Fast UART and SPI Protocol for Scalable IoT Platform

Rizka Reza Pahlevi[1], Aji Gautama Putrada S[2] and Maman Abdurohman[3]

School of Computing
Telkom University
Bandung, Indonesia
[1]rizkareza.p@gmail.com, [2]ajigps@telkomuniversity.ac.id, [3]abdurohman@telkomuniversity.ac.id

***Abstract*—This paper proposes the enhancement of Universal Asynchronous Transmitter Receiver (UART) and Serial Peripheral Interface (SPI) protocols for improving the node performance in the Internet of Things (IoT) platform. Implementation of UART communication protocol on IoT in microcontroller ATmega328 has limitation of Serial Hardware scalability so it must do bit banging which can affect performance and lack of speed. SPI requires change of SPI Control Register (SPICR) to become Slave Select which can affect speed and also have limited of Slave Select. There are many previous methods that have been proposed to solve this problem. However, none of them are scalable and speed aware. This proposed method uses parallelism concepts by implementing hardware environment. Through some experiments, this research succeeded to implement UART on FPGA where the speed is 67,3 % faster than the microcontroller ATmega328 and is able to reach four UARTs. SPI Slave on FPGA has speed up to 73.43 % faster than the microcontroller ATmega328 and is able to expand to two SPI Slaves.**

Keywords*—*Microcontroller, UART, SPI, FPGA, IoT

I. Introduction

Today, technological developments are very rapidly included in developments of hardware and software. One of the most discussed is IoT [1]. IoT makes electronic devices such as home appliances, medical devices, and sensors become part of an internet network [2]. The use of IoT continues to increase along with the users demand with the number of devices are up to 30,000,000 by 2018 [2] and covers all sectors of life [3] [4]. Many of them are platform based IoT application [5] [6]. But the development of IoT devices has many challenges, for example in scalability and speed.

The IoT device helps users by communicating with each other to collect data. This communication can be wired or wireless. That communication in hardware is known as inter machine communication. This is commonly referred to as Machine-to-Machine Communication or commonly referred to as M2M Communication. Some types of communication protocol used in hardware are UART and SPI. UART and SPI communication protocols are used for communication between Integrated Circuits (IC). The utility consideration of a communication protocols on a hardware device can improve the performance of communication data between machines.

The development of IoT device uses microcontroller as the main processor. Microcontrollers have many limitations such as scalability, speed, and storage space. On the microcontroller, UART communication protocol has limited amount of hardware Serial so when expanding the UART's scalability, it requires bit banging technique which can disrupt communication performance. On microcontroller, SPI especially SPI Slave requires changes to SPICR which can reduce speed performance. From IoT device problems which must consider scalability and speed [7], one of the devices that can handle it is FPGA. Field Programmable Gate Arrays or often referred to as FPGA is an electronics and semiconductor in an Integrated Circuit (IC) that can be configured by the programmer for the purposes of system simulation [8]. FPGA supports high speed, low power, low latency, better deterministic, adequate scalability, and parallel execution.

Main contribution of this paper is proposing UART and SPI protocol communication that is fast and scalable to solve the speed and scalability problem on the development of IoT. Fast UART and SPI is a way to communicate the chips on the developing IoT and also enables the scalability of communication.

II. Literature review

A. Related Works

This paper [9] attempts to make a design modeled into VHDL that intends to test UART communication protocols capability of being modeled and stable. The paper provides an overview of the use of the protocol.

Meanwhile, this paper [10] attempts to design and synthesis an UART block protocol communication. The research in this paper applies the Baud Rate Generator, Transmitter, and Receiver Modules.

In the paper [11], the research makes a design that can run SPI communication protocol in FPGA. The paper also describes how Clock Polarity and Phase which is a mode that can be used for synchronization between master and slave. The paper describes the port description used both on SPI Master and SPI Slave.

In the paper "SPI Implementation on FPGA" [12], the research also tried to implement SPI communication protocol on FPGA. In the proposed architectural design, the design

ISBN: 978-1-5386-4571-0 (c) 2018 IEEE

uses registers on SPI Master and SPI Slave. Also in this architectural design is the use of Clock Generator.

