Novel Accelerated Protocol for Faster Sensor Data Fetch in High Speed Robots

Chinmay Vilas Samak [#1], Tanmay Vilas Samak [#2]
[#] Department of Mechatronics Engineering, SRM Institute of Science and Technology
SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamil Nadu, India
[1] cv4703@srmist.edu.in
[2] tv4813@srmist.edu.in

***Abstract*— Reliable on-board state estimation demands high sensor sampling rate so as to update the CPU with present state of the robot fast enough. Since on-board CPUs often have limited computational capacity, a robot must be designed to use this resource efficiently. Current protocols force the CPUs to spend a considerable amount of time fetching sensor data, which delays the state estimation and affects crucial operations like planning and execution of control algorithms. This problem becomes predominant in high-speed robots, where state estimation needs to be extremely fast and accurate. This research, therefore, presents an enhanced communication protocol, aimed at minimizing the CPU intervention time required for fetching sensor data. The work also proposes deployment of the said communication protocol on a hardware accelerator for a drastic reduction in the CPU intervention time. The design was first simulated using Xilinx ISE Design Suite and subsequently implemented on a PSoC 5LP for hardware experimentation. The performance of serial and parallel implementations of the proposed communication protocol was analyzed comparatively against that of the standard I²C protocol.**

***Keywords* —** Computer communication, hardware acceleration, multi-sensor systems, robotics

I. INTRODUCTION

State estimation is the fundamental operation of autonomous robots, wherein the CPU collects raw data from the sensors, processes it and produces reliable values so as to estimate the present state of the robot. The estimated state is then passed on to the planner, which plans the future action(s) accordingly. Finally, the controller provides a desirable response by aptly controlling the actuation elements. In most cases, robots use multiple sensors to acquire information about various environmental parameters, which is then processed using data fusion algorithms in order to reduce the uncertainty in state estimation and make more informed decisions.

Currently, Inter-Integrated Circuit (I^2C) [1]-[9] and Serial Peripheral Interface (SPI) [9]-[12] are the two preferred communication protocols for multi-sensor data transfer. SPI is relatively fast, but has more hardware requirements than I^2C, due to which interfacing of multiple sensors becomes complicated. I^2C, on the other hand, is a bit slow but the fact that it uses only 2 lines for communication reduces the system complexity to a great extent. Other advantages of I^2C include the possibility of multi-master-multi-slave communication and acknowledgement of every byte of data transferred, which makes it an ideal protocol for interfacing multiple sensors with the CPU.

However, if I^2C is implemented on a software-level (i.e. serial implementation), the CPU wastes a considerable amount of time fetching sensor data, which introduces a time-lag derived error in state estimation. This problem becomes predominant in high-speed robots, where state estimation needs to be as fast as real-time, and a delay of even a few milliseconds can lead to catastrophic consequences.

This research, therefore, proposes the design of Robot Sensor Interface (RSI), a communication protocol derived from the I^2C protocol, which enjoys the benefits of reduced latency, and suggests its deployment on a software-level for reducing the CPU intervention time. It further suggests implementation of the RSI on a hardware-level (i.e. parallel implementation), which accelerates the sensor data fetch cycles by running parallel to the CPU, thereby increasing the sensor sampling rate and reducing the CPU intervention time drastically.

II. BACKGROUND

I^2C is a widely used low-speed, synchronous, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors) in 1982. It was originally intended to enable communication between multiple slave integrated circuits (ICs) and one or more master ICs within a short distance, and has become a de facto standard. It employs only 2 lines, serial clock (SCL) for synchronizing the communication and serial data (SDA) for transferring the data. Both the lines need to be pulled up to +Vdd using appropriate pull-up resistors. I^2C offers handshaking feature for improved error handling. It is predominantly employed where the notions of hardware simplicity and inexpensive implementation surpass the communication speed requirement.

