Report for ARM Microcontroller Project

uLoader WiSe 2014

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

April 8, 2015

Contents

List of Figures

List of Tables

Listings

1. Introduction

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

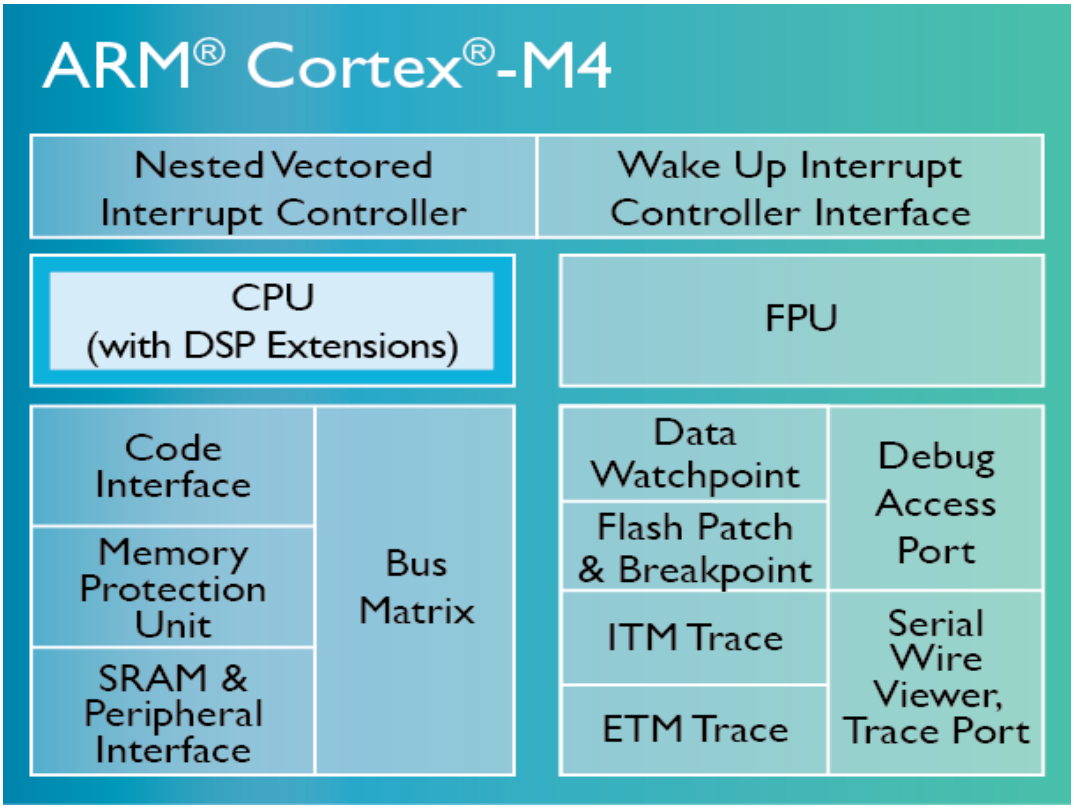


Figure 1.1.: Prinzipieller Aufbau

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:

Table 1.1.: Boot-Pin Function

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

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

Bei der Entwicklung kam die Serial Wire Debug Technologie zum Einsatz. Hierbei handelt es sich um einen Debug-Port, der speziell dafür entwickelt wurde um MCU bzw. Projekte mit MCU, bei denen so wenig wie möglich Pins verwendet werden sollen.
Dieser Port besteht aus den Leitungen folgender Tabelle:

Die anderen Leitungen des bisherigen 20-poligen Anschlussen wurden weggelassen, weil sie entweder für SWD uinteressant sind oder sie auf GND gelegt sind. Egal was davon zutrifft, sie haben keinen Einfluss auf die SWD-Kommunikation.

Es ist eine neue, sehr interessante Weise zu debuggen. Bisher war JTAG das Debugger-Interface. Die Vorteile dieser Technologie sind (frei von der ARM-Website übersetzt):

- Nur 2 Pins werden belegt
- JTAG TAP controller kompatibel
- Erlaubt dem Debugger ein weiterer AMBA-Bus-Master zu werden um auf Register / Speicher zuzugreifen.
- High Datarates - 4Mbytes/sec @50MHz
- Low Power - keine zusätzlichen Versorgungsspannung
- gute "'built in"' Fehler-Erkennung

Table 1.2.: SWD PINOUT

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.

