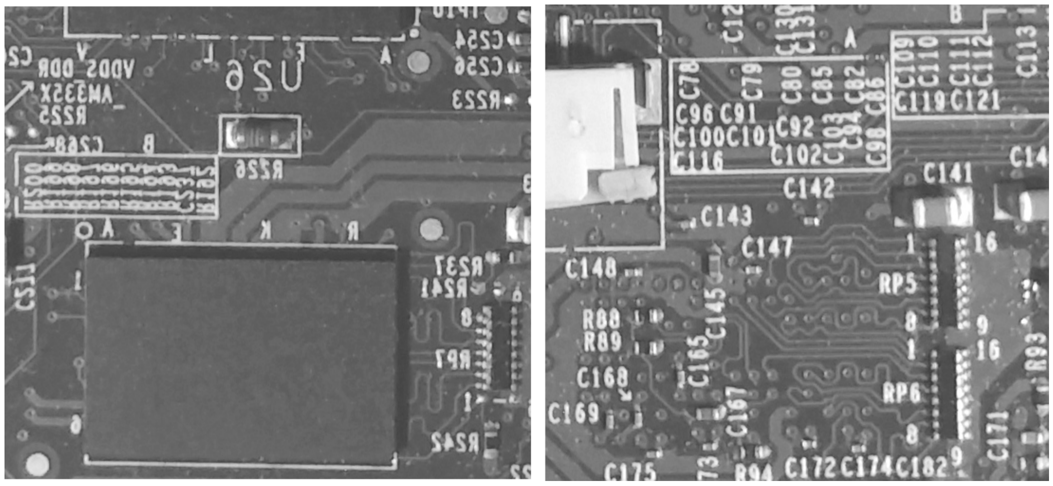


Figure 6.8: Board cut-out of the DDR3 SDRAM, bottom and top layers

- DDR_DQSN1, DDR_D[15:0], DDR_A[14:0], DDR_BA[2:0],
- DDR_ODT, DDR_DQM1, DDR_DQM0, DDR_CKE

V$_{D}DQ$ is powered by VDDS_DDR, and so is the V$_{D}D$. V$_{R}EFCA$ is connected to DDR_VREF, which is provided by the previously mentioned TPS51200.

6.3.3 SD Card Interface

Now, we will explain how the SD card socket is interfaced.
The SD card socket connected to the MMC0 port of the MPU, and the socket used is *SCHA5B0200*. The pin assignment of an SD card is shown in Table 6.7.

Since the communication with the SD card is done via two interfaces, SPI and MMC, in two different stages of booting, we show both pin assignments.

Pin	SD interface	SPI interface
1	DAT2	
2	CD/DAT3	CS
3	CMD	SDI
4	VCC	VCC
5	CLK	SCLK
6	GND	GND
7	DAT0	SDO
8	DAT1	

Table 6.7: SD card pinout

The schematics how the SD card is connected to the MMC0 port are shown in Figure 6.9. Inherently, there is nothing complicated here. In the lower left corner, the connector and the SD interface signals are shown. They are pulled high to VDDSHV4, and connected to the AM335x MMC interface. Besides this, an ESD protection is also incorporated for all the SD signals. The module which takes care of this is shown in upper right corner, and its name is TPD6E001. It offers a low capacitance +/-15-kV ESD-protection diode array for high speed communication lines. There is also an ESD protection for the CS line signal, the *TPD2E001*.

In Figure 6.10, we see the SD card connector soldered to top side of the board, with an SD card in it. Also, we see the bottom of the PCB, where two ICs responsible for the ESD protection are visible.

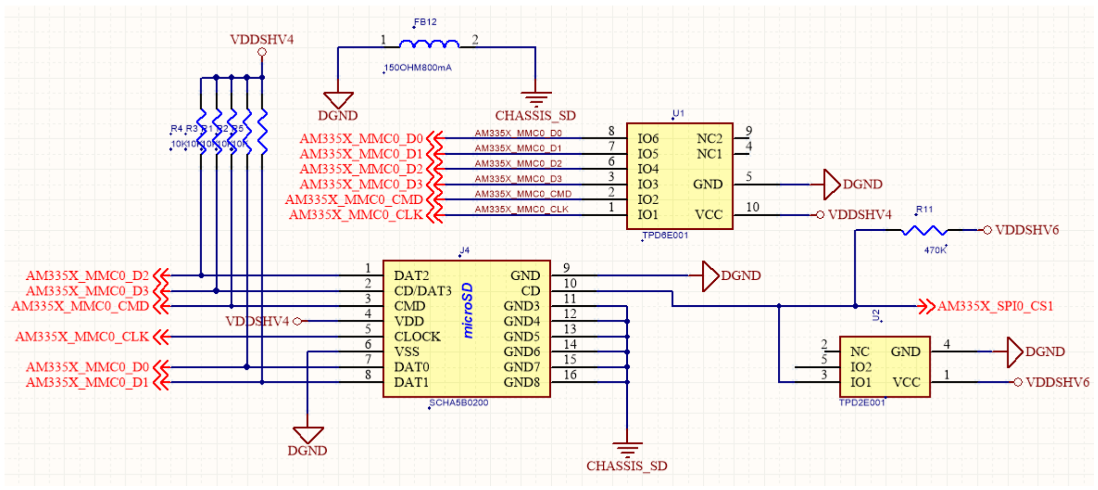


Figure 6.9: Interface between the AM335x and the SD Card slot

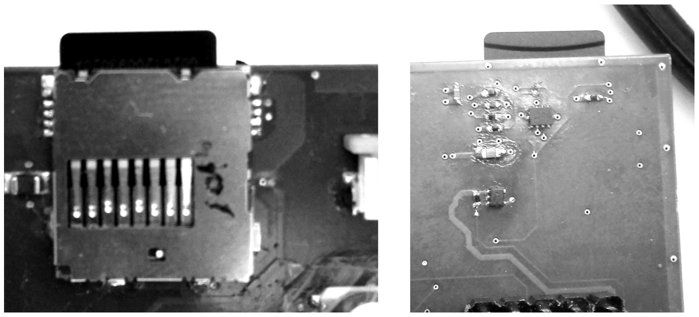


Figure 6.10: SD card slot soldered on the board

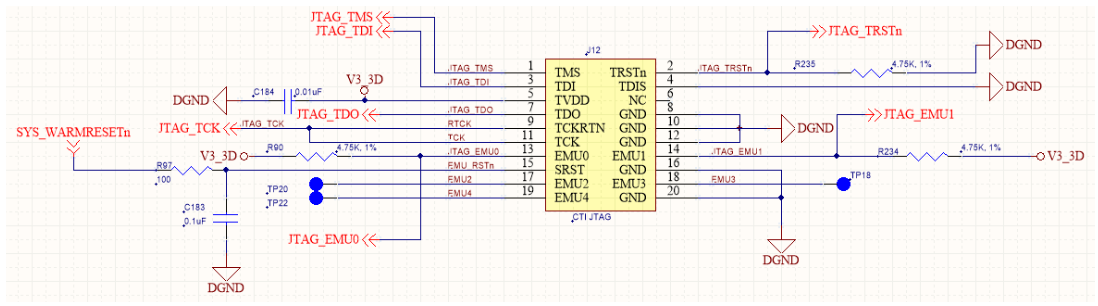


Figure 6.11: JTAG interface schematics for our board

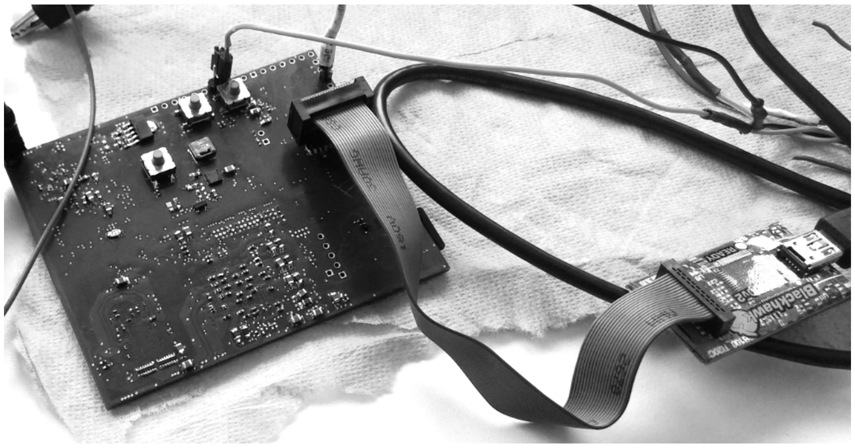


Figure 6.12: XDS200 JTAG probe connected to our board

6.3.4 JTAG

For the debugging purposes of our design, we will implement a JTAG header.
The schematics we came up with for the JTAG interface are visible in Figure 6.11. Here, we see a 20 pin header JTAG interface, with standard JTAG signals, which are then routed directly to the board. We had to make sure the data signals TD0, TRSTN, TDI, TCK, EMU0, TMS and EMU1 all have the same trace length. In our case, the length of the traces was around 95 millimetres.
We will use the XDS200 JTAG Probe by TI, with a 20 pin header. In Figure 6.12 we see the connected probe and our board. We used the probe to get the board to boot, since there were some hardware issues with the board.

6.3.5 Wireless Module

As mentioned in the chapter where we wrote about the hardware design, we decided to use a WiFi module based on RaLink *RT5370* SoC.
The dimensions of the module are 14 mm x 27 mm x 3 mm (WxLxH).

We decided not to solder the module on the PCB for space purposes. The module will rather be placed next to the board in the final product. This way, the height of the computer will not be affected.
For the communication between the AM335x and the module, we will use one of the USB interfaces of the MPU, and for on/off functionality, we will use a GPIO pin.
Since we are not using the WSP function, we will pull this terminal to the power supply, because the pin is an active low [17].
The pinout of the module is given in Table 6.8.

Pin name	**Description**
WIFI TXEN	RF on/off
VCC 3V3	Power supply
USB D-	USB Data signal
USB D+	USB Data signal
GND	Ground terminal
NC	Not used
WSP	WSP function (inverted logic)

Table 6.8: TSC_ADC_SS subsystem external interface signals

Interfacing the module should be fairly easy, since we only need to connect USB differential pair data signals to the data terminals of the module, and provide 3.3V voltage source to the module.

