Bachelorthesis

David Niederwestberg

Implementation of a Host-System for Communication with Secure Elements by Dual-wire Protocol on an ARM-Cortex M3 Microcontroller

David Niederwestberg

Implementation of a Host-System for Communication with Secure Elements by Dual-wire Protocol on an ARM-Cortex M3 Microcontroller

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bernd Schwarz
Zweitgutachter: Prof. Dr. Wolfgang Tobergte

Eingereicht am: 3. August 2017

David Niederwestberg

Thema der Arbeit

Entwicklung eines Host-Systems für die Kommunikation mit Secure Elements mittels eines Dual-wire Protokolls auf einem ARM-Cortex M3 Microcontroller

Stichworte

Dual Wire Protocol (DWP), Single Wire Protocol (SWP), Chipkartenleser, Schlüsselwort 2

Kurzzusammenfassung

Dieses Dokument beschreibt die Implementierung des Dual Wire Protocol auf einem Cortex-M3 μC. Die Implementierung ist für einen bestehenden Chipcartenleser konzipiert, der bei der Validierung von Mikrochips für sichere Anwendungen eingesetzt wird. Nach einer Beschreibung der Protokollgrundlagen wird die Erweiterung entworfen und implementiert. Mit verschiedenen Tests wird das System angepasst um die Spezifikation zu erfüllen und seine Funktionalität zu beweisen.

David Niederwestberg

Title of the paper

Implementation of a Host-System for Communication with Secure Elements by Dual-wire Protocol on an ARM-Cortex M3 Microcontroller

Keywords

Dual Wire Protocol (DWP), Single Wire Protocol (SWP), smart card reader,keyword 2

Abstract

This document descries the implementation of the Dual Wire Protocol on a Cortex-M3 μC. This implementation is designed for an existing reader platform, which is used in the validation of microchips for secure applications. After a description of the protocol basics, the extension is designed and implemented. With different tests the system is than adjusted to meet the specification details, and to prove its functionality.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Ambition . . . 2
1.3 Concepts of realization . . . 2
1.4 Chapter overview . . . 4

2 Technical overview **5**
2.1 DWP Protocol . . . 5
2.1.1 DWP Basics . . . 5
2.1.2 Timing constraints . . . 7
2.1.3 DWP-Frame structure . . . 8
2.1.4 Logical Link Control . . . 8
2.2 CRC calculation . . . 11
2.3 Target-system options . . . 12
2.3.1 FPGA . . . 12
2.3.2 Microcontroller . . . 13
2.4 Platform . . . 13
2.4.1 Spectro TP . . . 13
2.5 FreeRTOS . . . 14

3 Analyses of the current micro controller **15**
3.1 Timer and PWM . . . 15
3.2 GPIO . . . 17
3.2.1 C / Assembler . . . 20
3.2.2 delay compensation . . . 20

4 Architectural Design **22**
4.1 System structure . . . 22
4.2 Program flow for the communication . . . 24
4.3 State machine design . . . 27
4.4 Test-pattern generator design . . . 28

5 Realization **30**
5.1 Programming languages . . . 30
5.2 Code implementation . . . 30
5.2.1 GPIO solution . . . 31
5.2.2 PWM solution . . . 32

5.2.3 DWP interface 33
5.2.4 DWP waveform decoders 33
5.3 integration into the SaReader firmware 34
5.4 test-pattern generator 36

6 Testing **37**
6.1 Test setup 37
6.2 Test Pattern 38
6.3 Timing Analyzes 38
6.3.1 Problems with optimization in the GPIO solution 39
6.3.2 Slave resume time with PWM solution 41
6.4 Interrupt Latency with RTOS 42

7 Timing driven Design Modifications **47**
7.1 Changes to code positions 47
7.2 Semaphores and Queues in RTOS 48

8 Conclusions **50**
8.1 Results 50
8.2 Summery 50
8.2.1 Prospect 51

9 Appendix **53**
9.1 Abbreviations & Definitions 53
9.2 Charts & Pictures 54
9.3 Measurements 55
9.4 Code 59
9.4.1 Generation S1 with GPIO 59
9.4.2 Generation S1 with PWM 59
9.4.3 DWP interface 59

List of Figures

1.1 System overview with different communication protocol modules in the SaReader, that can be selected. 2

2.1 Bit encoding of signals S1, host to slave; The logical 1 is defined as a PWM with a duty cycle of 75% the logical 0 with 25%, the clock is derived from the period. [ETS12, p. 20] 6
2.2 Bit encoding of signals S2, slave to host; The logical state of the S2 bit is determined during the high state of S1, its state may thereby only be changed while S1 is low. [ETS12, p. 22] 6
2.3 DWP transition from a suspended state to active state after a slave request. . . 7
2.4 The structure of a DWP-frame with the area where bit stuffing is applied. The frame data consists of the (Logical Link Control) LLC-control field, the LLC-Payload and the 16 bit CRC. Each frame always starts with an Start Of Frame byte (SOF) and ends with the End Of Frame (EOF). 8
2.5 Program flow for detection of SOF & EOF in a received flow of data with a buffer.
The buffer is initiated with 0, every received bit is than appended to the end, the buffer is than checked for a match of SOF or EOF. 9

3.1 PWM-Timing measurement of entering the ISR; C2(red): PWM, IR generated on reset; C3(blue): GPIO pin output for measurement 16
3.2 Program flow for testing first GPIO output with wait cycles 18
3.3 Timing compare of setting GPIO pins with register and dynamic set function from the GPIO-library 19
3.4 Program flow for delay compensation in different length timing paths in an IF-statement 21

4.1 System structure with the designated functions. The SaReader board includes a socked for the LPC1769 and a DUT (cf. section 9.1). On the LPC the needed segments for the communication can be seen. 22
4.2 Communication structure of the DWP module in the reader; green: communication section; blue (FSM): state machines 23
4.3 Basic program flow for generating one bit output on S1 (cf. figure 2.1) with GPIO pins. The flow is implemented in the Com-module in figure 4.2. 25
4.4 S2 input buffer and program flow for processing one bit of S2 (cf. section 2.1.3) 26

4.5 ACT state machine for processing the activation sequence of the DWP (cf. figure 4.2)
blue: internal event; green: action; red: communication 27
4.6 SHDLC state machine for activation of the SHDLC protocol and processing incoming frames during activation (cf. figure 4.2); blue: internal event; green: action; red: communication (w = window size) 28
4.7 Program flow for S2 generator; left: ISR to synchronize the set value with the S1 signal from the reader; right: function to activate the message sending procedure 29

5.1 Basic program flow of the main tread of each communication module. Command (cmd) and data are included in the queue element 35
5.2 Section of the SaReader board circuit: The optocoupler is driving the S2 pin (IO1<) of the LPC board, for connecting the test-pattern generator it needs to be severed. 36

6.1 Test setup for testing timings and communication. S1 and S2 are recorded on oscilloscope and analyzed by the PC. 37
6.2 To suspend the PWM MR1 is moved above the MR0 value, this results in a continues high signal. To resume the PWM MR1 is moved back and the PWM counter is set to below MR1(red dot). 41
6.3 Shifting PWM resume caused by not setting the PWM counter to a specific value when resuming the PWM. Jitter: $4\mu s \equiv PWMperiod$
(Persistent measurement without RTOS, measured on auxiliary signal as S1 could not be used for a precise measurement cf. figure 9.3) 42
6.4 Interrupt latency measurement of the slave resume $\Delta t1$ time with RTOS & PWM counter reset . Setup time measurement with S2 and GPIO pulse in ISR. Jitter: $\Delta t2_{max} - \Delta t2_{min} \approx 550ns$; slave resume time: $\Delta t1_{max} \approx 2.85\mu s$; number of measurements: 2017 43
6.5 Flow of critical section to visualize the effects of the ISR delay. The Interrupt is delayed until after the critical section. This results in a longer pulse on the GPIO pin. 43
6.6 Time measurement from PWM reset to read in of S2 with latency caused by critical section. Wide pulses on C3 caused by interrupt during critical section. C4, Z1(green, yellow): PWM signal (S1); C2, Z2(red): readin of S2 finished; C3(blue): critical section; light blue area: measured time 44
6.7 Received frame with disabled WDT-Feed during the communication.
C1(yellow): S2 (slave to host); C2(red): S1 (host to slave); C4(green): readin of S2 finished, cf. setup time left; C3(blue): critical section of the WDT-Feed . . . 46

7.1 Measurement with a Semaphore_ give in the ISR. Due to the long ISR one Interrupt is lost (one spike on DGB1 is missing in the marked area).
DBG1: ISR finished; DBG2: sem_ give duration 48

7.2 Measurement with a software interrupt in the ISR. Here no Interrupt is lost, as the SW-ISR has a lower priority. 49

9.1 LPC1769 connected to the SaReader board and a secure element in a CLCC68 socket 54
9.2 Time measurement from PWM reset to read in of S2
Jitter: $\Delta t1_{max} - \Delta t1_{min} \approx 480ns$;
S2 worstcase: $\Delta t1_{max} < \frac{1}{4}T$ $(940ns < 1\mu s)$ 55
9.3 Persistent measurement of the PWM-resume without setting counter 56
9.4 Persistent measurement of the PWM-resume with setting counter 56
9.5 Measurement of the execution duration of a Semaphore give function in the ISR after a received frame.
Duration: 4,29µs 57
9.6 Measurement of the execution duration of a Queue send function in the ISR after a received frame.
Duration: 4,84µs 58

1 Introduction

1.1 Motivation

In the modern world encryption, is part of our daily life even if it is not always obvious for us. For credit cards, access control, the encryption in cellphones (SIM card), trusted platform modules (TPM) in notebooks, contactless car keys and many other devices secure platforms are needed. We have to trust these devices not only to operate correct but also to be tamper-resistant. These so called Secure Elements (SE) are under constant attack from criminals, governments or other people. These attacks may not be limited to the Secure Element but also to the communication, like attacks on Keyless Go [Rüs15] system, in the automotive industry. Leaks like "Vault 7" [Bec17] show that even governmental organizations, in this case the CIA, use security vulnerabilities. So analyzing is an essential part in every security assessment process, as hardware cannot simply be updated unlike software. For these reasons with a Secure Elements it is important to validate not only the functionality of the product, but particularly the security. A new product has to withstand all sorts of penetration tests, during these tests it is important to test not only within the specification limits but also to be able to test outside of these limits. This functionality is not possible with a standard of the shelf Smart Card Reader that can be bought, because to control as many parameters as possible a unique solution is thereby needed. Single Wire Protocol (SWP) and Dual Wire Protocol (DWP) are widely used protocols for communication with Secure Elements. Both protocols where developed for communication between Secure elements and a contactless frontend which provides a wireless Interface. The DWP is derived from the SWP which is specified by ETSI (European Telecommunications Standards Institute) in TS 102 613 [ETS12]. Only the physical transmission layer of the two protocols is different, because the DWP uses two wires for the communication instead of a single one in the SWP. If an interface is implemented in a Secure Element it has to be tested.

1.2 Ambition

To verify the secure devices only implementing the Dual Wire Protocol (DWP), an extension for an existing Card Reader is needed. The current implementation of this reader, the SaReader (picture of the SaReader cf. figure 9.1), is not able to communicate over Dual Wire Protocol. Standard micro controllers don't provide SWP or DWP peripherals, as the protocols are almost exclusively used in secure elements, therefore a custom solution is needed. The main focus of this thesis is a basic implementation and the testing of the achievable performance on the given hardware. Flexibility and control of as many parameters as possible are important features of the implementation, as further extensions may for example need to be able to send faulty information and other corrupt data like bit flips or bit stuffing errors. The results of the performance tests will determine if a given microcontroller can be used, or if for example an additional FPGA is required. The Timing is essential for the communication because the implementation will work directly on the physical layer, OSI layer 1, and the Media Access Control (MAC) sublayer of OSI layer 2. The focus lies hereby on the development and implementation for this two layers .In figure 1.1 the basic structure of the SaReader is displayed, the extension has to be a compatible communication module like the T0/T1 or the ISO 14443 mobile.

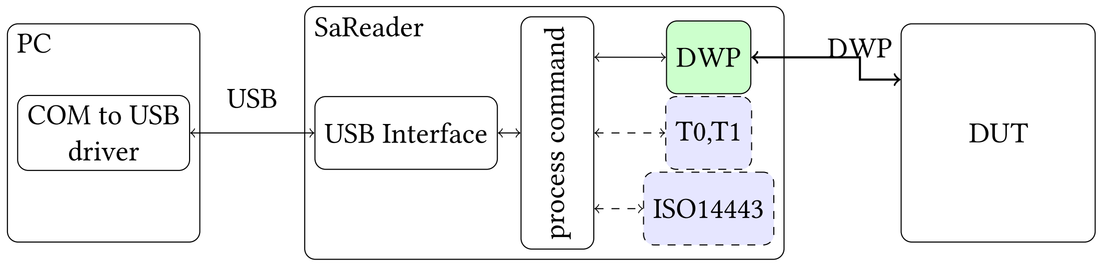


Figure 1.1: System overview with different communication protocol modules in the SaReader, that can be selected.

1.3 Concepts of realization

The main goal is to realize the communication on the existing LPC1769 micro controller, based on a Cortex M3-core. As the protocol needs to be implemented on the physical layer a solution with a FPGA is also an option. There are two basic concepts for this implementation on the microcontroller that will further be analyzed. As one signal of the Dual Wire Protocol is a

Pulse-width modulation the use of the PWM module seems obvious. The second concept to generate this signal is the use of standard GPIO pins and use wait cycles to generate the correct period. Both concepts have its pros and cons and need to be further evaluated.
For the PWM solution these are:

+ A precise hardware generated timing
+ No CPU idle time
- Potentially slower due to the use of slower connected peripherals and use of an ISR

For the GPIO solution:

+ Faster as only the fast GPIO pins are needed
+ Simple to implement
- No tasks can run in the background
- More measurements to generate the correct timing
- Code changes cannot simply be implemented, as the entire timing changes

With an FPGA:

+ A precise hardware generated timing
+ A fast data transmission is possible
- Not all readers could use it
- Expensive as readers need a hardware upgraded
- Different reader versions need to be supported

In the thesis these considerations are further evaluated to determine the best solution for the given case of application.
To analyze the timings of both solution they will be tested with an oscilloscope. To test the systems input and integrity a independent system has to be used. Here a Comprion Spectro TP is available, it is a validation platform for smart card testing. For testing chips the SWP is supported on this platform. With a separate translation module, that is available, it is possible to translate it to DWP. If the Spectro does not support functionality of simulating a smart card, the alternative is to use a second board LPC board, which will then simulate a secure element.

1.4 Chapter overview

The thesis is divided in eight chapters, the fist part is an introduction and an overview of the topic.
In the second chapter the basics of the protocol and the system are introduced. This is the technical foundation for understanding the thesis. As the implementation is supposed to be an extension to an existing system, the microcontroller is further analyzed in chapter three, on bases of the two implementation options.
With the basics of the previous two chapters all necessary components are designed and explained in chapter four. In the design also the existing system is taken into account, and program flows for the communication are developed. Also a test-pattern generator is described here. The designs of these different components are implemented, the implementation details are discussed in chapter five.
The tests of the project realization are included in sixth chapter. Here the test setup is presented and different test techniques are discussed. Also the problems which accrued during these tests and its solutions are included in this chapter. Changes that had an greater impact on the design were moved to the next chapter, where also design changes are visualized, with which the timing problems where solved.
The eights and last chapter includes a presentation of the results, a summary and a prospect of the needed steps for a full integration of the DWP in the current reader system.

2 Technical overview

2.1 DWP Protocol

The Dual Wire Protocol (DWP) is derived from the Single Wire Protocol (SWP), which was developed by Gemplus in 2007 for the use in integrated circuit cards on mobile phones [RE08, p. 302]. On a SIM card with USB and T=0 protocol only one of the eight pins was available to be used for this functionality. The main difference between Single and Dual Wire Protocol is the use of two wires for communication instead of a single one. The digital data and frame structure is hereby identical. Since an official DWP specification was not available and the protocols are almost identical, an altered SWP spec was used instead for the preformed developments. The close relation of these protocols can also be derived from a US patent 9350831 [Tro16], where the DWP is descried as "a low-power alternative to the SWP".

2.1.1 DWP Basics

Single Wire Protocol communication on one wire works by the host modulating the voltage and the slave modulating the current. The advantage is, that only a single wire is needed for full duplex communication. This is also the reason for the name. A disadvantage is, that there is additional hardware needed for the conversion of the current modulation to a voltage modulation that can be processed with a microprocessor or other standard CMOS logic. With the Dual Wire Protocol this is not necessary, as the slave communicates over a second wire with the host by also modulating the voltage in synchronization with the host signal.

In both protocols the clock signal is integrated in the host to slave signal (S1) (cf. figure 2.1). The clock-integration in the data stream is achieved by the use of a pulse-width modulation (PWM). The clock is derived from the period of the modulation, the bit-value by the duration of the high state. One bit has to be thereby defined as having two rising edges, the duty-cycle is 0,75 for the logical 1 and 0,25 for the logical 0.

