Documentation VM1

Stefanie Schmidiger

October 18, 2017

Hallo, I'm a sentence on the title page.

Abstract

This will be an abstract - sometime in the future.

Introduction

Hier wird eine Einleitung geschrieben.

To Do

Aufgaben welche noch zu erledigen sind:

- **Bla** (Zeitaufwand ca. x Stunden)
  foo
- **BlaBla** (Zeitaufwand ca. x Stunden)
  foo
- **BlaBlaBla** (Zeitaufwand ca. x Stunden)
  foo

To do list / Reminder

Those things need to be done:

- **Target definition**
  What is the main outcome of VM1? Define it as exactly as possible. Graph, matrix, flow chart?
- **Approach documentation**
  Document (also with easy graphics) what the approach is and how we want to work. Interfaces between tracks?
- **Time planing**
  Refine the time table and plan accordingly. Where are the interfaces? FFT in tensorflow depends on research.
- **Papers to follow**
  Follow 3 and 4, especially the people.

Reminders and informations

Points to note:

- **Main focus**
  Tool chain is not relevant. It's about what is possible and not how exactly can that be done. For details see target definition.
- **FFT**
  Elia VM2 is reference, he also has a MATLAB model which may be useful.
- **Sobol**
  Possible use case is truck detection / count of truck axles.

Table 1: Caption for the table.

Date	Hours	Work package
19.09.2017	7	Kickoff meeting, installing Altium and LaTex software, gathering information
20.09.2017	8	Updating altium library with new components
24.09.2017	7	Updating altium library with new components, starting on schematic for HW V1
26.09.2017	7	Weekly meeting, setup LaTex documentation, schematic for HW V1
27.09.2017	10	7:30-8:00: Meeting with E.Styger and B.Imbach for coordination/update of this project, set up of old SW environment, commissioning of old HW
29.09.2017	3	Commissioning of old HW
30.09.2017	7	Design of Teensy 3.6 adapter board
01.10.2017	4	Design of Teensy 3.6 adapter board
03.10.2017	8	Weekly meeting, completion of commissioning of old HW, ordering of adapter board for Teensy 3.5 / Teensy 3.6 from HSLU, setup of programming environment (Kinetis Design Studio 3.2)
04.10.2017	7	First program on Teensy 3.5 (LED blinking), first try of SD card reading draft of changes on UAV schematic
10.10.2017	7	Weekly meeting, SD Card and RTOS initialized Teensy adapter board V2 done
11.10.2017	7	Attempt to convert c++ project to c project, SPI setup
13.10.2017	4	Read .ini file from SD card
14.10.2017	4	Read and save integers in csv list, inside .ini file
17.10.2017	7	Weekly meeting, modification Teensy 3.5, understanding SW
18.10.2017	9	Analyzation of SPI interface with old SW
–	7	botRul

Protocol

Work progress:

Changes on hardware

For a new hardware, the following features had to be considered and evaluated for a possible implementation:

- Interface for hardware debugging, e.g. SWD
- Dynamic switching between RS232 and TTL possible, e.g. by changing a jumper position
- Evaluate use of hardware flow control on interface between new hardware and modem. Does it solve the problem of package losses or does it only move the problem to the Teensy side?
- 3.3V output for powering of connecting devices

ComponentEvaluation

The hardware should at least satisfy the following requirements:

- Optimization of size and weight
- Hardware fit for field use (e.g. connectors, plugs, housing)
- Powerful processor with more memory and RNG/encryption support (e.g. K64 or K66)
- SWD/JTAG debugging
- UART hardware flow control
- SD card (regular or micro)
- 3.3V power supply pins available
- Each interface should be configurable to either RS232 or TTL output

Processor selection

Andreas Albisser used a Teensy 3.1 in the previous project.
The Teensy development boards are breadboard compatible USB development boards. They are small, low-priced and yet equipped with a powerful ARM processor.
The previously used Teensy 3.1 does not support encryption, therefore a more powerful processor had to be selected. Currently, the only Teensy with an ARM processor that supports encryption is the Teensy 3.6.

Debugging interface selection

The Teensy development boards all come with a pre-flashed bootloader to enable programming over USB. They use a less powerful processor as an interface to the developer to enable the use of Arduino libraries and the Arduino IDE. The only downside is: Hardware debugging is not possible on the Teensies because of that processor interface.
The Teensy 3.6 has the SWD (Serial Wire Debug) debugging pins available on pads on the bottom side though so that a header can be soldered onto those pads.In order to enable hardware debugging, the interface processor has to be removed because it can not be stopped from communicating with the ARM processor. A detailed description of the removal process can be found on mcuoneclipse.com.
The SWD debug interface has the following pins defined:

- DD -¿ Debug Data

- GND

To enable JTAG pins being mapped directly onto a SWD debug interface, the following pins are defined additionally:

- DC -¿ Debug Clock, can be connected to JTAG TCK

- SWO -¿ Serial Wire Output, can be connected to JTAG's TDO

For more information, see:
`http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/DDI0314H_coresight_components_`
`trm.pdf`
Pinout: see p. 357

SD card

Read more: `http://elm-chan.org/docs/mmc/mmc_e.html`

Software validity

CRC vs Checksum

Anytime data is stored in a computer with the intent to transmit it, there is a need to ensure that the data is not corrupted. If corrupted data was sent, there would be inaccurate data transmitted and it may not work as desired. There is, therefore, a need for an error detection system that checks that all the data entered is okay and not corrupt before any encryption or transmission occurs. There are two main methods to check the data.
Checksum is arguably the oldest methods that has been used in the validation of all data prior to its being sent. Checksum also helps in authenticating data, as the raw data and the entered data should conform. If an anomaly is noticed, referred to as an invalid checksum, there is a suggestion that there may have been a data compromise in a given method.
Cyclic redundancy check, or CRC as it is commonly referred to, is a concept also employed in the validation of data. The principle used by CRC is similar to checksums, but rather than use the 8 byte system employed by Checksum in checking for data consistency, polynomial division is used in the determination of the CRC. The CRC is most commonly 16 or 32 bits in length. If a single byte is missing, an inconsistency is flagged in the data as it does not add up to the original.