6.3.6 LSM303D, Compass Module

The hardware implementation of the eCompass module we decided to use will be explained now.
This module supports I^{2}C and SPI interfaces [2]. We will use the I^{2}C interface in our implementation.
The *LSM303D* is factory calibrated to be powered by 2.5V, but it supports a supply voltage in range between 2.16V and 3.5V, so we will use the 3.3V voltage supply. This module requires two capacitors. The reservoir capacitor has a nominal value of 4.7μF , and the set/reset capacitor has a value of 0.22μF . We decided to stick with these values. To prevent coupling and skewing of the values of the magnetic field read by the module, we set a margin of 5 millimetres between the IC and the capacitors.
We will use the first I^{2}C interface of AM335x. The schematics for the module implementation can be obtained from Figure 6.13. How these schematics were utilized during the PCB design, can be seen in Figure 6.14. We see the positions of the capacitors, and we see the I^{2}C traces on the bottom.
The module is placed on the bottom side with the rest of the active components.

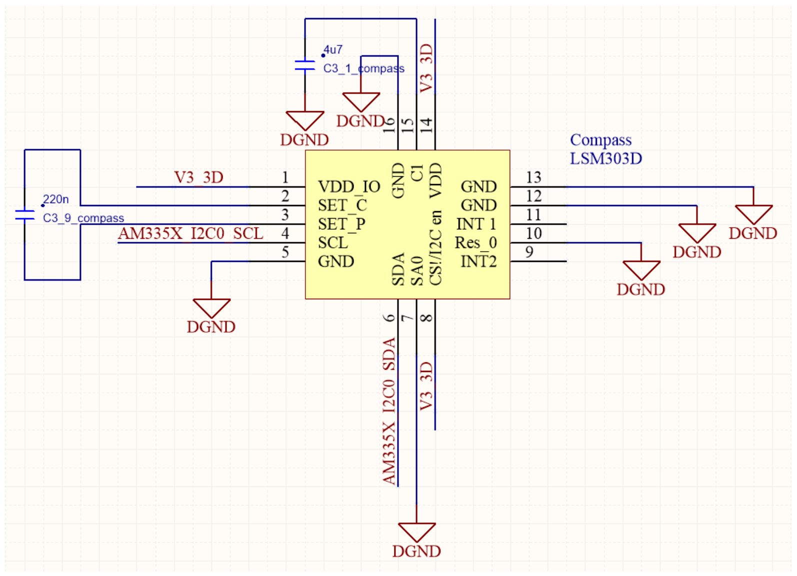


Figure 6.13: Schematics for the eCompass module

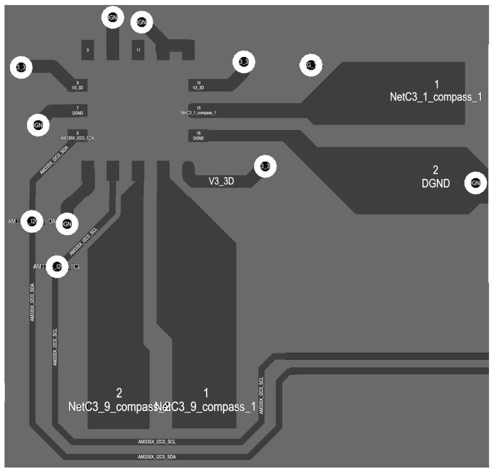


Figure 6.14: PCB design for the eCompass module

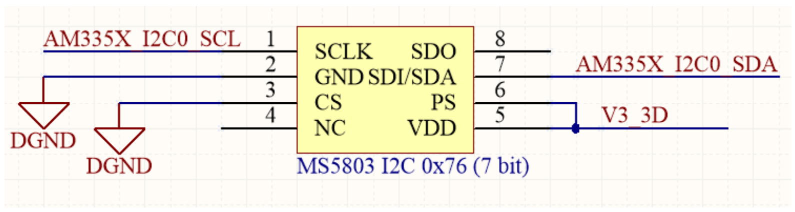


Figure 6.15: Schematics for the pressure and temperature sensor module

6.3.7 MS5803, Pressure and Temperature Sensor

Besides the eCompass, we use the sensor for pressure and temperature. This module also supports I^{2}C for communication, and we will use the same I^{2}C bus as for the eCompass module, i.e. the first one. We pull the protocol select (PS) signal to high, to select I^{2}C. The CS signal is pulled to the ground, to select the 0x76 address for the module [4].
In Figure 6.15, we show how this module looks in reality. It has a significant height profile, and we placed it on the same size as other active components.

6.3.8 Power Supply and Power Management

The EVM-SK is powered by a 5V power supply, which is provided as the power input to the Power Manager *TPS65910*. The *TPS65910* takes care of the power requirements and power sequencing of the AM3358. The PMIC is controlled by AM3358 via I^{2}C.
TPS65910 device is an integrated PMIC dedicated to applications powered by Li-ion battery, or a 5V input. It has three step-down converters, one step-up converter, and eight LDOs [29].
Two of the step-down converters provide power for dual processor cores. The third converter provides power for the I/Os and memory in the system.
We will slightly modify the layout of the PCB concerning the powering of MPU and the peripherals, specifically the position of the PMIC relative to the other components. Other than that, the PMIC configuration and the connection with the MPU and the DDR IC will stay the same.

6.3.9 LCD and Touchscreen Implementation

An LCD module we decided to use is the *P320X-35ALWS* by Electronic Assembly. Internally, it has an *HX8238-D* controller, and is interfaced over 24-bit RGB interface. Besides this interface, SPI interface is also present, but is merely used for changing display settings, and is not crucial for the functioning of the display or showing an image on it.
For this interface, we use the following signals between the AM335x and the LCD: HSYNC, VSYNC, DATA[23..0], CLK, and DISEN.
The 24-bit wide RGB signals, control signals and touch screen signals are terminated in a flex PCB with a 40 pin connector connected to board.
In Figure 6.16, the schematics of the interface between the AM335x and the LCD module are given.

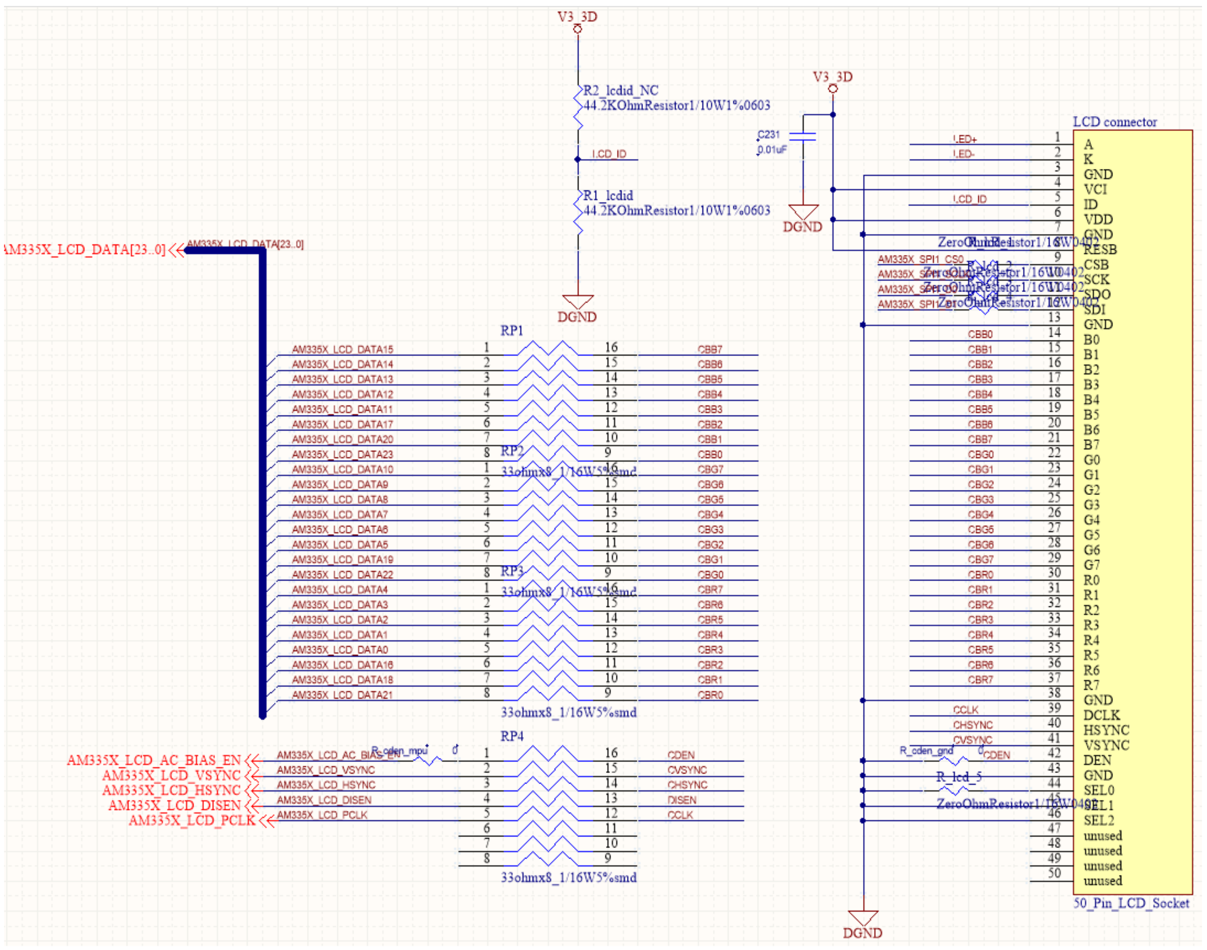


Figure 6.16: Schematics for the LCD module

On the left side, we see the data signals, which are connected to the LCD module via 33Ω termination resistors. The We also notice the control signals of the LCD controller on the bottom, and how they are connected to the display. The connector for the display will be placed on the top side with the active components. When the LCD is attached, the cable will wrap around the board, and the backplate of the display will cover the other side of the board, where the passive components are.
As for the powering of the controller, a voltage of 3.3V is required, and the typical power consumption is around 9mA.
As for the backlight, a voltage in range of 19.2V to 21V is needed, and the typical current supplied to it should not exceed 48mA.
Enable pins of both step-down and buck-boost converter are connected to VMMC power rail of the AM335x.
As for the buck-boost converter, the enable pin only charges the outer capacitor. Full operation is enabled by the LCD_BACKLIGHTEN pin coming from the AM335x.

