Report for ARM Microcontroller Project

uLoader WiSe 2014

Appel, Dennis (s813783)
Voigt, Alexander (s814526)
Merrikhi, Pedram (s827711)

April 11, 2015

Contents

Glossary

AHB AHB, please refer to AMBA. 1

AHB1 Advanced High-performance Bus is part of the Advanced Microcontroller Bus Architecture (AMBA) of the IP-manufacturer ARM Limited (ARM).. 1

AMBA Advanced Microcontroller Bus Architecture is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. It facilitates development of multi-processor designs with large numbers of controllers and peripherals.. 1

APB The Advanced Peripheral Bus (APB) is an internal bus for System-on-Chips (SoC) to connect low power peripheral devices. The APB bus is part of the AMBA-achitecture which is designed for low power and simple interface. It can be used with the standardized buses like AHB.. 1

CAN Controller Area Network ist ein serrieller BUS der asynchron arbeitet. 1 Mbit/s ist hierbei die Maximale Datenrate. Wird meist in Fahrzeugen eingesetzt.. 1

CMSIS Microcontroller Software Interface Standard, is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces.. 1

GPIO General-purpose input/output is a generic pin on an integrated circuit whose behavior, including whether it is an input or output pin, can be controlled by the user at run time. 1

JTAG Joint Test Action Group was formed in 1985 to develop a method of testing finished printed circuit boards after manufacture. In 1990, the effort was codified as a standard by the Institute of Electrical and Electronics Engineers with the designation IEEE Std. 1149.1-1990 entitled Standard Test Access Port and Boundary-Scan Architecture.. 1

LVM Logical Volume Manage provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes. In particular, a volume manager can concatenate, stripe together or otherwise combine partitions (or block devices in general) into larger virtual ones that administrators can re-size or move, potentially without interrupting system use.. 1

MCU Microcontroller Unit is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of Ferroelectric RAM, NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM.. 1

NVIC Nested Vectored Interrupt Controller,facilitates low-latency exception and interrupt handling, controls power management, implements System Control Registers. The NVIC supports up to 240 dynamically reprioritizable interrupts each with up to 256 levels of priority.. 1

ROM Read Only Memory is a class of storage medium used in computers and other electronic devices.. 1

SWD Serial Wire Debug is an alternative 2-pin electrical interface that uses the same protocol. It uses the existing GND connection. SWD uses an ARM CPU standard bi-directional wire protocol, defined in the ARM Debug Interface v5.. 1

UART universal asynchronous receiver/transmitter is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485. The universal designation indicates that the data format and transmission speeds are configurable.. 1

USART universal synchronous/asynchronous receiver/transmitter, modern ICs now come with a UART that can also communicate synchronously. 1

USB Universal Serial Bus is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication, and power supply between computers and electronic devices.. 1

List of Figures

List of Tables

Listings

1. Introduction

Nowadays, an ARM-MCU could be used in every aspect of everyday life. Additionally, the ARM processor is the number one architecture of choice in many market segments cause it aims to have the best performance to power consumption ratio..

This project is based on the development of a bootloader and its implementation inside a network. The usage stm32f4-discovery Board is a preferred and viewed as an "'Allrounder"' for such a project. The reasoning behind this is the "value for money" and user-friendliness. This allows for an easy introduction into the world of ARM Microcontroller unit programming.[STMicroelectronics, 2015]

The ARM-Cortex-M4-Processor found on the STM32f4-discovery board possesses the principle parts shown in the figure below.

The aim of the project is to research the feasibility to create a quick, cheap and easy to use way of utilizing an STM32 Microcontroller to communicate between a user and a remote device.[ARM Ltd, 2014]

The purpose of the application is meant to be a first step fundamental strategy to creating a product for future projects. It is hoped that by utilizing a boot loader, a fast and light application could be used to fulfill the desire of a user to achieve a particular objective such as, three way handshake signal to verify a particular device in order to transmit information such as codes or messages, using a TCP/IP protocol stake.

As it can be demonstrated the different applications could be endless.

Analogously, a new and different diagram would be used, in order to illustrate the change in the usage-concept of the processors.

But first, an overview of the discussed focal points would be outlined.

1.1. Bootloader

The purpose of a Bootloader program is to allow the installation and utilisation of any program that could be reloaded. Whereas the program that is currently loaded is also being run.

Next it is necessary to initialise the hardware, that would in turn be needed to load the program.