- Schutz vor Fehlern bei Kontaktverlust

1.3. startup

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

1.4. CMSIS

Der ARM Cortex Microcontroller Software Interface Standard ist eine Händlerunabhängige Abstraktionsschicht für die Cortex-M Prozessoren.
Dabei ist CMSIS unterteilt in:

- CMSIS-CORE - API zum Zugriff auf den Prozessorkern sowie Perepherie-Register.
- CMSIS-Driver - Generischer Zugriff auf Perepherie für die Middleware (reusability).
- CMSIS-DSP - DSP Bibliothek mit über 60 Funktionen
- CMSIS-RTOS API - Standardisiertes (RTOS kompatibel)

- CMSIS-Pack - Beschreibung der wichtigen Bestandteile (Nutzersicht)
- CMSIS-SVD - Beschreibung der wichtigen Bestandteile (Systemsicht)
- CMSIS-DAP - Debug Access Port

Zusammengefasst erlaubt CMSIS einheitliche und simple Software Schnittstelle zu Prozessor und die Perepherie, sowie Echtzeit OS (RTOS) und Middleware.

1.5. Nested Vectored Interrupt Controller

Der NVIC bietet die Möglichkeit gewisse Interrupts zu konfigurieren (Priorität, aktiviert, deaktiviert...).
Neben vorgegebenen Interrupts sind hier auch die Implementierungsabhängigen Interrupts konfigurierbar. Da die ersten 15 Interrupts vorgegeben sind, können die implementierten Interrupts in der Anzahl von 0 bis 240 reichen.

1.6. Unterschiede

GNU, KEIL iar

1.7. Netzwerk

was sollte zum besseren verstehen hier einen platz finden.

