Conference Presentation

Algorithmic analysis and hardware implementation of a two-wire-interface communication analyser

Schinagl, P. and Sharp, A.

This is a paper presented at the 7th IEEE Int. Conference on Internet Technologies and Applications ITA-17, Wrexham, UK

Algorithmic Analysis and Hardware Implementation of a Two-Wire-Interface Communication Analyser

Peter Schinagl, Andrew Sharp
School of Applied Science, Computing and Engineering,
Glyndwr University, Plas Coch, Mold Road,
Wrexham, LL11 2AW, UK

***Abstract*—This paper discusses the development of an algorithm for the data analysis to monitor Two-Wire-Interface operation in order to improve the reliability of communication. This algorithm is designed to improve code-efficiency with regards to hardware modelling. An algorithm for the protocol used in the Standard-Mode, Fast-Mode, Fast-Mode Plus and High-Speed-Mode was developed. The proposed algorithm has been derived using the bus protocol specification and implemented in hardware via a hardware description language. The correct operation of the algorithm was proofed by applying the hardware system on a sample communication. The paper also describes the development process of embedded systems and provides information on aspects regarding hardware modelling including a mathematical description of the TWI protocol is provided.**

Keywords—I2C bus, two-wire-interface, FPGA, reliability

I. Introduction

With the rise of microelectronics, in particular integrated circuits (IC), communication in the form of bus systems gained importance in order to allow complex arrangement of controllers and peripherals to execute their tasks both more efficient and faster. Most commonly used Inter-Integrated-Circuit (I^2C) serial bus also known as Two-Wire-Interface (TWI) has been developed by NXP Semiconductors in 1982 [1],[2]. Because of TWI's significant role, the bus is a potential failure source that must be considered in terms of troubleshooting. Furthermore, it is crucial for reverse engineering to understand ongoing bus communication in a system [3],[4].

The bus operates using two lines: a data line (SDA) and a clock line (SCL). Apart from the Start and the STOP-Command, every intended signal must be applied on the data line before the clock changes its state to high and must not change until the next low period (Fig. 1). A falling edge of SDA during a high cycle of the clock represents a Start or Repeated-Start-Command whereas a rising edge indicates a Stop-Command [1].

The frequency at which the master drives the SCL wire of the bus depends on the operating mode. A complete listing of the maximum SCL frequencies is shown in Table I. The master oversees the clock cycle which is applied on the SCL line. An exception is clock stretching, with the help of which a slave can pause the communication. A TWI network can accommodate 127 devices using a 7-Bit long address. An address extension to 10-Bit was later implemented into the standard [1],[2].

Communication via this bus must be seen as a data exchange between a master and a slave notwithstanding that the specification allows multi-master-operation. In the latter case one of both masters must behave as a slave. The device which initiates a communication by a Start-Command is holding the status of the master. It is crucial to emphasise that the Master and Slave statuses do not correspond with the Transmitter and Receiver status. Instead, the data flow direction is variable. TWI therefore is classified as a half-duplex communication.

An extra bit after every byte is provided for verification purposes. During the reserved Acknowledgement-Bit slot, the receiver of the previous transmitted byte must pull the SDA line on low to indicate the transmitter that it received the preceding byte and is prepared for further communication. If the receiver holds the status of the master, the bit is used to indicate if another byte is expected. A positive Acknowledgment is referred to as an ACK whereas a negative Acknowledgement is referred to as NACK. Each data frame must consist of at least one byte followed by an acknowledge bit. This arrangement can be repeated as often as desired within one frame. Within a byte, the bits follow a descendant order from the Most-Significant-Bit (MSB) to the Least-Significant-Bit (LSB) [1],[2]. The bitwise representation of TWI sequence is shown in Fig. 2.

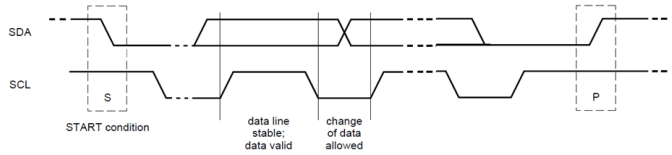

Fig. 1. Signal change at Start and Stop-Commands [1].