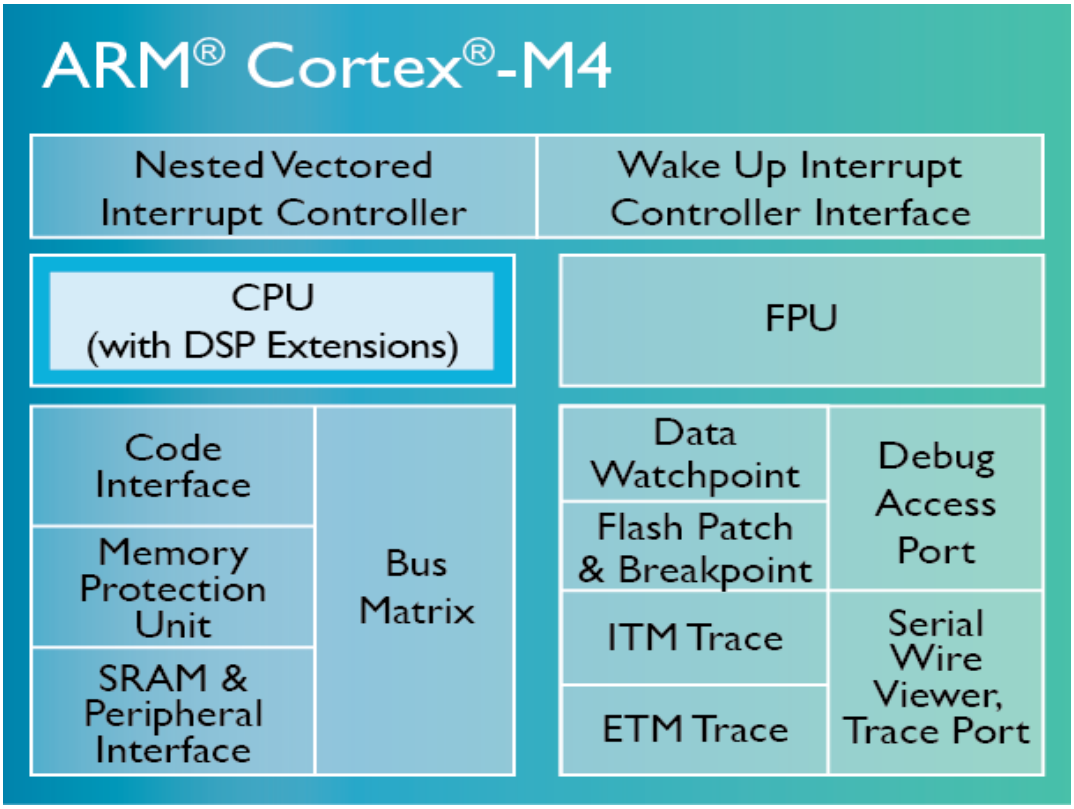


Figure 1.1.: Standard Layout of ARM Cortex-M4 MCU

The "STM32f407 discovery board" offers three different methods to boot up the hardware.[Robert Boys, 2012]

In order to switch between the three different boot methods, the Boot-Pins BOOT1 and BOOT2 could be set:

BOOT1	BOOT2	Boot-Mode	Addresse
x	0	Flash Memory (User Flash)	0x8000_0000
0	1	System Memory	0x1FFF_F000
1	1	SRAM	0x2000_0000

Table 1.1.: Boot-Pin Function

The ROM memory is included by the manufacturer along with the bootloader.

It is very important to set the correct address of the program, that is located in the memory, in order for the reloading of the program to work.

A step by step example of the the sequence after the boot loader is already loaded ,is as follows:

1. Hardware initialize (USB / USART / RCC ...)
2. Wait for running program (pending other tasks)
3. Write a program to address XY.
4. Point to address XY

After these steps, the newly setup program is then responsible for the initialization of the hardware.

1.2. SWD

During the development of the Boot Loader, a serial wire debug technology was used. The reason behind this is to utilize a Debug-Port that has been specially developed to cater to a MCU that makes allows the use of the least amount of pins possible.

This port consists of pins shown in the following:

Pin	Signal	Type	Description
1	VTref	Input	This is the target reference voltage. It is used to check if the target has power, to create the logic-level reference for the input comparators and to control the output logic levels to the target. It is normally fed from Vdd of the target board and must not have a series resistor.
7	SWDIO	I/O	Single bi-directional data pin
9	SWCLK	Output	Clock signal to target CPU. It is recommended that this pin is pulled to a defined state of the target board. Typically connected to TCK of target CPU.
13	SWO	Output	Serial Wire Output trace port. (Optional, not required for SWD communication)
15	RESET	I/O	Target CPU reset signal. Typically connected to the RESET pin of the target CPU, which is typically called "nRST", "nRESET" or "RESET".
19	5V-Supply	Output	This pin is used to supply power to some eval boards. Not all JLinks supply power on this pin, only the KS (Kickstart) versions. Typically left open on target hardware.

