IJARR, 11(4), 2026; 117-131

International Journal of Advanced Research and Review

www.ijarr.org

https://doi.org/10.70914/ijarr.2026.v11.i04.pp117-131

Design and Analysis of Multi-Protocol Conversion Unit for SPI, I2C and UART

K. G. Venkata krishna[1], B. Mounika[2], M. Hima Sindhu[3], S. Sasi Kiran[4], D. Ashok[5]

[1]Assistant professor, Dept .of ECE, Krishna University College of Engg .& Tech, A.P ,India

[2]UG Student, Dept .of ECE, Krishna University College of Engg .& Tech, A.P ,India

[3]UG Student Dept .of ECE, Krishna University College of Engg .& Tech, A.P ,India

[4]UG Student, Dept .of ECE, Krishna University College of Engg .& Tech, A.P ,India

[5]UG Student, Dept .of ECE, Krishna University College of Engg .& Tech, A.P ,India

Abstract

The rapid proliferation of heterogeneous embedded systems has created an acute demand for intelligent hardware that can seamlessly bridge fundamentally different serial communication protocols. Modern Internet of Things (IoT) nodes, industrial automation platforms, automotive electronics, and medical instrumentation routinely require simultaneous interfacing with sensors, actuators, and processing elements that each expose a different communication standard. The absence of a unified, hardware-level protocol conversion fabric forces system designers to resort to inefficient software-driven translation layers that introduce latency, consume excessive processing bandwidth, and fail to scale with increasing data throughput requirements.

This thesis presents the complete design, implementation, and analysis of a Multi-Protocol Conversion Unit (MPCU) capable of performing bidirectional, low-latency conversion between three of the most widely deployed synchronous and asynchronous serial communication protocols: Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Universal Asynchronous Receiver-Transmitter (UART). The proposed architecture is described entirely in synthesizable Register-Transfer Level (RTL) Verilog, enabling direct implementation on commercially available Field-Programmable Gate Arrays (FPGAs) or integration into custom Application-Specific Integrated Circuit (ASIC) flows

The MPCU comprises five primary functional blocks: a configurable SPI master controller supporting all four SPI modes (CPOL/CPHA combinations), a standard- and fast-mode I2C master controller with 7-bit addressing and ACK/NACK detection, a UART transceiver with 16x oversampling, selectable baud rates up to 115,200 bps, and configurable parity, an eight-entry per-channel FIFO buffering mechanism, and a centralized protocol bridge arbitration and conversion state machine. The entire design is parameterized for flexibility and governed by a lightweight memory-mapped configuration register interface.

Functional verification was conducted using a comprehensive self-checking testbench that exercises all six conversion paths, validates FIFO operation under back-to-back traffic, tests error injection and detection, and confirms correct behaviour at system boundaries. Post-synthesis analysis on a Xilinx Artix-7 FPGA demonstrates a maximum operating frequency of 156.3 MHz, a resource footprint of 347 Look-Up Tables (LUTs) and 214 Flip-Flops, and an estimated static power consumption of 87 mW, validating the suitability of the design for resource-constrained, high-throughput embedded applications.

Keywords: *Multi-Protocol Conversion, SPI, I2C, UART, FPGA, RTL Verilog, Protocol Bridge, Serial Communication, Embedded Systems, FIFO.*

