Report for ARM Microcontroller Project

uLoader WiSe 2014

Appel, Dennis (s813783)
Voigt, Alexander (s814526)
Merrikhi, Pedram (s882217)

April 9, 2015

Contents

Glossary

AHB1 Advanced High-performance Bus is part of the Advanced Microcontroller Bus Architecture (AMBA) of the IP-manufacturer ARM Limited (ARM).. 6

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.. 6

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

List of Figures

List of Tables

Listings

1. Introduction

CAN Logical Volume Manager (LVM) 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.

This project is based on the development of a bootloaders and its implementation inside a network. The usage stm32f4-discovery Board is a prefered and viewed as an "'All-rounder"' 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.
The ARM-Cortex-M4-Prozessor found on the STM32f4-discovery board processes the principel 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.

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, threeway 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.

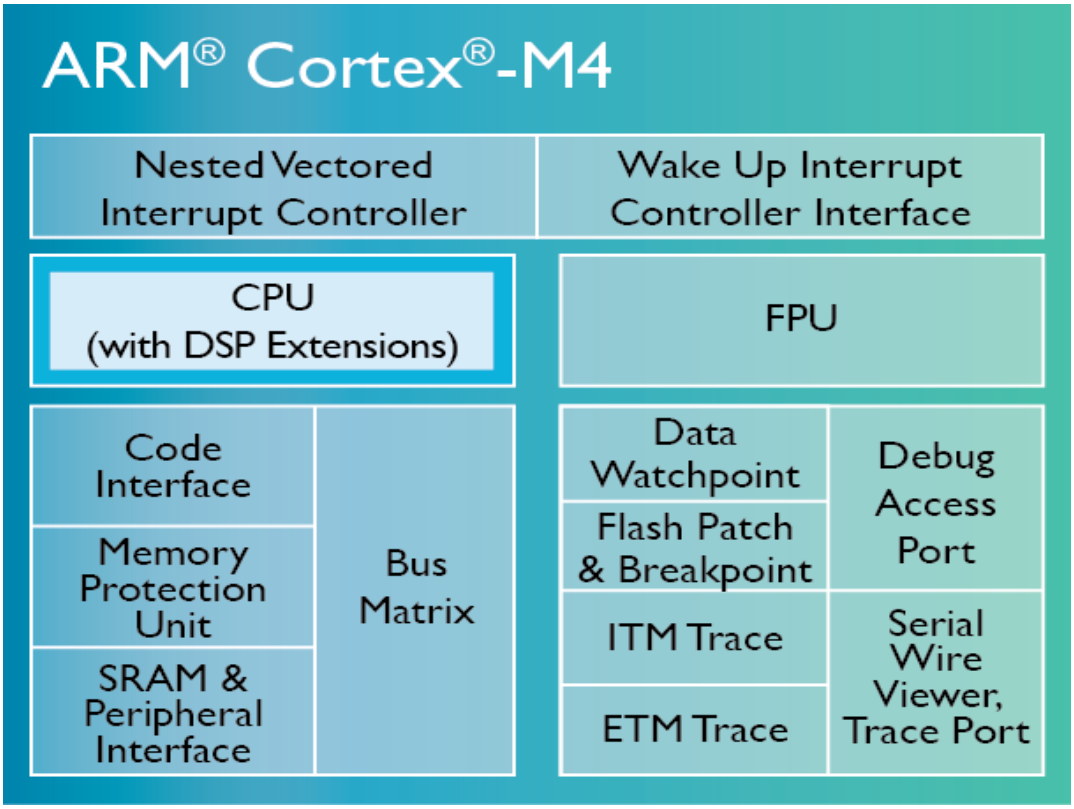


Figure 1.1.: Prinzipieller Aufbau

Next it is necessary to initialise the hardware, that would in turn be needed to load the program.
The "STM32f407 discovery board" offers three different methodes to boot up the hardware.
In order to switch between the three different boot methods, the Boot-Pins BOOT1 and BOOT2 could be set:

BOOT1	BOOT2	Boot-Mode	Adresse
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. Startup

- stack, program counter, interrupt, vector table, initial system clock

1.4. 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 kernal 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.