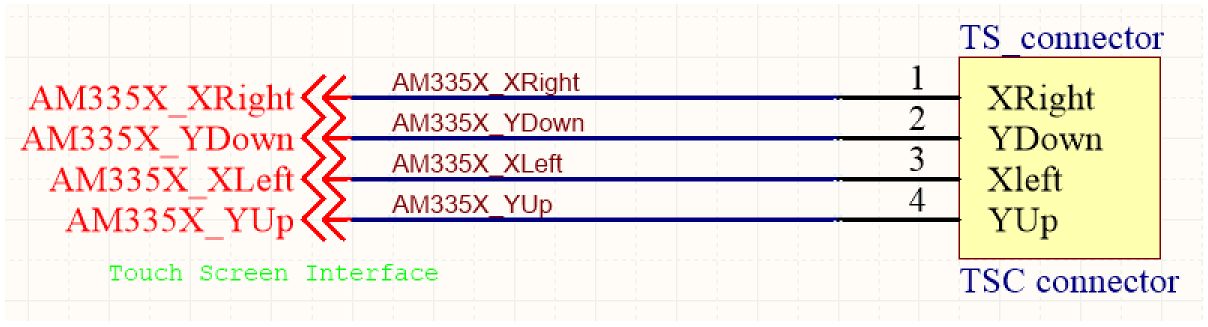


Figure 6.17: Schematics for the resistive touchscreen

6.3.9.1 Resistive Touchscreen

Besides the LCD module, we use 4 pin resistive touch screen which works under water. Interfacing this screen is fairly easy, we just need to connect the four analogue signals to the ADC module pins A0 to A3 of the AM335x. This is shown in Figure 6.17.
A resistive touchscreen consists of several layers. On the top and bottom side, conductive layers which are made of Indium-Tin-Oxide (ITO), a transparent conductive material are present. Between these layers, an insulating layer of air (or in our case, oil) is present. Besides these thee basic layers, other layers may be employed (on the outside) to ensure the sturdiness of the sensor.
There are four wires connected to the ITO layers. On the right and the left side of the X layer, and upper and lower side of the Y layer. When a touch is performed, the top and bottom layers touch. Since voltage is applied to one of the layers, a voltage divider is created.

To calculate the Y coordinate of the touch, X right is driven to a known voltage, and X left wire is driven to ground. When a touch occurs, a voltage drop is read, and used to derive the Y coordinate, i.e. the ratio of the measured voltage to the drive voltage applied is equal to the ratio of the y coordinate to the height of the touchscreen [30].
The same is done for the Y layer, to read the X coordinate. This coordinate pair gives us an approximate position of the touch. This concept is visible in Figure 6.18.

To implement a touchscreen sensor interrupt, a pull up resistor, whose resistance must be higher than the one of the touch sensor is used. A positive voltage is applied to Y+ through a pull up resistor, and X- is driven to ground. With no touch present, Y+ is pulled to positive voltage. In case there is a touch, Y+ is pulled to ground.

6.3.10 Overview of the Finished Board

This short chapter will now visually present the board which was made during the course of this work.
In Figure 6.19, we see the bottom and top layers of the our board. Number 1 shows where the AM335x is placed. Number 2 tells us where we placed the DDR3 chip. Next up is the PMIC

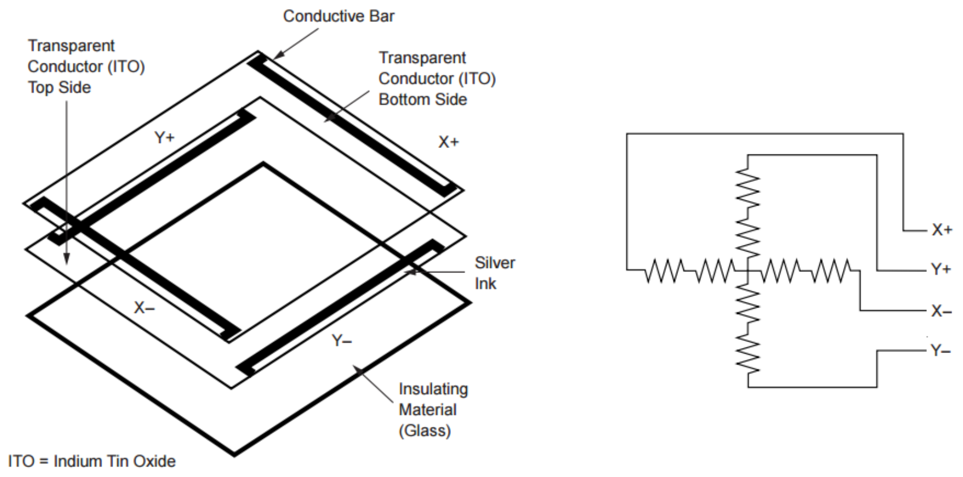

Conductive Bar
Transparent Conductor (ITO) Top Side
Transparent Conductor (ITO) Bottom Side
Y+
X+
Silver Ink
X–
Y–
Insulating Material (Glass)
ITO = Indium Tin Oxide
X+
Y+
X–
Y–

Figure 6.18: Resistive touchscreen conceptual design

labelled by a 3. SD card socket is labelled by a 4 on the left side, and on the right side of the figure, number 4 shows the position of the pull up resistors for the SD card signals, and its ESD protection.
Number 5 shows the position of the USB hub. Sensors are placed around number 6 - compass on the left side, and the pressure and temperature sensor on the right. Number 7 on the left side is the position of the LCD connector, and on the right side, the position of the backlight circuitry.
Label 8 shows us the position of the JTAG connector, and number 9 is the touchscreen connector.
In Figure 6.20, we see the signal layers. Here we see how these subsystems are interconnected.
In Figure 6.21, the front side of board with an LCD and the touchscreen are visible.
In Figure 6.22, the back side of the board is visible. Here, the most notable component is the JTAG connector.
We also made some pictures during the hardware debug phase. In Figure 6.23 we see the SPI clock measured on the SD card.
In Figure 6.24, we see this first built prototype, where we improvised with an SD card adapter, instead of the socket, which was not available at the time.
The workspaces for the soldering and power measurement, and for the software development are visible in Figure 6.25 and Figure 6.26, respectively. In Figure 6.27, we see an LCD attached to the board, whose timings were not properly configured at the time, so a distorted image was shown.