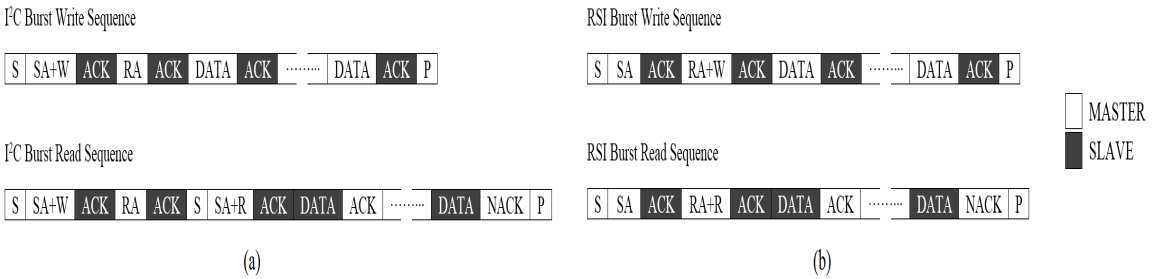

Fig. 1. (a) Standard I²C Communication Protocol with 7-bit Addressing and (b) RSI Communication Protocol

TABLE I. Legend for Figure 1

Signal	Description	
	I²C Protocol	**RSI Protocol**
S	**Start Condition:** SDA goes low while SCL is high	**Start Condition:** SDA goes low while SCL is high
SA	**7-bit Slave Address**	**8-bit Slave Address**
RA	**8-bit Internal Register Address**	**7-bit Internal Register Address**
W	**Write bit (0)**	**Write bit (0)**
R	**Read bit (1)**	**Read bit (1)**
ACK	**Acknowledge:** SDA goes low at 9th clock cycle	**Acknowledge:** SDA goes low at 9th clock cycle
NACK	**Not-Acknowledge:** SDA stays high at 9th clock cycle	**Not-Acknowledge:** SDA stays high at 9th clock cycle
DATA	**8-bit Data Packet**	**8-bit Data Packet**
P	**Stop Condition:** SDA goes high while SCL is high	**Stop Condition:** SDA goes high while SCL is high

Fig. 1 (a) represents the standard I²C protocol. The communication starts when the master node generates a start condition by pulling the SDA line low while SCL is high. The master then sends a 7-bit slave address followed by a write bit and waits for the corresponding slave to acknowledge by tri-stating the SDA line. Every slave connected to the I²C bus then compares this address with its own and upon a successful match, generates an acknowledgement (ACK) signal by pulling the SDA line low. The master then writes the internal register address to the slave and again waits for it to acknowledge. The further sequence depends upon whether the master writes to the slave, or reads from it.

In case of a write operation, the master, upon receiving acknowledgement, writes the first data byte to the slave and again waits for acknowledgement. The data transfer continues until the last bit is written to the slave and acknowledged, after which the master generates a stop condition by pulling the SDA line high while SCL is high.

Conversely, in case of a read operation, the master, upon receiving acknowledgement, generates a repeated start condition, followed by the slave address and a read bit. The slave then responds with an acknowledgement and sends the first data byte to the master and waits for it to acknowledge. The data transfer continues until the master sends a not-acknowledge (NACK) signal by pulling the SDA line high, immediately followed by the stop condition.

Fig. 1 (b) represents the proposed RSI protocol. It is quite similar to I²C; however, it comes with some significant modifications. In RSI, the internal register address is not written to the slave, instead, it is sent on the SDA line immediately following the slave address and the slave is programmed to acknowledge only if it holds a register with that particular address. This omits the requirement of a repeated start condition during the read operation, which saves 10 crucial clock pulses during a single sensor data fetch cycle. Another difference between the two is that the read/write bit in RSI is sent along with the 7-bit register address and the slave address is expanded to 8-bits, which allows interfacing of 256 slaves with a single master device. This is proposed following a general observation that, for a complex robotic system, the number of sensors (slaves) to be interfaced with the CPU is generally more than the number of internal registers within each sensor.

To sum up, RSI not only supports important I²C features such as handshaking and hardware simplicity, but also enjoys additional benefits of reduced latency and a two-fold increase in the number of slaves per master.

III. RESEARCH METHODOLOGY

A. Problem Statement

The major limitation of the I^2C protocol is its slow speed. For instance, the standard mode of I^2C protocol runs at 100 kHz clock frequency, which translates to a data transfer rate of no more than 12.5 kilobytes per second. This means that any data package larger than 12.5 bytes will reduce the sampling rate down to less than 1 kHz, which is bound to happen since most sensors have data package sizes greater than this value. This problem intensifies in case of multi-sensor systems, wherein there are various data packages to be transferred across the bus. Furthermore, due to serial implementation, the communication sub-system remains idle while the processor is functioning and vice-versa. Hence, the CPU unnecessarily wastes a lot of time waiting for sensor data in each cycle, thereby reducing the sampling rate and adversely affecting the state estimation.