Differences

One of the differences noted between the 2 is that CRC employs a math formula that is based on 16- or 32-bit encoding as opposed to Checksum that is based on 8 bytes in checking for data anomalies. The CRC is based on a hash approach while Checksum gets its values from an addition of all truncated data which may come in 8 or 16 bits. CRC, therefore, has a greater ability to recognize data errors as a single bit missing in the hash system which changes the overall result.
The checksum, on the other hand, requires less transparency and will provide for ample error detection as it employs an addition of bytes with the variable. It can, therefore, be said that the main purpose of CRC is to catch a diverse range of errors that may come about during the transmission of data in analog mode. Checksum, on the other hand, can be said to have been designed for the sole purpose of noting regular errors that may occur during software implementation.
CRC is an improvement over checksums. As earlier noted, checksums are a traditional form of computing, and CRC's are just a mere advancement of the arithmetic that increases the complexity of the computation. This, in essence, increases the available patterns that are present, and thus more errors can be detected by the method. Checksum has been shown to detect mainly single-bit errors. However, CRC can detect any double-bit errors being observed in the data computation. In understanding the differences between the two data validation methods, knowledge is gathered as to why these two methods are used hand-in-hand in Internet protocol, as it reduces the vulnerability of Internet protocols occurring.
If you enable ECC (which is highly recommended), then the data rate you can support is halved. The ECC system doubles the size of the data sent by the radios. It is worth it however, as the bit error rate will drop dramatically, and you are likely to get a much more reliable link at longer ranges.
Read more: `DifferencesBetweenCRCAndChecksum|DifferenceBetweenhttp://www.differencebetween.net/technology/software-technology/differences-between-crc-and-checksum/#ixzz4trxUtflQ`

Error correction in Modem

When using the RFD868x model, a variety of configurations can be made that will affect data throughput and errors.
The ECC (Error Correcting Code) parameter makes a big difference to the data rate you can

support at a given AIR_SPEED. If you have ECC set to zero, then no error correcting information is sent, and the radio uses a simple 16 bit CRC to detect transmission errors. In that case your radio will be able to support data transfers in one direction of around 90% of the AIR_SPEED.
Read more: `http://ardupilot.org/copter/docs/common-3dr-radio-advanced-configuration-and-te`
`html#common-3dr-radio-advanced-configuration-and-technical-information`

CRC

Header: 11 Bytes -¿ 10 Data bytes + 1 byte CRC (CRC8). CRC8 ¿ correction up to 256 bit possible Payload: max 150 (normally 50...100 bytes). 2 byte CRC -¿ Fehlerwahrscheinlichkeit?

Encryption

General information about encryption

The purpose of encrypting data is to secure information so that only the intended receiver can decypher and read it. It is not entirely impossible to decrypt a encrypted message without the encryption key but considerable computational resources would be required. In encryption, the original data or message, in cryptography referred to as plaintext. There are different algorithms that can be used for encryption and all of them require an encryption key. The encrypted plaintext is then referred to as ciphertext. The ciphertext can be decrypted easily by using the intended key.
There are two techniques to ensure confidentiality: symmetric and asymmetric encryption.

Confidentiality: symmetric and asymmetric encryption

In symmetric encryption, both parties are in possession of the same key and that key us used for encryption and decryption. This is the oldest and best-known technique and was often used in letters. The key then consisted of a word or a number and was applied to the message in a particular way such as shifting the alphabet in a particular order. With the knowledge of the encryption key, the message could be deciphered easily. One profound issue with symmetric encryption is the transmission of the shared key to the other party without it falling into the wrong hands.
In asymmetric encryption, there are two separate keys: a private key and a public key. When encrypting a message with a public key, it can only be decrypted with the corresponding private key and vice versa. This solves the problem of having to secretly transmit the key to the receiver because the key that is intended for the opposite party is a public key. There is no way to compute the private key when knowing the public key. So for two users to communicate with encrypted messages, they would have to agree on an encryption algorithm and use the other users public key to encrypt a message. Both users can send a query over the network to receive the other users public key. The disadvantage of the asymmetric encryption is that it requires far more processing power than the symmetric encryption.
Encryption ensures the confidentiality of a message but other measurements are still required to ensure integrity and authenticity.

Integrity

There are two types of data integrity threats: passive and active threads.
Passive threads are due to accidental change of data such as noise on the transmission channel or data getting corrupted upon saving it. The simplest way to detect those errors is by adding a error-correcting code or a checksum.
In an active thread, a third party tampers with the original message and might even replace the error-correcting code or checksum with a newly calculated one. One way to prevent it is by using hash functions. Hash functions map arbitrary sized data to data of fixed size (called hash codes, digests or simply hashes). By only changing one bit of the original data, the hash codes will change almost completely.

Authenticity

Authenticity proves that the message originated from a specific entity. This can be done by adding an authentication tag. When the message is altered with, the authentication tag becomes invalid.