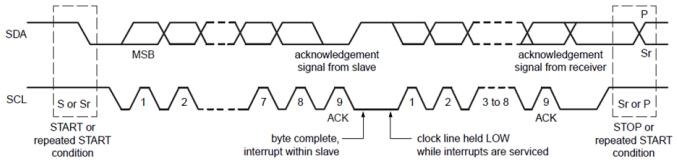

Fig. 2. Bitwise representation of a TWI sequence [1].

978-1-5090-4815-1/17/$31.00 ©2017 IEEE

TABLE III. MAXIMUM FREQUENCY OF THE TWI MODES [1]

Operating Mode		Max. Frequency	Unit
Standard-Mode		100	kHz
Fast-Mode		400	kHz
Fast-Mode Plus		1	MHz
High-Speed Mode	100 pF max	3.4	MHz
	400 pF max	1.7	MHz
Ultra-Fast Mode		5	MHz

Previous studies had been carried out with the proposed work to develop a hardware model of a TWI-bus participant. Dependent on the mode of participation, the models' extent ranged beyond the scope of an analyser. Any development of a master device implied, that monitoring a TWI communication can be realised utilising a state machine in Hardware Description Languages (HDL) code [5]-[7]. However, the previously published papers are not focused on mathematical elaborations and analysis. In general, there was a lack of papers which investigated the derivation of an algorithm from the TWI bus. This work focuses on the performance analysis of the TWI in terms of assessment of the reliability of communication. It includes analysis of code-efficiency with the aid of an analysis of the protocol and its mathematical interpretation.

The other main technology used in this study is Field Programmable Gate Array (FPGA) where function of ICs can be programmed after manufacturing. This enables the design of a specific IC in order to meet the requirements of a particular project [8]. The implementation of the desired logical function into the FPGA is realised using HDL.

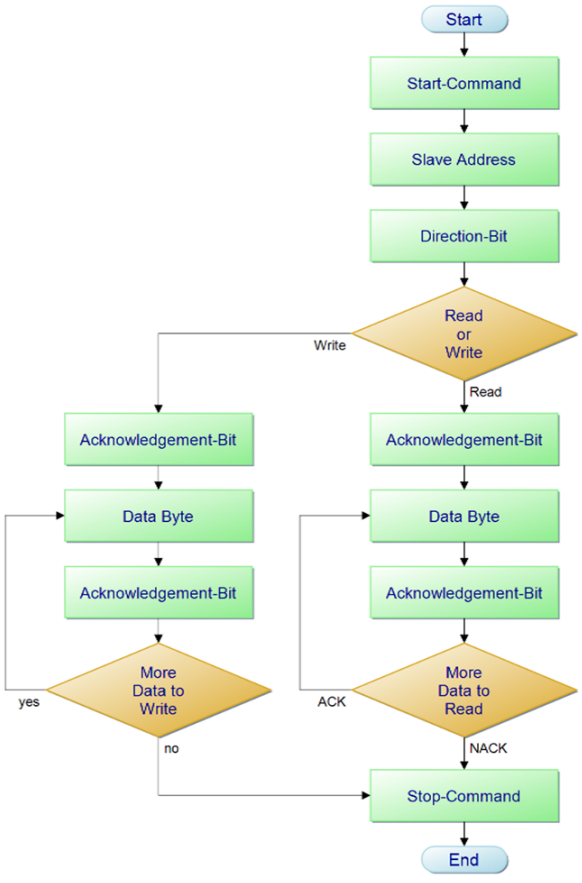

Fig. 3. Flowchart of TWI protocol.

HDLs differ from software programming languages substantially due to the fact that they describe the interconnection and thus the hardware behaviour instead of a sequence of instructions [9]. In addition, HDLs are counted among parallel languages. This is due to the nature of hardware programming, which does not follow the concept of a processor polling its commands in serial order [10],[11].