Table 1.2.: SWD PINOUT

The other pins of the 20-pole connection are going to left out, meaning, the other pins are useless for the SWD or they will be used as a GND. Regardless of the pin allocation, it is important that the communication of the SWD would not be interrupted or effected.

This technique represents a new and more effective way to debuggen. Until now JTAG represented the Debugger-Interface.

The advantages of this technology are:

- Only 2 Pins are used
- JTAG TAP controller compatible
- Allows the Debugger to become an extra AMBA-Bus-Master, in order to accomidate an extra access capability to the Register or Memory
- High Datarates - 4Mbytes/sec @50MHz
- Low Power - no extra power supply
- Error recognition "'built in"' that performs well
- Protection against errors that cause disconnection

1.3. CMSIS

The ARM Cortex Microcontroller Software Interface Standard is a manufacturer independent abstraction layer for the Cortex-M processors.

Thereby the CMSIS is subdivided into:

- CMSIS-CORE - API access to the processor kernel and peripheral register.
- CMSIS-Driver - Generic access on peripheral devices for Middleware (reusability).
- CMSIS-DSP - DSP Liberary with over 60 functions
- CMSIS-RTOS API - Standardised (RTOS compatible)
- CMSIS-Pack - Description of the most important components (User view)
- CMSIS-SVD - Description of the most important components (System view)
- CMSIS-DAP - Debug Access Port

Summarized, CMSIS allows a consistent and simple software interfaced to the processor and peripheral devices, as well as Real-time OS (RTOS) and Middleware.

1.4. Nested Vectored Interrupt Controller

The NVIC offers the possibility to configure special interrupts (Priority, Activate, Deactivate...).

Aside from the given interrupts, there are also the configurable implementation dependent interrupts. Because, the first 15 interrupts are allocated, the number of implimented interrupts could be from 0-240.

1.5. Differences in Development

As we work in teams and sometime on the same file at the same time, a distributed version control system was used to manage all the work to stay in sync.

The System of choice was Git and therefore the public Hostplatform Github.

Repository Link

There you can track down who did what, when (blame) and get updates from one central point when they happen. Also included is a readme where is described how to install needed tool to compile the source code. In addition the wiki should been read too. There is a collection of all used resources from the Web and Books.

The presentation has been kept via prezi.com so the url for that is:

ARM CORTEX-M4 uLoader

The whole project was developed with Vim, make and gcc. The advantage of that is, that it is not necessary to *learn* how to use an IDE. There are a lot of IDE's which offer programming ARM. Theoretically one could learn to work with a new IDE every project. Vim on the other hand is every time the same, GCC and make also. The only thing one has to do is to write the makefile which can be very time consuming. The disadvantage is the missing luxury. Nearly every IDE offers things like an address editor or a nice graphical way to debug. But as we are professional engineers, we don't need such tools that do all the work for you in the background so that you don't need to know what you are actually doing.

What is a linker script, how does make / building work, you will probably never learn that if you utilize ready to use IDEs all day. And the most important fact, KEIL and iAR cost so much money with low benefit, why should anyone actually use them? Cause user are to lazy to learn what it means to be a real engineer!

This project is a Bare Metal Project not a click two buttons to be ready with your work project. All the three programs have a steep learning curve, are Open Source, and have the advantage of letting you do what ever you want you just need to know what you are doing.

An Engineer should get payed for their knowledge not for the tools they are using (IMHO).

1.6. Network

The stm32f407 discovery board itself has no possibility for any direct network connection. At least no connector and transceiver but it hat the Ethernet feature already in it. Thus the BaseBoard (BB) is needed (and used). The BB adds new capabilities like ethernet or rs232. There are other PCB's just for the Network feature available but most of them have range issues.