B. Proposed Design

The serial implementation of the proposed system requires the incorporation of the RSI block as an intellectual property (IP) core into the architecture of the future system on chip (SoC) devices. This will enable communication using the proposed RSI protocol, similar to the I^2C protocol.

The parallel implementation, on the other hand, requires a much complex system to be developed, of which RSI block is just a part. The top-level design of such a system principally constitutes of a CPU, a hardware accelerator [13] and the RSI block itself. Detailed architecture of the described system is represented in Fig. 2, where the sensors (slaves) are assumed to be able to communicate using the proposed RSI protocol.

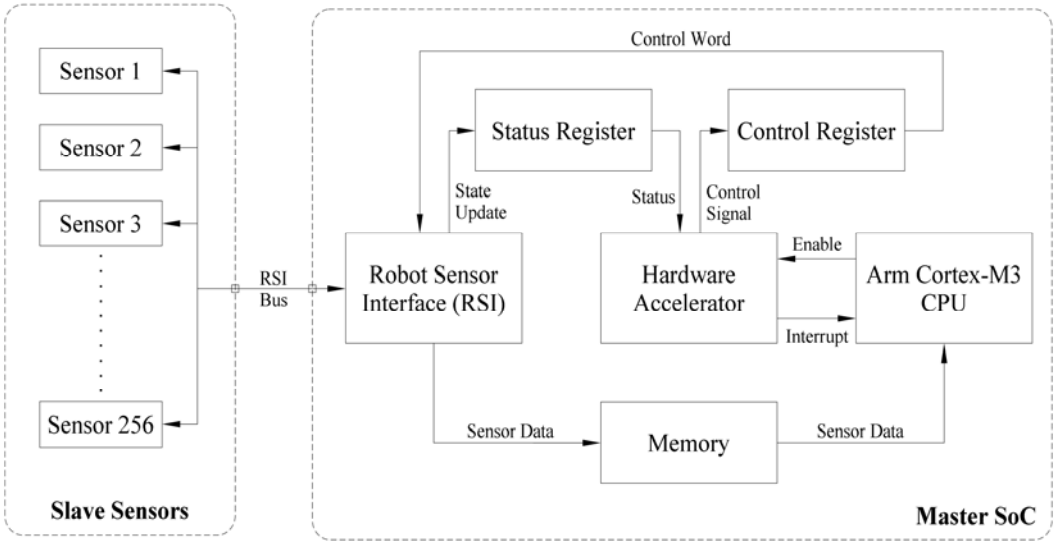

Fig. 2. Architecture of Hardware Accelerated Implementation of RSI

The entire process of hardware accelerated sensor data fetch operation is elucidated in Fig. 3, showcasing the intermediate steps of the proposed protocol in detail with reference to Fig. 1 (b) and Fig. 2. The CPU generates an enable signal to initialize the hardware accelerator, which then modifies the control register to command the RSI block to begin the sensor data fetch operation. The CPU can now perform other compute-intensive tasks while the RSI block fetches the sensor data (using the proposed RSI protocol) and loads it into the memory element (buffer) so that the CPU can easily access it as and when required (by servicing the interrupt requested by the hardware accelerator). Status of the RSI block is continuously monitored by the hardware accelerator with the help of status register. Once the entire sensor data fetch cycle is completed, the hardware accelerator generates an interrupt indicating the same and disables itself. The CPU can re-enable it whenever necessary to read the same sensor again, or a different one. A finite state machine (FSM) can be designed to sequentially read data from multiple sensors. Note that the red dashed lines in Fig. 3 represent the enable (EN) and interrupt (INT) signals, and are not an integrated part of the process flow.