In order to implement a bus-monitoring function into a FPGA chip with the aid of a HDL, an algorithm was determined to be designed to provide a mathematical model which acted as a pattern, based on which the programme was coded. Furthermore, an improvement of the code's efficiency was expected by utilising such an algorithm.

II. TWI PROTOCOL

Fig. 3 illustrates a complete sequence on the TWI-bus including the potential branches from the master's point of view. The master executes a Start-Command to initiate the communication, followed by the seven-bit long slave address and the indicator, whether it desires to write or read data. In both cases, the slave must acknowledge that it has been addressed. Depending on the data flow direction either the master or the slave must send a byte which must be acknowledged if the slave is the receiver, whereas the master as a receiver uses this acknowledgement to indicate if it expects another byte. In the position of the transmitter, the master can send another byte without declaring upfront. Either way the communication must be terminated by a STOP condition from the Master.

Two supplementary notes on the process are listed below:

- Instead of a Stop-Command, the Master can apply another Start-Command on the bus, which is referred to as a Repeated-Start.
- Acknowledgement-Bits which the slave is responsible for must not carry a NACK. Otherwise, an error will occur and the Master is in charge of ending the communication.

III. MATHEMATICAL BACKGROUND

A. Bit Index

The protocol of a serial bus purports the appearance of high and low states on the data line with regards to the transmitted data. Therefore, under consideration of the possible branches of the protocol, the number of bits to be transmitted can be predicted. By using Boolean algebra, above mentioned branches can be interpreted and utilised to identify the current state of the communication and predict the number of following bits. In the following, an elaboration of the number of the appearing bits dependent on the course of the communication is provided.

The communication must start with a Start-Command, which occupies one bit of the sequence. The end or restart of the communication is indicated by a Stop-Command or

respectively a Repeated-Start-Command. Both of the latter mentioned occupy one bit as well. Another fixed slot is captured by the Direction-Bit. Therefore, three bits of the sequence can be predicted to appear regardless of the course of the communication. The address of the slave which is transmitted as the first information after the Start-Command seizes seven bits [1].

The core of the communication is occupied by data bytes. The number of bits concerning data to be transmitted therefore is equal to the sum of all data byte's bits and can be obtained multiplying the number of data bytes with eight.

All data bytes must be followed by an Acknowledgment-Bit. Hence, the number of data bytes also determines the number of Acknowledgement-Bits. Because the direction-indicating-bit is also followed by an acknowledgement bit, its quantity is equal to the number of data bytes plus one.

Combining these elaborations, the overall number of bits can be determined:

$$\begin{aligned} \text{Overall Bits} &= C_s + A + n \times 8 + (1+n) \\ \text{Overall Bits} &= C_s + A + n \times 9 + 1 \end{aligned} \tag{1}$$

where the constant C_s represents the number of bits that must appear on every issued communication, consisting out of the Start-Bit, the direction-indicating-bit and the Stop-Bit respectively the Repeated-Start-Bit. Hence, this constant is equal to 3; the constant A depicts the address which in this study is assumed to be 7 bit long; n indicates the number of transmitted data bytes. This number is either known upfront or, in case of a monitoring function, derived during the monitoring process by evaluating the binary values of the acknowledgement bits.

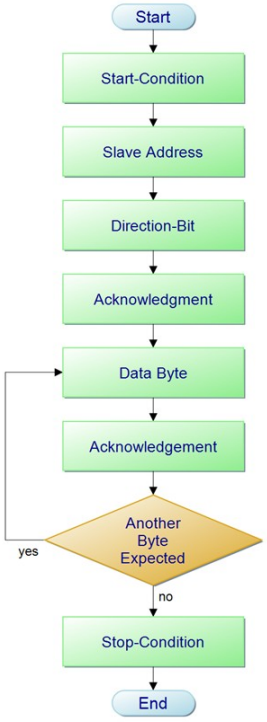

Fig. 4. Flowchart of a whole sequence of the algorithm

B. *Boolean Branch Operations*

The prediction whether another data byte follows an Acknowledgment-Bit must be obtained by using Boolean algebra. As shown in Fig. 3 the acknowledgement that follows every data byte that is read from the master is used to indicate whether the master expects another byte to read. The logical composition of direction status and the acknowledgement value (ack) therefore was utilised to forecast the next state of the communication.