2. ARM M4

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

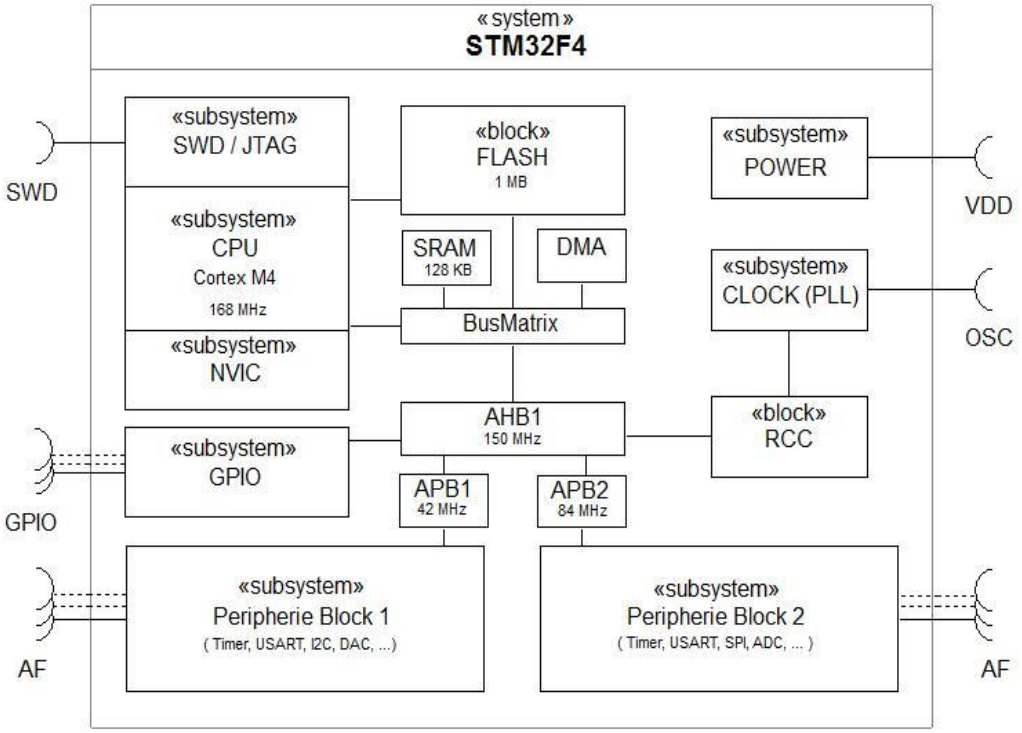


Figure 2.1.: Simple M4 Diagram

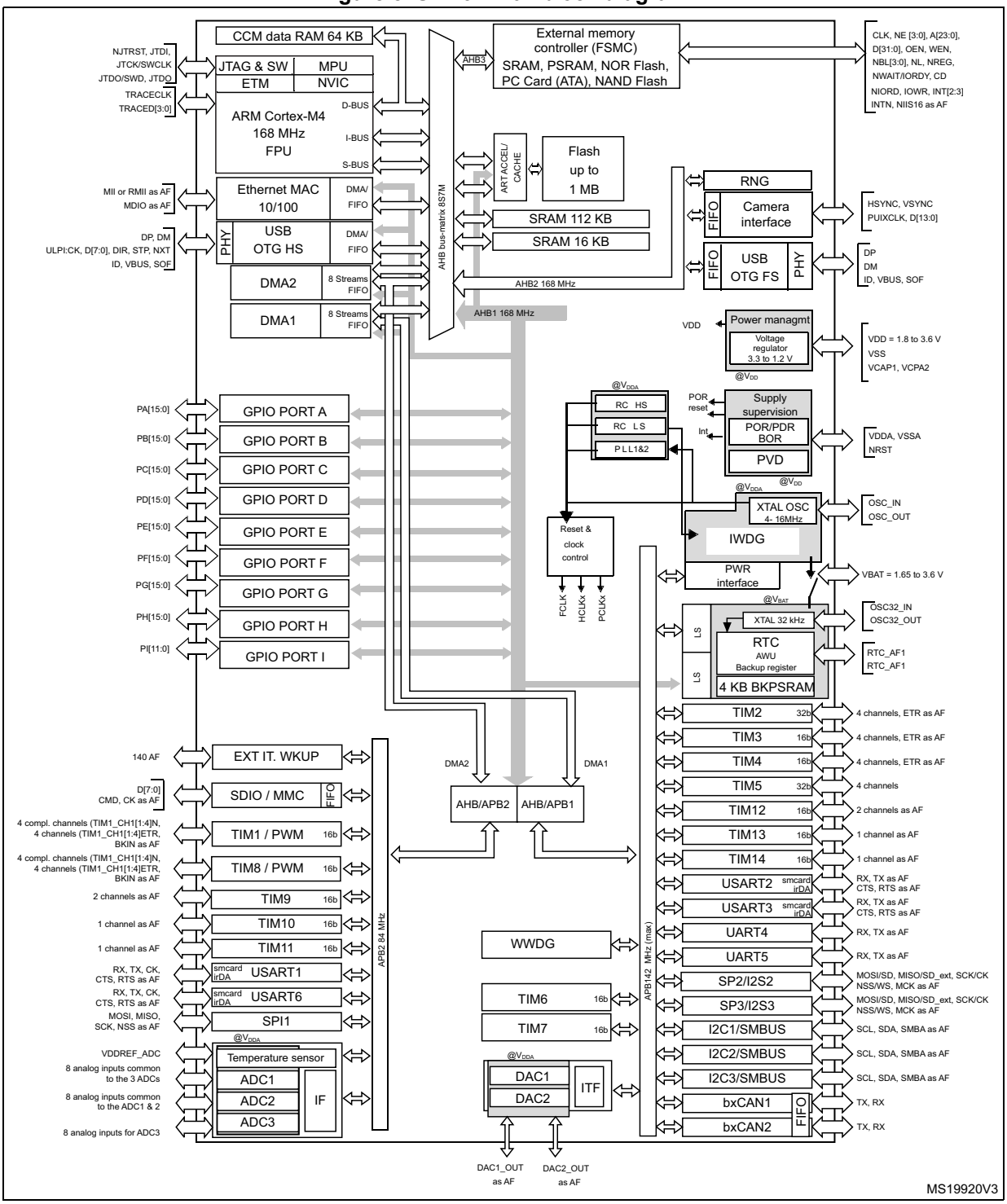
The diagram doesn't show, which peripherie systems are connected to which APBX-Bus, but that can be seen in the datasheet. What it does show is, that the GPIO-Pins, are connected to the AHB1-Bus.
The reason for that is, that it's much faster what is recommended for most demands of GPIO. The subsystems on the other hand aren't necessarly 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 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 said, 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 is not part of the NVIC configuration.
The clock is mentioned because it is an essential part to do that, neither U(S)ART[1..6] or whatever is used. Of course the the right APBX has to be configured (precise diagram of stm32f4).
The configuration is done by writing the parameters into a structure and bind and than load that struction 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;
```

As it is shown in the code the configurationprocess is done like described above.

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