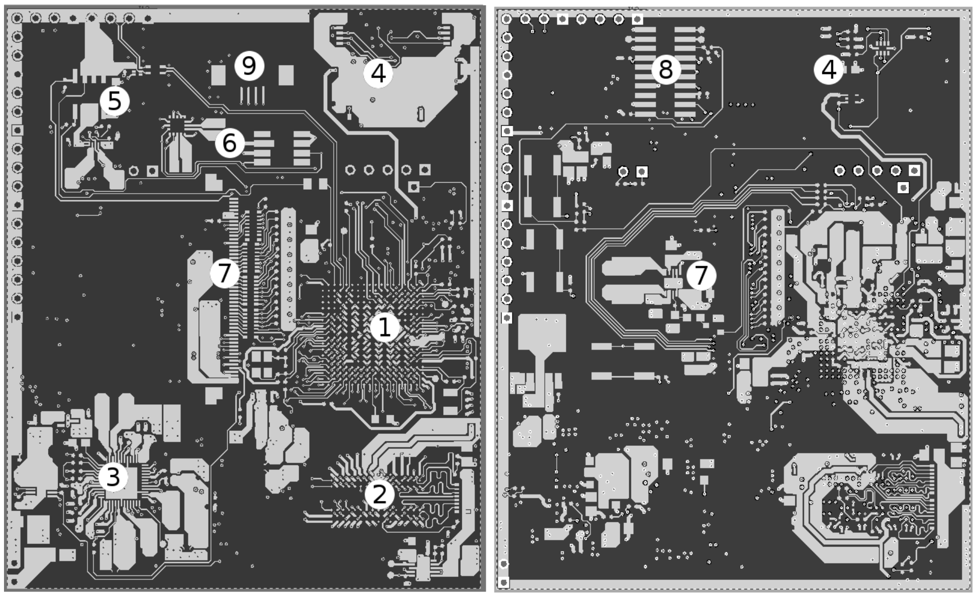


Figure 6.19: AM335xbottom and top layers of the prototype board

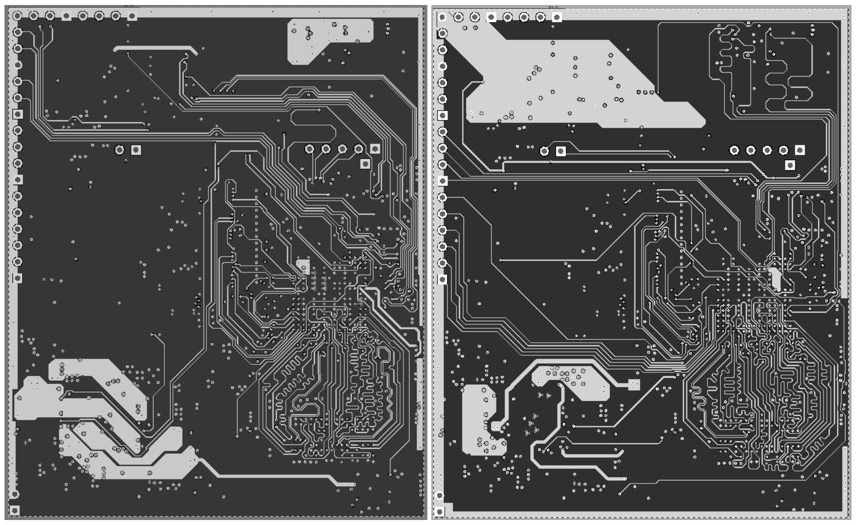


Figure 6.20: AM335xSIG1 and SIG2 layers of the prototype board

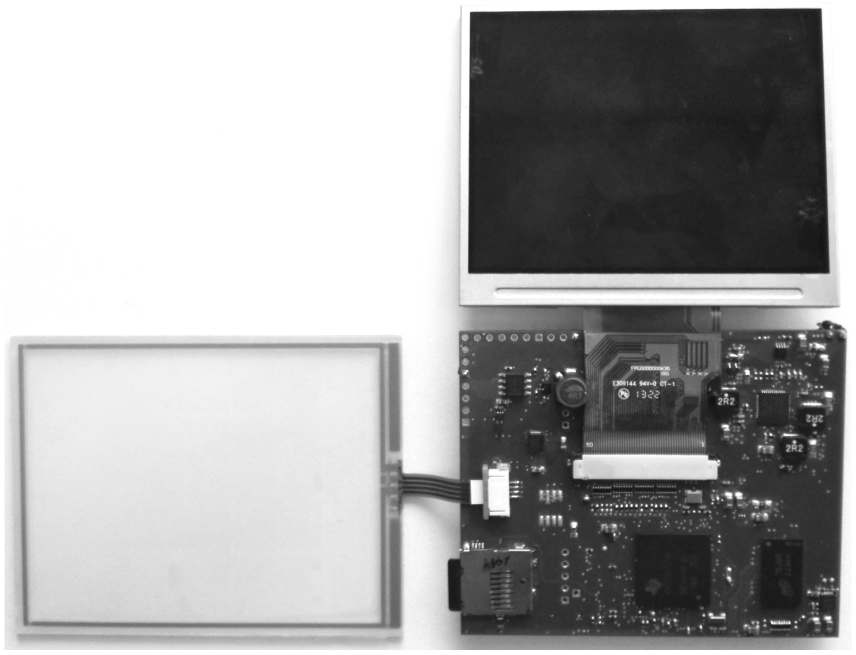


Figure 6.21: Front side of the board, with LCD and touchscreen attached to it

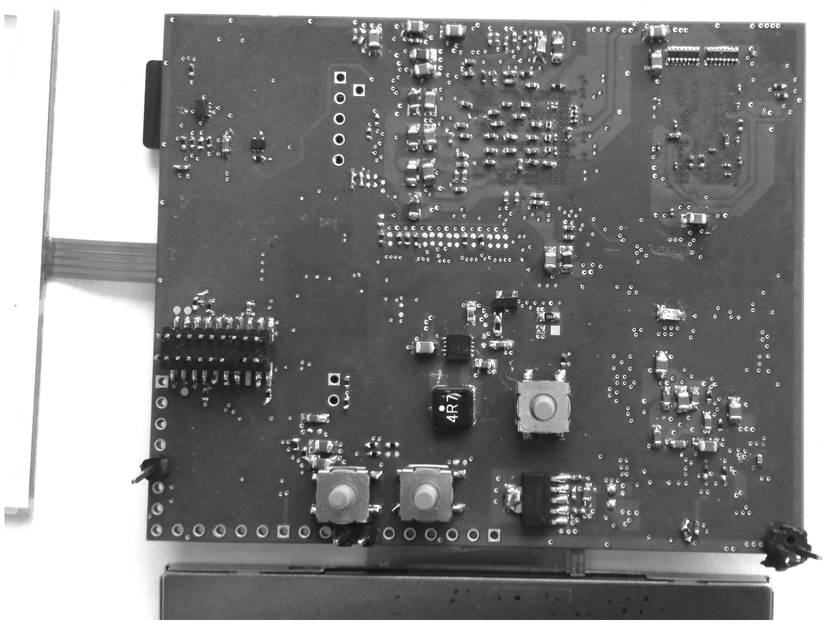


Figure 6.22: Back side of the board

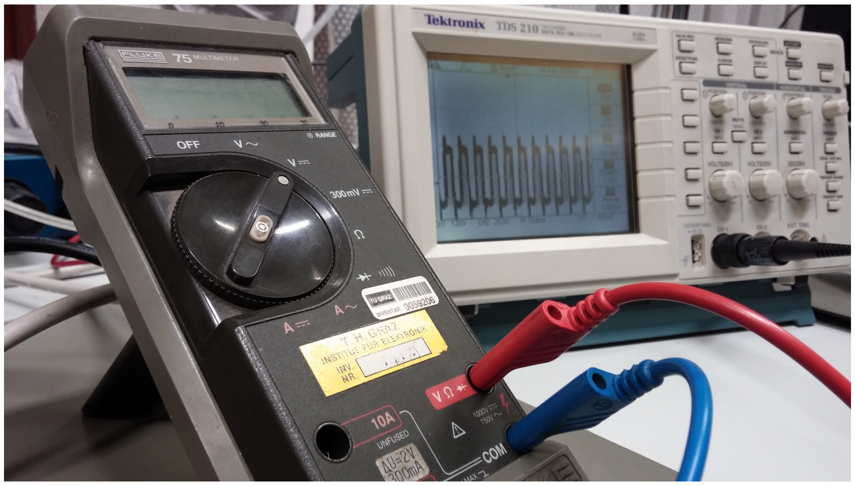


Figure 6.23: Measuring the SD card SPI clock signal on first prototype board

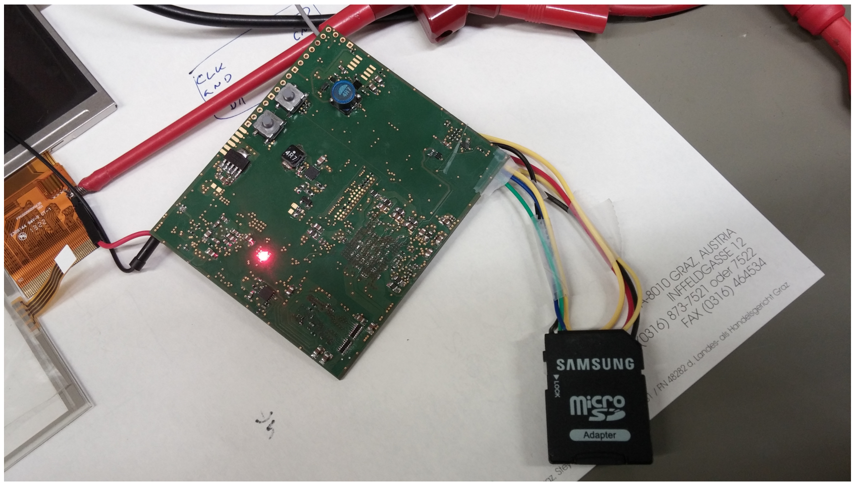


Figure 6.24: First build prototype with improvised SD card connector

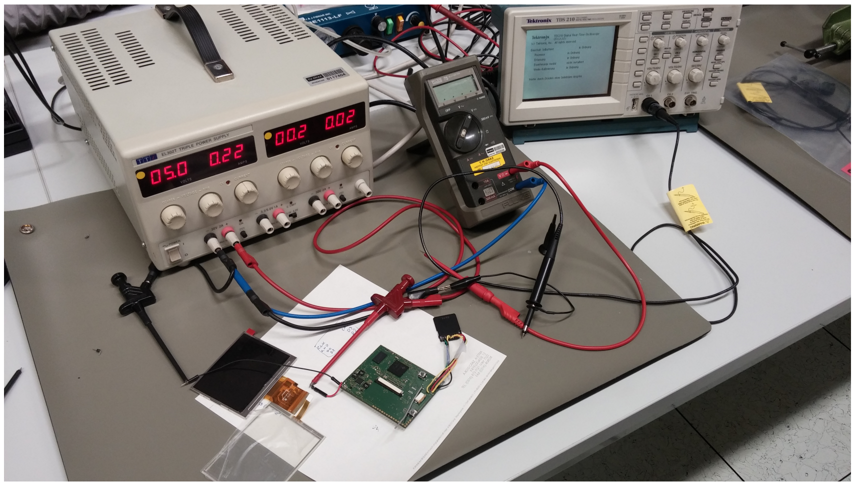


Figure 6.25: Workspace for power measurement

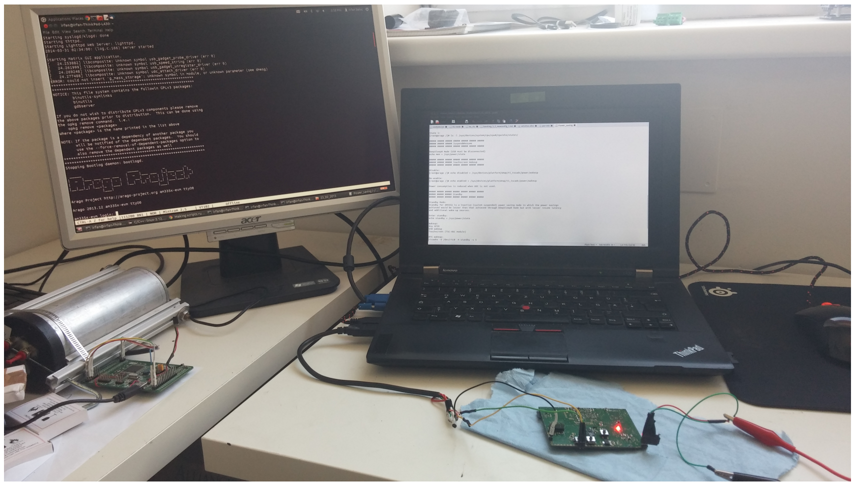


Figure 6.26: Workspace for software development

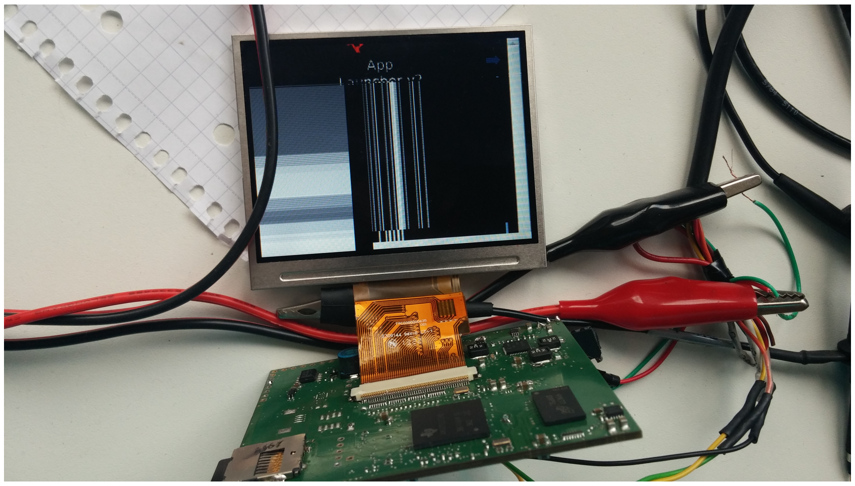


Figure 6.27: Second prototype with LCD with wrong timing config connected to it

Chapter 7

Software Implementation

In this chapter, we briefly explain what software modules were implemented in the course of this work.
Besides the newly implemented software, like the Qt application, some of the pre-existing software modules were changed to suit the needs of out board.