2. ARM M4

The ARM M4 MCU is the heart of the stm32f407. ARM offers the IP of the CORE-M4 to manufacturers and they can customize the IP as they see fit. A very simple diagram of the used MCU is the following:

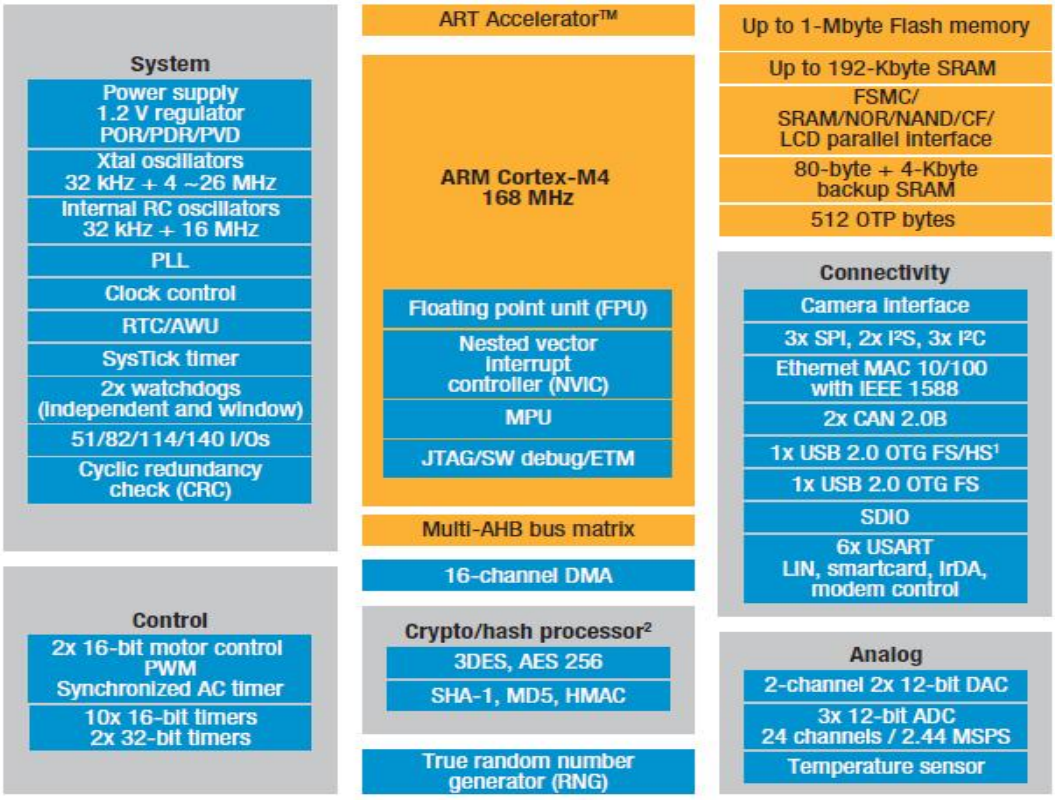


Figure 2.1.: Simple M4 Diagram

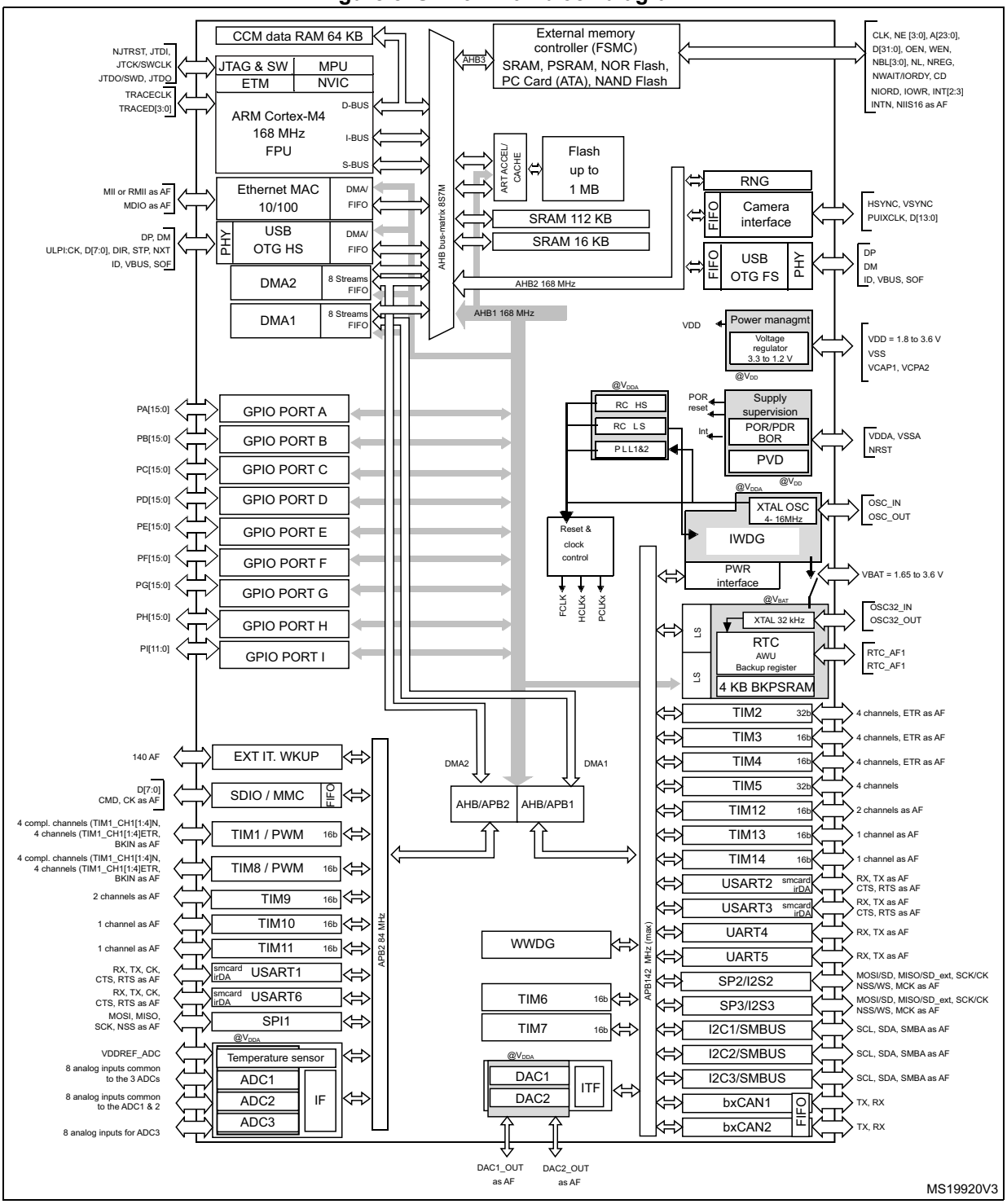
The diagram doesn't show, which peripheral systems are connected to which APBX-Bus, but that can be seen in the datasheet. What it does show is, that the GPIO-Pins, are connected to the AHB1-Bus.
The reason for that is, that it's much faster then what is recommended for most the part of what a GPIO demands. The subsystems on the other hand aren't necessarily needed to react that fast.

For example the USART6, which is used in the project, is connected to APB2. A much more precise representation of the STM32F4 is illustrated in the following diagram.

The AHB1 is the system bus and therefore should be really fast.

2.2 Device overview

Figure 5. STM32F40x block diagram



1. The timers connected to APB2 are clocked from TIMxCLK up to 168 MHz, while the timers connected to APB1 are clocked from TIMxCLK either up to 84 MHz or 168 MHz, depending on TIMPRE bit configuration in the RCC_DCKCFGR register.
2. The camera interface and ethernet are available only on STM32F407xx devices.