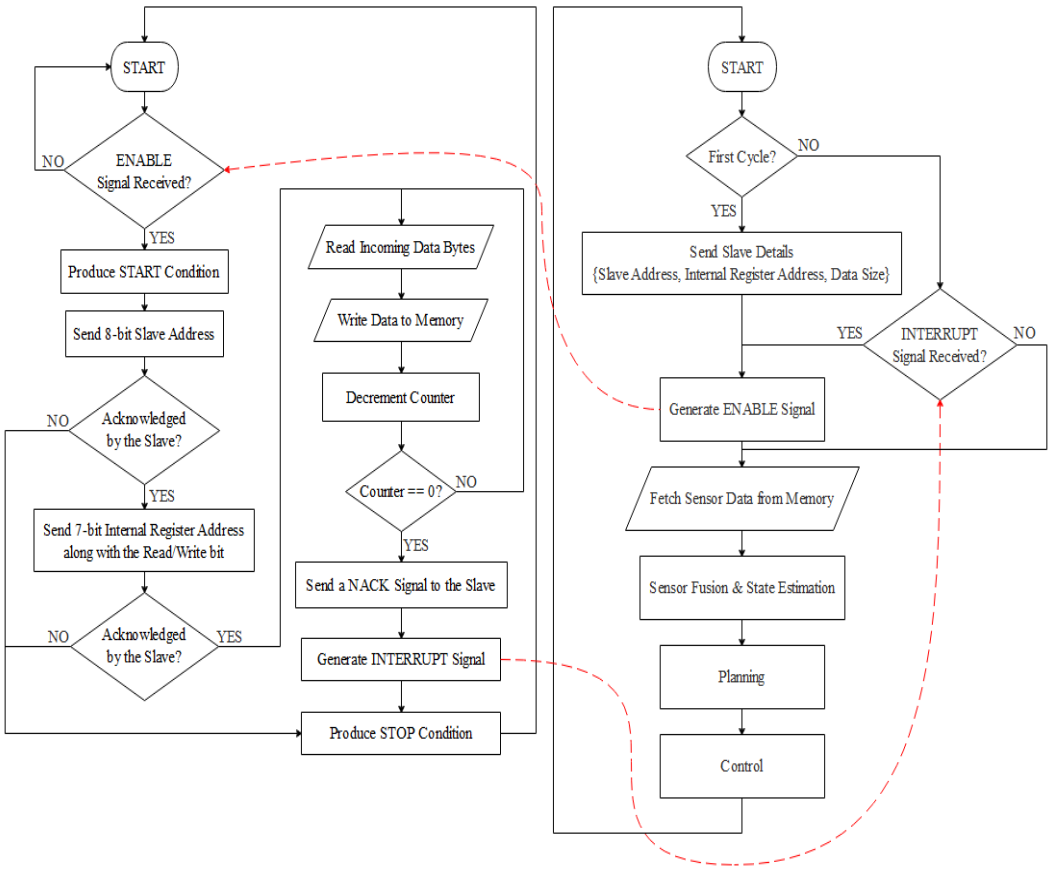

Fig. 3. Flowchart of Hardware Accelerated Implementation of RSI

C. *Software Simulation*

The I²C as well as RSI protocols were designed in Xilinx ISE Design Suite 14.7 [14] using Verilog hardware description language (HDL). The notion of finite state machines (FSMs) was used to conditionally switch between different stages of the communication protocols, depending upon the master and slave responses. The behavioral models of both the communication protocols were individually simulated using the ISim [15] simulation tool and their timing diagrams (Fig. 4 and Fig. 5) were verified. The slave algorithms were developed separately with an intention to emulate slave devices employing the I²C and RSI protocols, respectively.

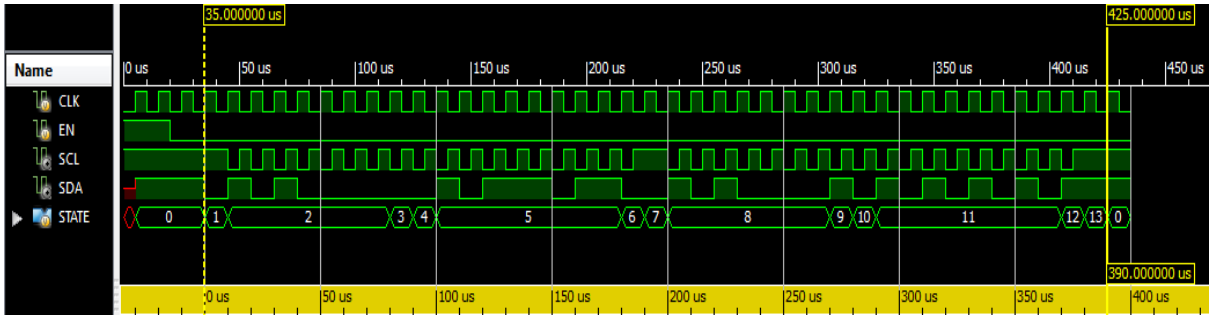

Fig. 4. Simulated Timing Diagram of I²C Communication Protocol at 100 kHz Clock Frequency