The signal from the slave to the host (S2) is only valid during the high state of the S1 signal (cf. figure 2.2). This originated from the fact that with the SWP it is easier to measure a current if there is a higher voltage. Consequently S2 may only be changed during the low state of S1, so

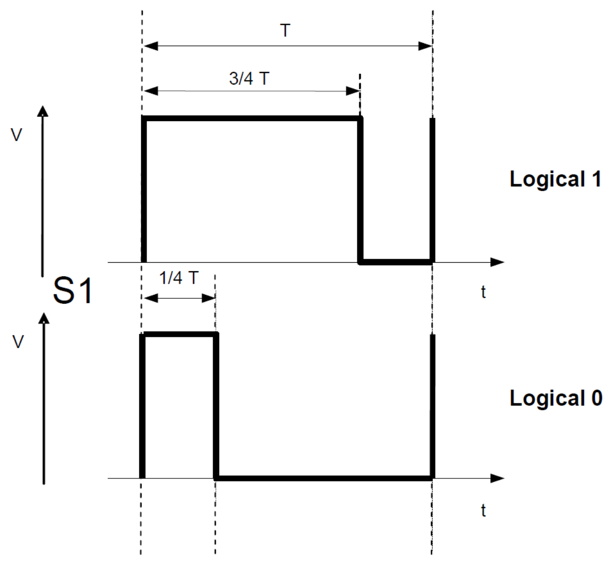


Figure 2.1: Bit encoding of signals S1, host to slave; The logical 1 is defined as a PWM with a duty cycle of 75% the logical 0 with 25%, the clock is derived from the period. [ETS12, p. 20]

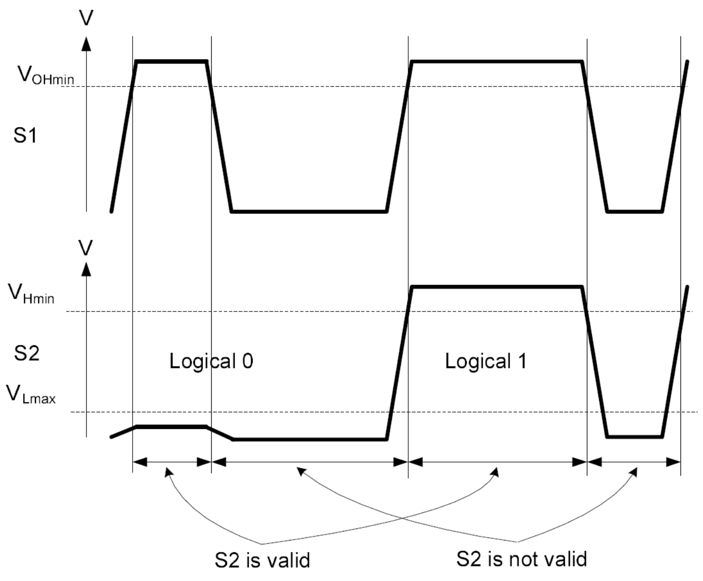


Figure 2.2: Bit encoding of signals S2, slave to host; The logical state of the S2 bit is determined during the high state of S1, its state may thereby only be changed while S1 is low. [ETS12, p. 22]

there is a defined time frame to read the state of S2. If no data has to be send by either side the host can suspend the communication. in this case the S1 signal is in high state and S2 in low state. If the slave wants to send data while the communication is in suspended state it can raise S2 to high to indicate to the host to resume the communication cf. figure 2.3.

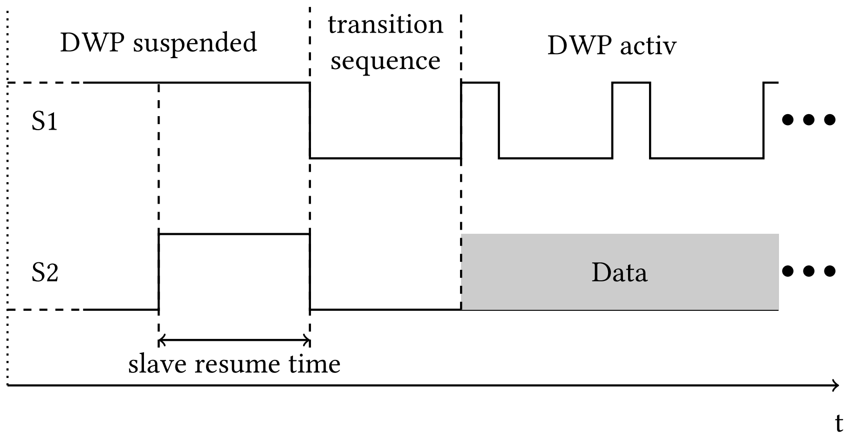


Figure 2.3: DWP transition from a suspended state to active state after a slave request.

2.1.2 Timing constraints

The most significant timings for the S1 signal are listed in Table 2.1. In the specification is also stated, identical bit-duration's are not required, as long as the high / low ratio is correct. "The bit-duration may be different for each transmitted bit." [ETS12, p. 20]. This can potentially be used to increase the transmitting speed by reducing the duration of bits with a high idle time. The second important timing is the "resume by slave time", the time in which reader has to acknowledge the slave's resume request, by setting S1 from high to low state cf. figure 2.3. The

Parameter	**Minimum**	**Nominal**	**Maximum**
Period T (bit duration)	1 µs	-	5 µs
H state of S1 for logical 1	0,70 x T	0,75 x T	0,80 x T
H state of S1 for logical 0	0,20 x T	0,25 x T	0,30 x T
Resume by slave time	-	-	5 µs
Extended bit-duration's	0,59 µs	-	10 µs

Table 2.1: Timing specification of the DWP for S1 signal; cf. figure 2.1 for the slave resume figure 2.3

length of the transition sequence is not further specified, thereby a maximum of 0,75 x T will be used for this implementation.

2.1.3 DWP-Frame structure

The communication is based on frames for both the S1 and the S2 signal cf. figure 2.4. Each frame begins with a Start Of Frame byte (SOF) it has the value '0x7E' and an End Of Frame byte (EOF) with the value '0x7F'. A frame consists of a payload with a maximum length of

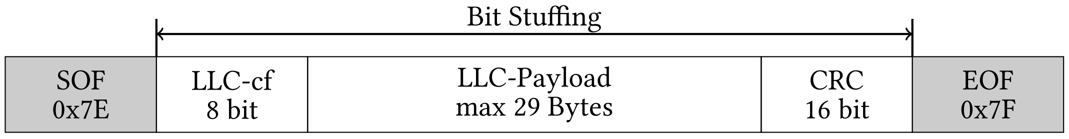


Figure 2.4: The structure of a DWP-frame with the area where bit stuffing is applied. The frame data consists of the (Logical Link Control) LLC-control field, the LLC-Payload and the 16 bit CRC. Each frame always starts with an Start Of Frame byte (SOF) and ends with the End Of Frame (EOF).

30 bytes and a 2 byte cyclic redundancy check (CRC) based on ISO/IEC 13239. During the transmission of the payload and the CRC bit stuffing is used to avoid an unintentional send of SOF or EOF. Bit-stuffing works by inserting a logical '0' following five consecutive '1' these stuffed bits have to be filtered out later by the receiving side. An exception is after the last bit of the CRC no stuffing bit will be inserted here. Bit stuffing is also implemented in other communication protocols like Ethernet or USB, as implemented in the USB protocol it can also be used for clock synchronization [RE08, p. 298].
As shown in figure 2.5 to detect both SOF and EOF in the communication of S2, a software shift register with a 2 Byte buffer and a left shift for every incoming bit will be used. SOF or EOF can then be detected by a match check of bits 7-0. Bit 8 will contain the bit that has to be processed. To filter out the SOF the processing must be delayed until the first bit of the payload is in position 8. When detecting an EOF the last bit of the payload is in position 8. After processing this last bit the receive process can be stopped until a SOF is detected again. An example of this flow can be seen in table 2.2.

2.1.4 Logical Link Control

Three Logical Link Control (LLC) layers are defined in the current SWP/DWP-specification.

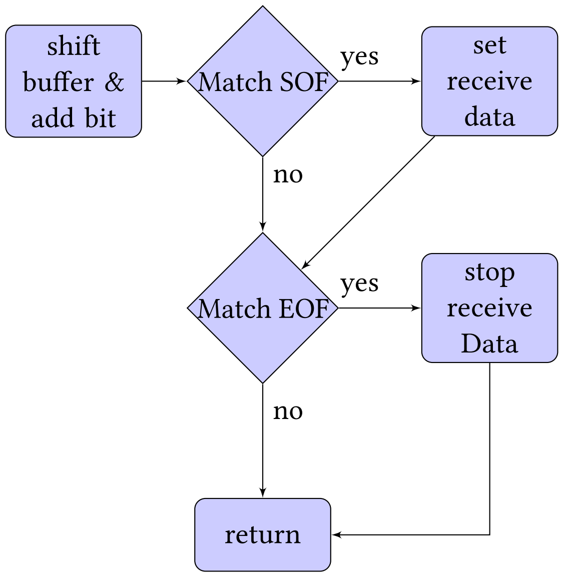


Figure 2.5: Program flow for detection of SOF & EOF in a received flow of data with a buffer. The buffer is initiated with 0, every received bit is than appended to the end, the buffer is than checked for a match of SOF or EOF.

step	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
init	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	**0**
2	0	0	0	0	0	0	0	**0**	**1**
3	0	0	0	0	0	0	**0**	**1**	**1**
7	0	0	**0**	**1**	**1**	**1**	**1**	**1**	**1**
8	0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**
r1	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	d0
r9	d0	d1	d2	d3	d4	d5	d6	d7	d8
rn	dm	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Table 2.2: Buffer (8 bits) function example:
In the steps 1 to 8 the SOF is written to the buffer and detected in step 8. The receive is set and in r1 to rn the data bits **d** are shifted into the buffer. In step r9 the first data bit (d0) is processed, with the detection of the EOF in rn the last bit is processed and the receive is stopped.

- SHDLC: SHDLC (Simplified High Level Data Link Control) is the generic LLC used for most communication in contactless transaction. It is a simplified version of the HDLC (High Level Data Link Control) protocol specified in ISO 13239.
- CLT: CLT (ContactLess Tunnelling) is optional and used for protocol handling for proprietary UICC communication it will not be implemented at this point.
- ACT: The ACT LLC (ACTivate) is used for the activation of the interface. The support it is thereby mandatory.

The LLC-type and the frame specific settings are encoded in the LLC control field, the first byte of the payload. All bytes starting with a leading '1' are SHDLC type frames, CLT frames start with '010' and ACT with '011'. The start combination '00' is reserved for future use.

ACT frame

At least the first three frames after power up are always of this type. The ACT-frame has the basic structure described in Table 2.3.

	LLC Control field (1 byte)						ACT payload (0-3 byte)	
Position	7	6	5	4	3	2 - 0	0-2 byte	0-1 byte
Value	0	1	1	FR	INF	ACT ctrl	ACT data	ACT information

Table 2.3: Structure of the ACT Frame

- The FR bit may only be send from the host to the slave. It indicates to the slave to repeat the last ACT-frame. A FR bit is send, when no or a corrupt frame was received by the host.
- The INF bit may only be send by the UICC. If it was set to '1' the last byte of the payload contains an information field. In the information field the extended capabilities of the SE (secure element) are encoded. These include an extended slave resume (cf figure 2.3) time, bit duration between 5 µs and 10 µs as well as bit duration below 1 µs to 0,590 µs (cf. figure 2.1).
- The ACT ctrl bits are for sending the ACT commands, size and information of the ACT data field is defined by this command. In the current specification only three commands exist: ACT_READY, ACT_POWER_MODE, ACT_SYNC.

SHDLC frame

The Simplified High-level Data Link Control (SHDLC) is the standard protocol for sending upper-level information. It is responsible for the integrity of the received data. This means it is ensured that all data is error free, it is in the right order and the data is complete after it was processed by the protocol. SHDLC is also designed with a minimum overhead. To ensure the data integrity of the carried information, the CRC is used to check for errors. A 3 bit sequence number is used to determine the data order and completeness. SHDLC uses a window size between 2 and 4 frames it is negotiated during the link setup. To manage the control and transfer of data three frame types with different tasks exist.

- **I-Frame (Information Frame):** I-frames are designed to carry the information of upper-layer protocols. They are the only frames carrying send sequence numbers. The receive sequence number is also included, this way it is possible to acknowledge a received I-frame in the return frame without an extra frame for acknowledgement.
- **S-frame (Supervisory Frame**): S-frames carry commands and responses for the flow control of the LLC. They are also used to acknowledge I-frames because they carry a field for a receive sequence number. There are 4 types of S-frames. **RR** Receive Ready is usually used to acknowledge received I-frames. **RNR** Receive Not Ready indicates to the other side that the receive-buffer is full and further frames can currently not be processed. **REJ** Reject is to reject a frame if it was not transmitted correctly or was lost. All frames beginning with the rejected frame have to be resend. An optional **SREJ** Selective Reject that can be implemented. It differs from the Reject, as only the indicated frame has to be resend. All S-frames are not supposed to carry an information field.
- **U-frame (Unnumbered Frame):** U-frames are used to control the data link. They carry a 5 bit modifier allowing for 32 possible commands. In SHDLC only two commands are used, **RSET** for the reset of the link and **UA** to acknowledge a received RSET. With a reset of the link also the window size is negotiated between host and slave.

2.2 CRC calculation

A cyclic redundancy check (CRC) is a technique to detect bit errors in a block of data, a correction of these errors is not possible. It is generally used in sequential transmitted data, where the check value is added to the end of the message. CRC is commonly used because it can easily be implemented in hardware and it is very good in detecting multiple bit flips due to

noise errors that originate from outside sources. Many different types of CRC-algorithms exist but they all work on the same basic principal. To calculate the check value the message can be seen as one big number, bits can also be added at the end and the begin of the message. This number is than divided by a defined generator polynomial. The remainder of this division is than divided over and over again, the quotient is not needed and discarded during this process. The last remainder of this process is than added to the original message as the check value. This process can be done during the transmission of a sequential data stream. After the message is fully received the CRC is also finished, it can then be processed without further delay. [RE08, pp. 133-135]
The DWP uses a 16 bit CRC defined in ISO/IEC 13239 commonly known as "CRC-16/CCITT-False", it uses the polynomial:

$$X^{16} + X^{12} + X^{5} + 1$$

2.3 Target-system options

To implement the protocol a Field Programmable Gate Array would be the obvious solution. But for extending the existing SaReader other factors have to be considered too. The main goal is to try to implement the protocol on the available microcontroller. In the following section the differences and advantages of the two options are discussed.

2.3.1 FPGA

The advantage of a Field Programmable Gate Array (FPGA) is a very precise timing. The FPGA would handle the communication while the µC handles the protocol, the µC is then free for other tasks. On the FPGA several steps can be done in parallel during the one bit cycle to save processing time. The S1 signal can be generated with a counter match combination just like in a PWM module. The detection of S2 is done during the high phase of S1, it can then be written to a shift register. For the detection of SOF/EOF the content of this register will just be compared to the SOF/EOF values. A benefit of the FPGA solution is, it is also possible to check the CRC during the communication to relieve the µC further. A serial interface like a I²C or an other interface included in the microcontroller peripherals can be used for communication between µC and FPGA. In the FPGA a state machine controller would handle the communication with the device and the microcontroller.
The disadvantage with this solution is that only readers with a FPGA extension would be able

to use the protocol. Extending all existing SaReader's would be costly. Also the development of a FPGA based implementation is more complex.
With this disadvantages a FPGA based solution is only practical in this use case, if a microcontroller solution is not possible.

2.3.2 Microcontroller

The preferred solution is the implementation on the available microcontroller LPC1769, an ARM Cortex-M3. It is mounted on a small development board, the LPC1769 LPCXpresso board. The board was designed and produced by Embedded Artists. There are two distinct areas on the board. The first section is the target board with the microcontroller, the other is the program board with an included JTAG debugger. The board can be separated into these two parts to save space if the debugger and programming board is not needed. It can be reconnected by use of a pin connector. The microcontroller has many common peripherals, these include GPIO, UARTs and PWM, the µC provides also a Watchdog Timer and a USB 2 Interface. The clock has a maximum frequency of 120 MHz.[NXP16]
The use of an existing setup avoids huge redesigns of the existing interface board and no extra hardware is needed. This also prevents multiple versions with different features and is thereby cheaper and easier to maintain.
The downside of this solution is the timing of the transmission. All the signal processing has to be done in the period of one transmitted bit. The consequence is that it is more difficult to achieve the fastest specified transmitting speed (cf. 2.1). To generate a signal with the micro controller there are two alternatives, the first is to use a PWM signal, this guarantees a precipice timing for each transmitted bit. The other option is to generate the output with the standard GPIO pins and inserting assembler code to instruct the CPU to idle and thereby generating a correct timing. This solution includes many measurements, as the timing for S1 needs to be adjusted manually in the code itself. Later extensions are not as easily implemented, as the timing measurements and adjustments have to be repeated, to adjust for the timing changes in the code.

2.4 Platform

2.4.1 Spectro TP

The Spectro TP is a validated conformance platform for testing potocols on smart cards. It has predefined test cases for various protocols including SWP. According to the specification it is

also possible to emulate a smart card with SWP [Com]. The benefit of using this system are the validated test cases, as they don't need to be implemented and verified again. The SaReader DWP extension could be tested with these, for the translation from DWP to SWP an existing interface board exists.
Further analyses and an information request to the support have revealed that it's not possible to use the Spectro TP for this kind of tests. As a consequence an own signal generator has to be developed. The generator is based on the same system as the reader, to reduce the development effort.[Com16]

2.5 FreeRTOS

On the micro controller a FreeRTOS (Free Real-Time Operating System) is running. The operating system supports multiple threads or tasks, mutexes and semaphores for thread synchronization, as well as software timers. The kernel schedules all running tasks, each task is also given a priority for more efficient scheduling. Each task can also suspend itself to wait for resources or delay itself, during this time other tasks can use the processing time. Tasks are switched by priority and a round-robin scheduling, for the interval a hardware timer is used.[Ltd16]
FreeRTOS was chosen, because this system is supported on the chosen platform. A benefit is also, it is simple to use and also slim, this is especially important, as the debug limit of LPCXpresso studio is only 256k (License Free Edition). FreeRTOS is distributed under a modified GPL, the modified license permits to use FreeRTOS without the bound to open source applications, this benefits commercial use.
On the current implementation version 6 of the RTOS is used. This version has not implemented the software timers yet so an update to a newer version will be mandatory, if they shall be used to generate the timeouts needed for the Dual Wire Protocol. As an alternative also hardware timers can be used.