Presupposing that 'Write' correlates with the false state of the Boolean value of the direction (dir) and 'Read' corresponds with its true value, the following equations could be set up to predict the following state:

$$\text{Stop-Command} = \left(\overline{dir} \cap \overline{ask}\right) \cup \left(dir \cap ask\right) \tag{2}$$

$$\text{Error} = \overline{dir} \cap ask \tag{3}$$

$$\text{Next Data Byte} = \left(\overline{dir} \cap \overline{ask}\right) \cup \left(dir \cap \overline{ask}\right) \tag{4}$$

$$\text{Stop-Command} = \overline{dir} \cap \overline{ask} \tag{5}$$

The equations' results are summarised in Table II.

IV. ALGORITHM

The electrical specification of the bus defines that every transmitted bit must hold a consistent logical level over the whole period of the clock. The Start, Stop and Repeated-Start commands however differ from that specification. Instead of operating on a Boolean value, these commands are indicated by providing a rising or respectively falling edge of the data signal during a high clock level. [1]

This restriction allowed to identify a command that issues the start or end of a communication and therefore were utilised as the entry and exit conditions of the algorithm. Once the algorithm is entered, the number of bits to appear until the first branch is expected to have an impact on the communication is counted. Due to the fact that every information can be identified via the corresponding bit index, the direction can be obtained and stored for later usage.

When eventually the bit to indicate the branch is transmitted, its value – together with the direction value – is fed into the Boolean equations that form the truth table shown in Table II. The results of these Boolean operations determine the following progress of the communication as shown in Fig. 4 and hence allow the algorithm to predict which information is expected to follow.

TABLE II. TRUTH TABLE THE BRANCH OPERATION

Direction	Acknowledge	Prediction
Write	0	Stop-Command or next byte
Write	1	Error
Read	0	Next data byte
Read	1	Stop-Command

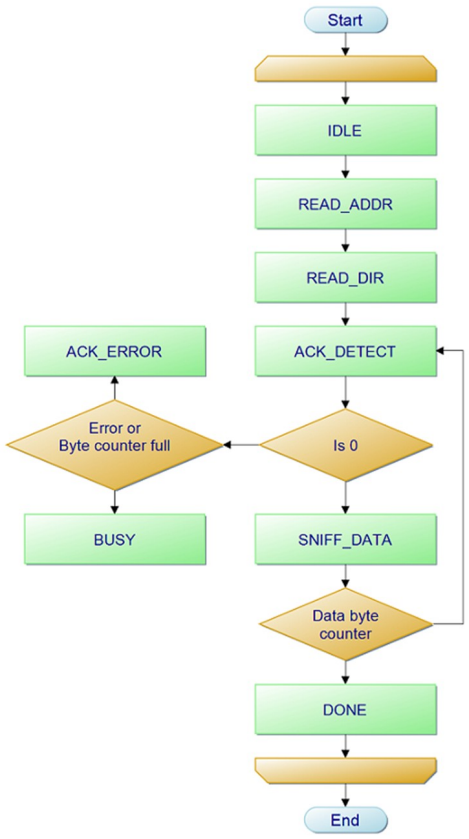

Fig. 5. Flowchart of the state machine.

V. Algorithm Implementation

As the target system, a Cyclone II chip from the manufacturer Altera was elected. The developed algorithm was transferred into a VHDL code. The sequential nature of the algorithm advocated in favour of the utilisation of a state machine to implement the algorithm into hardware [12],[13]. Due to hardware limitations, the maximum number of data bytes was limited to 2. This restriction however did not confine the algorithms capabilities because further communication is subject of the same recurring procedure of the algorithm. An illustration of the state machine is given as a flowchart depicted in Fig. 5. The single states of the state machine were derived from the algorithm and its corresponding entities of communication within a sequence of a TWI communication.