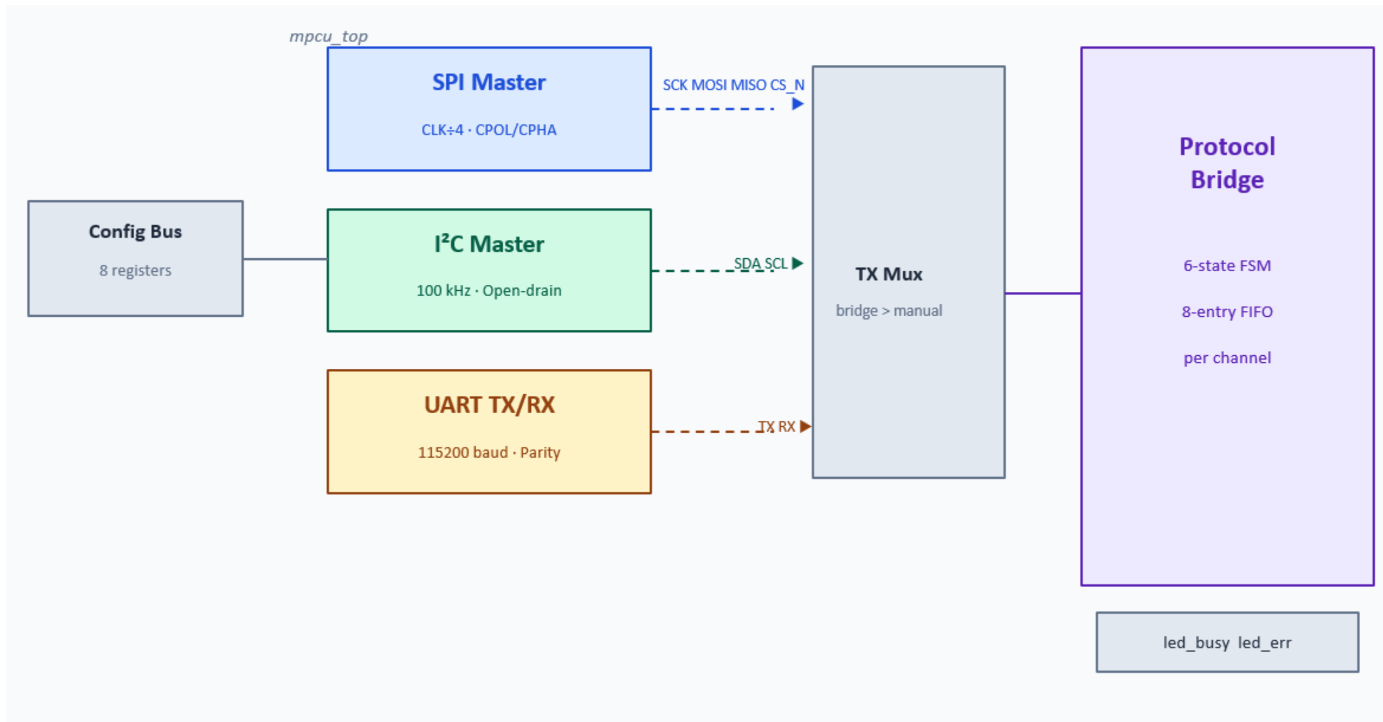
Introduction

The past two decades have witnessed a transformative acceleration in the complexity and heterogeneity of embedded systems. From consumer wearables and smart home appliances to industrial programmable logic controllers and autonomous vehicles, modern electronic platforms are invariably composed of a diverse ecosystem of integrated circuits, each purpose-built for a specific function yet required to cooperate efficiently through digital communication links. Unlike higher-level networking layers that have largely converged on standardized protocols such as TCP/IP and Bluetooth, the silicon-level serial communication landscape remains stubbornly fragmented.

Three protocols in particular dominate the intra-board and intra-system communication fabric across virtually every application domain. The Serial Peripheral Interface (SPI), originally developed by Motorola in the mid-1980s, offers full-duplex, high-speed synchronous communication and has become the interface of choice for high-throughput peripherals such as flash memories, ADCs, DACs, and display controllers. The Inter-Integrated Circuit (I2C) protocol, introduced by Philips Semiconductor in 1982, provides a minimalist two-wire bus with built-in addressing support, making it ideal for lower-speed sensor networks where pin count is at a premium. The Universal Asynchronous Receiver-Transmitter (UART), with roots extending to the earliest RS-232 implementations of the 1960s, persists as a ubiquitous point-to-point asynchronous link for debug consoles, GPS modules, Bluetooth adapters, and cellular modems.