3 Analyses of the current micro controller

3.1 Timer and PWM

To generate the S1 signal a PWM with capture and compare can be used. The PWM signal rate is equivalent with the transmission frequency of the S1 signal. For the bit transmission a match register will have to be adjusted for every bit change. The first major question that has to be answered with this solution is, whether it is possible to adjust the pulse width for every bit by also maintaining the correct timing. The second problem that has to be solved is the analysis of the S2 signal during the high period of the PWM. The issue with the generation of the S1 signal may be solved by using an ISR, the microcontroller supports loading the new PWM match values with the PWM reset. For this shadow registers are used and the new values are loaded when a match 0 accrues (cf. PWM1LER and PWM1MCR registers [NXP16, pp. 528,532]. This option is useful, because it helps to avoid accidentally setting a new match register value and thereby accidentally changing a bit in the process.
For the PWM solution the speed for entering the ISR is essential, as the S2 signal needs to be read in the ISR, if it is not fast enough the S2 signal may be read during the invalid state (cf. figure 2.2). Using the PWM ISR also has its benefits, there will be only insignificant jitter and the timing of the signal is thereby very precise. To measure the timings a PWM-module is configured to generate an interrupt when the PWM resets at the beginning of each period (cf. PWM1MCR-register [NXP16, p. 528]; listing 3.1 line 14).
To analyze the response time and code duration of the different options, test code was implemented. A PWM was used to generate an interrupt. In figure 3.1 it can be seen, that the ISR is ready after approximately 400 ns, so with 100 MHz about 40 cycles (1 cycle $\widehat{=}$ 10 ns). This also includes the clearing of the interrupt. Setting of a GPIO pin is finished after 450 ns. The read instruction for the S2 pin should lay in the first $\frac{1}{8}$ of the period to always be in the high state of the PWM output. The calculated minimal period T would be 8*450 ns = 3.6 µs with a max T of 5 µs (cf. table 2.1) it is within the specification limit.
Further analyses have shown that the delay is not caused by the ISR itself but with the communication of the PWM module and CPU. The CPU needs 120 ns to enter the ISR as specified for

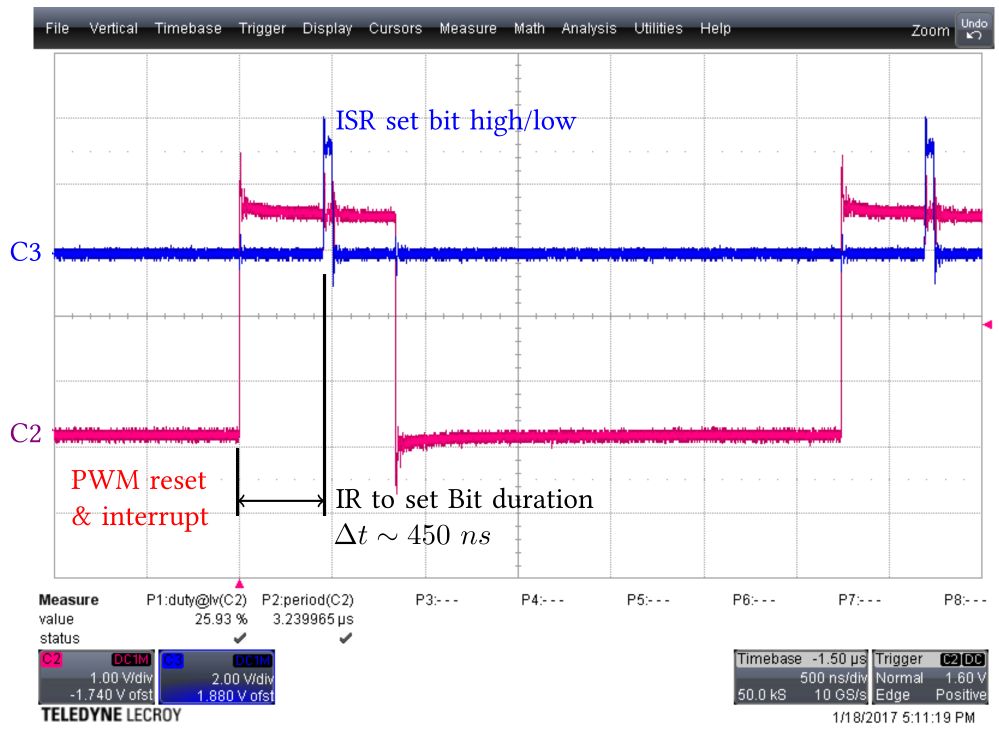


Figure 3.1: PWM-Timing measurement of entering the ISR; C2(red): PWM, IR generated on reset; C3(blue): GPIO pin output for measurement

the Cortex M3 [ARM]. The additional delay in figure 3.1 is generated by clearing the interrupt in the PWM module. The communication takes about 300 ns, this time also applies to the setting of a new match register value. Conclusively minimal PWM-ISR needs at least 120 ns to enter, 2*300 ns for clearing the interrupt and setting a new match value and additional 100 ns to exit, in total 820 ns, this doesn't include any signal processing.
The advantages of this solution is, there are no wait cycles needed so other processes can run in the background during the communication after the processing of the bit has finished. The timing is very precise because the PWM signal is independent from the signal processing. The drawback is that the communication with the PWM module is slow.

```c
#include "lpc17xx_pwm.h"
#include "lpc17xx_gpio.h"

#define S2_PIN    (1 << 25)
#define PWM_WAIT4 40                    // 1/4 T for logical 0
#define PWM_WAIT3_4 (PWM_WAIT4*3)       // 3/4 T for logical 1
```

```c

void init_PWM() {
    GPIO_SetDir(1, S2_PIN, 1);           // set GPIO for IRQ measure
    LPC_PINCON->PINSEL3 = (2 << 20);     // Bits 21:20 to '10'
                                         // for PWM1.6 out
    LPC_PWM1->TCR = (1 << 0) | (1 << 2);// Counter enable; PWM enable
    LPC_PWM1->PR = 0x0;                  // No Prescalar
    LPC_PWM1->MCR = (1 << 0) | (1 << 1);// IR on match R0;
                                         // Reset on match R0
    LPC_PWM1->MR0 = (PWM_WAIT4 << 2);    // 4*PMW_WAIT for full cycle
    LPC_PWM1->MR6 = PWM_WAIT4;           // Set MR6 to 1/4 T PWM
    LPC_PWM1->LER = 1 << 6;              // enable load new value
                                         // on reset for MR
    LPC_PWM1->PCR = (1 << 14);           // enable PWM output
    NVIC_EnableIRQ(PWM1_IRQn);           // enable IRQ in NVIC
}

void PWM1_IRQHandler(void) {
    PWM_ClearIntPending(LPC_PWM1,PWM_INTSTAT_MR0); //CLR IRQ
    LPC_PWM1->IR = 0xFF & PWM_IR_BITMASK;   //CLR IR register
    LPC_GPIO1->FIOSET = S2_PIN;             //Set GPIO Pin
    LPC_GPIO1->FIOCLR = S2_PIN;             //CLR GPIO Pin
}
```

Listing 3.1: Code for first test measurements of the timing of the PWM ISR and to test different PWM settings.

3.2 GPIO

The second approach is to use the GPIO pins with static code and idle instructions to generate the delays between the state changes of the S1 signal. In this configuration it is important to always have the same timing for every possible path in the code. Also clock speed is essential for this solution to work, it has to be fast enough to do all the computing in one bit cycle. With the clock running at max frequency of 120 MHz and a max signal period of 5 µs there is a maximum of 600 clock cycles per period. With the current clock setting of 100 MHz only 500 cycles for instructions remain, a change of the clock is not preferred, as existing code is balanced to 100 MHz. The timing is crucial for the transmitting to work so all interrupts have to be disabled during the transmission process. A first implementation to measure the timing the sequence in figure 3.2 was used. Multiple code duration measurements on an oscilloscope

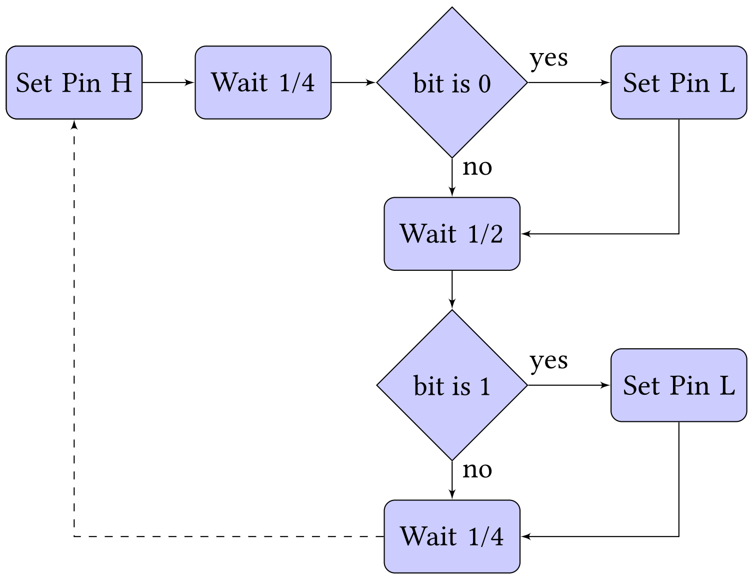


Figure 3.2: Program flow for testing first GPIO output with wait cycles

show that it is probably possible to achieve the timing constraints with this solution. In this rudimentary function the jitter between the different bits was quite low. The measurement in figure 3.3 shows that there is potential to save processing time in the use of direct register writes (line 1& 2) instead of the given functions (line 3& 4). A more accurate analysis of the used bitSet-Functions show, that the flexible code for setting different pins in this function is not needed and can thereby be discarded to make the code as slim as possible. The increased duration is a result of different factors, the GPIO port is selected with a switch case, the function call also takes processing time, as variables need to be stored on the stack.

```
1 LPC_GPIO1->FIOSET = S2_PIN;
2 LPC_GPIO1->FIOCLR = S2_PIN;
3 GPIO_SetValue(1, S2_PIN);
4 GPIO_ClearValue(1, S2_PIN);
```

Listing 3.2: Code for generating wave in figure 3.3

The benefit of this solution is that almost only the processing time for the signals is needed. Besides the fast GPIO-pins no slow peripherals are needed, a higher transmission frequency is

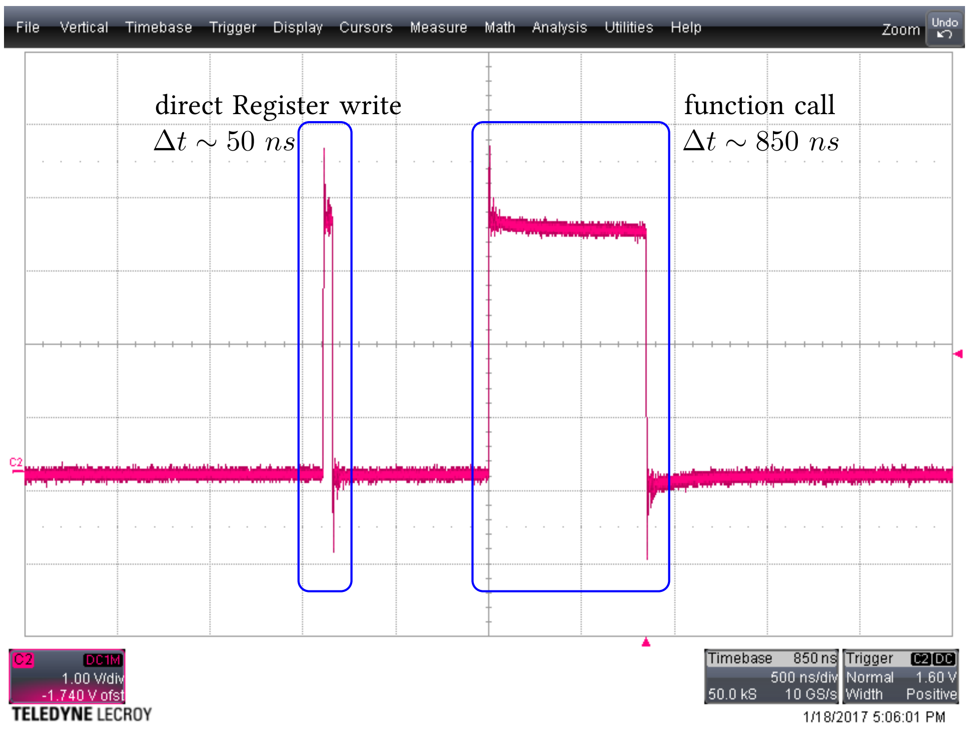


Figure 3.3: Timing compare of setting GPIO pins with register and dynamic set function from the GPIO-library

possible. The downsides are the CPU has to idle to generate the signal. A second disadvantage is, the timing has to be remeasured and adjusted for every code change.

3.2.1 C / Assembler

To understand the timing of the code it is helpful to have the C-code disassembly. It benefits not only with the timing but also to check whether the compiler included the NOP's (No OPeration) correctly which are needed later. The disassembly of the C-code, with the compiler option -o0 (no Optimization) for example only uses registers R0 to R3 and R7, other registers are not used. By use of -o3 (most optimization) all of the Low Registers are used.
In figure 3.3 it takes 50 ns to set a GPIO pin, this translates into 3 assembler instructions. When adding up the cycles for each instruction *LDR* (2 cycles), *MOV* (1 cycle) and *STR* (2 cycles) it corresponds exactly to the measured time[ARM15].

```
LPC_GPIO1->FIOSET = S2_PIN;
```

Listing 3.3: C-Code

```
35e:    4b10         ldr     r3, [pc, #64]   ; (3a0 <33c+0x64>)
360:    f04f 7200    mov.w   r2, #33554432   ; 0x2000000
364:    619a         str     r2, [r3, #24]
...
3a0:    2009c020     .word   0x2009c020 //adr. GPIO
```

Listing 3.4: Disassemble of the C-Code section

3.2.2 delay compensation

As the timing is significant for the implementation, every path of the GPIO solution's communication method has to be almost identical in length, the proportion between the high and the low time in a bit has to stay within the specification limits (cf. table 2.1). To achieve this, special attention has to be given to IF-statements and other instructions that can have different timings depending on the input. For example as can be seen in figure 3.4 for an IF-statement, the Yes-path would be longer than the No-path, to keep the timing an idle time has to be included in the shorter No-path. Different bit duration's are not a significant problem, if the duty-cycle and the duration are within the specification limit (cf. table 2.1).

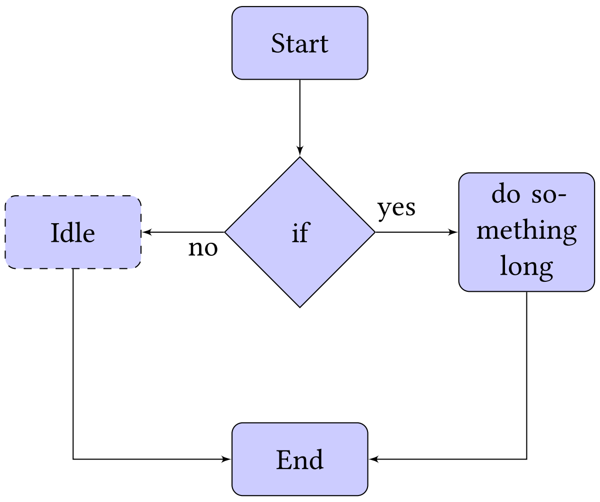


Figure 3.4: Program flow for delay compensation in different length timing paths in an IF-statement

4 Architectural Design

4.1 System structure

To communicate with a device the SaReader system is split in two parts, as the reader will have to communicate with the secure element, as well as a PC if necessary at the same time. The connected PC can set the configuration, for example the selection of the protocol, on the reader. For this communication side a driver on the PC is required. All the standard DWP communication with the DUT will later be handled by the micro controller. For this a state machine will analyze the received frames from the DUT and respond accordingly. Hereby the CRC has to be checked and the frame type will be determined to be processed by the right handler for the frame type. SHDLC I-Frames can carry upper-layer information, which might not be handled by the SaReader but be passed on to the PC to be processed there.

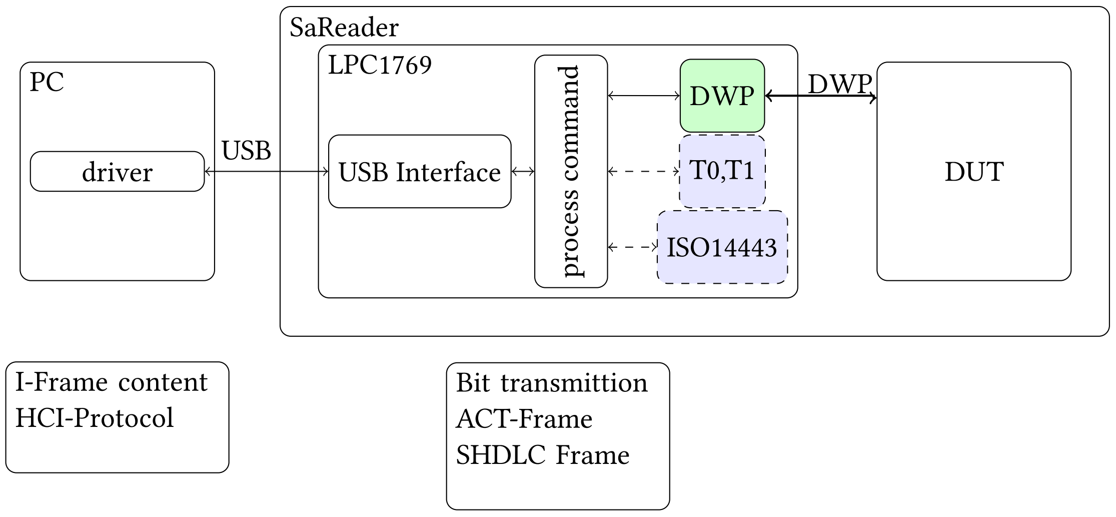


Figure 4.1: System structure with the designated functions. The SaReader board includes a socked for the LPC1769 and a DUT (cf. section 9.1). On the LPC the needed segments for the communication can be seen.