Fig. 4 shows simulation of single byte read sequence of the standard I²C protocol. Initially, the I²C block is in idle condition (State 0). The communication starts when the EN signal goes low. First, a start condition is produced (State 1) and 7-bit slave address, 0x50 is sent (State 2) along with write bit (State 3). Upon reception of acknowledgement from the slave (State 4), an 8-bit internal register address, 0xBB is written to it (State 5). The slave acknowledges again (State 6), after which, a repeated start condition is produced (State 7) and the 7-bit slave address, 0x50 is sent again (State 8), this time followed by a read bit (State 9). The slave then responds with an acknowledgement (State 10) and delivers an 8-bit data packet, 0xAA to the master (State 11), which then sends a NACK signal (State 12) to indicate completion of data transfer. The process is subsequently terminated as the master generates a stop condition (State 13).

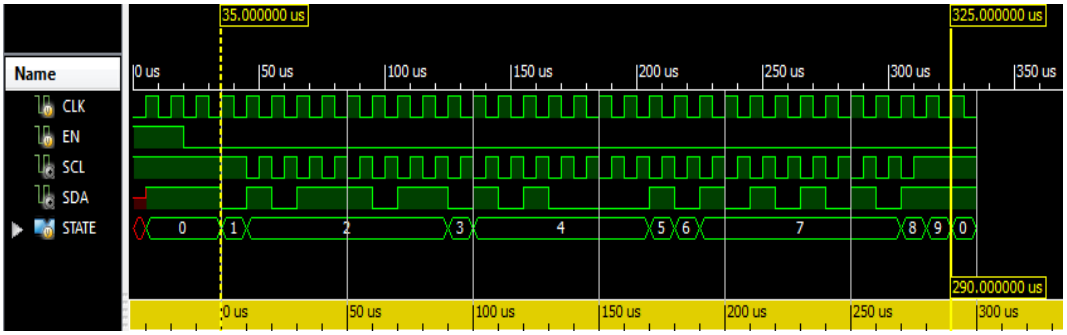

Fig. 5. Simulated Timing Diagram of RSI Communication Protocol at 100 kHz Clock Frequency

Fig. 5 shows simulation of single byte read sequence of RSI protocol with similar slave details and data to be communicated as in case of I²C protocol simulation. However, the timing diagram appears to be different in this case since the FSM built for simulation was based on Fig. 1 (b). The communication begins with a start condition (State 1) followed by an 8-bit slave address, 0xBB (State 2). Once the corresponding slave acknowledges (State 3), a 7-bit internal register address, 0x50 is sent (State 4) along with a read bit (State 5). The slave acknowledges again (State 6) and delivers an 8-bit data packet, 0xAA to the master (State 7), which then sends a NACK signal (State 8) to indicate completion of data transfer. The process is subsequently terminated as the master generates a stop condition (State 9). It is thus evident that RSI offers a latency reduction of 10 clock pulses (100 µs at 100 kHz) in a single data fetch cycle.

D. Hardware Implementation

The proposed design was implemented on the CY8CKIT-059 PSoC 5LP [16], which is a low power programmable system on chip prototyping board developed by Cypress Semiconductor. It incorporates a powerful Arm Cortex-M3 CPU, a 24-bit digital filter block (DFB), 24 universal digital blocks (UDBs) and a high-performance direct memory access (DMA) controller. PSoC Creator IDE 4.2 [17] was used to transfer the top-level design of the proposed system as functional blocks to be compiled for hardware implementation on PSoC 5LP [18]. The on-board UDBs were configured using Verilog HDL to prototype our design.

The hardware implementation was carried out in 4 stages, viz. serial I²C, serial RSI, parallel I²C and parallel RSI. First, real-time performance of the data transfer was analyzed by interfacing commercially available sensors, viz. accelerometer (ADXL345) [19], gyroscope (L3G4200D) [20] and magnetometer (HMC5883L) [21] with PSoC 5LP via I²C bus. A secondary PSoC 5LP device programmed with multi-slave algorithm was then used for comparative analysis of both the protocols.

IV. RESULTS AND DISCUSSION

The proposed RSI protocol exhibited superior behavior as compared to the standard I²C protocol. Simulation results confirmed that the RSI protocol is faster than the standard I²C protocol as it saves 10 clock pulses corresponding to 4 state transitions per cycle. This translates to the conservation of 100 µs per data fetch cycle at 100 kHz clock frequency.