1.5. 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.

1.6. Differences in Development

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. Theorically one could learn to work with a new IDE every project. Vim on the other hand is everytime 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. And to be fair, it is not that hard to learn how to use an IDE.

1.7. Network

The stm32f407 discovery board itself has no possibility for any network connection. Thus the BaseBoard (BB) is needed (and used). The BB adds new drivers like network or rs232.

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:

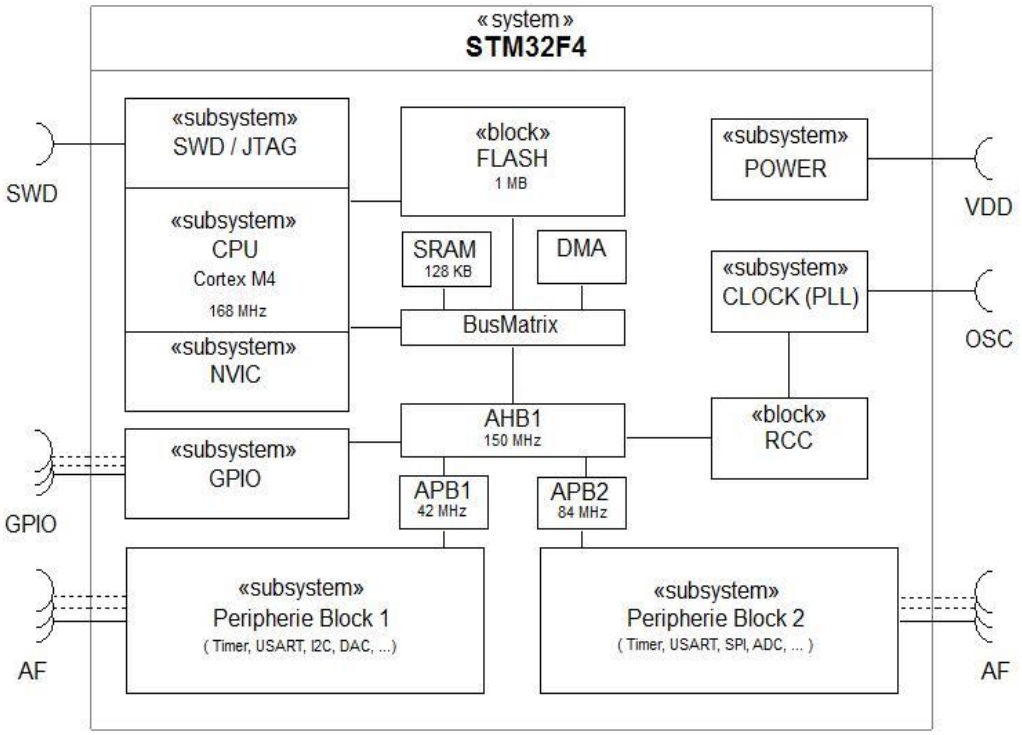


Figure 2.1.: Simple M4 Diagram

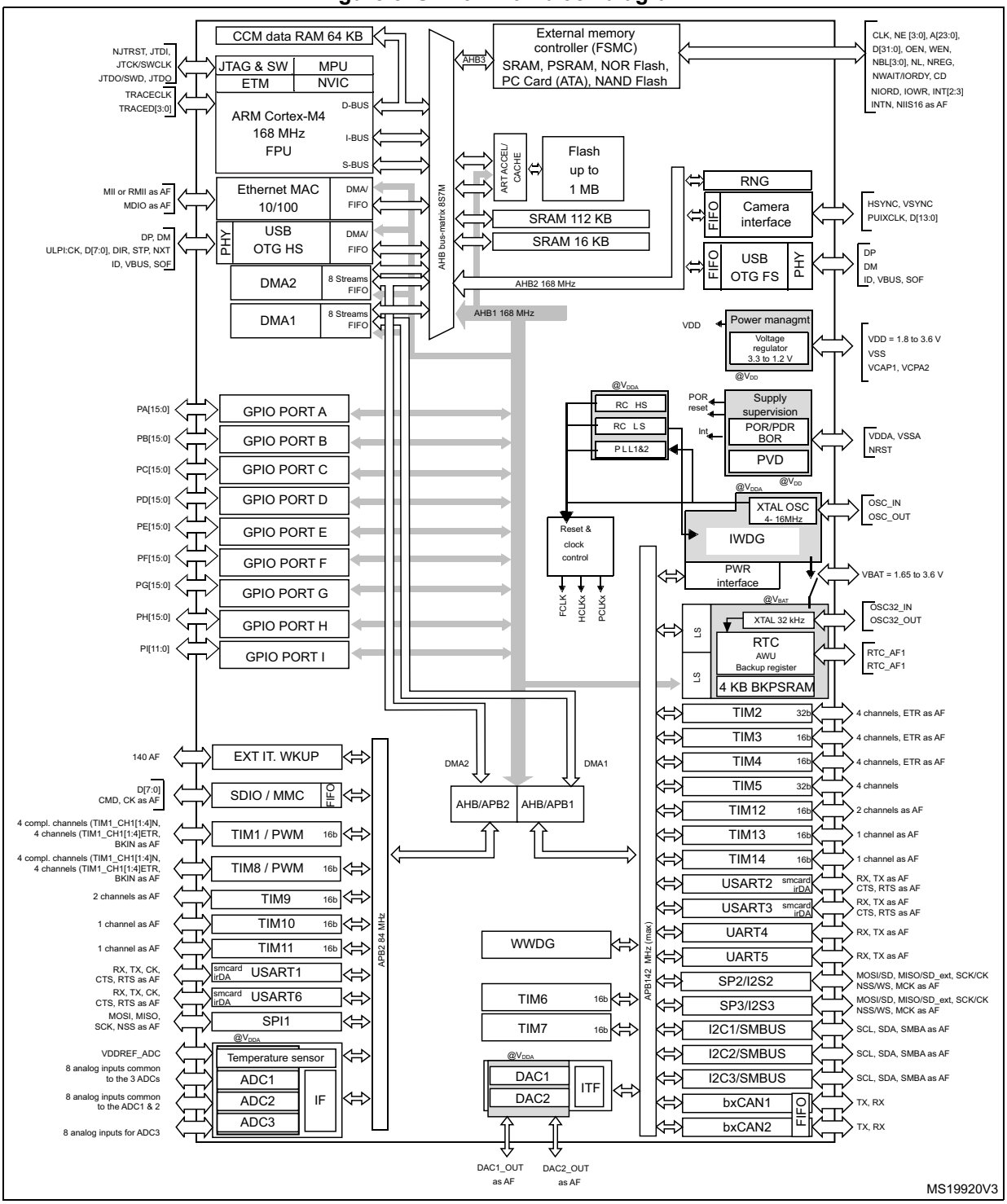
The diagram doesn't show, which peripheral systems are connected to which APBX-BusAPB, but that can be seen in the datasheet. What it does show is, that the GPIO-Pins, are connected to the AHB1-BusAHB1.
The reason for that is, that it's much faster then what is recommended for most the part of what a 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.

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. Additonally 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 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.

```
/* We are initialized */
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(USART
NVIC_Init(&NVIC_InitStruct);

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

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 whichs proves the method that was choosen 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 requiremenst are met, the nibbles are clocked out of the transcreiver on the falling edge of RXCLK. RXCLK is the 25MHz outpu 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 ransceiver on the falling edge of XTAL1/CLKIN (REF_CLK).