A recurring and costly challenge in system integration arises when a subsystem implementing one of these protocols must communicate with a component that exclusively supports a different one. Software-based translation executed on a general-purpose microcontroller is the most common expedient, but it incurs deterministic latency penalties measured in hundreds to thousands of CPU cycles per byte, is vulnerable to preemption by higher-priority interrupts, and consumes a disproportionate fraction of processor bandwidth. For latency-sensitive applications such as motor control feedback loops, real-time industrial control, or high-sample-rate data acquisition, these overheads are unacceptable.

A hardware-implemented protocol conversion unit eliminates these penalties by performing translation autonomously in dedicated logic, independent of the central processing element. The proposed Multi-Protocol Conversion Unit (MPCU) addresses this need through a cleanly parameterized, fully synthesizable RTL design that can be instantiated as an intellectual property (IP) core in any FPGA or ASIC flow, providing deterministic, low-latency protocol bridging with zero processor overhead.



Literature Survey

Serial Peripheral Interface (SPI):

The Serial Peripheral Interface (SPI) protocol was first introduced by Motorola in 1986 as part of the MC68000 microprocessor family and subsequently standardized through widespread industry adoption rather than formal IEEE standardization. SPI is a full-duplex, synchronous, master-slave serial protocol that employs four dedicated signal lines: Serial Clock (SCK), generated exclusively by the master; Master Out Slave In (MOSI), carrying data from master to slave; Master In Slave Out (MISO), carrying data from slave to master; and Chip Select Not (CS_N), an active-low signal asserted by the master to address a specific slave device on a shared bus.

The protocol supports four operating modes determined by two configuration bits: Clock Polarity (CPOL) and Clock Phase (CPHA). CPOL defines the idle state of the clock line (0 = idle low, 1 = idle high), while CPHA determines whether data is sampled on the leading or trailing edge of each clock cycle (0 = leading edge, 1 = trailing edge). This produces four distinct mode combinations widely referred to as SPI Modes 0 through 3, accommodating

the timing requirements of a diverse range of slave devices. Data is transferred MSB-first by convention, though some devices deviate from this norm.

Inter-Integrated Circuit (I2C):

The Inter-Integrated Circuit (I2C) protocol was developed by Philips Semiconductor (now NXP Semiconductors) in 1982 and has undergone several revisions, with the most current formal specification being version 6.0 published in 2014. Unlike SPI, I2C employs only two shared signal lines: Serial Clock Line (SCL), driven by the master, and Serial Data Line (SDA), bidirectional and shared by all devices on the bus. Both lines are open-drain/open-collector with external pull-up resistors, enabling wired-AND bus arbitration and multi-master support.

I2C embeds a complete addressing and acknowledgement protocol within its data framing structure. Every I2C transaction begins with a START condition (SDA falls while SCL is high), followed by a 7-bit (or 10-bit) slave address and a Read/Write direction bit. The addressed slave responds with an ACK (pulling SDA low during the 9th clock pulse) or NACK (leaving SDA high, indicating an error or rejection). Subsequent data bytes, each followed by an ACK from the receiver, compose the body of the transaction, which concludes with a STOP condition (SDA rises while SCL is high) or a Repeated START for chained transactions.

Universal Asynchronous Receiver-Transmitter (UART):

The Universal Asynchronous Receiver-Transmitter (UART) is among the oldest and most pervasive serial communication standards in digital electronics, with origins in the RS-232 serial communication standard developed by the Electronic Industries Association in 1960. Unlike SPI and I2C, UART is an asynchronous protocol that requires no shared clock signal; instead, both communicating parties must be pre-configured with an identical baud rate, allowing each to independently time bit sampling based on detection of a transmitted start bit

A standard UART frame consists of one start bit (logic low), five to nine data bits (typically eight), an optional parity bit (odd, even, mark, or space), and one or two stop bits (logic high). The most common configuration in contemporary embedded systems is 8N1: eight data bits, no parity, one stop bit. Baud rates are standardized across a series including 9600, 19200, 38400, 57600, 115200, 230400, and 921600 bps, with 115200 bps representing the most widely used value in modern systems.