We give more information about the software adaptation made to the bootloader provided by TI, and the operating system. Besides this, we explain the implementation of the software modules which are directly used by the user, which were shortly introduced in Chapter 5.

The chapter is divided as follows: In Chapter 7.1, we talk about the device tree adaptation to our board, configuration of the bootloader including the main memory configuration, and the configuration of Linux.

In Chapter 7.3, we introduce the implementation of the drivers for the pressure and temperature sensor, and the eCompass, the Qt application, and in Chapter 7.4, we will give a brief overview of some of the useful tools which are offered by the Linux kernel, and we will explain their purpose and how to use them.

7.1 Configuring the Hardware

This chapter explains the adaptation which had to be made to the software which serves as the abstraction layer between the operating system and the hardware. This includes the device tree and the bootloader.

7.1.1 Device tree

As our starting point for our board's device tree, we will use the device tree written for AM335x Starter Kit. In Linux kernel code, it is located in arch/arm/boot/dts/am335x-evmsk.dts. It

includes all of the periphery which is present on the AM335x Starter Kit.
The nodes in the device tree include the CPU, memory, battery regulators, status LED's, GPIO buttons, LCD panel and backlight, sound, UART, I^{2}C, USB, PMU, MMC, touchscreen interface, and the pinmux configuration.

For the most part, we will keep the device tree, as most of the hardware is the same, but we will need to remove some nodes from it. We will also introduce some minor changes on our own, like the LCD panel timing configuration, and the I^{2}C chips we will use.
We will not embed the finished device tree in this document, as it would span to more than 5 pages. Rather than that, some snippets will be shown and briefly explained, just for the sake of understanding on what was done.
The device tree document starts with several statements which include other device tree files. The important one is the *am33xx.dtsi*, which is the device tree included file which contains the device tree source for the AM3xx SoC, ranging from the definition of the CPU core, to all the interface configurations it offers.
In the *am335x − evmsk.dts* file, the CPU configuration is expanded by adding the cpu0-supply entry and the *vdd1_reg*, which is a regulator for VDD_MPU domain controlled by the PMU.

```
#include "am33xx.dtsi"
#include <dt-bindings/pwm/pwm.h>
#include <dt-bindings/interrupt-controller/irq.h>

{
        model = "TI AM335x EVM-SK";
        compatible = "ti,am335x-evmsk", "ti,am33xx";

        cpus {
                cpu@0 {
                        cpu0-supply = <&vdd1_reg>;
                };
        };

        memory {
                device_type = "memory";
                reg = <0x80000000 0x10000000>; /* 256 MB */
        };

        ....
```

Next up is an example on how the LCD panel and backlight are configured. The values for the timings were taken from the datasheet of the panel we used.
This is an example on where our device tree differs from the AM335x Starter Kitdevice tree. We see the definition of the resolution of the display, and the timing values.

```
        backlight {
                compatible = "pwm-backlight";
                pwms = <&ecap2 0 50000 PWM_POLARITY_INVERTED>;
                brightness-levels = <0 58 61 66 75 90 125 170 255>;
                default-brightness-level = <8>;
        };

        panel {
                compatible = "ti,tilcdc,panel";
                pinctrl-names = "default", "sleep";
                pinctrl-0 = <&lcd_pins_default>;
                pinctrl-1 = <&lcd_pins_sleep>;
                status = "okay";
                panel-info {
                        ac-bias           = <255>;
                        ac-bias-intrpt    = <0>;
                        dma-burst-sz      = <16>;
                        bpp               = <32>;
                        fdd               = <0x80>;
                        sync-edge         = <0>;
                        sync-ctrl         = <1>;
                        raster-order      = <0>;
                        fifo-th           = <0>;
                };
                display-timings {
                        320x240 {
                                hactive         = <320>;
                                vactive         = <240>;
                                hback-porch     = <40>;
                                hfront-porch    = <8>;
                                hsync-len       = <4>;
                                vback-porch     = <12>;
                                vfront-porch    = <4>;
                                vsync-len       = <10>;
                                clock-frequency = <2000000>;
                                hsync-active    = <0>;
                                vsync-active    = <0>;
                        };
                };
        };
```

7.1.2 Boot Source Configuration

The AM335x processors support multiple boot sources for the second stage bootloader. The boot mode is defined by the pull up/down resistor combinations on the SYS_BOOT pins, which are multiplexed with the LCD_DATA0...LCD_DATA15 pins. These configuration pins are *parsed* when PORz pin is low.

The AM335x bootloader can be loaded over MMC/SD, SPI, Universal Asynchronous Receiver/-Transmitter (UART), etc.
The SYS_BOOT configuration on the AM335x Starter Kit is as follows: MMC0, SPI0, UART0, USB. We will keep this order, and we also boot from the SD card.

7.1.3 EMIF Module Configuration

This section will explain the external memory interface module configuration for a proper interfacing of the DDR memory. As explained in Section 4.0.9, several memory types can be interfaced by EMI, including LPDDR, DDR2 and DDR3. In our case, we use a DDR3 memory chip, so we will explain the steps taken to configure EMIF for DDR3.

7.1.3.1 EMIF Register Description

As a first step of the EMIF configuration, the SDRAM_CONFIG register at an address 0x4C000008 needs to be configured. The values in this register are used to initialize the memory module, and the values are then written to proper registers in the DDR3 memory device [1]. The values in this register are explained in Table 7.1.

Register bit	**Register field**	**Description**
31-29	REG_SDRAM_TYPE	Memory type: LPDDR1 (mDDR) = 1, DDR2 = 2, DDR3 = 3
28-27	REG_IBANK_POS	Internal bank position
26-24	REG_DDR_TERM	DDR2 and DDR3 termination resistor value
23	REG_DDR2_DDQS	Set to 1 when using DDR2/DDR3 for differential DQS
22-21	REG_DYN_ODT	These bits are only used in DDR3 mode
20	REG_DDR_DISABLE_DLL	Set to 0 for normal operation
19-18	REG_SDRAM_DRIVE	For DDR2, set to 0 for full drive strength
17-16	REG_CWL	CAS Write Latency; Only used in DDR3 mode
15-14	REG_NARROW_MODE	Set to 1 for 16bit mode
13-10	REG_CL	CAS Latency
9-7	REG_ROWSIZE	sets the number of row address bits
6-4	REG_IBANK	sets the number of banks
3	REG_EBANK	external chip select setup
2-0	REG_PAGESIZE	sets the number of column address bits

Table 7.1: SDRAM_CONFIG register fields and their description [1]

7.1.3.2 EMIF Register Configuration for MT41J128M-125 (2Gb)

The Starter Kit uses a *MT41J128M-125* chip my Micron with 2Gb capacity. The values calculated for this chip are given in Table 7.2

Register field	**Value**	**Explanation**
REG_SDRAM_TYPE	3	DDR3
REG_IBANK_POS	0	first bank
REG_DDR_TERM	1	RZQ2
REG_DDR2_DDQS	1	differential DQS
REG_DYN_ODT	2	Dynamic ODT RZQ4
REG_DDR_DISABLE_DLL	0	enable DLL
REG_SDRAM_DRIVE	0	drive strength RZQ7
REG_CWL	0	CAS write latency 7
REG_NARROW_MODE	1	16bit data bus width
REG_CL	2	CAS latency of 9
REG_ROWSIZE	5	14 row bits
REG_IBANK	3	8 banks
REG_EBANK	0	1 chip select
REG_PAGESIZE	2	10 column bits

Table 7.2: SDRAM_CONFIG register values for MT41J128M-125 (2Gb)

The value for the SDRAM_CONFIG we get for this chip is thus
0b01100001110000000100101010110010 or 0x61C24AB2.

7.1.3.3 EMIF Register Configuration for MT41J64M16-15E (1Gb)

The register value for the *MT41J64M16-15E*, the chip with 1 Gb capacity, used on our board is derived from the values in Table 7.3.

Register field	**Value**	**Explanation**
REG_SDRAM_TYPE	3	DDR3
REG_IBANK_POS	0	first bank
REG_DDR_TERM	1	RZQ2
REG_DDR2_DDQS	1	differential DQS
REG_DYN_ODT	2	Dynamic ODT RZQ4
REG_DDR_DISABLE_DLL	0	enable DLL
REG_SDRAM_DRIVE	0	drive strength RZQ7
REG_CWL	0	CAS write latency 7
REG_NARROW_MODE	1	16bit data bus width
REG_CL	2	CAS latency of 9
REG_ROWSIZE	4	13 row bits
REG_IBANK	3	8 banks
REG_EBANK	0	1 chip select
REG_PAGESIZE	2	10 column bits

Table 7.3: SDRAM_CONFIG register values for MT41J64M16-15E (1Gb)

We see that the difference to the 2Gb chip is in the number of row bits.
The value for the SDRAM_CONFIG we get for this chip is
0b01100001110000000100101000110010 or 0x61c04a32.

7.1.3.4 DDR3 Timing Settings

Besides the SDCFG register, DDR timing values also need to be properly configured. For this matter, TI provided a spreadsheet file. This file can be used to automatically derive values for SDRAM_TIMING1, SDRAM_TIMING2 and SDRAM_TIMING3 variables in the bootloader code.
The values which need to be inserted in this file must be obtained from the datasheet for the *MT41J64M16-15E* chip. Field names and values which need to be entered in the spreadsheet are visible in Table 7.4.