On the SaReader itself an operating system (FreeRTOS) handles the scheduling of different tasks, and the loading of different communication protocols modules. The DWP communication will be one of the modules that can be selected. The DWP module will consist of two main parts (cf. figure 4.2). The state machines (blue) will handle the protocol and a communication part (green) that handles the hardware interaction for the DWP. These 2 parts will further

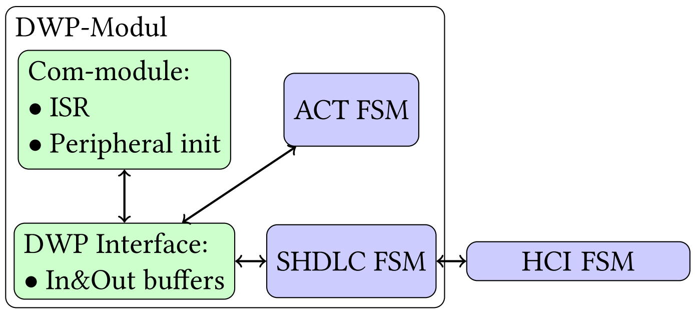


Figure 4.2: Communication structure of the DWP module in the reader; green: communication section; blue (FSM): state machines

be split to allow easier future extensions and testing. The communication section is split into a com-module contains all code that is used for the signal generation with the micro controller peripherals. This includes the necessary ISRs and also the initialization of all required components. In between the com-module and the state machines an application programming interface is included. It provides a buffer for incoming and outgoing frames and code for the CRC calculation. The benefit of this split is, that the com-module can be tested stand alone. Another benefit is, the com-module can be replacement to send test data to the state machine directly. This way the state machines can be tested without the use of the micro controller hardware.
There will also be multiple state machines, each state machine has a specific task. Here the split is done to keep the complexity as low as possible, but also to be able to easily add more protocols later without having to change and test the one big state machines again. For a basic communication there will be two state machines, one for each LLC (cf. 2.1.4). All high level protocols like HCI or other protocols are handled over SHDLC, they will also use own state machines. The ACT state machine is used for the activation of DWP and handles the first frames. If the activation is finished, the state machine for SHDLC initiates its protocol and handles all further communication with the DUT. The content of the I-frames will not be processed, but handed over to the protocol handler. These state machines are not required to run directly on the reader itself but can run on a PC instead.

4.2 Program flow for the communication

The communication is split into two flows that have to cycle through for each transmitted or received bit. The first cycle is the "timing" cycle to send a bit to the slave and generate a valid S1 signal (figure 4.3), the other is the "processing" cycle for the received bit from the slave (figure 4.4).
For the timing cycle(cf. figure 4.3) to work, it may not be interrupted. The cycle has to either run in an ISR with high priority or all ISRs have to be turned off during the transmission.

a: In the "timing" loop the first task is to set S1 to high state for at least $\frac{1}{4}T$. During this period the state on S2 can be read and be written into a buffer for later processing.

After this time it has to be determined whether a logical '0' or '1' has to be send this also includes stuffing bits after five consecutive '1'.

b,d: If a logical '0' is sent S1 is set to low state and a wait cycle with $\frac{1}{2}T$ is entered.

c: For a logical '1' the wait cycle will be entered first and then the signal will be set to low.

If a Bit was sent in path **c** or **d** the next bit for transmitting is set. To complete the sequence a $\frac{1}{4}T$ low state is added at the end. The CPU doesn't have to idle during all the wait cycles, it makes sense to use the longest cycle for processing the received bit. With this solution the minimum duration of each cycle is also dependent on the processing time for S2. It is either two times the duration for S2 or the duration of S2 and S1 combined.
If a PWM is used for signal generation the flow is almost identical. The difference is, there is no need for the wait cycles and instead of setting a GPIO pin to low a new match value is loaded. The PWM solution also sets the timing value for the next bit instead of the current bit as in the GPIO solution.
When analyzing the S2 input (cf. figure 4.4), first it has to be determent if data is transmitted. For this the Start Of Frame (SOF) sequence has to be detected. If a SOF is detected, the input is set true and a counter to -8 to dispose the first 8 bits containing the SOF. After these eight cycles bit[8] contains the first bit of the payload. The input check is included to save time in the longer cycles to skip the SOF check again after detection. If the input is true the bits[7-0] will be checked for the End Of Frame (EOF) sequence, if found the input will be set to false again. After an EOF detection the last bit of the message has not been processed yet, the processing cycle will be run through a last time to include the last bit of the message.
As mentioned above the first 8 Bits will not be stored as they contain the SOF. After this initialization bit 8 is checked each cycle to update the stuffing counter, if the five consecutive

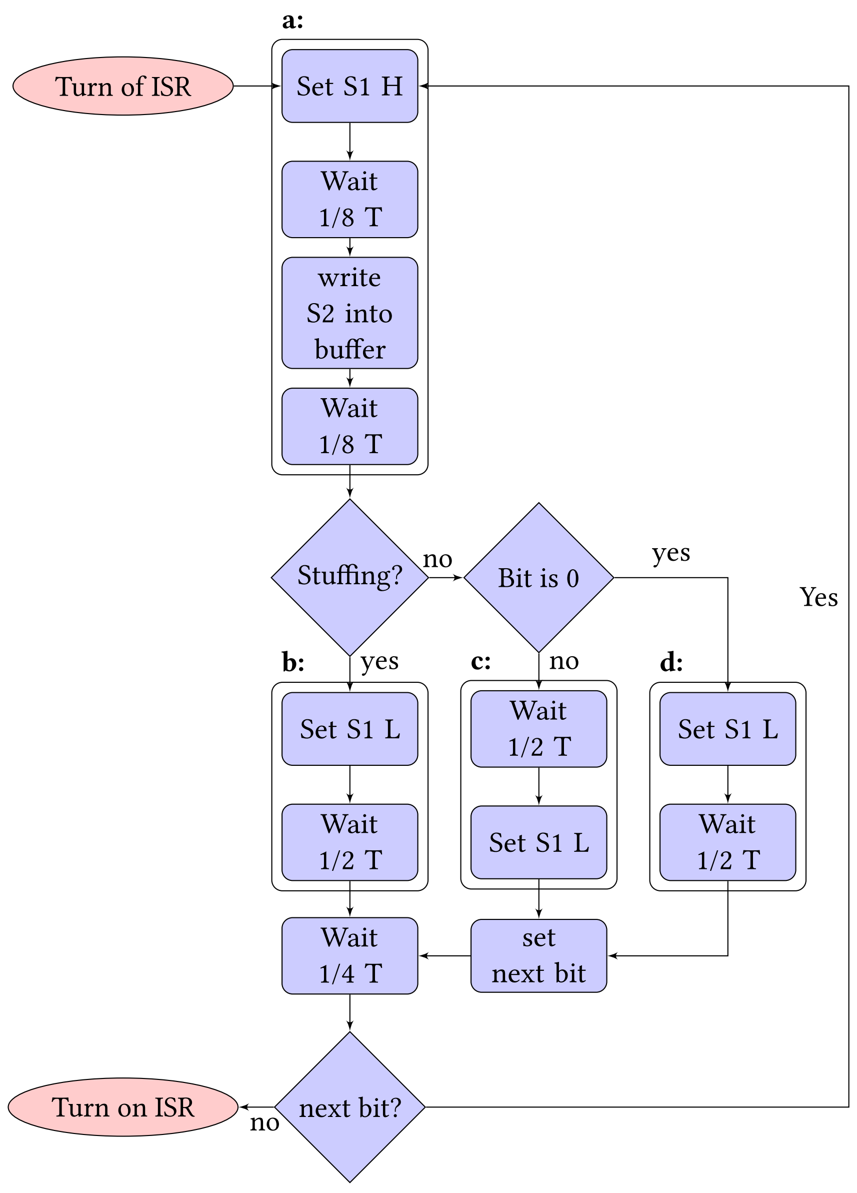


Figure 4.3: Basic program flow for generating one bit output on S1 (cf. figure 2.1) with GPIO pins. The flow is implemented in the Com-module in figure 4.2.

Uint16_t Buffer	Bits 15-9	Bit 8	Bits 7-0
S2_buffer	unused	process Bit	SOF EOF detection

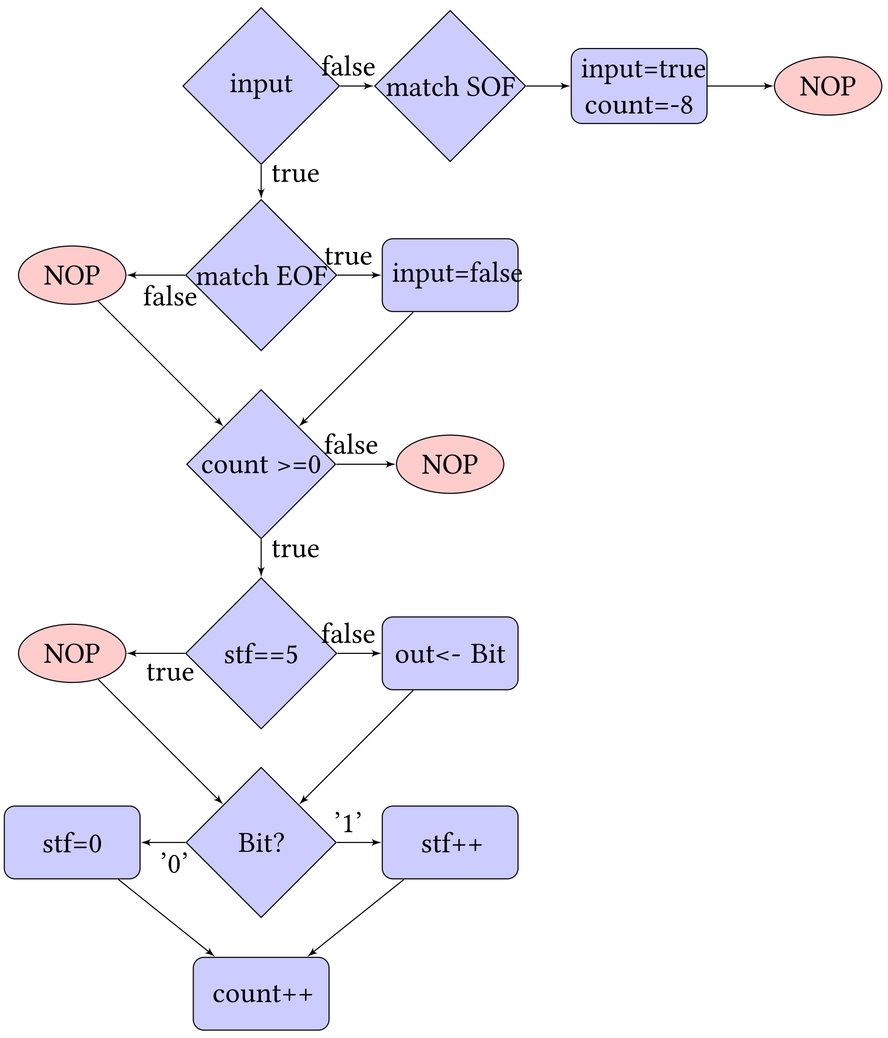


Figure 4.4: S2 input buffer and program flow for processing one bit of S2 (cf. section 2.1.3)

'1' are detected a stuffing bit was inserted and this bit will not be stored. If a '0' is received the counter is reset to 0, this also applies to the stuffed bit as well, the inserted '0' will reset the counter. When the bit isn't a stuffing bit it will be written to the output buffer. To ignore the first 8 bits the initialization counter is incremented by one.
The S2 signal flow for the PWM and the GPIO concept has no differences in the program flow, with the exception that also no delay compensation is needed and is thereby not included.

4.3 State machine design

For the DWP protocol handling multiple state machines will handle the communication. For a basic DWP implementation only the two state machines shown in figure 4.2 are needed. The HCI state machine is not part of the DWP implementation, as it is a different protocol. As an extension a CTL state machine could also be included.

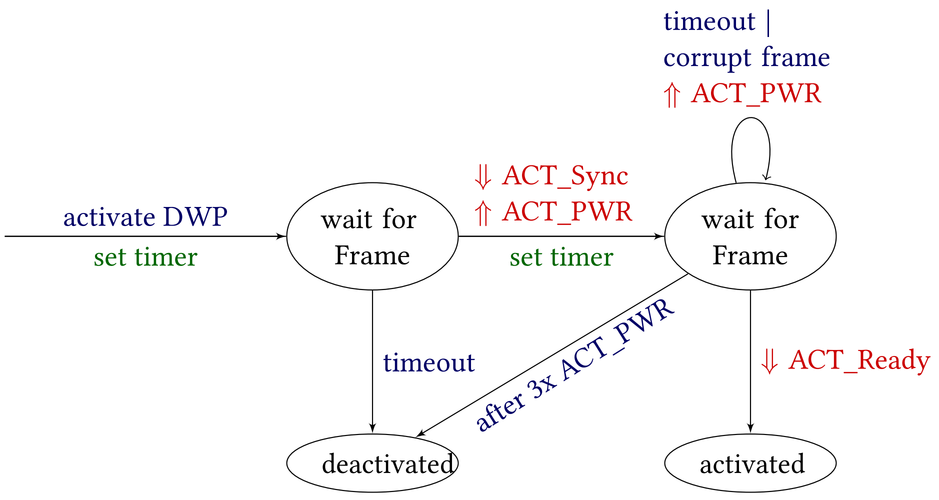


Figure 4.5: ACT state machine for processing the activation sequence of the DWP (cf. figure 4.2) blue: internal event; green: action; red: communication

The state machine for activating the DWP interface is the first to be executed. On initialization it resets all other state machines and sets all basic setting that are needed for communication. Last it powers up the DUT and set the S1 pin to high. It then waits for the first frame sent by the DUT. If a correct ACT-Sync frame is received the reader will reply with an ACT-Frame containing the power mode. With a last ACT-Ready frame from the DUT, the DWP link is established and the state machine is in the activated state. This is a typical three-way handshake for a connection between two nodes. During this process also information about the capability of both devices is exchanged. After the activation the fist frame is always send from the host to the slave. This way it is possible to establish a connection over the CLT LLC if it is needed. Is this not the case and CTL is not needed a SHDLC link is established.

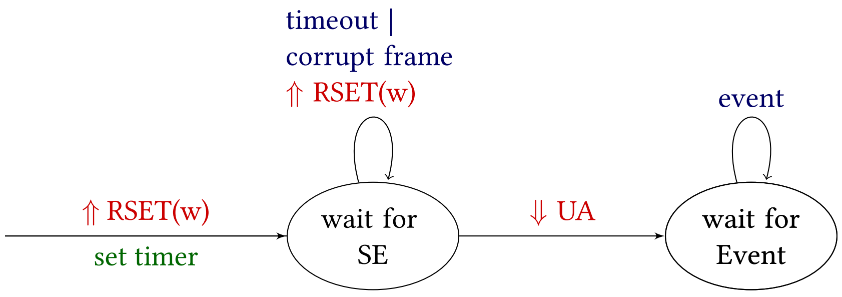


Figure 4.6: SHDLC state machine for activation of the SHDLC protocol and processing incoming frames during activation (cf. figure 4.2); blue: internal event; green: action; red: communication (w = window size)

The state machine for the SHDLC LLC is very simple it has only 2 states for setting up the link and determining the window size. After the link setup the state machine will only wait for events to respond to.
The following events are specified (cf. section 2.1.4 SHDLC) and can occur:

- send I-frame
- receive I-frame
- receive Ready-frame
- receive Reject-frame
- acknowledge timeout
- transmit timeout

The rejection of a selective frame will not be supported at this point.

4.4 Test-pattern generator design

For generating valid signals on the input (S2) of the com-module a second module for the SaReader was implemented cf. figure 4.7. The functionality is limited to sending messages, as the received output of the reader can't simply be interpreted with a micro controller and an

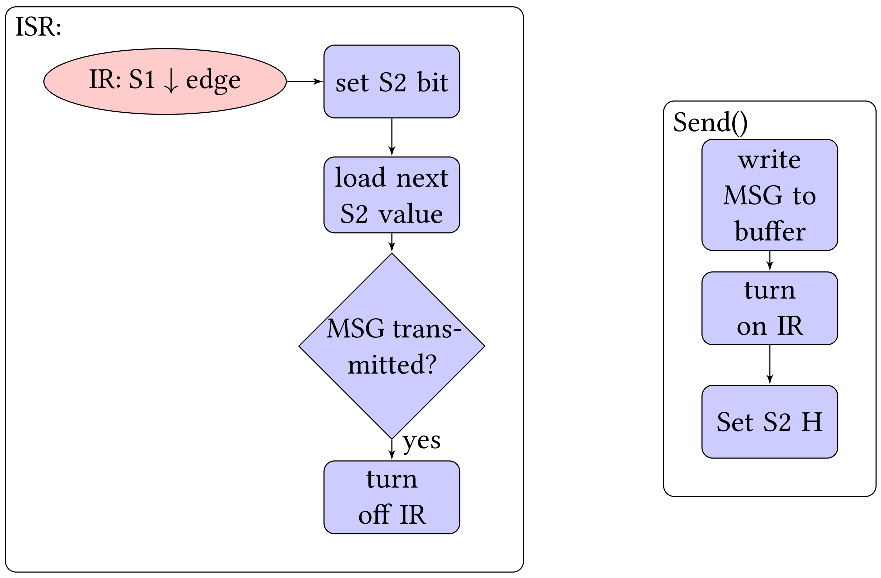


Figure 4.7: Program flow for S2 generator; left: ISR to synchronize the set value with the S1 signal from the reader; right: function to activate the message sending procedure

oscilloscope is more suitable for this job. An oscilloscope has also the advantage that the signal structure can be analyzed and checked for compliance of the specification, a micro-controller is not precise enough for this. The messages that are send can be provided by a PC. Thereby for the full emulation of a secure element all three components are needed, the generator for the emulated output, the oscilloscope for the input and a PC to operate both.
The generator simulates a secure element which requests to send a message to the reader. As the DWP may be in the suspended state it raises S2 to make a resume request (cf. figure 2.3). To synchronize the S2 signal with the PWM provided by the reader an interrupt on the falling Edge of S1 can be used. This ensures the state change of the S2 signal is in the specified time frame (cf. figure 2.2).