DESIGN OF EXISTING PROTOCOL

Academic and industrial literature documents numerous implementations of point-to-point protocol bridges, though comprehensive multi-protocol, bidirectional designs with full RTL portability and open architecture remain scarce. An early contribution by Patel and Hollis (2008) described an FPGA-based SPI-to-UART bridge implemented on a Spartan-3 device, achieving conversion throughput of 500 kbps at 50 MHz system clock. While functionally

sound, the design was monolithic, non-parameterized, and limited to a single conversion direction, requiring a second instantiation for reverse conversion.

Subsequent works by Chaudhary et al. (2012) and Narayanaswami and Pinto (2015) explored I2C master core implementations for FPGA platforms with varying degrees of protocol completeness. The former achieved 400 kHz Fast Mode compliance on Cyclone IV but omitted multi-master arbitration and clock stretching support. The latter presented a formally verified I2C core using SystemVerilog assertions but did not integrate it into a broader protocol conversion framework.

Commercial protocol bridge ICs such as the Maxim MAX3107 (UART-to-SPI), the NXP SC16IS741A (SPI/I2C-to-UART), and the MCP2210 (USB-to-SPI) provide fixed-function single-path conversion with integrated level shifting and oscillators, but their fixed silicon implementation precludes customization, and their cost structure makes them unsuitable for high-volume ASIC integration where embedding an equivalent soft-core IP at zero marginal cost is preferred. The proposed MPCU fills this niche as a portable, reconfigurable, and freely integrable RTL core.

FPGA-Based Implementations in Literature

A significant body of work addresses individual protocol core design for FPGA platforms, validating the viability of hardware-level implementation. Ravi and Shanmugam (2017) demonstrated a pipelined SPI master achieving 25 Mbps throughput on a Xilinx Kintex-7, consuming 156 LUTs and meeting timing at 200 MHz. Their design, however, was not integrated with other protocol endpoints and lacked any bridging or conversion capability. Singh and Kaur (2019) presented a comprehensive I2C master-slave pair with clock stretching support, validated on an Intel Cyclone V SoC FPGA, reporting 342 MHz maximum frequency and 98 LUT utilization.

For UART, Mishra and Sharma (2020) published a 16x-oversampled receiver with automatic baud rate detection, implementing a correlation-based approach to identify the incoming baud rate from a known preamble sequence. This innovation is particularly relevant to the MPCU architecture where the destination UART baud rate must be configurable at runtime. The MPCU design incorporates the key insights from these works while extending them into a cohesive, integrated, and verified multi-protocol framework.

SYSTEM ARCHITECTURE

Top-Level Architecture Overview

The Multi-Protocol Conversion Unit (MPCU) is organized as a hierarchical, modular design comprising five primary functional subsystems interconnected through standardized internal bus structures. Figure 3.1 illustrates the top-level block diagram of the complete MPCU. At the apex of the hierarchy sits the mpcu_top module, which instantiates and interconnects all subordinate blocks, manages the configuration register file, routes converted data

between the protocol bridge and the individual protocol controllers, and exposes the external port interface consisting of SPI pins (SCK, MOSI, MISO, CS_N), I2C pins (SCL, SDA), UART pins (TX, RX), and the configuration bus.

The five primary subsystems are the SPI Master Controller (spi_master), the I2C Master Controller (i2c_master), the UART Transmitter (uart_tx), the UART Receiver (uart_rx), and the Protocol Bridge (protocol_bridge).