B. Serial Peripheral Interface

In its implementation, the SPI protocol uses two modules, Master and Slave [12]. This communication protocol requires 3+N paths where the number of N depends on the number of slaves. The paths are MISO, MOSI, SCLK, and SS [13].

1) Architecture of SPI: SPI communication protocol has three important parts that is Clock Generator, Master Module, and Slave Module. In Fig. 1, the master has three output paths that is MOSI, SS, and SCLK. MOSI used by the master to send data to the slave by making the slave select value go to the low. The slave module has an output of only 1, MISO. The path is used for the slave to provide data according to the speed given by the Slave Clock of the master.

2) Timing Diagram of SPI: SPI communication protocol has several modes. With Fig. 2, there are four applicable models [12].

- Model 0
  Model 0 it uses CPOL = 0 and CPHA = 0. In this mode, SCK starts with a low state. The data will be captured when the clock rises (low to high transition) and data data will be received when the clock drops (high to low transition).
- Model 1
  Model 1 it uses CPOL = 0 and CPHA = 1. In this mode, SCK starts with a low state. The data will be captured when the clock drops (high to low transition) and the data will be increased as the clock rises.
- Model 2
  Model 2 is using CPOL = 1 and CPHA = 0. In this mode, SCK starts with a high state. The data will be captured when the clock rises (low to high transition) and data will be received when the clock drops (high to low transition).
- Model 3
  Model 3 it uses CPOL = 1 and CPHA = 1. In this mode, SCK starts with a high state. The data will be captured when the clock rises (low to high transition) and data will be received when the clock drops.

C. Universal Asynchronous Receiver-Transmitter (UART)

It is the protocol for communicating that has asynchronous characters [9] [14]. It also could communicate in full-duplex, which will be used in data communications and control systems [9]. This protocol has two signal paths that is RXD and TXD for full-duplex communication [9].

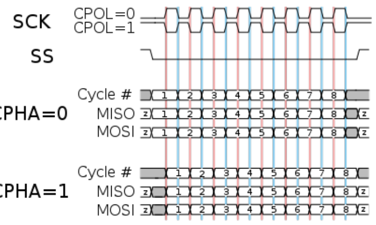

Fig. 2. Timing diagram SPI [12]

1) Architecture of UART: UART communication protocol has three important parts, those are transmitter and receiver modules and Baud Rate generator as shown in Fig. 3.

- Baud Rate Generator
  It has function as clock divider. There are two factors affecting the baud rate frequency such as clock of the clock oscillator and baud requests [9]. The purpose of using a baud rate generator is to speed up the asynchronous serial data [9] [10].
- Receiver Module
  Receiver Module is responsible for receiving data from RXD input pins. In the process the displacement from signal 1 to 0 means the beginning of the reception of data [10] [9].
- Transmitter Module
  The function of the Transmitter Module is to change from parallel to serial data, adding some bits to data for controlling purpose [10] [9].

2) Timing Diagram of UART: UART communication protocol has a timing diagram as in Fig. 4.

Fig. 4 indicates that idle state is high or 1, when there is a displacement of signal 1 to 0, it can be said as the start bit, and after that the data is beginning to be delivered to up to 8 bits. After the 8 bits data has been sent, the last bit is the stop bit which indicates that the data has been read and then the signal will come back to idle or become a high signal.

III. Methodology and System Design

A. Methodology

The methodology of the analysis and implementation of this communication protocol cover the design of architecture and communication protocols and how the communication protocol verification tests have been implemented towards the FPGA as shown in Fig. 5.

The design of the communication protocol is a process design block that will be required in accordance with the

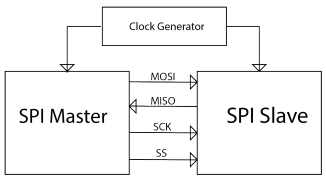

Fig. 1. Architecture SPI [12]

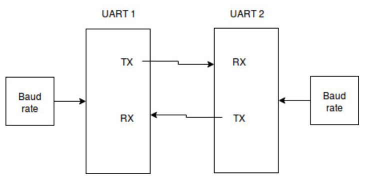

Fig. 3. Architecture UART

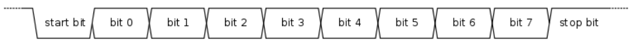

Fig. 4. Timing diagram UART

standard provisions of the communication block and then the design with VHDL. After the design is finished, the next step is synthesizing the design. After completion of verification, it is then applied on the FPGA. Then on the verification of communication protocol, it will be done by communicating with Raspberry Pi.