5 Realization

5.1 Programming languages

For the project different programming languages are used, depending on the purpose and the platform the code is running. The code for LPCXpresso board has to be written in C, as the current source code for the SaReader is also in C. The developed algorithms (cf. section 4.2) for encoding and decoding the signals are partly tested first in Java for functionality, tools like JUnit are helpful during this testing process. The reason for the use of Java is that no extra hardware is needed for developing and testing the algorithms, especially during the debugging process this is a benefit. The algorithms are then translated to C for the micro controller.
Instead of using integer variables in Java for the buffer, strings are used. The reason for using strings is, while debugging the content of the buffer does not have to be translated from integer to a bit-string, the buffer content can simply be read. The string class provides all required functions that can easily be used. It is also simple to change a specific single bit while debugging without calculating a new integer value. A last but significant advantage of strings is, that it is possible to insert other chars as placeholders for bits like a 's' for a stuffing bit, this makes manual analyses easier and helps to solve potential bugs.
For the analyses of S1 and automated measurements, it is required to control the oscilloscope from a PC (cf. 5.2.4). A Java class for the communication via TCP/IP existed, and is already used in other NXP projects. It is also possible to communicate with the SaReader OS from Java, a package for the SaReader also exists. The use of the same language seems obvious, as both functionalities can be combined if needed. The combination of these with other functionalities of Java, like JUnit this test environment can be extended to a system for simulating and testing a secure element and the DWP module.

5.2 Code implementation

To compare the considered DWP solutions, GPIO and PWM, a basic functionality for the com-module(cf. figure 4.2) was implemented. As both are based on the same algorithms (cf. section 4.2), they will mostly differ in the use of different hardware components, this also

includes the use GPIO-pins and the PWM but also the delay compensation for the CPU. These differences will be looked at in more detail below. The Test code for the GPIO solution is in the Appendix section 9.4.1. The final code with the PWM solution can also be found in the Appendix section 9.4.2.
The function of the S1 waveform analyses software is included in this section.

5.2.1 GPIO solution

To generate the S1 PWM-signal with GPIO-pins it is important to have an interrupt free environment, otherwise CPU generated timing will be distorted when the CPU switches to handle the interrupt. The interrupt free environment can be achieved in two different ways, the code can be executed in an ISR with the highest user priority itself or it has to be run in a so called "critical section" where all interrupts are disabled. To implement first test-code this was not necessary as there are no other applications running in parallel on the board.
For the first basic tests both program flows in figure 4.3 and 4.4 where implemented (Code extraction for S1 cf. section 9.4.1). For the first tests to generate an input signal for S2 with only one board, a debug-pin (designated GPIO-pin) was used and connected to the S2 input. This debug pin was than adjusted at the end of the signal generating loop. The benefit of this integration of the input signal is, it always changes at the end of the bit-cycle, this synchronization method is simple and effective for this use case. The disadvantage of this test solution is, it can only be used during the first test phases, as it alters the timing characteristics of the entire system, by adding additional processing time for setting the GPIO pin to the end of the function.
To generate the timing compensation that is needed for the GPIO solution a macro with an assembler No OPeration (NOP) command is used, for better code readability also a macro with multiple NOPs was also used.

```c
#define NOP1 asm("NOP");
#define NOP5 asm("NOP");asm("NOP");asm("NOP");asm("NOP");asm("NOP");
```

The use of delay compensation has the effect on the two flows, that every IF-statement also has an else path for the delay compensation (cf. 3.2.2). For visualization these NOP's where included into the flow in figure 4.4, as these are not needed for the PWM also the IF-path can differ from the GPIO.

5.2.2 PWM solution

The PWM is updated in an Interrupt Service Routine, the interrupt is generated by the PWM module with the reset of the counter (cf. section 3.1). As the output signal is generated by the PWM peripheral the timing is not as critical as in the GPIO solution. The only timing constraint is, to read the S2 signal in time. As it has to be read-in during the high state of S1, it has to be in the beginning of the ISR. As the ISR is periodically executed (PWM period) it is effectively a loop that has to be broken when the communication is finished. This is the case, if no more bits have to be received or transmitted.(line 2 & 8) Also communication errors on S2 have to be considered (line 9), a simple bit flip in a received EOF could result in an infinite loop, the reader would interpret the faulty EOF as data and wait for a correct EOF sequence, which cannot be received. If this error handling is not included this could potentially result in a undesired buffer overflow. The loop in the PWM solution is stopped by disabling the interrupt (line 10). During the shutdown sequence the MR1-register is set to a value higher than MR0-register (line 11) for a continuous high signal. To detect a slave resume request the interrupt on the S2 pin is turned on again (line 16).

```
1  //Break ISR Cycle if Communication is done
2  if (((S1_position) - (out_Frame->size_byte)) > 1) {
3     if (DWP_Bufferoutsize()) { // check for frame in out buffer;
4        out_Frame = DWP_getoutFrame();
5        S1_bufferout = 0;
6        S1_position = -2;
7        S1_buffermask = 0x01; // at least 1 Idle bit needed
8     } else if ((((S2_inbuffer & 0xFF) == 0) && (S2_input == 0))
9           || (S2_position > 32)) { //nothing to send
10       LPC_PWM1->MCR = (0 << 0) | (1 << 1); // IR disabled
11       LPC_PWM1->MR1 = (PWM_WAIT4 << 3); //MR1 disabled out of range
12       LPC_PWM1->LER = (1 << 1);
13       // -->Reseting all needed global variables not included here<--
14       g.rdr.wdtFeedUpdate=TRUE; //enable WDT
15       LPC_GPIOINT->IO0IntClr = S2_PIN;  // CLR potential IR
16       LPC_GPIOINT->IO0IntEnR |= S2_PIN; // SET IR for slave request
17    }
18 }
```

Listing 5.1: Code section where the loop is stopped.

5.2.3 DWP interface

The DWP interface (cf. figure 4.2; code cf. section 9.4.3) are two FIFO buffers, one for the input and one for the output. They are designed to have a minimal timing delay on the timing critical com-module side, as these are executed in the ISR. All time consuming activities, like array copy, are on the state machine side. As the number of buffer elements is written to by the ISR and the FSM-tread it is needs to be protected by a critical section. In the interface are also included the functions for appending and checking the CRC.
When a SOF is detected in the com-module, it gets a pointer from to the next empty in-buffer frame, the received data is then written to this frame. After detection of an EOF only the buffer counter is increased and a software interrupt is generated, the interrupt is needed for the time consuming notification of the state machines(cf. section 7.2).

5.2.4 DWP waveform decoders

For the analyses of a DWP communication sequence two waveform decoders were implemented in Java. The first, more complex one is for the decoding of a waveform captured with a Picoscope (USB powered oscilloscope), it was developed to decode the entire activation sequence between an existing reader and a secure element. The data is first written into a csv-file, this file is then loaded into the decoder. The decoder analyses both, the communication from and to the reader (S1 and S2). The signal is decoded in steps:

1. The file is read-in line by line and split into a string array containing the time and the voltage of S1 and S2. The voltage is translated into a string with "1" or "0" representing the high or low state of the signal. On a state change of S1 or S2 a new set of data containing a time stamp and the S1 and S2 values is added to a list. When there is no state change on the signals the data set is discarded, as it does not contain required information. To determine the duty-cycle of S1, the time difference between two data sets is analyzed.

2. In the second step the data in the list is translated into two strings, one for each signal, of '0' and '1' representing the transmitted binary data. If the time difference between two sets of data exceeds a predetermined value, it indicates that the communication was suspended during this time. The two strings are than written to an output file with an indication for the end of a transmission sequence, for better readability the strings are analyzed for SOF and EOF and converted to Hex values, potential stuffing bits are filtered out during the conversion process.

A section of a decoded output of a communication sequence:

```
-----------Sequence---------
Host:    000000000000000000000000000000000000000000000000000000000000
         00000000000000000
Client: 010101011111100110100110011011001101010000011000111100000101
         101111111000000000000
Host:
Client: Bin: SOF 011010011001101100110101000001100011110000001011 EOF
         Hex: SOF 699b35063c0b EOF
```

The second decoder was implemented to decode the content of a S1-signal with an oscilloscope. It was developed for an easier analyses and to have the capability in the test setup (cf. section 6.1) to read-in the data send to the device. The oscilloscope has to be set up with a specific configuration, as the decoder makes use of its measurement capabilities. For this a WaveScan [LeC] is used to analyze the duty cycle of the PWM. The "WaveScan" feature of a LeCroy oscilloscope is designed to analyze a captured waveform, this makes it possible to measure all displayed PWM-cycles simultaneously, each cycle is written to a table which can be readout by the PC to be converted. The values are processed and can be displayed or further analyzed. The processing in this decoder works with the same algorithm that is also used on the S2 signal of the reader.

5.3 integration into the SaReader firmware

The SaReader firmware running on the LPC1769 is based on FreeRTOS. Multiple threads for different tasks run simultaneously. For the communication with a secure element (SE), different communication modules can be loaded, each module implements a specific protocol. Already included protocols are the contact based ISO 7816 and the contactless ISO 14443. If a module is loaded on the SaReader a new task is created. The basic structure of this task is shown in Figure 5.1.

1. On creation the task first initializes the module, for the DWP-module this includes the input output buffers, the PWM, the ISRs and their priorities, semaphore for the input buffer, and all other needed variables. The output signals and the power for the device under test are not set at this point.

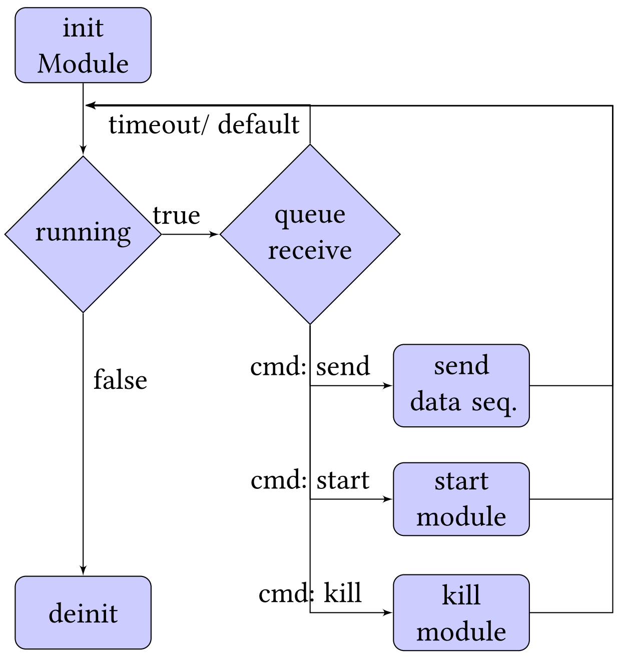


Figure 5.1: Basic program flow of the main tread of each communication module. Command (cmd) and data are included in the queue element

2. After initialization the tasks enters the modules main running loop, this loop can be broken by different termination conditions to exit the module. The xQueueReceive command blocks and waits for user commands.
3. The first command to be sent has to be the activation of the device. With this step the DUT is powered on and all signals are set to their required initial state, for the DWP this is a rising the S1 signal to high.
4. After the activation the user can communicate with the DUT.
5. To deactivate the module the user sends a command to the queue, the local run variable is set for the main loop to be stopped. During the deinitialization the used interrupts have to be disabled, and all used memory needs to be cleaned up.

5.4 test-pattern generator

For generation of test signals for S2 a generator was implemented, it is also based on the LPC1769, the first development version is not based on the RTOS and it can only send pre-programmed messages. This very simple version is used for the first test, as well as sending the same message multiple times without the need of a connection to a PC. The sending of the message is triggered by an external signal, in combination with an function generator, the message is sent periodically. Another benefit is that the timing is always identical, as no other tasks run in the background, this is especially important when balancing the timing of the GPIO solution.
A second, more complex version is based on the SaReader firmware it is integrated in the firmware as an independent module like the DWP-module. This requires the reader board with a USB connection to a PC. The benefit of this integration is, it can also be accessed the same way as the SaReader, including access from different programming languages. For the input of S2 to the reader, the pin which is also connected to the DUT are used. This poses a problem, as the pin is also driven by an optocoupler (cf. figure 5.2). If the two sources try to drive opposite voltage states, it can result in a voltage that cannot clearly be interpreted by reading side. To avoid this interference, the connection between the reader-board and the

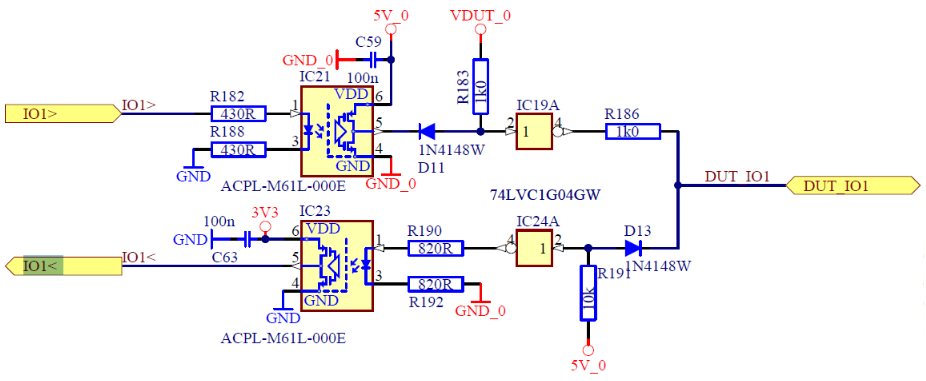


Figure 5.2: Section of the SaReader board circuit: The optocoupler is driving the S2 pin (IO1<) of the LPC board, for connecting the test-pattern generator it needs to be severed.

LPC-board was severed for this pin, the boards are then connected to each other. A small board with jumpers was developed and inserted between the LPC board and the SaReader board. with the jumpers the input for LPC board can be selected.

6 Testing

6.1 Test setup

To test the implementations a basic test setup is needed. Some components like the Com-module have to be tested on hardware. To test the timings and function on hardware, additional equipment is required. The signal S1 has to be interpreted and evaluated to test the output. For testing the input a S2 signal has to be generated. To merely test the output of the com module it is not necessary to have an external source to generating a input signal S2 it can be set to ground and no frames are received. But if both directions are tested, it is necessary to generate a valid S2 frame. It is possible to generate the signal on the DWP board in the ISR, but for an independent testing this is not a viable option, as the code has to be changed after testing. Also the timing differs because of the included code in the ISR. To test independently the setup in figure 6.1 was used. This setup can be used for multiple hardware tests including

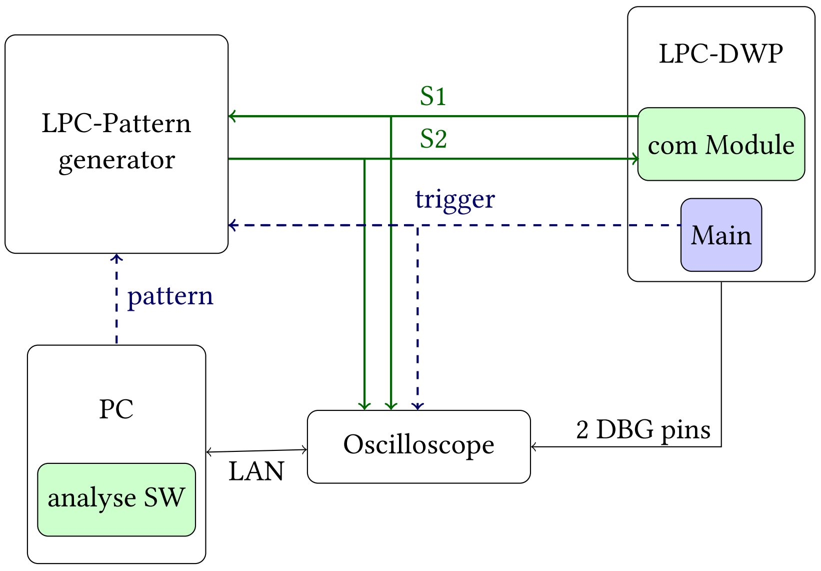


Figure 6.1: Test setup for testing timings and communication. S1 and S2 are recorded on oscilloscope and analyzed by the PC.

timing tests of the output, but also simulating a device. For more flexibility and also to analyze and debug the software on the LPC-DWP board additional GPIO pins are used (DBG pins).
For a simple input and output test the LPC-DWP board generates a trigger signal in its main method. The pattern generator on the second LPC board starts transmitting a predefined sequence. After the message is received by the DWP-board it is sent out on S1 to validate if the message was received correct. The oscilloscope then analyses the waveform for the duration of each PWM-cycle. The determined values can then be fetched by the PC, they are translated to bytes again and then checked if they match the original message that was sent. If only the output is tested, no trigger is sent and only the output function in the DWP module is used. S1 is analyzed the same way as before with the oscilloscope.
Even more complex tests like simulating a device are possible with this setup. To accomplish this, a PC can send a pattern to the generator it is than processed and sent over the DWP connection. A possible response can then be picked up by the oscilloscope and analyzed on the PC. The benefit is, this way it is possible to also partly test the state machines (cf. section 4.3) on real hardware too. Possible problems are the timeouts of the state machines, due to the increased processing and analysis time of the signals in the PC. To avoid this problem the timeout values have to be increased or the timeout has to be turned off.