Register name	**Bit length**	**DDR3 symbol**	**Value**	**Unit**
REG_T_RP	4	tRP	13.5	ns
REG_T_RCD	4	tRCD	13.5	ns
REG_T_WR	4	tWR	15	ns
REG_T_RAS	5	tRAS	36	ns
REG_T_RC	6	tRC	49.5	ns
REG_T_RRD	3	tRRD	4	ns
REG_T_WTR	3	tWTR	4	ns
REG_T_XP	3	tXP	3	ns
REG_T_ODT	3	ODTlon	3	tCK
REG_T_XSNR	9	tXS	120	ns
REG_T_XSRD	10	tXSDLL	512	tCK
REG_T_RTP	3	tRTP	4	tCK
REG_T_CKE	3	3	tCKE	tCK
REG_T_PDLL_UL	4			
REG_T_ZQCS	6	tZQCS	64	tCK

Table 7.4: DDR3 ratio seed spreadsheet values for MT41J64M16-15E

The values for the timing registers we end up with, when we enter them in the file, are given in Table 7.5.

Register name	**Value**
SDRAM_TIMING1	0x0888A39B
SDRAM_TIMING2	0x26247FDA
SDRAM_TIMING3	0x501F821F

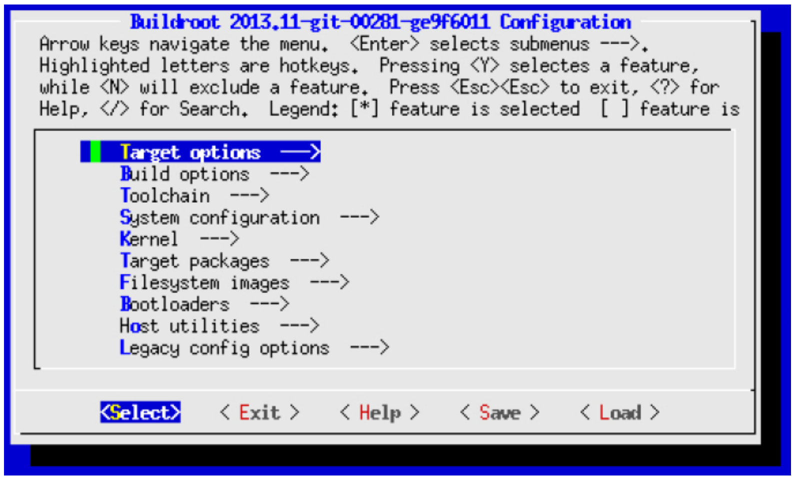
Table 7.5: DDR3 timing register values for MT41J64M16-15E

This concludes the part about the DDR configuration.

7.2 Buildroot

We used Buildroot to generate the embedded Linux image, which we then flashed to our SD card.

It is said that Buildroot is a set of Makefiles which automate the process of building of the complete Linux environment, starting from bootloader, to the kernel we configured, to the filesystem. It is a convenient system, where one can configure what bootloader to use for a specified platform, set up the boot arguments, the path to the device tree, kernel tree, and even checkout a certain revision.



```
Buildroot 2013.11-git-00281-ge9f6011 Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is

        Target options  --->
        Build options  --->
        Toolchain  --->
        System configuration  --->
        Kernel  --->
        Target packages  --->
        Filesystem images  --->
        Bootloaders  --->
        Host utilities  --->
        Legacy config options  --->

    <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 7.1: Schematics for the pressure and temperature sensor module

After the bootloader and kernel were chosen and configured, we can choose what userspace programs we want to include in our system. The applications are divided in a category, and picking the right ones we needed (e.g. Qt framework, some utilities we used, RaLink firmware and so on) was self-explanatory.

A welcome screen of a Buildroot can be seen in Figure 7.1.

Buildroot is suitable for this type of projects, since the generated result (image) is as small as possible, and boots fast, since only the necessary modules are loaded during boot.

Buildroot is free and open-source software, maintained by Peter Korsgaard and licensed under version 2 or later of the GNU General Public License (GPL).[4] The project started in 2001, with initial intentions to serve as a testbed for uClibc. New releases are made available every three months [31].

7.2.1 Software Configuration for the WiFi Module

For some of the hardware we mentioned in the chapter about the hardware design, we will require some configuration steps for the Linux kernel. For our WiFi requirements, we are using a RaLink *RT5370* based chip. Linux kernel supports the RaLink 802.11n USB chipsets since version 2.6.31. Among these devices is our *RT5370* [32].
The driver in question is the $rt2800usb$. It is already in the kernel main tree. so we will need to include this driver in our kernel. Since we have the USB1 interface ready on our board, we must also make sure the USB Wireless Device Management is supported.
In the userspace, we will require the RaLink firmware. The package we need is called $firmware-ralink$ firmware. It is closed source, and contains the binary firmware for the wireless card we using.
We will be using the WiFi in access point mode, so we will need some basic network interface configuration, by editing the /etc/network/interfaces file, and setting the IP address and netmask

for it.
Enabling the driver and the firmware for the WiFi module will allow us to use the WiFi on our board, but to add usefulness to it, we will need a server to which the clients will connect, and browse the files, or upload a new software version. We write about the server in Chapter 7.3.4.

7.3 Sensors Driver Implementation

In this part, we give an overview of the implementation of the user-space drivers for the eCompass and the pressure and temperature sensor. These drivers serve as the interface for the modules which need to communicate with these devices. Besides the drivers, we introduce the Qt application which was made during the course of this work.

7.3.1 Pressure and Temperature Sensor Driver

Here, we will briefly introduce the software implementation of the sensor for the pressure and temperature used on the board. The implementation of the driver was done in C++. A single class called *MS5803* was implemented.
The following, summarized header file gives us the information on the member variables and functions used.

```
class MS5803
{
        private:
                int i2c_fd;
                uint16_t calib_coeff[NUM_CALIB];
                uint32_t read_value(uint8_t command);
                int  read_calibration_data(void);
                void convert_sensor_values(uint32_t press_raw, uint32_t tem_raw,

        public:
                MS5803();
                ~MS5803();
                int ms5803_init(const char *f_dev, unsigned char address);
                int ms5803_get_sensor_values(int32_t *pressure, int32_t *tempera
                int ms5803_close();
}
```

We also define some commands which are sent to the module. These are give in Table 7.6.

Command	**Value**	**Description**
OSR_4096_PRESS	0x48	Convert D1 with highest resolution (4096)
OSR_4096_TEMP	0x58	Convert D2 with highest resolution (4096)
PROM_READ_START	0xA0	PROM read start address
RESET_COMM	0x1E	Reset

Table 7.6: Commands used in MS5803 driver [4]

The bus used in our application is located on path $/dev/i2c-1$.
i2c_fd holds the number of the I^{2}C file descriptor, which is used by other functions to access the I^{2}C bus. This way, the driver can write to, and request data from sensor.
By calling *ms5803_init*, the I^{2}C bus is set up, the sensor is reset, and the *read_calibration_data* function is called. In this function, calibration data is read out and stored in *calib_coeff* array.
Reading pressure and temperature values is done by calling
ms5803_get_sensor_values, which reads out uncompensated pressure and temperature data, then calls *convert_sensor_values* to compensate the values.
When done using the driver, the caller should call *ms5803_close* to close the file.
Initially, this was a standalone executable, but was later integrated in the Qt application which will later be more expanded upon.

7.3.2 eCompass Driver

As for the eCompass driver implementation, it is similar to the pressure and temperature sensor. They both use the same way of communication over the I^{2}C interface. The difference is in the commands which are sent to the sensor, and in the values the driver receives back.
Here, we have an overview of the *LSM303* class, and the member functions it has.

```
class LSM303
{

#define ACC_REG 0x30>>1;
#define MAG_REG 0x3C>>1;
  public:
    enum register_addr
    {
                // register addresses
                ...
         }
    vector<int16_t> acc_val;
    vector<int16_t> mag_val;
         void init();

    void write_register(uint8_t addr, uint8_t reg, uint8_t val);
    void read_register(uint8_t addr, uint8_t reg, uint8_t nr_bytes);

    void read_accel_values();
    void read_mag_values();

    float heading();
```

The implementation was done in C++. Member variables acc_val and mag_val hold the read data from the accelerometer and magnetometer respectively. Each of the sensors sends 6 bytes of data, 2 bytes for each axis. A high and low byte value of an axis are then stored in an signed integer, since the values are two's complement.
The *init()* function initializes the I^2C bus, and tries to establish a communication with the sensor. If the communication is successful, configuration values are written in the respective registers. write_register(...) function is used to write a value *val* to a sensor with address *addr*, to register *reg*. We use the function read_register(...) to read *nr_bytes* many bytes from a sensor on address *addr*, after we send it a command *reg*. read_accel_values() is a wrapper for these two functions with which we read out the 6 bytes for the accelerometer values. *read_mag_values()* helps us read out the 6 bytes for the magnetometer.
The read out values are stored in their respective vectors we explained earlier.
With the *heading()* function, we calculate the angular difference between the vector sensor is pointing to, and the north vector, in the horizontal plane, in degrees. To do this, first we find the North vector with the calibration data. Acceleration data is then used to determine the vector pointing upwards.
Cross product of north vector and this vector is the vector pointing east. The vector pointing east and the one pointing north form the horizontal plane. The vector the sensor is pointing to is projected to this plane, and the angle between the projected vector and projected north is calculated and returned.

7.3.3 QT GUI Implementation and Driver Integration

As noted in Chapter 5.3.1, a proof of concept GUI, which shows the capability of Qt, with some functionality exposed to the user, will be implemented.
For this matter, QT framework for embedded systems was used, and a small Qt application was implemented.
We used Qt Creator for this task, which is an IDE tailored to the needs of Qt developers.
For our application, we used QtCore and QtGui modules offered in the framework. All other modules rely on QtCore. It offers the already mentioned concept for the object communication called signals and slots, and QtGui is the GUI tool-kit offering the graphical components for the design [25]. In our project, we have several files:

- DiveComp.pro
- Makefile

- main.cpp
- ms5803.cpp
- ms5803.h
- lsm303.cpp
- lsm303.h
- main_screen.cpp
- main_screen.h
- main_screen.ui
- gas_chooser.cpp
- gas_chooser.h
- gas_chooser.ui

The main.cpp contains the apps' starting point, the *main()* function. Here, we initialize the pressure sensor, and start our GUI. The *ms5803.cpp* and *ms5803.h* contain the pressure and temperature sensor driver. *lsm303d.cpp* and *lsm303.h* contain the code for the eCompass sensor. These sensors use the same bus, so only one initialization of the bus is needed, whose file descriptor is then shared.
The *main_screen** files hold the functionality of the main screen, where we show some basic data, and gas_chooser* files are responsible for the menu where the user can change the gas they are using.
The .ui file is an XML file holding the information about the graphical layout of the elements on the screen.
In the *main_screen.cpp* we used signals for buttons which can be pressed, and implemented functions which will we called when a signal occurs.
After we cross-compiled our application for the target, we need to run it.
A Qt application needs a server application to be running on an embedded device, or it can be a server application itself. We make our application a server application by using the $-qws$ flag when starting the application.
The app is started in background by typing