1) Designing Architecture of Communication Protocol Design: In the Design of this Communication Protocol, communication protocols will be built, designed, and implemented towards the FPGA. The task of designing is to design a protocol to have standardized communication validation. The block design is built on the design and communication protocols as shown in Fig. 6.

2) Designing Architecture of Communication Protocol Verification: Verification of this communication protocol is a way used to know whether the design built on the FPGA can run on a standard communication protocol. This section uses Raspberry Pi hardware as the equipment that verifies it. Fig. 7 is a block design of communication protocol verification. In Fig. 7 it contains two processes, which are, designing the code and sending and reading data from the FPGA that has been designed.

IV. Results and Discussion

A. Designing Architecture of Communication Protocol Design

Designing architecture of communication protocol will present the design and synthesis results of the design of UART communication protocol, SPI Slave and SPI Master.

1) UART:

- Baud Rate Generator
  The Baud Rate Generator uses 9600bps. To get the speed a divider is needed or in the design is called clock divider. Clock divider is

$$M = \frac{CLK}{baud\ rate} \tag{1}$$

  Where M is the clock divider, CLK is the internal clock of the FPGA, and baud rate is the number of the baud rate speed. The clock divider is the result of the clock used inversely proportional to the baud rate used. So if in a design using clock equal to 50 MHz or 50.000.000 Hz then a baud rate equal to 9600 bps is desired, then result of the clock divider obtained from equation (1) will be 5208, where this number will be used in the Receiver Module and Transmitter Module which is used as a marker whenever data will be sent and received.
- Receiver Module
  This module requires a clock entity, rx line which is used as a marker that there are data coming on the Receiver Module. Data is used as a container of incoming bits, and busy which gives a flag or sign to the module that the Receiver Module is receiving data. If flag is low and rx line low, it will give busy signal to high and flag high. In the data retrieval process, the clock divider is starting to be used. Data on UART consists of 10 bits, bit 0 is called start bit, 1 to 8 is data, and bit 9 is stop bit.
- Transmitter Module
  In this module the clock entity is required, start as a marker of the trigger, busy which gives signal to other modules that Transmitter Module is sending data, data as input bit, and tx line as bit data transmission path. When the flag is low and start is high, busy signal will be high.

In the synthesis results report, UART design uses 117 logic elements and 81 registers. In the design four pins are used. Pin UART RX is a pin that is used for data paths that come to the FPGA board and the UART TX pin is used for data paths coming out of the FPGA. Then after synthesis, the RTL result is obtained from the design that is made. In Fig. 8, the Receiver Module and Transmitter Module have been successfully created. Clock has been entered into both modules. In Fig. 8 shows that the clock, uart rx, and key acts as input, and uart tx acts as output.

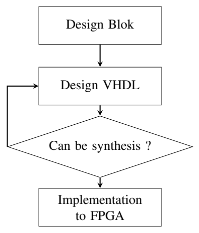

Fig. 6. Flowchart of communication protocol design

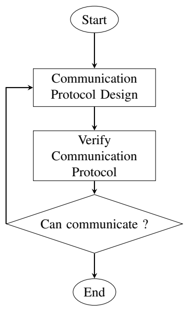

Fig. 5. General flowchart system

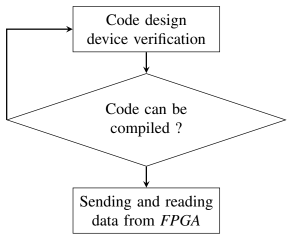
Code design device verification
Code can be compiled ?
Sending and reading data from *FPGA*