The hardware implementation also yielded positive results. In case of serial implementation, the CPU wasted a lot of time waiting for sensor data until the entire communication process was finished and hence, the CPU intervention time was equal to the communication time. Since each of the three sensors (and emulated slaves) used for the hardware implementation had 3 8-bit registers holding the inertial measurements w.r.t. the 3 orthogonal axes, a total of 3-byte data was to be fetched per cycle, 1 byte from each of the 3 registers. This was accomplished by reading the 3 registers sequentially, by changing the internal register address each time. Thus, RSI protocol saved about 300 µs per sensor (slave) per data fetch cycle. The RSI protocol will prove to be even more beneficial as the number of sensors and/or internal registers increases.

On the other hand, in case of parallel implementation, the CPU intervened only during the interrupt, while the entire communication process was handled by the hardware accelerator. Therefore, the CPU intervention time was much less than the actual communication time. Furthermore, due to parallel execution, the sampling rate doubled, which in turn increased the accuracy of state estimation. The RSI protocol proved to be beneficial even in this case, since it fetched data faster than I²C, thereby reducing the load on hardware accelerator. The results obtained from the hardware experiments are presented in Table II. Note that the observations of intervention time in Table II are the statistical mode values of ten consecutive readings.

TABLE II. Results of Hardware Experiments

Communication Protocol	Master-Slave Configuration	Implementation Type	Slaves Interfaced	Hardware Accelerator Intervention Time (µs)	CPU Intervention Time (µs)
I^2C	PSoC-Sensors	Serial (Software-Level)	Acc	-	1942
			Acc + Gyro	-	4663
			Acc + Gyro + Mag	-	7365
		Parallel (Hardware Accelerated)	Acc	1926	1
			Acc + Gyro	4652	2
			Acc + Gyro + Mag	7362	3
	PSoC-PSoC	Serial (Software-Level)	1-Slave Algorithm	-	1914
			2-Slave Algorithm	-	4640
			3-Slave Algorithm	-	7356
		Parallel (Hardware Accelerated)	1-Slave Algorithm	1918	1
			2-Slave Algorithm	4643	2
			3-Slave Algorithm	7359	3
RSI	PSoC-PSoC	Serial (Software-Level)	1-Slave Algorithm	-	1617
			2-Slave Algorithm	-	4046
			3-Slave Algorithm	-	6462
		Parallel (Hardware Accelerated)	1-Slave Algorithm	1621	1
			2-Slave Algorithm	4050	2
			3-Slave Algorithm	6466	3

V. CONCLUSION

This research was conducted with a vision to develop an enhanced communication protocol for faster on-board state estimation, particularly in high-speed robots. Such a communication protocol was expected to fetch the sensor data faster and more efficiently than the existing protocols, like I^2C. A novel communication protocol, the RSI, was therefore designed with an intention to reduce the latency of standard I^2C protocol. Simulation results confirmed that RSI protocol had a latency reduction of 10 clock cycles (100 µs at 100 kHz) and was therefore faster as compared to the standard I^2C protocol. Serial implementation reduced the CPU intervention time from about 1900 µs, 4600 µs and 7300 µs for 1, 2 and 3 slaves respectively in case of standard I^2C protocol to about 1600 µs, 4000 µs and 6400 µs for the respective number of slaves in case of RSI protocol. Hardware acceleration further reduced the CPU intervention time to just 1 µs, 2 µs and 3 µs for both the protocols, but RSI still proved to be more efficient in terms of reducing the data fetch time on the hardware accelerator side. All in all, hardware accelerated RSI was elected as the most desirable protocol in terms of fast and efficient multi-sensor data fetch operations.

Future consideration of this research includes integrating the proposed protocol design as an IP core to be embedded in the upcoming SoC architectures and testing the real-time performance of the system with actual master-slave devices employing RSI.