6.2 Test Pattern

To test the implementation of the communication algorithm, defined in section 4.2, different test patterns are used. With these patterns it is possible to evaluate different sections of the code. Each pattern is designed for a specified task. They can also be used to determine the timing of code sections. All patterns can be used for both the S1 as well as the S2 signal. Each pattern should begin with a SOF and end with EOF otherwise the bit steam may not be picked up by the other side and is ignored. In table 6.1 some test patterns and their use cases are listed. Of cause it is also possible to generate commands that trigger responses, the response can than be checked whether it is valid.

6.3 Timing Analyzes

For the timing analyzes an oscilloscope is the optimal measurement tool. With the correct settings it can analyze and prepare most of the data itself. The measured values can then easily be analyzed. Each measurement should be run multiple times to also measure worst

Description	Hex	transmitted bit stream
default no data sent	0x00	00000000|00...
stuffing test	0xFF	11111(0)111|11(0)11111(0)1|...
Stuffed bit & buffer reload	0x1F	00011111|(0)1...
	0x0F 8	00001111|1(0)...
Toggle patterns	0xAA	10101010|10...
	0x55	01010101|01...

Table 6.1: Selection of possible test patterns

case values in case of jitter or other timing effects. To verify the correct implementation of the protocol the following timings have to be checked.

- Code duration:
  To check the duration of a code fragment a GPIO-pin will be set to high state at the beginning of the block and set to low at the end of this block. As a result a pulse is generated, which can be measured. With the waveform analyses of an oscilloscope it is possible to measure multiple pulses at the same time.

- the timing of S1:
  The timing measurement includes the frequency and the duty cycle of the S1 signal. These values have to be in the limit shown in table 2.1. For this measurement the extended statistics of an oscilloscope can be used. The statistics include the minimum and maximum values of all measurements which both have to be in the specification limit.

- Jitter during read-in of S2 in a ISR:
  As specified S2 has to be read during the high state of S1. This can be a problem with the PWM solution as it uses a ISR. The duration to enter the interrupt service routine can vary depending on other ISRs or critical sections running in parallel.

- Resume by slave time:
  The host has to respond to a request by the slave in a max of $5\mu s$. To resume the communication different steps have to be taken in this time frame, so measuring the max resume time has to be verified.

6.3.1 Problems with optimization in the GPIO solution

With compiler optimization set to -o3 the following problem emerged: The duration of the C-code in listing 6.1 is static so it can be assumed, that the timing should always be identical

for each pass. An exception is the loading of a new value into the buffer. In the table 6.2 an extraction of the timings for sending bytes with the value '0xFF' are listed. Stuffing bits where included but not measured, as they were generated outside of this code section. In this extraction two effects can be pointed out. Further data shows that the default duration Δt for this code fragment is 5 cycles, if a new value is loaded into the buffer a extend duration of 12 cycles is needed. The first effect that can be seen is, the duration for the first pass through is 2 cycles longer then for a normal pass. This effect was also seen in other measurements which where analyzed.
The second more severe effect is, if a stuffing bit was inserted the duration for the next processed bit is 4 CPU-cycles longer. This type of effect can unfortunately not easily be countered in the optimized C-code. A solution would be the use of no optimization, but in this case the PWM solution with optimization has in the worst case almost the identical or even a better transmissions speed. In the best case the ISR with the PWM is faster and the spare CPU time can be used for processes in the background.

```c
if (stf_count < 5) { //stuffing bits not included in measurement
        LPC_GPIO1->FIOSET = S1_PIN; // for loading SET
        LPC_GPIO1->FIOCLR = S1_PIN; // and CLR into register
        NOP5 // WAIT
        LPC_GPIO1->FIOSET = S1_PIN; //start of measurement

        output = (output << 1);         //shift bufffer
        output += bit;                  //write bit to buffer
        buffermask >>= 1;
        if (buffermask == 0) {          //to load new byte to buffer
                buffermask = 0x80;
                s->out_Buffer[position] = output;
                output = 0;
                position++;
        } else { // reserved for NOP's
        }
        LPC_GPIO1->FIOCLR = S1_PIN; // end of measurement
}
```

Listing 6.1: C-Code section of the measured timing.

Bit	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1
Δt	**7**	5	5	5	5	na	**9**	5	12	5	5	na	**9**	5	5	5	5	na	**16**	5

Table 6.2: Number of cycles needed for each iteration through the C-code after a stuffing bit was sent the number of cycles increases.

6.3.2 Slave resume time with PWM solution

To suspend the communication the PWM interrupt is turned off and the second match register value is moved out of range from the reset value. This way a second match event does not occur and the signal stays in high state (cf. figure 6.2). The PWM-MCR-register is set to disable

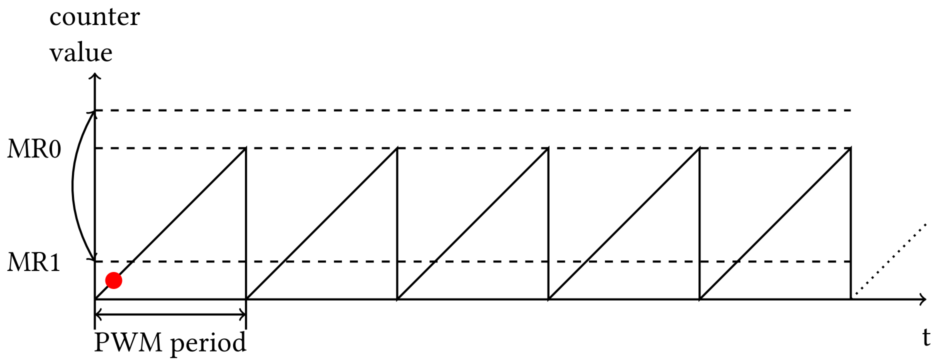


Figure 6.2: To suspend the PWM MR1 is moved above the MR0 value, this results in a continues high signal. To resume the PWM MR1 is moved back and the PWM counter is set to below MR1(red dot).

the interrupt, so the CPU does not enter the ISR and no processing time is lost. When the slave signals the host to resume the communication, the match register and interrupt have to be turned on again. The value of the PWM counter is between reset and max value. Dependent on the counter the first match can occur somewhere in the period of the PWM. This effect does not pose a problem, as long as the max value is within the limit. To visualize the effect a persistent measurement is helpful. The oscilloscope measures multiple resumes and lays them graphically on top of each other. Minimal and maximal resume time can then easily be determined. This measurement can be seen in figure 6.3. If the counter is not set to a specific value the falling edge is somewhere in the time frame of the PWM. If the counter is set to a fixed value, the PWM always resumes at the same position. The analyses of the slave resume time also lead to another design change, in the first code version all variables where reset before the resume of the PWM. This was changed and all variables are now reset when S1 is

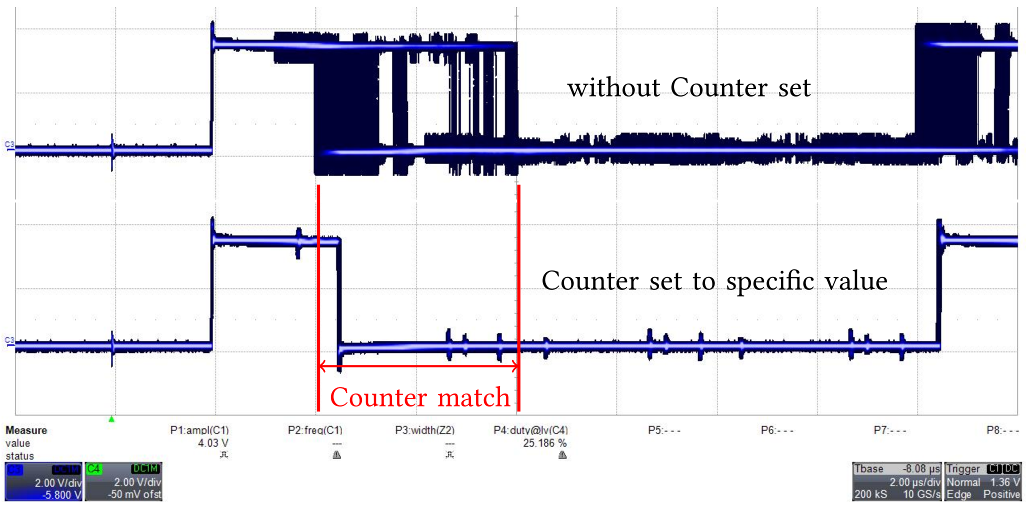
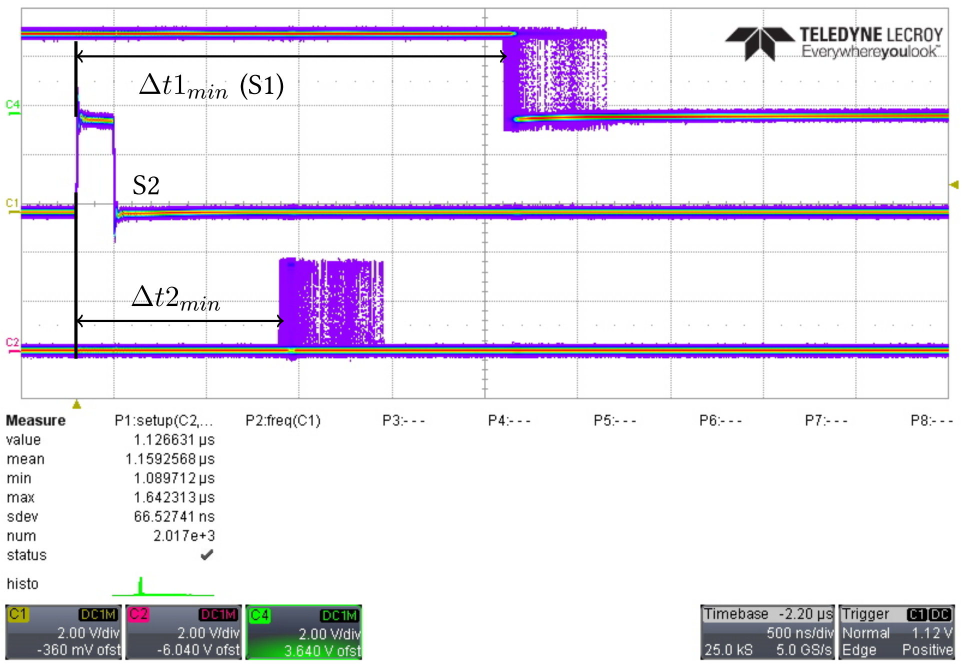


Figure 6.3: Shifting PWM resume caused by not setting the PWM counter to a specific value when resuming the PWM. Jitter: $4\mu s \equiv PWMperiod$
(Persistent measurement without RTOS, measured on auxiliary signal as S1 could not be used for a precise measurement cf. figure 9.3)

put in suspended mode. Due to this change the resume time was cut by almost $4\mu s$ the time of one PWM cycle.

6.4 Interrupt Latency with RTOS

On a comparison of figure 6.3 where no RTOS was used and figure 6.4 with a OS, it can be observed that the OS generates jitter when entering an ISR. The latency applies to the slave resume request, shown in figure 6.4, as well as the PWM reset ISR, in figure 6.6. For the slave resume time the worst case, $\Delta t1_{max}$, with 2.85 µs is well below the specified 5 µs(cf. table 2.1) . The worst case latency for reading in S2 with 0.94 µs is barely within its limit of the 1µs high phase set in the current PWM (measurement: cf. figure 9.2). In both cases the ISRs have the highest priority of all implemented ISRs. On the Cortex-M3 architecture lower priority interrupts can be preempted by higher priority interrupts, the latency for a preempted interrupt is 0. The delay can also be caused by a critical section in the Code, where all interrupts are disabled. A precise analyses of the code revealed, that there is a critical section in the IDLE thread running. To visualize and prove that this section is responsible, a GPIO pin is set and cleared right before and after the critical section. If an interrupt occurs in the critical section it is suspended until after this section but before the clear command of the GPIO pin. This results



```
Measure   P1:setup(C2,...   P2:freq(C1)   P3:---   P4:---   P5:---   P6:---   P7:---   P8:---
value       1.126631 µs
mean       1.1592568 µs
min         1.089712 µs
max         1.642313 µs
sdev        66.52741 ns
num           2.017e+3
status               ✓
histo

C1 DC1M        C2 DC1M        C4 DC1M                    Timebase  -2.20 µs  Trigger    C1 DC
  2.00 V/div     2.00 V/div     2.00 V/div                          500 ns/div  Normal     1.12 V
-360 mV ofst   -6.040 V ofst   3.640 V ofst               25.0 kS   5.0 GS/s  Edge     Positive
```

Figure 6.4: Interrupt latency measurement of the slave resume $\Delta t1$ time with RTOS & PWM counter reset . Setup time measurement with S2 and GPIO pulse in ISR.
Jitter: $\Delta t2_{max} - \Delta t2_{min} \approx 550ns$; slave resume time: $\Delta t1_{max} \approx 2.85\mu s$; number of measurements: 2017

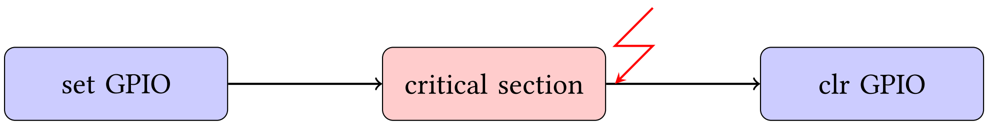
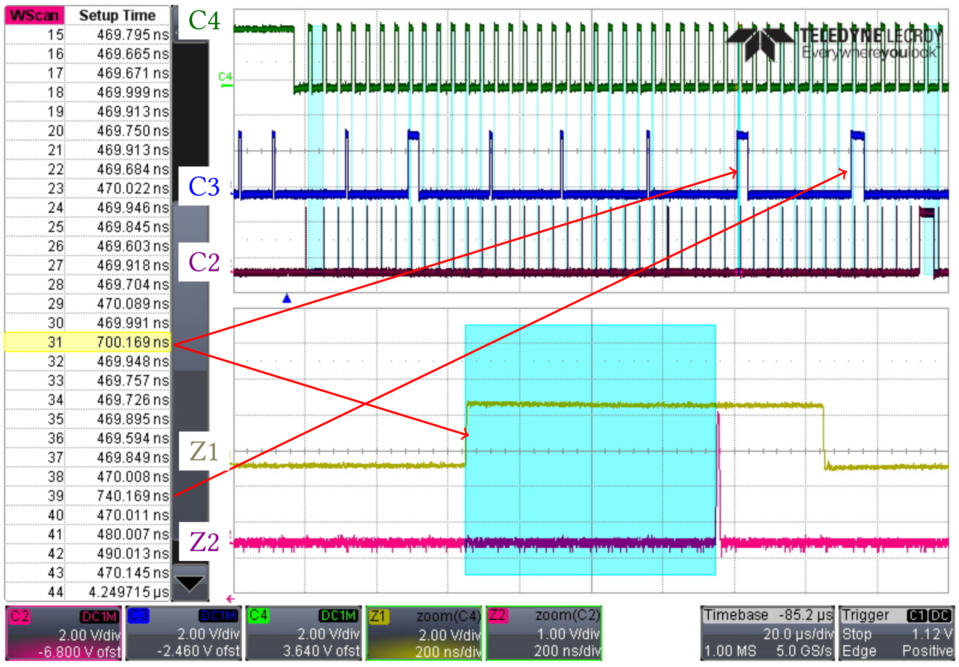


Figure 6.5: Flow of critical section to visualize the effects of the ISR delay. The Interrupt is delayed until after the critical section. This results in a longer pulse on the GPIO pin.