Fig. 7. Flowchart of architecture communication protocol verification

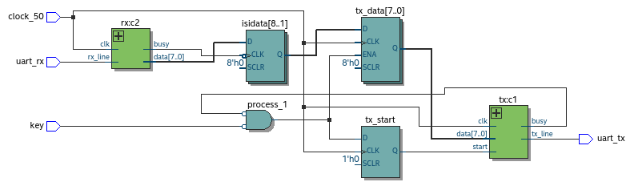

Fig. 8. RTL results of UART synthesis

2) *SPI:*

- SPI Slave
  Speed of clock will affect the retrieval and sending of data, so it is of great concern. Clock counter is used to calculate the length of the received data. SPI Master will send 16 bits of data which consists of 8 bits of instruction and 8 bits of data. In this design used 0x00 as the instruction to write on SPI slave and 0x01 as command to read existing data in SPI slave. In the synthesis report, the SPI Slave design uses 64 logic elements and 53 registers. In the design of the SPI Slave five pins are used.
  SPI Clock pin is the input from of the Master that contains the clock. SPI MOSI pin is the pin that becomes the path of the data coming from the Master and the SPI MISO pin is the path of the data going out to the Master. Afterwards the SPI Slave Select pin is used by the master to activate a device that will be selected. From Fig. 9, an overview of the final design is provided. The Fig 9 illustrates that there are four given inputs: sys-clk which is the internal clock of the FPGA, *SCLK* as the Slave Clock from the Master, *MOSI* as MOSI, and *SS* as Slave Select, output is also given through *MISO* as MISO.
- SPI Master.
  Because the SPI Master acts as the center of SPI, it should have a standard just as Fig. 2 that representative of SPI. SPI Master using four pins. SPI Master can be work under 500KHz, so in the design should step down the speed of clock. This design use cspol 0, cpha 0, dan cpol 0. SPI Clock is

$$spiclock = \frac{clock}{clock\ div \cdot 2} \tag{2}$$

  Where spiclock is speed maximum of SPI is 500 KHz, clock is internal clock of FPGA, and clock div is numbers given so that the clock spi will be worth 500 KHz.
  With equation 3, with the known internal clock of 50 MHz, the div clock is obtained 50. In the reporting of the synthesis result, the SPI Master design uses 86 logic elements and 60 register. In SPI Master design uses six pins.

B. *Verify Communication Protocol*

1) *UART:* In the previous design, the built UART communication protocol on the FPGA can communicate with Raspberry Pi by sending characters according to the ASCII table. Then once the FPGA successfully receives data from the Raspberry Pi, the FPGA responds to the input and sends it back to the Raspberry Pi.

2) *SPI:*

- SPI Slave
  In a previously designed Slave SPI, the design can communicate with a Raspberry Pi and receive command blocks and data blocks. The FPGA can understand the instruction bits and the data from the Master. The FPGA can send the data back to the FPGA if the Master asks for the read instruction.
- SPI Master
  The SPI Master testing will run directly on the FPGA. In the previous test of SPI Slave it is stated that the SPI Slave has a good validation and can communicate with the master. Then after the slave receives the data, the slave will reply to the master in the form of a 16 bits long data. The SPI Master design can communicate with a previously verified SPI slave.

C. *Comparison of FPGA with* Microcontroller

In this comparison test ATmega328 as the microcontroller chip and the FPGA Altera EP4CE6E22C8N is used.

1) *UART on FPGA and Microcontroller:*

- Speed
  ATmega328 has been equipped with UART communication protocol. ATmega328 has one Hardware Serial located on PCINT16 as RX and PCINT17 as TX. The speed obtained from the crystal oscillator used by ATmega328 is 16 MHz While the FPGA used in this test has a speed of 50 MHz moreover the clock will be distributed on each block diagram. Then on the test which is done by sending 10000 bits of data to the FPGA and the FPGA will reply the command in the message. As in Fig. 10, UART communication between ATmega328 with Raspberry Pi with baud rate 9600bps is 3.33382 seconds while on the FPGA it is 1.08737 seconds. Testing in Fig. 10 is only one UART because on the ATmega328 there is only one Serial Hardware.
- Scalability
  Scalability is the ability of a system or device in dealing with the addition or expansion of the given load. The problem is that when using more than one UART protocol, it uses the bit banging technique. Bit Banging is a technique for providing new UART protocols using