The peripheral systems are addressed through the busses. For example the GPIOA and USART6. The address range of the peripherals is 0x4000_0000 - 0x5FFF_FFFF. The peripheral base address is 0x4000_0000. To configure and write to / read from the USART6 we need the peripheral base address and an offset. In the case of USART6 the offset is a combination of two offsets, as seen in the table:

Table 10. STM32F40x register boundary addresses (continued)

Bus	Boundary address	Peripheral
APB2	0x4001 4C00 - 0x4001 57FF	Reserved
	0x4001 4800 - 0x4001 4BFF	TIM11
	0x4001 4400 - 0x4001 47FF	TIM10
	0x4001 4000 - 0x4001 43FF	TIM9
	0x4001 3C00 - 0x4001 3FFF	EXTI
	0x4001 3800 - 0x4001 3BFF	SYSCFG
	0x4001 3400 - 0x4001 37FF	Reserved
	0x4001 3000 - 0x4001 33FF	SPI1
	0x4001 2C00 - 0x4001 2FFF	SDIO
	0x4001 2400 - 0x4001 2BFF	Reserved
	0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3
	0x4001 1800 - 0x4001 1FFF	Reserved
	0x4001 1400 - 0x4001 17FF	USART6
	0x4001 1000 - 0x4001 13FF	USART1
	0x4001 0800 - 0x4001 0FFF	Reserved
	0x4001 0400 - 0x4001 07FF	TIM8
	0x4001 0000 - 0x4001 03FF	TIM1
	0x4000 7800- 0x4000 FFFF	Reserved

1. Peripheral Base: 0x40000000
2. APB2: Peripheral Base + 0x10000 = 0x40010000
3. USART6_Base: APB2 + 0x1400 = 0x40011400

At the USART6_Base is equal to the USART Status register. To transceive data via USART6 the DR register (16-bit) with an additional offset of 0x04 is right one.