WScan	Setup Time
15	469.795 ns
16	469.665 ns
17	469.671 ns
18	469.999 ns
19	469.913 ns
20	469.750 ns
21	469.913 ns
22	469.684 ns
23	470.022 ns
24	469.946 ns
25	469.845 ns
26	469.603 ns
27	469.918 ns
28	469.704 ns
29	470.089 ns
30	469.991 ns
31	700.169 ns
32	469.948 ns
33	469.757 ns
34	469.726 ns
35	469.895 ns
36	469.594 ns
37	469.849 ns
38	470.008 ns
39	740.169 ns
40	470.011 ns
41	480.007 ns
42	490.013 ns
43	470.145 ns
44	4.249715 µs

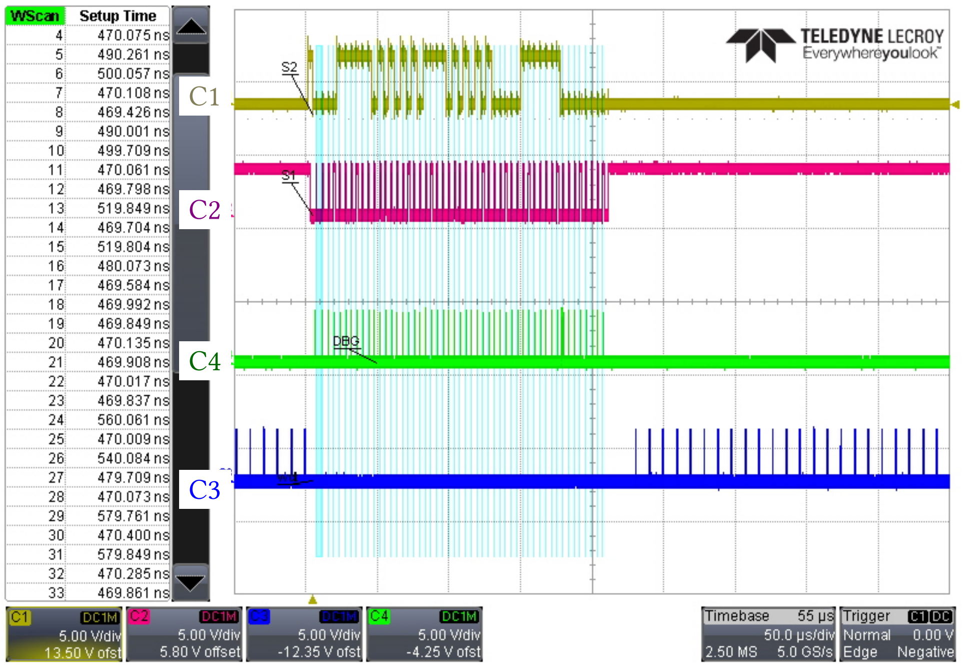
C2 DC1M	C3 DC1M	C4 DC1M	Z1 zoom(C4)	Z2 zoom(C2)	Timebase -85.2 µs	Trigger C1 DC
2.00 V/div	2.00 V/div	2.00 V/div	2.00 V/div	1.00 V/div	20.0 µs/div	Stop 1.12 V
-6.800 V ofst	-2.460 V ofst	3.640 V ofst	200 ns/div	200 ns/div	1.00 MS 5.0 GS/s	Edge Positive

Figure 6.6: Time measurement from PWM reset to read in of S2 with latency caused by critical section. Wide pulses on C3 caused by interrupt during critical section.
C4, Z1(green, yellow): PWM signal (S1); C2, Z2(red): readin of S2 finished; C3(blue): critical section; light blue area: measured time

in a longer pulse on this GPIO pin, as can be seen in figure 6.6. The ISR latency correspond exactly with an interrupt event that occurred during this critical section. Because the critical section is passed through periodically roughly every 900 cycles it is very likely an interrupt during this critical section occurs. The section is located in the idle thread of the SaReader system and is caused by a watchdog feed. As this critical section poses a problem while the PWM is running it has to be disabled. During the slave resume interrupt this is not a problem, as the ISR does not require as strict timings as the PWM. Thereby the WDT-Feed can just be disabled if the PWM is started and enabled when the communication is finished. This can be achieved with setting a flag that is checked before the watchdog feed. If the flag is set the watchdog will not be feed and the critical section is not entered (cf. listing 6.2 line 14). This flag was added and the analyses in figure 6.7 shows that with a disabled watchdog these timing spikes don't occur during the transmission. As the watchdog timer is set to 5 minutes, it does not impact its functionality if the WDT is enabled when the communication is completed. Also other timing sensitive interrupts could profit from this functionality of disabling this critical section for a period of time.

```c
1  //WDT_Feed function with a critical section
2  void WDT_Feed (void)
3  {
4      // Disable irq interrupt
5      __disable_irq();
6      LPC_WDT->WDFEED = 0xAA;
7      LPC_WDT->WDFEED = 0x55;
8      // Then enable irq interrupt
9      __enable_irq();
10 }
11
12 //...
13 //Code extraction of the IDLE-tread with WDT_Feed command and fix
14 if (g.rdr.wdtFeedUpdate) { //Flag To disable WDT_Feed
15     SET_DBG_Pin2 //Set GPIO H before critical
16     WDT_Feed();
17     CLR_DBG_Pin2 //Set GPIO L after critical
18 }
19 //...
```

Listing 6.2: Code extraction of the WDT-Feed function and the its call in the IDLE thread. With the a global variable used in line 14 the WDT-Feed can be disabled.



WScan	Setup Time
4	470.075 ns
5	490.261 ns
6	500.057 ns
7	470.108 ns
8	469.426 ns
9	490.001 ns
10	499.709 ns
11	470.061 ns
12	469.798 ns
13	519.849 ns
14	469.704 ns
15	519.804 ns
16	480.073 ns
17	469.584 ns
18	469.992 ns
19	469.849 ns
20	470.135 ns
21	469.908 ns
22	470.017 ns
23	469.837 ns
24	560.061 ns
25	470.009 ns
26	540.084 ns
27	479.709 ns
28	470.073 ns
29	579.761 ns
30	470.400 ns
31	579.849 ns
32	470.285 ns
33	469.861 ns

C1
C2
C4
C3
S2
S1
DBG
TELEDYNE LECROY
Everywhereyoulook
C1 DC1M 5.00 V/div 13.50 V ofst
C2 DC1M 5.00 V/div 5.80 V offset
C3 DC1M 5.00 V/div -12.35 V ofst
C4 DC1M 5.00 V/div -4.25 V ofst
Timebase 55 µs 50.0 µs/div 2.50 MS 5.0 GS/s
Trigger C1 DC Normal 0.00 V Edge Negative

Figure 6.7: Received frame with disabled WDT-Feed during the communication. C1(yellow): S2 (slave to host); C2(red): S1 (host to slave); C4(green): readin of S2 finished, cf. setup time left; C3(blue): critical section of the WDT-Feed

7 Timing driven Design Modifications

7.1 Changes to code positions

In a timing critical environment the position of the code can be significant, as certain code sections need to be executed in specific time frame. Also a long execution time in the wrong position can have impact, as timing requirements will then not be met. In the DWP implementation this can be seen in multiple code changes.

- **Variable initiation:** In the first versions the initiation of the variables used for the communication was done before the signal generation was started. This had of cause impact on the length of the slave resume time, in a worst case scenario where the interrupt from S2 was delayed due to a critical section it was possible to get timings outside of the specification limits. This problem was solved by resetting the required variable during the shutdown sequence where the PWM-IR is disabled, here the increased time consumption is less likely to impact the communication.
- **Position of the slave resume in IRQ Handler:** For the slave resume request the EINT3_ IRQHandler is used. This handler is also used by other interrupt sources, UART and other GPIO pins, the right positioning of the interrupt source check is essential, as every check takes time. This becomes even more important if multiple interrupts occur at the same time, they all are processed delaying the desired interrupt further. Sorting the checks and execution by timing priority can minimize the risk of this problem.
- **Interrupt acknowledge:** The interrupt acknowledge can take around 300 ns (cf. read measurements with the PWM in section 3.1), by not putting it in the beginning of the ISR, as done for the readin of S2 it reduces the response time to the IR. As the Interrupt occurs periodically it is ensured that the ISR is finished before the next interrupt. For this a period was chosen that is longer then the worst case execution time of the ISR.

7.2 Semaphores and Queues in RTOS

For the communication between threads, FreeRTOS provides semaphores and queues. Both can block and wait for resources. The communication with the state machines is also based on a loop with a blocking queue. For accessing a queue from an ISR, special macros are provided in the FreeRTOS code. The timings of both semaphores and queues macros was tested as they were to be used for communication with the state machines. As can be seen in the measurements (cf. figure 9.5 and 9.6) both durations are over 4 µs and thereby too long to be used in the PWM-ISR, as the maximum duration of this ISR is equal to the PWM period with 4 µs.
In figure 7.1 it can be observed that an interrupt is lost due to the long duration of the ISR. This results in an unread S2 signal and potentially in a double sent bit on S1. As an alternative to

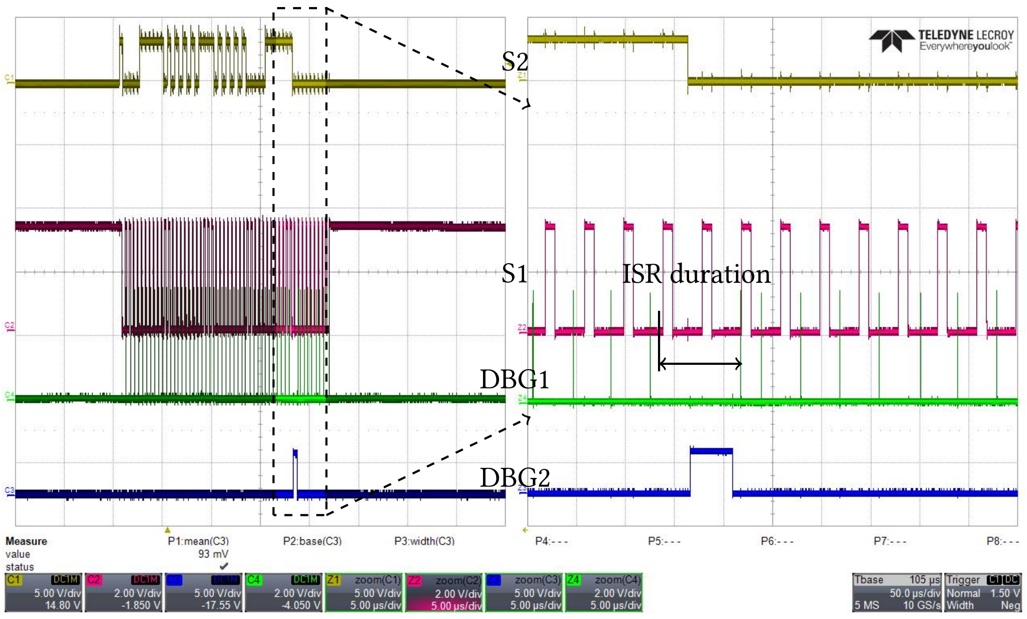


Figure 7.1: Measurement with a Semaphore_ give in the ISR. Due to the long ISR one Interrupt is lost (one spike on DGB1 is missing in the marked area).
DBG1: ISR finished; DBG2: sem_ give duration

the integration of the state machine notification in the PWM-ISR the following concepts:

- **Polling:** To inform the state machines of a new received frame, polling could be used. A counter with the number of frames would then be polled. The handicap of polling is that CPU time is used that other tasks in the background could otherwise use, this can result in an undesired delay in this tasks if they run in the background. Also the watchdog needs to be fed to avoid a reset of the entire system.

- **SW triggered interrupt:** To not execute the queue send command in the main run-time critical ISR a second lower priority ISR can be used. The advantage is, the time demanding operations can be interrupted. To send to the queue only the faster software interrupt is needed in timing critical ISR.
- **Code positioning:** The third solution is to move the queue send command to the end of the communication where the PWM Interrupt is disabled. As no more data is transmitted in this section a longer execution time has no impact on the communication. In this case a separate counter would count the number of received frames. The drawback of this solution is, the delay between the message arrival and the processing can be very long. This problem particularly emerges if multiple frames are transmitted.

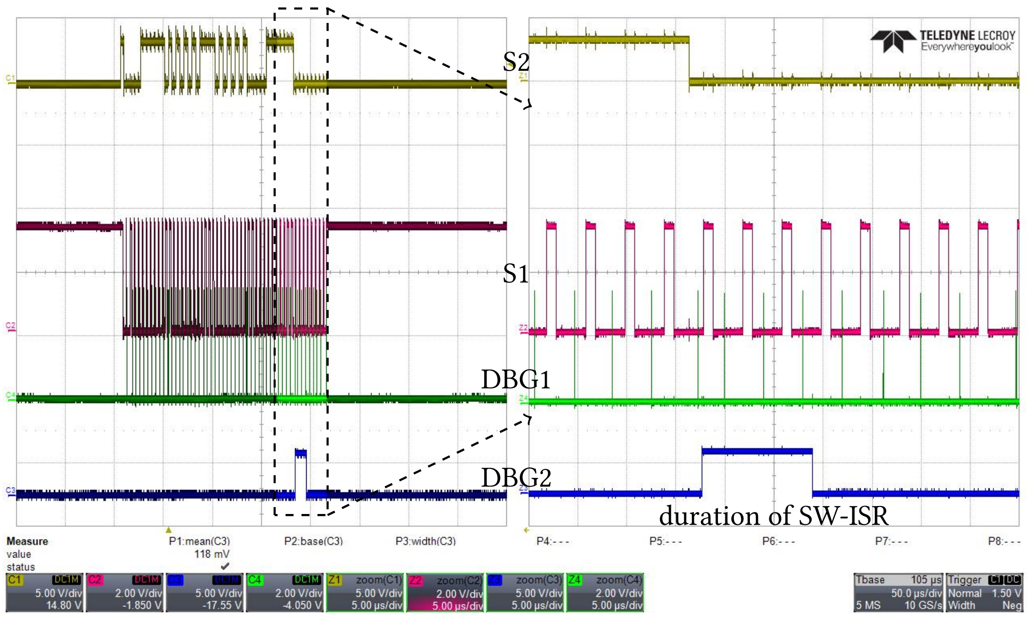


Figure 7.2: Measurement with a software interrupt in the ISR. Here no Interrupt is lost, as the SW-ISR has a lower priority.

8 Conclusions

8.1 Results

The results of the measurements have shown that it is possible to achieve the required timing specification, on a pure microcontroller based system. But the thesis has also clearly shown the limitations of the microcontroller, especially for faster communication speeds. In this application type the communication between the CPU and the peripherals, especially the PWM module, are the most time consuming operations due to the connection via APB. Also due to the use of the required ISRs, valuable time is lost during the resulting context switches. A second interesting aspect that was found is the problem of adjusting the processing length for the GPIO solution, which is mentioned in section 6.3.1. As soon as the compiler optimization is enabled it very difficult to control the timing.
The measurement, with the persistent feature of the oscilloscope on the interrupt latency was vital, as it revealed a critical section that was effecting all ISRs. It clearly shows the importance of multiple measurements. In the case of the watch dog critical section, detection was rather simple because of two factors:

- the ISR and the critical section were executed regularly.
- the interrupt also has a hardware generated signal, against which the delay of the IRQ Handler could be measured.

8.2 Summery

The main goal of this thesis was, to analyze if it is possible to implement a Dual Wire Protocol (DWP) communication with the required specified timing parameters that are specified by the single wire protocol (SWP) specification [ETS12], on an existing Cortex M3 microcontroller. This µC is the base of an existing smart card reader, which is used for the validation of future security devices. In the measurements of this thesis it turns out that the simplest solution to achieve the needed timings is to implement a pulse width modulation (PWM) based solution. In the beginning of this thesis the protocol basics, including the physical characteristics as

well as data structure of the protocol are presented. In addition the target-system options are compared including the currently used microcontroller with the operating system .
On this base in chapter three the two possible microcontroller based solutions are further analyzed, the first solution is a PWM with an periodic interrupt for the signal update, the second is dependent on GPIO pins with idle instruction in the CPU to generate the required timings. For this, different timings are measured to evaluate if it is possible to accomplish the required timings with the different solutions.
After these analyses of the microcontroller the architecture for the system is developed (4.1). For generating the correct transmission of the data on the physical layer two program flows, one for each direction, are designed. For the two required frame types of the DWP state machines are designed for later protocol implementation For later testing and validation of the system the base for a test-pattern generator is designed.
With these designs test code for both solutions of chapter three, as well as the interface between the state machines and the communication module is implemented. Furthermore the integration in the system of the SaReader is described. To test the system also the designed pattern-generator and a waveform-analyzer for the DWP output are implemented.
In section 6.1 a setup for testing the functionality is shown. Different analyses techniques are discussed and problems presented that where found with these techniques. In the last chapter design changes are demonstrated which result from timing issues that were exposed during the testing of the entire system.

8.2.1 Prospect

During the thesis a full implementation of DWP-layers 1 and 2 on a μC of the given reader system was developed. For a fully working solution of the DWP with real targets further steps are required that are outside of the scope of this thesis:

- The implementation and testing of the DWP state machines, as they are required for the activation and the communication with a secure element. A possible design is shown in section 4.3.
- The implementation of the HCI state machine. The HCI protocol is used to send actual APDUs (Application Protocol Data Unit) to the device.
- Testing with a physical device. In this step also problems can emerge that didn't occur before as only the simulation of a device was used.

- further optimization of the protocol. As the transmitting speed is based on the measured worst-case-time, to increase it the duration of the ISR can to be reduced this could be done for example by temporarily disabling the full duplex mode if only one side is transmitting.

For future designs, the integration of an FPGA for a faster data transmission can still be considered, this could probably also be a benefit for other protocols. For the DWP the FPGA could speed up the communication by the factor of 4 or even up to 8, if the secure element supports faster transmission rates. Due to the design of the implemented software, for this integration the com-module would simply have to be replaced. This can potentially make it possible to also run different reader designs. As the readers all should run the same firmware, to make them interchangeable, an identification for the hardware configuration would then be required.