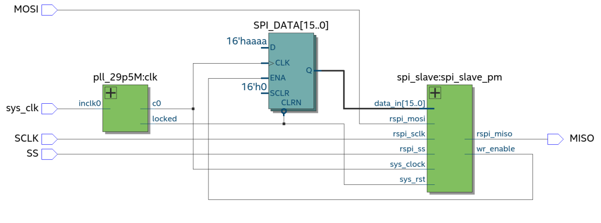

Fig. 9. RTL results SPI Slave synthesis

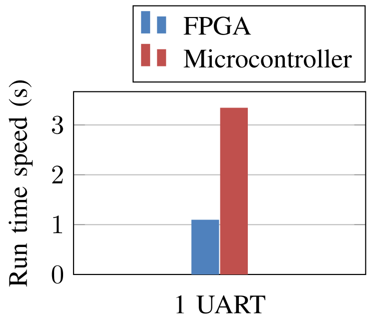

Fig. 10. UART Speed testing on FPGA

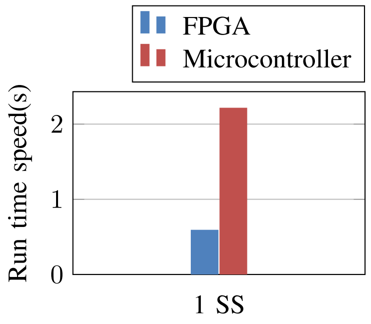

Fig. 12. SPI Slave speed testing on FPGA

software control. In the microcontroller the are some types of serial protocol.

1) *HardwareSerial* is a serial protocol that exists physically as it is on microcontroller and very good because it can make communication that reliability and can continuously.
2) *SoftwareSerial* is the choice of serial software which can use almost any pin, but only 1 can be active at once. Can disturb other program libraries.
3) *AltSoftSerial* is the choice of serial software which can transmit and receive data simultaneously and uses a 16 bit timer which will disturb other programs that also use a timer.

As shown that the second and third serial port types that can be used by the microcontroller have disadvantages, among others, can disturb the timer, disturb the interrupt, only one can transmit data at a time. However with FPGA, Hardware Serial manufacture can be done as much as possible as long as the number of pins are sufficient. To handle the problem then tested the manufacture of Hardware Serial on UART design for FPGA. As shown in Fig. 11, it is found that the UART of the FPGA is capable of being expandable at a stable speed with two UARTs the obtained time is 1.25278 seconds, three UARTs is 1.13758 seconds, and four UARTs is 1.08741 seconds.

2) SPI on FPGA and Microcontroller:

- Speed
ATmega328 has been equipped with an SPI Master Communication protocol but is requires a change to become a SPI Slave. The pins used for SPI Master transmission are on PCINT5 pins for SCK, PCINT4 for MISO, PCINT3 for MOSI, and PCINT2 for SS. When testing the ATmega328 into SPI Slave and doing data transmission of 16000 bits of data that was performed with Raspberry Pi as the SPI Master the time was 2.21041 seconds while on the FPGA it was found to be 0.58709 seconds as Fig. 12. The test is done on one SPI Slave because in microcontroller there is only one pin for SS.
- Scalability
With only one SS pin so ATmega328 can only control one slave only. Then applying two SPI slaves on the FPGA with purpose that its use can be expanded. In the test as in Fig. 13 it provides evidence that the FPGA is capable of being loaded by two SPI slaves and the second Slave has a transfer time of 0.64525 seconds.