```
export QWS_MOUSE_PROTO=Tslib:/dev/input/event0
./DiveComp -qws &
```

QWS_MOUSE_PROTO is the environment flag which specifies the driver for pointer handling. In our case, this is the touchscreen. *DiveComp* is the name of the compiled executable, which is executed with the $-qws$ flag, so $DiveComp - qws$. A screenshot of the start screen is visible in Figure 7.2
Here, we see the current dive time, depth, the gas which is used, and the no-dive time dummy values.
On the right side, we see four buttons: settings, gas settings, compass and a log book. When the settings button is pressed, a shell script to toggle WiFi state is executed. This shows how

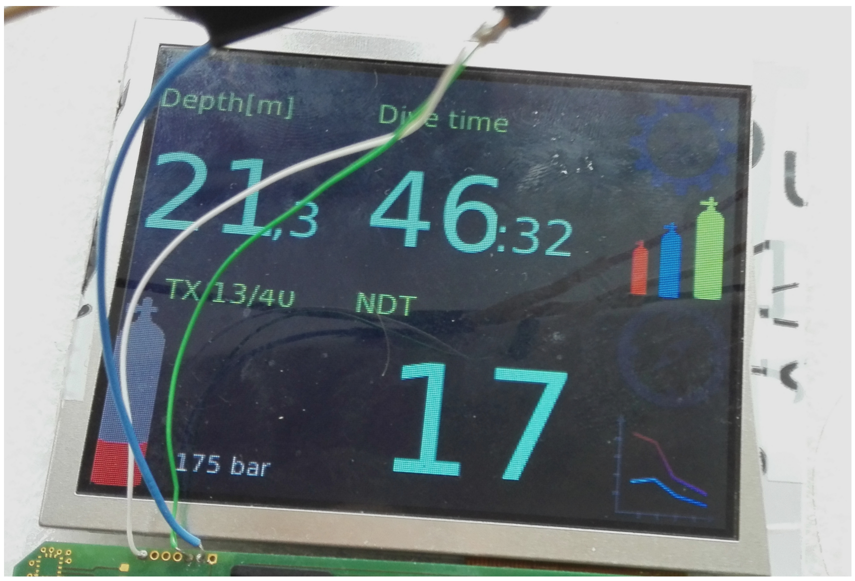


Figure 7.2: Qt application dive mode, a proof of concept

we can interact with Linux command tools and bash scripts.
Besides the settings and gas settings buttons, where the gas can be changed, these are dummy buttons, i.e. they lack functionality.

7.3.4 Services Offered over WiFi

As a proof of concept of what can be done over WiFi, a small software app was implemented during the course of the project. The basic idea was to configure the WiFi module in the Access Point mode, where the user can connect with their PC or smartphone. We implemented a small NodeJS server.
The server offers a static HTML web page, with some JavaScript code embedded in it. The user can, by clicking a button, trigger a bash script which is present on the dive computers filesystem. Several examples include recalibrating the touchscreen, or updating the kernel by uploading a compressed file with the device tree, bootloader and kernel binaries.

7.4 Software Tools Offered by Linux

In this part, we explain some of the drivers found in the kernel, which could be useful for embedded symbols.

7.4.1 Using Power Saving Mechanisms in Linux

Power management is important in embedded systems running on a limited power supply, like a battery.
There are several mechanisms for power management offered by Linux kernel in userspace, which are supported by the AM335x processors. The tools used in our work are *cpufreq*, governors, and sleep states.

7.4.1.1 CPU frequency scaling

In general, CPU frequency scaling enables Linux to increase or decrease the clock speed of the CPU (VDD_CORE), to increase performance, or to save power, respectively. CPU scaling is implemented in the kernel for various architectures, including ARM, and this is referred to as the *cpufreq* framework.
AM335x also supports this framework by stating its operating points in the device tree. In *arch/arm/boot/dts/am33xx.dtsi*, operating points for the processor are defined, and are given in Table 7.7.

kHz	**uV**
720000	1285000
600000	1225000
500000	1125000
275000	1125000

Table 7.7: Operating points for AM335xas defined in the device tree

An essential part of the cpufreq framework are the governors. Governors are so called power schemes for the CPU. They offer some sort of of abstraction, and adjust the values for the clock and voltage [33]. Several governors are offered by the kernel, and they each come with their advantage and disadvantage. For example, the *performance* governor always lets the CPU run at the maximum frequency. On the other side, *powersave* lets the CPU run at the minimum frequency. The default governor is *userspace*, where the CPU frequency is user-defined (but still choosen from a pool of available frequencies).

When we include the *cpufreq* driver in the kernel, *cpufreq* driver is then available in userspace in */sys/devices/system/cpu/cpuX/cpufreq/*.
Viewing all supported governors is done by executing

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Setting the governor is not complicated:

```
echo $governor > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

If we set the *userspace* governor, we can set any of the frequencies offered by the CPU. To see available frequencies, we execute

```
cat /sys/devices/system/cpu/cpu0/cpufreq/
                        scaling_available_frequencies
```

Current CPU frequency can be shown with

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

Setting the minimum or maximum CPU frequency is done with

```
echo 275000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
echo 720000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

Processor voltage is adjusted accordingly.
To manually set the VDD_CORE voltage to, e.g. 0.95V or 1.1V, we may use

```
i2cset -f 0 0x2D 0x25 0x1F
i2cset -f 0 0x2D 0x25 0x2B
```

0x2D is the address of the PMIC module, and this command writes the wanted voltage in the register at addres 0x25, the VDD2_OP_REG.

We use the *powersave* governor.
Unfortunately, we cannot go below the minimum voltage defined in the device tree, because the several submodules are not supported when in a lower voltage range, since we are then in the OPP50 state. Several peripherals and features are not supported when operating in OPP50. Among these are for us relevant DDR3 and LCDC in LIDD mode [1]. For this reason, we must remain in the OPP100 voltage range.

7.4.2 Suspend to RAM

Another important concept are the sleep states. AM335x, just like many other devices, support several sleep modes. This way, the system can transition to a lower power state, if some functionalities are not needed.
We use the *Suspend to RAM* state, where most parts of the CPU are not powered, and RAM is kept in the self refresh state, where its content is preserved. This makes a fast restoration of machine state possible, since userspace and kernel do not have to be loaded again from the SD card to the working memory.
We enter the *Suspend to RAM* state with

```
echo mem > /sys/power/state
```

Waking up the CPU can be done by GPIO0 and touchscreen interfaces.
To enable waking up from touchscreen touch, we must execute

```
echo enabled > /sys/devices/platform/omap/ti_tscadc/power/wakeup
```

before going to sleep.
To wake up by the RTC, we use the *rtcwake* utility. Here, we set the time when we want to wake up the CPU, and the state we want to transition into, i.e. the state we want to be awaken from.
We do this with