9 Appendix

9.1 Abbreviations & Definitions

ACT	ACTivate LLC
CLT	ContactLess Tunneling LLC
CRC	Cyclic Cedundancy Check
EOF	End of Frame
DWP	Dual-Wire Protocol
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
ISR	Interrupt Service Routine
LLC	Logic Link Control
MAC	Medium Access Control
PWM	Pulse Width Modulation
SE	Secure Element
SHDLC	Simplified High Level Data Link Control
SoC	System on Chip
SOF	Start of Frame
SWP	Single-Wire Protocol
S1	signal from master to slave
S2	signal from slave to master

9.2 Charts & Pictures

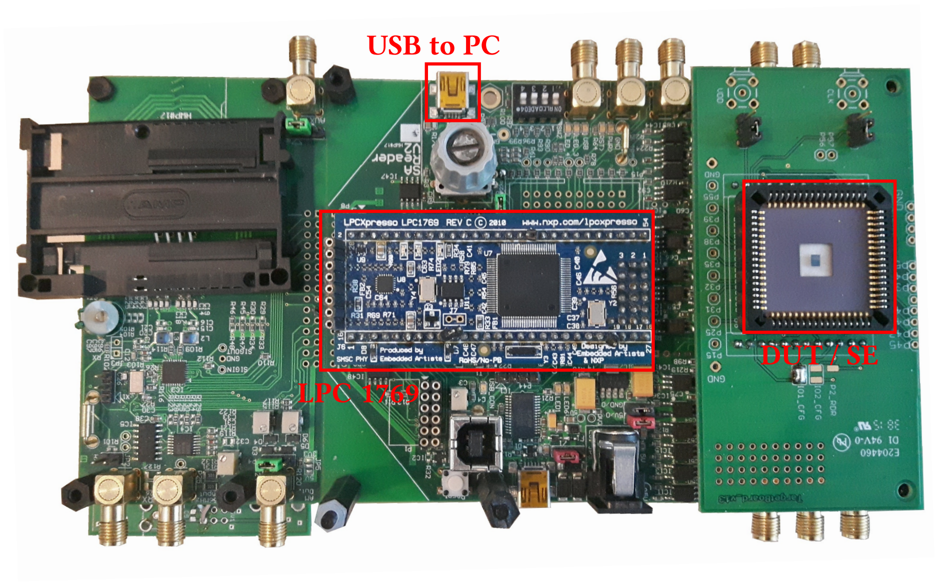
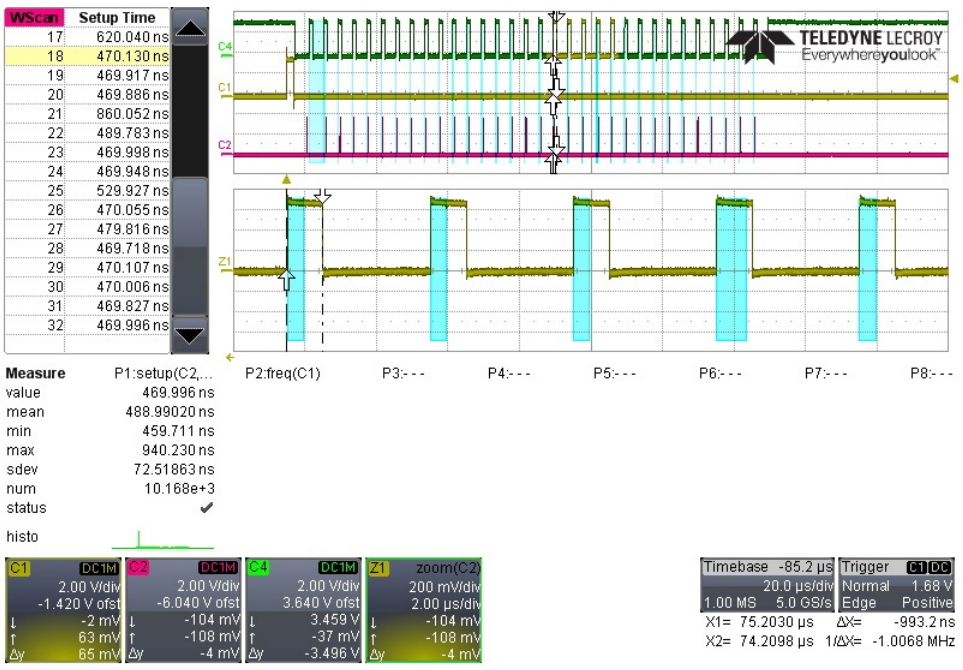


Figure 9.1: LPC1769 connected to the SaReader board and a secure element in a CLCC68 socket

9.3 Measurements



WScan	Setup Time
17	620.040 ns
18	470.130 ns
19	469.917 ns
20	469.886 ns
21	860.052 ns
22	489.783 ns
23	469.998 ns
24	469.948 ns
25	529.927 ns
26	470.055 ns
27	479.816 ns
28	469.718 ns
29	470.107 ns
30	470.006 ns
31	469.827 ns
32	469.996 ns

Measure	P1:setup(C2,...	P2:freq(C1)	P3:- - -	P4:- - -	P5:- - -	P6:- - -	P7:- - -	P8:- - -
value	469.996 ns							
mean	488.99020 ns							
min	459.711 ns							
max	940.230 ns							
sdev	72.51863 ns							
num	10.168e+3							
status	✓							
histo								

C1 DC1M	C2 DC1M	C4 DC1M	Z1 zoom(C2)
2.00 V/div	2.00 V/div	2.00 V/div	200 mV/div
-1.420 V ofst	-6.040 V ofst	3.640 V ofst	2.00 µs/div
↓ -2 mV	↓ -104 mV	↓ 3.459 V	↓ -104 mV
↑ 63 mV	↑ -108 mV	↑ -37 mV	↑ -108 mV
Δy 65 mV	Δy -4 mV	Δy -3.496 V	Δy -4 mV

Timebase -85.2 µs	Trigger C1 DC
20.0 µs/div	Normal 1.68 V
1.00 MS 5.0 GS/s	Edge Positive

X1= 75.2030 µs ΔX= -993.2 ns
X2= 74.2098 µs 1/ΔX= -1.0068 MHz

Figure 9.2: Time measurement from PWM reset to read in of S2
Jitter: $\Delta t1_{max} - \Delta t1_{min} \approx 480ns$;
S2 worstcase: $\Delta t1_{max} < \frac{1}{4}T$ ($940ns < 1\mu s$)

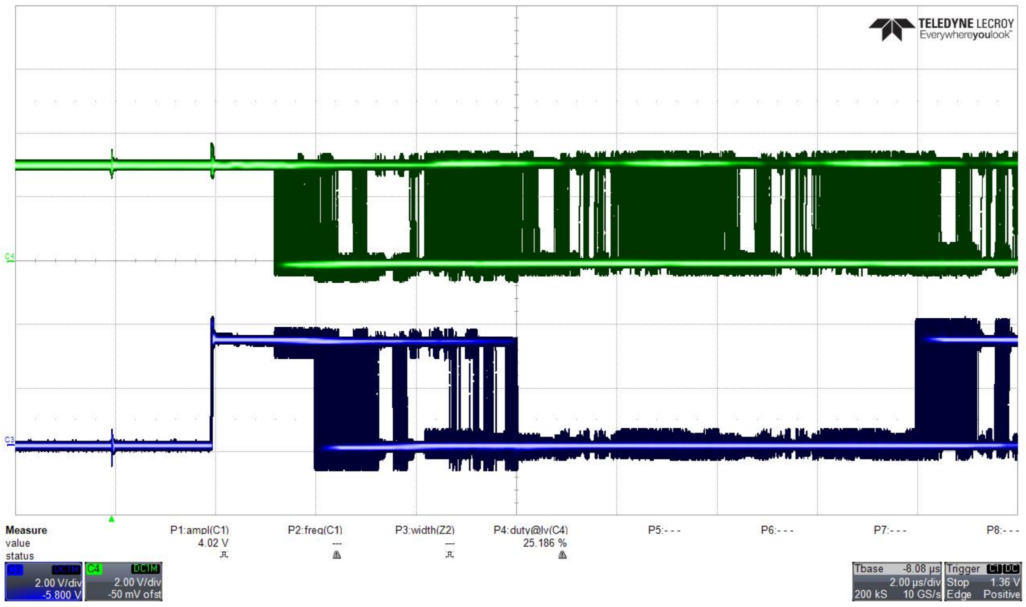


Figure 9.3: Persistent measurement of the PWM-resume without setting counter

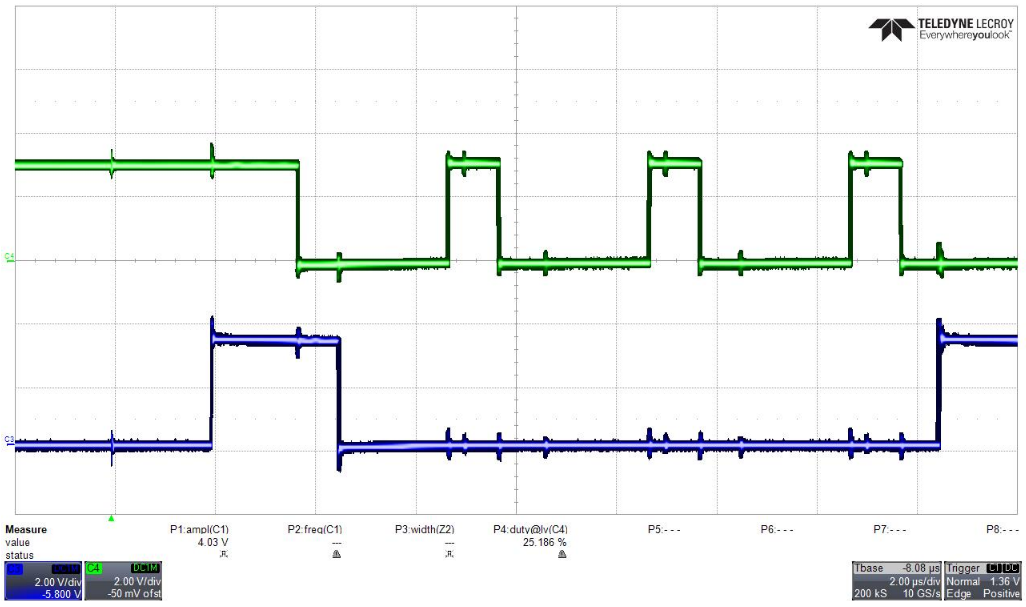
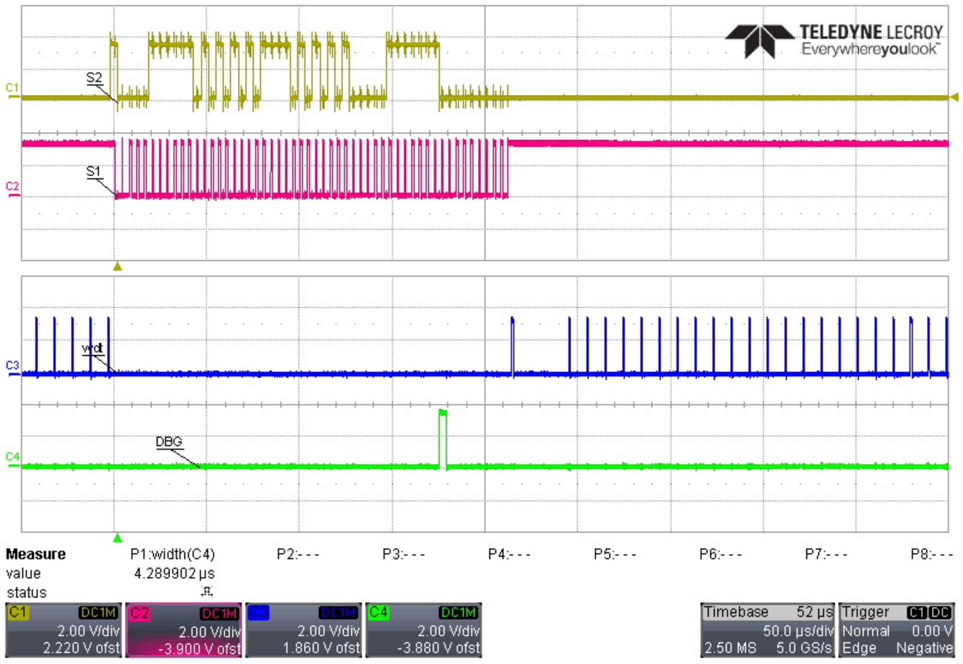


Figure 9.4: Persistent measurement of the PWM-resume with setting counter



```
Measure   P1:width(C4)   P2:---   P3:---   P4:---   P5:---   P6:---   P7:---   P8:---
value     4.289902 µs
status

C1 DC1M 2.00 V/div 2.220 V ofst | C2 DC1M 2.00 V/div -3.900 V ofst | C3 DC1M 2.00 V/div 1.860 V ofst | C4 DC1M 2.00 V/div -3.880 V ofst
Timebase 52 µs 50.0 µs/div 2.50 MS 5.0 GS/s | Trigger C1 DC Normal 0.00 V Edge Negative
```

Figure 9.5: Measurement of the execution duration of a Semaphore give function in the ISR after a received frame.
Duration: 4,29µs

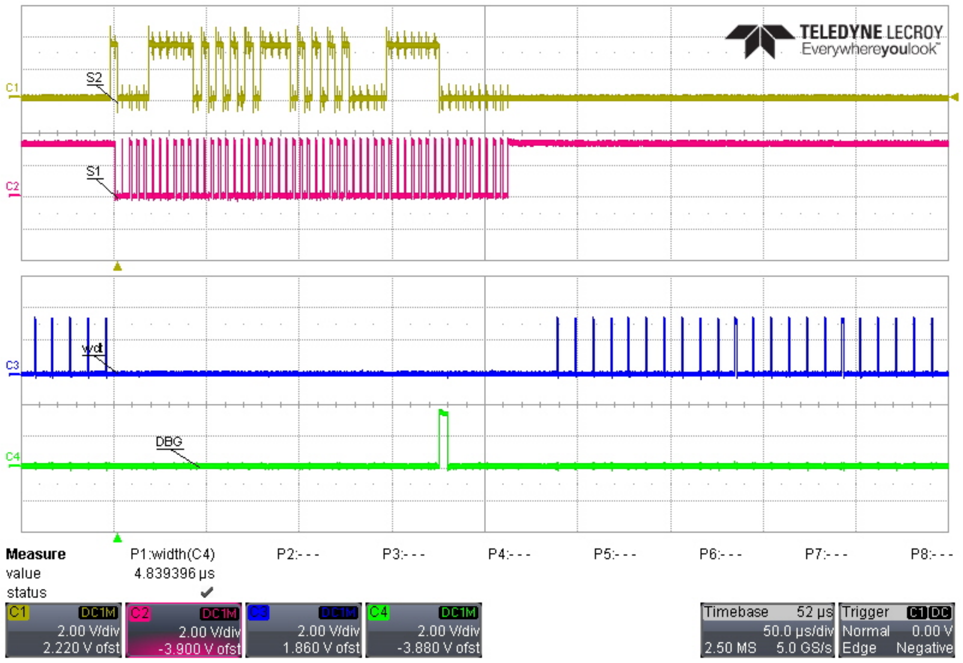

TELEDYNE LECROY
Everywhereyoulook
C1
S2
C2
S1
C3
wdt
C4
DBG
Measure
value
status
P1:width(C4)
4.839396 µs
P2:- - -
P3:- - -
P4:- - -
P5:- - -
P6:- - -
P7:- - -
P8:- - -
C1 DC1M 2.00 V/div 2.220 V ofst
C2 DC1M 2.00 V/div -3.900 V ofst
C3 DC1M 2.00 V/div 1.860 V ofst
C4 DC1M 2.00 V/div -3.880 V ofst
Timebase 52 µs 50.0 µs/div 2.50 MS 5.0 GS/s
Trigger C1 DC Normal 0.00 V Edge Negative

Figure 9.6: Measurement of the execution duration of a Queue send function in the ISR after a received frame.
Duration: 4,84µs

9.4 Code

9.4.1 Generation S1 with GPIO

```
This code section is included on the CD.
```

Listing 9.1: Code extraction: Test code for the generating of S1 with GPIO pins, without needed timing compensation.

9.4.2 Generation S1 with PWM

```
This code section is included on the CD.
```

Listing 9.2: Code for the generating of S1 with a PWM module.

9.4.3 DWP interface

```
This code section is included on the CD.
```

Listing 9.3: Code of the DWP interface with buffers.

Bibliography

[ARM] ARM. *What is the true interrupt latency of Cortex-M3 and Cortex-M4 for interrupt entry and exit?* [Online; accessed May 2017]. URL: `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka16366.html`.

[ARM15] ARM. *ARM®Cortex®-M3 Processor. Technical Reference Manual.* Rev r2p1. [Online; accessed May 2017]. 2015. URL: `http://infocenter.arm.com/help/topic/com.arm.doc.100165_0201_00_en/arm_cortexm3_processor_trm_100165_0201_00_en.pdf`.

[Bec17] Leo Becker. "Apple: Keine Verhandlungen mit Wikileaks über "Vault 7"". In: *C't* (2017). [Online; accessed May 2017]. URL: `https://www.heise.de/security/meldung/Apple-Keine-Verhandlungen-mit-Wikileaks-ueber-Vault-7-3663486.html`.

[Com] Comprion. *Spectro TP. Fact Sheet.* [Online; accessed May 2017]. URL: `https://www.comprion.com/fileadmin/user_upload/comprion/Products/Spectro_TP/Spectro_TP_FS.pdf`.

[Com16] Comprion. *User Manual. Card Test Center.* R2.6. 2016.

[ETS12] ETSI. *ETSI TS 102 613.* V11.0.0. 2012.

[Fin12] Klaus Finkenzeller. *RFID Handbuch.* 6. Auflage. Hanser, 2012.

[LeC] LeCroy. *Feature: WaveScan.* [Online; accessed May 2017]. URL: `http://teledynelecroy.com/features/featureoverview.aspx?modelid=2107&capid=102&mid=556`.

[Ltd16] Real Time Engineers Ltd. *The FreeRTOS™Reference Manual. API Functions and Configuration Options.* Rev 9.0. 2016. URL: `http://www.freertos.org/Documentation/RTOS_book.html`.

[NXP16] NXP. *UM10360. LPC176x/5x User manual.* Rev 4.1 - 19 December 2016. 2016.

[RE08] Wolfgang Rankel and Wolfgang Effing. *Handbuch der Chikarten*. 5. Auflage. Hanser, 2008.

[Rüs15] Kai Rüsberg. “Keyless gone”. In: *C’t* (2015). [Online; accessed May 2017]. URL: `https://www.heise.de/ct/ausgabe/2015-26-Autodiebe-tricksen-kontaktlose-Schliesssysteme-aus-3013915.html`.

[Tro16] Theodore Albert Trost. “System and method for enabling a dual-wire protocol”. US 9350831 B2. May 24, 2016. URL: `http://www.google.com/patents/US9350831`.

[Yiu14] Joseph Yiu. *The Definitive Guide to ARM®Cortex®-M3 and Cortex-M4 Processors*. Third Edition. Newnes, 2014.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. August 2017 David Niederwestberg