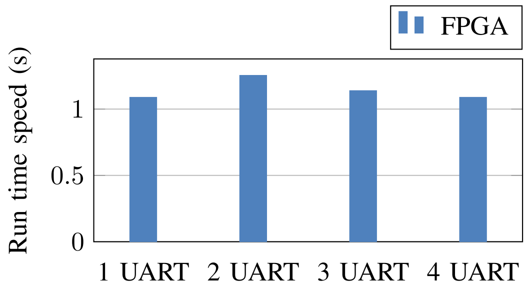

Fig. 11. UART scalability testing on FPGA

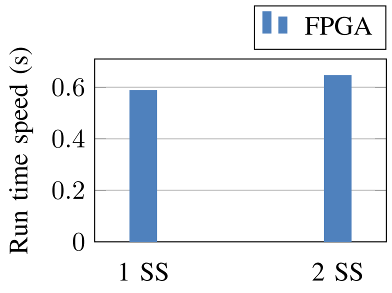

Fig. 13. SPI Slave scalability testing on FPGA

V. Conclusion

We have enhanced the UART and SPI protocol communication to solve the speed and scalability problem in IoT. UART and SPI communication protocols is allowed to be applied on an FPGA which can improve the performance of IoT. FPGA is capable of transmitting and receiving data of 10000 bits on UART requires 1.08737 seconds which is 67.3 % faster than ATmega328 and can be extended to handle scalability problems up to four UARTs. The SPI Slave requires 0.58709 seconds, which is 73.43 % faster than the ATmega328 and is also capable of handling scalability problems of up to two SPI Slaves and on the SPI Master is capable of communicating with the SPI Slave.

Acknowledgment

The authors would like to thank Telkom University that has for providing support for research. We also would like to thank the Hardware Embedded System Studio (HES Studio) Telkom University for providing the resources to do this research.

References

[1] statista, "Internet of things (iot): number of connected devices worldwide from 2012 to 2020 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2017, accessed : 2017-02-11.

[2] R. Buyya and A. V. Dastjerdi, *Internet of Things: Principles and paradigms*. Melbourne, Australia: Elsevier, 2016.

[3] M. Abdurohman, A. Sasongko, and R. Rawung, "Mobile tracking system based on event driven method," in *Applied Mechanics and Materials*, vol. 321-324. Trans Tech Publ, 2013, pp. 536–540.

[4] V. Suryani, A. Rizal, A. Herutomo, M. Abdurohman, T. Magedanz, and A. Elmangoush, "Electrocardiagram monitoring on openmtc platform," in *38th Annual IEEE Conference on Local Computer Networks - Workshops*. Sydney, NSW: IEEE, 2013, pp. 843–847.

[5] M. Abdurohman, A. Herutomo, V. Suryani, A. Elmangoush, and T. Magedanz, "Mobile tracking system using openmtc platform based on event driven method," in *38th Annual IEEE Conference on Local Computer Networks - Workshops*. Sydney, NSW: IEEE, 2013, pp. 856–860.

[6] M. Abdurohman, A. G. Putrada, S. Prabowo, C. W. Wijiutomo, and A. Elmangoush, "M2m device connectivity framework," *International Journal on Electrical Engineering and Informatics*, vol. 9, no. 3, pp. 441–454, 2017.

[7] W. Dargie and C. Poellabauer, *Fundamentals of wireless sensor networks: theory and practice*. Chichester, United kingdom: John Wiley & Sons, 2010.

[8] M. Abdurohman, *Perancangan Embedded System Berbasis FPGA*. Yogyakarta: Graha Ilmu, 2014.

[9] Y.-y. Fang and X.-j. Chen, "Design and simulation of uart serial communication module based on vhdl," in *Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on*. IEEE, 2011, pp. 1–4.

[10] G. B. Wakhle, I. Aggarwal, and S. Gaba, "Synthesis and implementation of uart using vhdl codes," in *Computer, Consumer and Control (IS3C), 2012 International Symposium on*. IEEE, 2012, pp. 1–3.

[11] V. D. Veda Patil, "Implementation of spi protocol in fpga," *International Journal of Computational Engineering Research (IJCER)*, vol. 3, no. 2, pp. 142–147, 2013.

[12] T. D. Shingare and R. Patil, "Spi implementation on fpga," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 2, no. 2, pp. 7–9, 2013.

[13] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui, "Fpga implementation of i 2 c & spi protocols: A comparative study," in *Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on*. IEEE, 2009, pp. 507–510.

[14] T. P. Blessington, B. B. Murthy, G. Ganesh, and T. Prasad, "Optimal implementation of uart-spi interface in soc," in *Devices, Circuits and Systems (ICDCS), 2012 International Conference on*. IEEE, 2012, pp. 673–677.