```
rtcwake -d /dev/rtc0 -m mem -s 20
```

In this example, we will wake up in 20 seconds from the *SuspendtoRAM* state.

Chapter 8

Future Work

In this section, we will give some information on how we plan to proceed with this project, after the academic part has been finished.
First of, a dive computer requires an user interface which offers more then just sensor values, if we want this product to be used commercial. For this matter, we present a small concept GUI, where we show our perspective on how such a GUI might look like, and what it should offer.

8.1 Full Featured Dive Computer Software

In order to keep the project in a manageable size, a full featured GUI was not required by the contracting authority, and was thus not implemented.
Such a GUI would include more features, like an appealing and user friendly interface, customizability for the information shown, and so on.

8.1.1 GUI Concept

We will now merely present a concept of this GUI. The reason for the large size of the project is not the GUI, but the algorithms which would have to be developed for this functionality to be achievable.
As for the GUI concept, there would be an area in the top of the display, where the meta information about the battery, WiFI status and no fly time will be shown. This area would be present all the time.
For the rest of the screen, the GUI would have two different states: the surface mode, and the dive mode.
In the surface mode, when the watch is not in water (e.g. environment pressure below 1.1 bar), information such as time and date, compass, gas type and temperature would be shown. Figure 8.1 gives a graphical representation of the surface mode concept.

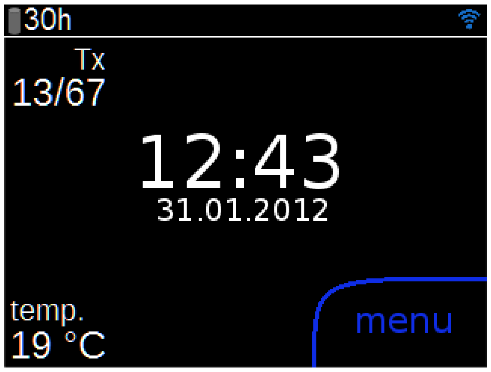


Figure 8.1: Surface mode concept

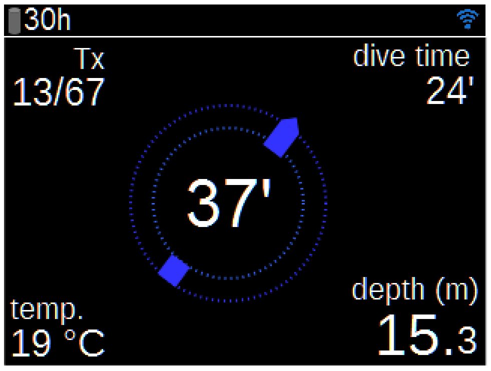


Figure 8.2: Dive mode concept

In the second mode, called the dive mode, a diving related information would be shown. This information consists, among other things, of the gas mixture type, depth, temperature, dive time, compass, time until decompression is needed, and so on, depending on the users preferences. Figure 8.2 shows the dive mode concept.
As one can see, the two states share a subset of information, so they would be shown at all times, and their position on the screen would be fixed, to lessen the confusion of the user.
We also introduce the concept of alarms. For example, if the diver is ascending too fast, or their tank pressure is lower than a predefined limit, a visual alarm might be shown. This is shown in Figure 8.3.

8.1.2 Desirable Dive Computer Features

Some features we would like to see implemented for the dive computer are:

- Alarms (depth, pressure, tank pressure, heart rate, hypothermia danger, ...)
- Time until flight allowed
- Programmable safety stops

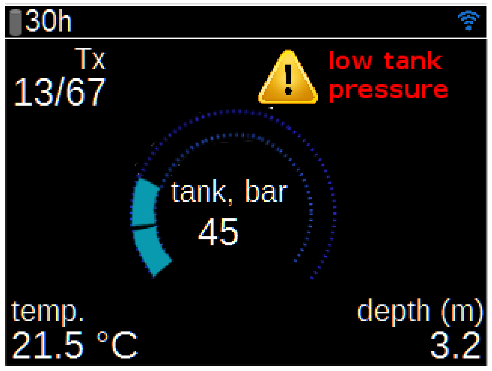


Figure 8.3: Dive mode concept with a triggered alarm

- Deco algorithms with various gas mixtures
- Dive simulator (user creates a dive profile and simulates it)
- Dive log viewer
- Following a compass direction ...

8.2 Hardware Changes

Besides the implementation of the software features, some changes in the hardware design would also be desired. For example, instead of using a DDR3 chip for our main memory, we would use slower and less power-hungry DDR2 chip. The initial reason we decided to use DDR3 was the lack of experience with interfacing high speed devices.

Chapter 9

Conclusion

Here, we will tie together the thesis statement, issues we encountered during this work, and we will draw some conclusions about this project, and what we have learned from it.

During this project, we build a hardware and software platform which can be used for a dive computer.

We based our hardware design on an evaluation board by Texas Instruments, which was then adapted to our needs, in terms of removing unneeded hardware, and adding our own. Me made a platform which supports a large LCD, with a touchscreen which is capable of working under water. Several sensors were also incorporated, which can be used by the diving software to calculate important parameters for the divers. Also, accessing the data like the logbook would be easy, via WiFi.

Besides the hardware aspect, we also had the software aspect to take care of. For this matter, we incorporated a rich OS in our project, which then served as the abstraction layer to the software modules we build, like the sensor drivers and the GUI application. These software modules served as a proof of concept for what can be implemented on a hardware platform we built.

This research gave us a good insight on how we can modernize the dive computers, with the help of other supporting technologies including semiconductor, sensor, display and battery technology.

The integral stages of this project were the layouting of the PCB, configuring and implementing the software, and assembling the PCB. All of these stages require a highly skilled personnel, if a guarantee is needed that a project will succeed.

Several issues were encountered during the project, e.g. issues with the components not being soldered properly, or misbehaving software. Several components did not work as expected, like the LCD. The problem is either in the layout, soldering, or in the configuration. Many dimensions of the problem, and not being able to pinpoint it in a short period is a good example which shows how difficult embedded systems development is.

We have learned a lot from this project, starting from the basic concepts concerning hardware and software, but also a general overview of how a project of this type needs to be managed. We also gained a good insight in many topics concerning embedded systems, communication protocols and designing a PCB. Besides this, we realized how powerful Linux is for embedded systems, and what potential it has to offer in the embedded world.

Appendix A

Abbreviations

GUI	Graphical user interface
MPU	Microprocessor unit
PCB	Printed circuit board
PMIC	power management integrated circuit
JTAG	Joint Test Action Group
I^2C	Inter-integrated circuit interface
EMI	Electromagnetic interference
EMC	Electromagnetic compatibility
LCD	Liquid crystal display
SPI	serial peripheral interface
RTC	Realtime clock
IC	Integrated circuit
RTCSS	Realtime clock subsystem
PLL	Peripheral Phase Locked Loop
ECC	Error code correction
SOC	system on chip
EMIF	External memory interface
CS	Chip select
CAS	Column access strobe
ADC	Analogue-to-digital converter
OS	Operating system
BSP	Board support package
IPL	Initial program loader
MBO	Minimal bootloader
DMA	Direct memory access
RTOS	Real-time operating system
PS	Protocol select
RS	Radio frequency
OPP	Operating performance points
DQS	Differential data strobe
UART	Universal asynchronous receiver/transmitter

Bibliography

[1] Texas Instruments. AM335x Sitara Processors Datasheet. http://www.ti.com/lit/ds/sprs717h/sprs717h.pdf, 2015. [Online; accessed 16-Dec-2015].

[2] ST Microelectronics. Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer. http://www.st.com/web/en/resource/technical/document/datasheet/DM00057547.pdf, 2015. [Online; accessed 16-Dec-2015].

[3] Micron. 2Gb: x4, x8, x16 DDR3 SDRAM Datasheet. https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/2gb_ddr3_sdram.pdf, 2015. [Online; accessed 16-Dec-2015].

[4] Measurement Specialities. MS5803-01BA Miniature Variometer Module. http://www.meas-spec.com/downloads/MS5803-01BA.pdf, 2015. [Online; accessed 16-Dec-2015].

[5] MA Lang and RW Hamilton Jr. Proceedings of the AAUS dive computer workshop. page 231, 1989.

[6] Texas Instruments. AM335x Overview. https://www.ti.com/lsds/ti/processors/sitara/arm_cortex-a8/am335x/overview.page?paramCriteria=no, 2015. [Online; accessed 16-Dec-2015].

[7] Karim Yaghmour, Jonathan Masters, and Gilad Ben. *Building Embedded Linux Systems, 2Nd Edition.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, second edition, 2008.

[8] Doug Abbott. *Linux for Embedded and Real-Time Applications.* Butterworth-Heinemann, Newton, MA, USA, 2003.

[9] P. Horowitz and W. Hill. *The Art of Electronics.* Cambridge University Press, New York, NY, USA, 1989.

[10] Texas Instruments. PCB Design Guidelines For Reduced EMI. http://www.ti.com/lit/an/szza009/szza009.pdf, 1999. [Online; accessed 16-Dec-2015].

[11] Powell Mark. *Deco for Divers: A Diver's Guide to Decompression Theory and Physiology.* AquaPress, 2014.

[12] Suunto. SUUNTO EON STEEL 1.1 User Guide. http://ns.suunto.com/Manuals/EONSteel/Userguides/Suunto_EONSteel_UserGuide_EN.pdf, 2015. [Online; accessed 16-Dec-2015].

[13] Seabear GmbH. H3 User Manual. `http://www.seabear-diving.com/downloads/manuals/H3/index.html?cover.htm`, 2015. [Online; accessed 16-Dec-2015].

[14] Texas Instruments. ARM Processors Selection Guide. `http://www.ti.com/lit/sg/sprt596f/sprt596f.pdf`, 2015. [Online; accessed 16-Dec-2015].

[15] Electronic-Assembly. P320X-35ALWS Datasheet. `http://www.lcd-module.de/eng/pdf/grafik/p320x-35a.pdf`, 2015. [Online; accessed 16-Dec-2015].

[16] Geoff Walker. A review of technologies for sensing contact location on the surface of a display. *Journal of the Society for Information Display*, 20(8):413–440, 2012.

[17] Acme Systems. WIFI-2 - OEM WiFi USB module. `http://www.acmesystems.it/WIFI-2`, 2015. [Online; accessed 16-Dec-2015].

[18] Johnson Randy and Intel Christie Stewart. JTAG 101, IEEE 1149.x and Software Debug. `https://www-ssl.intel.com/content/dam/www/public/us/en/documents/white-papers/jtag-101-ieee-1149x-paper.pdf`, 2015. [Online; accessed 16-Dec-2015].

[19] Petazzoni Thomas. Your new ARM SoC Linux support check-list. `http://elinux.org/images/a/ad/Arm-soc-checklist.pdf`, 2015. [Online; accessed 16-Dec-2015].

[20] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[21] Peter Jay Salzman. *The Linux Kernel Module Programming Guide*. CreateSpace, Paramount, CA, 2009.

[22] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.

[23] Buell Alex. Framebuffer HOWTO. `www.tldp.org/HOWTO/Framebuffer-HOWTO/`, 2010. [Online; accessed 16-Dec-2015].

[24] Atmel. AVR32416: AVR32 AP7 Linux LCD Panel Customization. `http://www.atmel.com/images/doc32105.pdf`, 2008. [Online; accessed 16-Dec-2015].

[25] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 3*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[26] Qt Foundation. Qt Documentation. `http://doc.qt.io/qt-4.8`, 2008. [Online; accessed 16-Dec-2015].

[27] Freescale Semiconductor. Designing for Board Level Electromagnetic Compatibility. `https://cache.freescale.com/files/microcontrollers/doc/app_note/AN2321.pdf?pspll=1`, 2008. [Online; accessed 16-Dec-2015].

[28] Soeser Peter Stoeckler Gerhard Winkler Gunter Deutschmann Bernd, Eichberger Bernd. Lecture Notes on Development of Electronic Systems. `https://online.tugraz.at/tug_online/lv.detail?clvnr=178781`, 2008. [Online; accessed 16-Dec-2015].

[29] Texas Instruments. TPS65910x Integrated Power-Management Unit Top Specification. `http://www.ti.com/lit/ds/symlink/tps65910.pdf`, 2008. [Online; accessed 16-Dec-2015].

[30] Downs Rick Osgood Skip, Ong CK. Touch Screen Controller Tips. `http://www.ti.com/lit/an/sbaa036/sbaa036.pdf`, 2000. [Online; accessed 16-Dec-2015].

[31] Buildroot Documentation. The Buildroot user manual. `http://buildroot.uclibc.org/downloads/manual/manual.html`, 2015. [Online; accessed 16-Dec-2015].

[32] Debian Wiki. rt2800usb. `https://wiki.debian.org/rt2800usb`, 2008. [Online; accessed 16-Dec-2015].

[33] Brodowski Dominik. Linux CPUFreq. `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`, 2008. [Online; accessed 16-Dec-2015].