It is one register to read to and write from. That is a good example, why an interrupt handler is needed (NVIC). The program would have a huge timing problem since the read operation on that register would cause a huge delay, because it would wait till data is read.

On this way all the peripherals are can be addressed. For the GPIOA it is theoretically the same but with other addresses. The GPIOA is connected via the faster AHB1 bus.

1. AHB1_Base: Peripheral Base + 0x20000
2. GPIOA_Base: AHB1_Base

These are just two examples of the addressing of the ARM peripherals, but at the bottom they are all the same - it's all about the addresses.

3. NVIC in use

As already mentioned, the NVIC is used to configure the interrupts. Since there is a bi-directional communication via USART6, the NVIC has to be configured to fit that demand.
Therefore the first thing to do is to enable the clock on APB2 to clock the USART6. Additionally configuration has to be done for the USARTX, but it is not part of the NVIC configuration.
The clock is mentioned because it plays an essential part of the configuration. It is not important which U(S)ART[1..6] pin is used. Naturally, the right APBX has to be configured (precise diagram of stm32f4).

The configuration is done by writing the parameters into a structure and binding them and than load it into the NVIC. The last thing to do is to enable USART globally.

```
u->Initialized = 1;

/* Disable if not already */
USARTx->CR1 &= ~USART_CR1_UE;

/* Init */
USART_Init(USARTx, &USART_InitStruct);

/* Enable RX interrupt */
USARTx->CR1 |= USART_CR1_RXNEIE;

/* Fill NVIC settings */
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = TM_USART_NVIC_PRIORITY;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = TM_USART_INT_GetSubPriority(
    USARTx);
NVIC_Init(&NVIC_InitStruct);

/* Enable USART peripheral */
USARTx->CR1 |= USART_CR1_UE;
```

Listing 3.1: enable usart

The code is part of a function therefor USARTx is a parameter, in this case it is USART6.

The process and as well as the code that was used, generated the desired results which proves the method that was chosen works.

4. Network

The Network on the Baseboard is realized through a SMSC LAN8720.

4-bit data nibbles are sent to the MII block. These data nibbles are clocked to the controller at a rate of 25MHz. The controller samples the data on the rising edge of RXCLK. To ensure that the setup and hold requirements are met, the nibbles are clocked out of the transceiver on the falling edge of RXCLK. RXCLK is the 25MHz output clock for the MII bus. It is recovered from the received data to clock the RXD bus. If there is no received signal, it is derived from system reference clock(XTAL1/CLKIN).

When tracking the received data, RXCLK has a miximum jitter of 0.8ns (provided that the jitter of the input clock, XTAL1/CLKIN, is below 100ps).

In RMII mode, the 2-bit data nibbles are sent to the RMII block. These data nibbles are clocked to the controller at a rate of 50MHz. The controller samples the data on the rising edge of XTAL1/CLKIN (REF_CLK). To ensure that the setup and hold requirements are met, the nibbles are clocked out of the transceiver on the falling edge of XTAL1/CLKIN (REF_CLK).

The blockdiagram 4.1 shows the hardware for the RMII for a network connection.

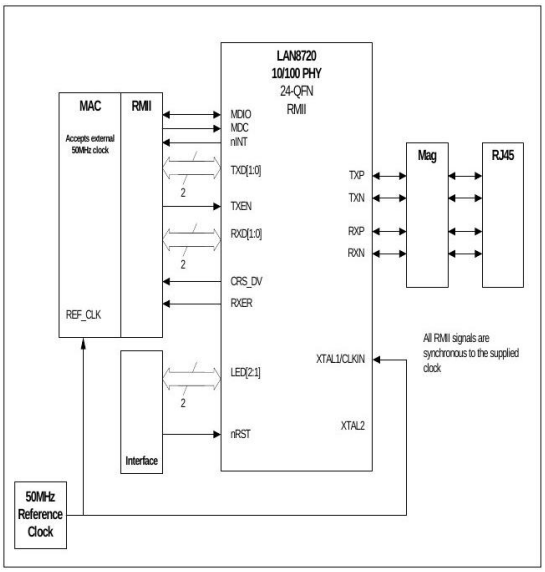


Figure 4.1.: Blockdiagram for PHY Connection

5. CMSIS

CMSIS is a huge advantage in code portability. The idea behind that standardized interface is to make the programmer independent of the hardware. Thus CMSIS is a hardware abstraction layer.