TABLE III. Truth Table of the Exit Conditions

Byte Count < 2	**Direction**	**ACK**	**Target State**
False	READ	0	BUSY
False	READ	1	ACK_ERROR
False	WRITE	0	BUSY
False	WRITE	1	BUSY
True	READ	0	SNIFF_DATA
True	READ	1	BUSY
True	WRITE	0	SNIFF_DATA
True	WRITE	1	ACK_ERROR

Hereinafter the states of the state machine and their transition conditions were explicated.

IDLE

When no communication has been tracked or a reset command was entered, the IDLE state is occupied. A Start-Command causes the transition into the READ_ADDR state.

READ_ADDR

To enable processing of the value using the same sub-entities as the data registers, the register in which the address is stored was defined to be eight bit long, carrying a ‘0’ as the MSB. For the duration of seven clock cycles on the SCL line, the programme tracks the level on the SDA line and stores the values in this register. While doing so, the decreasing counter value is used as the index of the bit of the register in which the value is stored. An abstraction of this procedure is presented in (6).

$$\text{adderss}\left(\text{bitcounter}\right) = SDA_{bitcounter} \tag{6}$$

When the bit counter ran through seven cycles, the READ_DIR state is entered.

READ_DIR

The bit which indicates the communication direction specified by the master is caught and sets the corresponding READ or WRITE flag. For further usage the value is also stored in a signal.

ACK_DETECT

This state may be entered from different stages. Every byte -long sequence is followed by an acknowledge bit and therefore demands this state to be run through. This applies on the data bytes as well as the seven bit address combined with the subsequently transmitted direction indicating bit. This state moreover possesses different branches to navigate the programme flow into. An overview of the possible exit states is given in Table III.

It was predicted that this state must be called for the second time after and the first data byte and hence could be used to count the overall number of data bytes. The state therefore is used to branch into the BUSY state when the processing capacity of data bytes is reached. In this test, this capacity was set by the hardware which allowed to display two data bytes. The exit condition into the BUSY state was therefore hardcoded to a compare condition of the data byte counter with the value of two. If the acknowledgement bit suggests an error, the current state changes to ACK_ERROR. If none of the above mentioned exit conditions are met, the state machine proceeds its normal procedure by entering the SNIFF_DATA state.

ACK_ERROR

Accessed by the acknowledge detection in case of an error, this state sets the error flag and awaits a reset command.

BUSY

The programme’s inability to track data frames beyond the length of two bytes due to hardware limitations legitimates the

TABLE III. TRUTH TABLE OF THE EXIT CONDITIONS

Information	Value
Slave Address	126
Data Byte 1	255
Data Byte 2	255

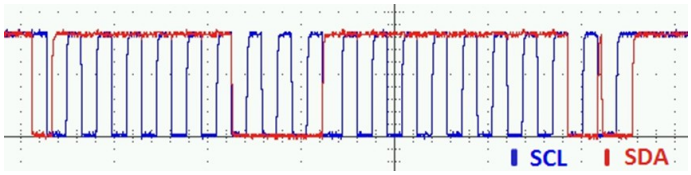

Fig. 6. Oscilloscope trace of the sample sequence.

existence of this state. When no further monitoring is possible, this state is called. This however is not an error handler. It is expected that the communication finishes with a Stop-Command which causes a transition into the DONE state.

SNIFF_DATA

The eight bits that follow this state's invocation are stored in a temporary register to be written to the entity's output afterwards. The bitwise value assignment was designed the same way as in the READ_ADDR state. Before accessing the ACK_DETECT state, the data byte counter is incremented.

DONE

Every error-free cycle of the state machine terminates in this state. An operator input is necessary to switch from here into the IDLE state to start a new measurement.

Supplementary Notes:

- Every state was provided with a reset handler. By resetting the system, all temporary registers and flags are set to their initial values and the IDLE state is accessed
- A STOP command terminates the state machine regardless of the current state.
- A START or REPEATED START command leads to a transition into the READ_ADDR state regardless of the current state.