REFERENCES

[1] UM10204 I^2C-bus specification and user manual, Rev. 6, NXP Semiconductors, Eindhoven, Netherlands, 2014.
[2] Z. Hu, "I2C Protocol Design for Reusability," 2010 Third International Symposium on Information Processing, Qingdao, 2010, pp. 83-86, doi: 10.1109/ISIP.2010.51.
[3] C. Liu, Q. Meng, T. Liao, X. Bao and C. Xu, "A Flexible Hardware Architecture for Slave Device of I2C Bus," 2019 International Conference on Electronic Engineering and Informatics (EEI), Nanjing, China, 2019, pp. 309-313, doi: 10.1109/EEI48997.2019.00074.
[4] W. Andrysiewicz, D. Kościelnik and M. Miśkowicz, "I2C hardware master serial interface for asynchronous ADCs," 2015 IEEE International Symposium on Systems Engineering (ISSE), Rome, 2015, pp. 77-81, doi: 10.1109/SysEng.2015.7302736.
[5] K. B. Bharath, K. V. Kumaraswamy and R. K. Swamy, "Design of arbitrated I2C protocol with DO-254 compliance," 2016 International Conference on Emerging Technological Trends (ICETT), Kollam, 2016, pp. 1-5, doi: 10.1109/ICETT.2016.7873672.
[6] Eswari, Bollam & Ponmagal, N. & Preethi, K. & SG, Sreejeesh. (2013). Implementation of I2C master bus controller on FPGA. 678-681. 10.1109/iccsp.2013.6577141.
[7] Płachno Łukasz, Oleksiak Marcin, Kościelnik Dariusz and Jabłeka Marek, "I2C Interface Design for Hardware Master Devices," IFAC Proceedings Volumes, vol. 45, no. 7, pp. 163-168, 2012.
[8] PSoC® Creator™ Component Datasheet, "I2C Master/Multi-Master/Slave," v3.30, Rev. *B, Cypress Semiconductor, San Jose, CA, USA, 2014.

[9] F. Leens, "An introduction to I2C and SPI protocols," in IEEE Instrumentation & Measurement Magazine, vol. 12, no. 1, pp. 8-13, February 2009, doi: 10.1109/MIM.2009.4762946.
[10] SPI Block Guide, v03.06, Motorola Inc., Schaumburg, IL, USA, 2001.
[11] S. Saha, M. A. Rahman and A. Thakur, "Design and implementation of SPI bus protocol with Built-in-self-test capability over FPGA," 2014 International Conference on Electrical Engineering and Information & Communication Technology, Dhaka, 2014, pp. 1-6, doi: 10.1109/ICEEICT.2014.6919076.
[12] D. N. Oruganti and S. S. Yellampalli, "Design of power efficient SPI interface," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), New Delhi, 2014, pp. 2602-2606, doi: 10.1109/ICACCI.2014.6968350.
[13] J. Su, "Hardware Acceleration for Sensor Data Fetch," RISS Working Papers Journal, vol. 16, pp. 129-132, Fall 2018.
[14] ISE In-Depth Tutorial, v13.1, Xilinx Inc., San Jose, CA, USA, 2011.
[15] ISim User Guide, v14.3, Xilinx Inc., San Jose, CA, USA, 2012.
[16] CY8CKIT-059 PSoC® 5LP Prototyping Kit Guide, Rev. *G, Cypress Semiconductor, San Jose, CA, USA, 2018.
[17] PSoC® Creator™ User Guide, Rev. *K, Cypress Semiconductor, San Jose, CA, USA, 2018.
[18] PSoC® 5LP: CY8C58LP Family Datasheet, Rev. *M, Cypress Semiconductor, San Jose, CA, USA, 2018.
[19] Digital Accelerometer ADXL345, Rev. 0, Analog Devices Inc., Norwood, MA, USA, 2009.
[20] L3G4200D MEMS motion sensor: three-axis digital output gyroscope, Rev. 2, STMicroelectronics, Amsterdam, Netherlands, 2010.
[21] 3-Axis Digital Compass IC HMC5883L, Rev. E, Honeywell International Inc., Morris Plains, NJ, USA, 2013.

AUTHOR PROFILE

Chinmay Vilas Samak is currently pursuing his bachelor's degree in Mechatronics Engineering at SRM Institute of Science and Technology, India, where he stands as one of the department toppers and is also a proud recipient of the Academic Excellence Scholarship. He is also associated with the Autonomous Systems Lab at the university and his research concentrates on robotics and autonomous systems. He has also been twice felicitated at the SRM Research Day for his contributions, with a silver medal in 2018 and a gold medal in 2020.

Tanmay Vilas Samak is currently pursuing his bachelor's degree in Mechatronics Engineering at SRM Institute of Science and Technology, India, where he stands as one of the department toppers and is also a proud recipient of the Academic Excellence Scholarship. He is also associated with the Autonomous Systems Lab at the university and his research concentrates on robotics and autonomous systems. He has also been twice felicitated at the SRM Research Day for his contributions, with a silver medal in 2018 and a gold medal in 2020.