The programmer develops code to access the CMSIS driver library instead of the hardware. This way the code becomes portable for all CORTEX MCU's. The following diagram gives an overview about CMSIS:[ARM Ltd, 2015]

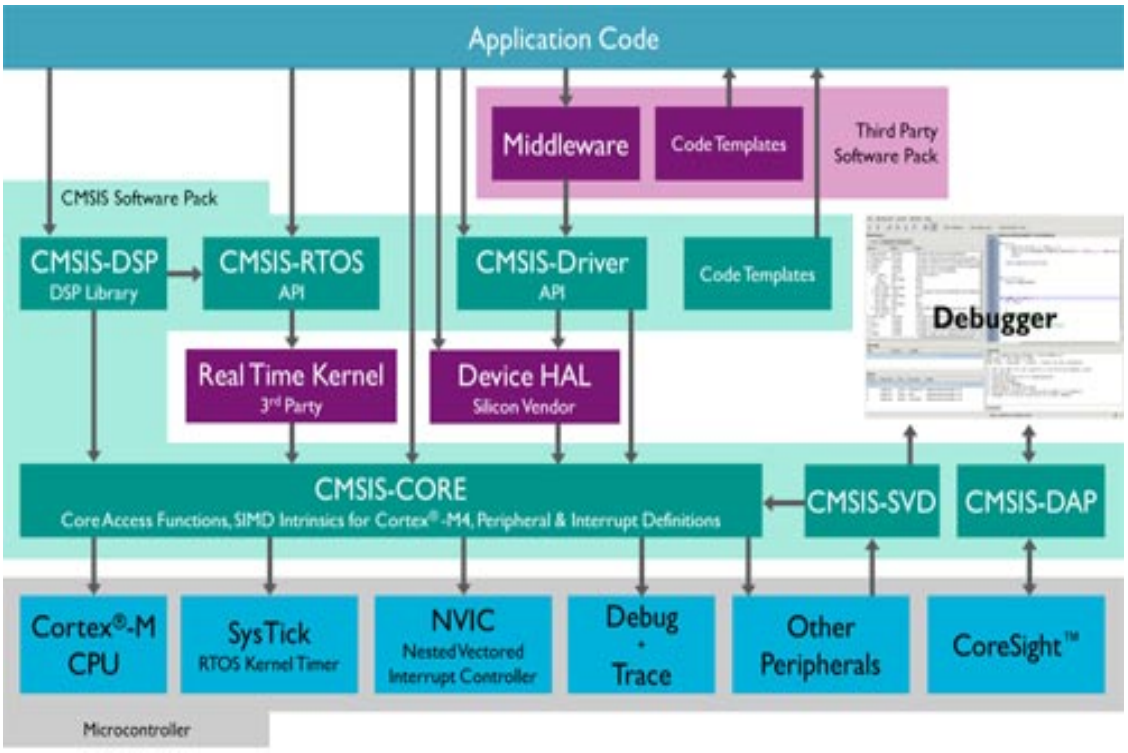


Figure 5.1.: Cortex Microcontroller Software Standard Interface

6. LWIP

LwIP is open source TCP/IP protocol suite designed for embedded systems.

The focus of the LwIP TCP/IP implementation is to reduce RAM usage while still having a full scale TCP. This makes LwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

LwIP includes the following protocols and features:[Swedish Institute of Computer Science, 2015]

- IP (Internet Protocol) including packet forwarding over multiple network interfaces
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- IGMP (Internet Group Management Protocol) for multicast traffic management
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit
- Raw/native API for enhanced performance
- Optional Berkeley-like socket API
- DNS (Domain names resolver)
- SNMP (Simple Network Management Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- AUTOIP (for IPv4, conform with RFC 3927)
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol) for Ethernet

7. Implementation

As mentioned earlier several makefiles are completely written from scratch. There is a separate makefile for the Network Stack (lwip). Also there is a config file (.config) that makes it possible to just compile in what is needed. (And not everything and let the linker find what is needed, as used in IDE's.)

To save time the CMSIS libs and standard peripheral drivers supplied by ST have been used. In addition the well written library from Tilen Majerle [Tilen Majerle , 2015] have been used too.

To be more precisely the UART and therefor GPIO parts of them. The advantage of them is that is now possible to make the input nonblocking, so that an event is only fired when the buffer is either full or has a specific item in it. (this corresponds to the canonical mode of *termios* programming under Linux)

This behavior is important due the fact that the network stack needs occasionally time to process pending work.