VI. TESTING

The verification of the algorithm's implementation was carried out by setting up a sample TWI-communication and let the system monitor it. As sample parameters, the slave address was set to 126 and the transmitted data was set to the value of 255. An oscilloscope track of the transmitted data frame is depicted in Fig. 6.

The system on which the algorithm was implemented was hooked up to the test system and a communication was issued. After verifying the operation to this point, a two byte long data frame with the same parameters was transmitted to provide a different branch decision for the algorithm. The second byte also was detected and correctly displayed. The monitored information is listed in Table IV.

The sample parameters and data which had been configured to be transmitted by the sample communication system had been monitored successfully. It could be shown that the algorithm meets the requirements and was mathematically correct.

V. CONCLUSION

It has been shown that the developed algorithm for the communication monitoring was correctly derived from the protocol specification. It was also demonstrated that a TWI communication can be transformed to a more abstract stage by utilising mathematical methods to index the transmitted bits and assign these indices to correlating information respectively states of the communication. An interpretation of the course of events on a Boolean level was elaborated and applied to provide a logical model for branches.

The findings could be reasonably implemented into hardware by transferring the worked out relations into hardware-description-language expressions and designing a finite state machine to assign each incremental step of the communication to its corresponding code segment.

Further work proposes the improvement implementation of the 10-Bit address range as well as the Ultra-Fast-Mode into the algorithm which has not been considered in this study.

REFERENCES

[1] NXP Semiconductors. (2014, 4 April). I2C-bus specification and user manual UM10204 [Online]. Avalable: http://www.nxp.com/documents/user_manual/UM10204.pdf

[2] I2C Info. (2017). [Online]. Available: http://i2c.info

[3] S. Freiberger, M. Albrecht and J. Kaufl, "Reverse engineering technologies for remanufacturing of automotive systems communicating via CAN bus," *Journal of Remanufacturing*, vol. 1, no. 6, pp. 1-15, Dec. 2011.

[4] Yan-Jie Chai, Ji-Yin Sun, Jing Gao, Ling-Jiao Tao, Jing Ji, and Fei-Hu Bao, "Improvement of I2C bus and RS-232 serial port under complex electromagnetic environment," in *Proc. Int. Conf. on Computer Science and Software Engineering*, 12-14 Dec. 2008, Hubei, China, pp. 178-181.

[5] R. Archana, and J. V. Rao, "Implementation of I2C master bus protocol on FPGA," *Int. Journal of Engineering Research and Applications*, vol. 4, no. 10, pp. 6-10, 2014.

[6] B. Eswari, N. Ponmagal, K. Preethi, and S.G. Sreejeesh, "Implementation of I2C master bus controller on FPGA," in *Proc. Int. Conf. on Communication and Signal Processing*, 3-5 April 2013, Melmaruvathur, India, pp. 678-681.

[7] A. Oudjida, M. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui, "FPGA implementation of I2C & SPI protocols: A comparative study," in *Proc. 16th IEEE Int. Conf. on Electronics, Circuits and Systems*, 13-16 Dec. 2009, Yasmine Hammamet, Tunisia, 2009, pp. 507-510.

[8] W.A. Najjar, and P. Ienne, "Reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 4-6, Jan-Feb 2014.

[9] D.J. Smith, "VHDL and Verilog compared and contrasted-plus modeled example written in VHDL, Verilog and C," in *Proc. 33rd Design Automation Conf.*, 3-7 June 1996, Las Vegas, NV, pp. 771-776.

[10] P.J. Ashenden, *The Student's Guide to VHDL*, 2nd ed. Burlington, USA: Elsevier, 2008.

[11] IEEE Computer Society, *IEEE Standard VHDL Language*. New York: IEEE, 2009.

[12] H. Wallace. (2003). Using state machines in your designs [Online]. Available: http://aqdi.com/articles/using-state-machines-in-your-designs-3

[13] J. Brownlee. (2002). Finite state machines (FSM) [Online]. Available: http://ai-depot.com/FiniteStateMachines/FSM-Background.html