This corresponds well to the implementation idea of the lwip stack to be asynchron. Lwip achieves that via callbacks. That are registered functions that just get called when the specified event has been triggered. Another advantage of that is that there is not the need to use an interrupt for every possible event.

Together those feature's make it possible to keep the Human Interface responsible while processing pending work in the background.

The Human Interface makes it possible to interact with the User. It can be compared to a Unix shell environment. Corresponding to that it has some build in functions that are similar to some known from Unix.

A prompt showing the name of the Program awaits input. This can be some calls like *env* or *printenv* that will show the actual value the global variables have at the time of the call. Some of them are the systemclock, IP and how many bytes the last file via TFTP was.

The call *tftp* will issue an tftp read request to a hardcoded server with a hardcoded file (blinky.bin).

But this will only work when there is a network connection and the device has acquired an IP address. Appropriate to that the commands *static* and *dhcp* are available. At system boot there are some attempts to get an IP from a DHCP server. If that fails

for some time an static IP will be set. To try the DHCP process again of switch from dynamic to static IP allocation these two command can be used at runtime.

The Trivial File Transfer Protocol [Uni Oldenburg, 2015] was completely implemented by us, via the RAW API from the lwip Stack. As we don't have an multi threaded system the other two API's can not be used. The netcon API needs an OS like Free RTOS and the Berkley socket API, known from UNIX Systems, is build on top of that.

7.1. Problems

As there where other classes we need to do work for, the time was way to short to implement all needed device drivers on our own, the help of the existing examples from *st* [element14, 2015] was therefore needed and welcome.

But the problem that has been arisen from the fact that they just use IDE's specially build and pre configured for those examples (project file), was so big and therefor time consuming, that we could not manage to implement the main feature of a bootloader, to load and start a downloaded program.

There is a command implemented to read the contents of the flash (*read*). It accepts an argument that stands for the number of bytes to read. The output looks like the canonical ascii format known from *hexdump*.

But a write command is still missing. Dues the fact that we could not find out in time how we need to prepare the downloaded binary to make it start from the address we we would put it. Guess it has to do something with the linker or startup script.

The most time consuming factor was to find out how to build the code and adjust the lwip stack to work with the System. As the lwip wiki is kind of useless and not much documentation is available for the now 3 year old last version of the stack. To find out what to do, we were in need to reverse engineer the existing examples.

7.2. Test

The command:

```
sudo picocom -b 38400 /dev/ttyUSB0 --omap crlf --echo
```

was used, to get an interface via the uart6 of the Base Board and an ftdi adapter connected to the PC. To verify that DHCP and TFTP work tcpdump was used.

Bibliography

[ARM Ltd 2014] ARM Ltd: *Cortex-M4 Processor*. http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php, 2014. – [Online, accessed 10-April-2015] 1

[ARM Ltd 2015] ARM Ltd: *CMSIS - Cortex Microcontroller Software Interface Standard*. http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php?tab=Videos, 2015. – [Online, accessed 10-April-2015] 5

[element14 2015] element14: *STM32F4DIS-BB Discover More Software Examples*. http://www.element14.com/community/docs/DOC-51670/l/stm32f4dis-bb-discover-more-software-examples, April 2015. – online, 11.04.2015 15.34 Uhr 7.1

[Robert Boys 2012] Robert Boys: *STMicroelectronics: Cortex™-M4 Training STM32F407*. 1.1. http://aces.shu.ac.uk/staff/engafh/STM32/Discovery_M4_LAB.pdf, July 2012 1.1

[STMicroelectronics 2015] STMicroelectronics: *Discovery kit for STM32F407/417 lines - with STM32F407VG MCU*. http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419, 2015. – [Online, accessed 14-March-2015] 1

[Swedish Institute of Computer Science 2015] Swedish Institute of Computer Science: *lwIP - lightweight TCP/IP*. http://savannah.nongnu.org/projects/lwip/, 2015. – [Online, accessed 11-April-2015] 6

[Tilen Majerle 2015] Tilen Majerle : *website all about stm32f4 Discovery Programming*. http://stm32f4-discovery.com/, April 2015. – online, 11.04.2015 14.25 Uhr 7

[Uni Oldenburg 2015] Uni Oldenburg: *website describing how tftp works*. http://einstein.informatik.uni-oldenburg.de/rechnernetze/tftp.htm, April 2015. – online, 11.04.2015 15.25 Uhr 7