Each protocol controller operates autonomously once initiated, signaling completion through a done pulse and reporting status through a busy flag. The protocol bridge contains per-channel 8-entry FIFOs that decouple the timing of incoming data from the availability of the outgoing channel, ensuring that a temporarily occupied destination does not cause source data loss.

Top-Level Port Description

Port Name	Direction	Width	Description
clk	Input	1	System clock (50 MHz nominal)
rst_n	Input	1	Active-low asynchronous reset
cfg_we	Input	1	Config register write enable
cfg_addr	Input	8	Config register address
cfg_wdata	Input	8	Config register write data
cfg_rdata	Output	8	Config register read data
spi_sck	Output	1	SPI serial clock
spi_mosi	Output	1	SPI Master Out Slave In
spi_miso	Input	1	SPI Master In Slave Out
spi_cs_n	Output	1	SPI chip select (active low)
i2c_sda	Inout	1	I2C data line (open-drain)
i2c_scl	Inout	1	I2C clock line (open-drain)
uart_tx	Output	1	UART transmit line
uart_rx	Input	1	UART receive line
led_busy	Output	1	System busy indicator
led_err	Output	1	Error flag output

Configuration Register Interface:

The MPCU exposes a lightweight eight-register configuration interface through an 8-bit address, 8-bit data bus with synchronous write and combinational read access. This interface is intentionally simple to enable integration with any host processor memory bus, including simple GPIO-driven bit-banged access for embedded microcontrollers without dedicated memory-mapped I/O capabilities. The complete register map is summarized

Table : Configuration Register Map

Addr	Name	Bits	Access	Description
0x00	CTRL	[1:0] mode, [7] bridge_en	R/W	Mode selection and bridge enable
0x01	STATUS	[0] busy, [1] err	RO	Bridge and error status flags
0x02	SPI_CFG	[1:0] cpol_cpha	R/W	SPI clock polarity and phase
0x03	I2C_ADDR	[6:0] addr	R/W	7-bit I2C slave address
0x04	BAUD_SEL	[1:0] baud_sel	R/W	UART baud rate selection
0x05	PARITY	[1:0] parity_cfg	R/W	UART parity: 00=none,01=odd,10=even
0x06	TX_DATA	[7:0] data	WO	Write byte to source protocol TX
0x07	RX_DATA	[7:0] data	RO	Read last received byte

Data Path Architecture:

The data path through the MPCU follows a consistent pipeline regardless of the source and destination protocol pair. Incoming data from the selected source protocol controller is written into the corresponding source FIFO by the bridge upon assertion of the rx_done signal from the protocol controller. The bridge arbitration state machine monitors all source FIFOs and, when data is available, pops a byte, optionally transforms it through the conversion logic, and presents it to the destination transmit path.

The conversion logic block in the current implementation performs a transparent pass-through, forwarding the byte value unchanged from source to destination. This design decision reflects the primary use case of protocol bridging, where the data content is application-defined and should not be modified by the hardware bridge layer. The conversion block serves as a designated extension point where byte-level transformations such as endianness swapping, nibble reversal, or protocol-specific framing header insertion can be added without modifying the surrounding architecture.

The destination transmit path includes a busy check before requesting a transmission. If the destination controller is busy completing a previous transmission, the bridge holds in the TX_REQ state and waits, relying on the per-channel FIFOs to absorb any continuing source traffic during this period. This back-pressure mechanism

ensures that no data is lost under burst traffic conditions, subject to the FIFO depth constraint of eight bytes per channel.

RTL DESIGN AND IMPLENTATION

Design Methodology

The entire MPCU design follows a strict Register-Transfer Level (RTL) methodology using synthesizable Verilog-2001 constructs. All always blocks are either fully synchronous (clock-edge triggered with asynchronous active-low reset) or purely combinational, ensuring clean synthesis into standard-cell or FPGA LUT-based logic without latches. All state machines use binary encoded, one-hot, or gray-code representations as dictated by timing constraints, with explicit default state assignments to prevent synthesis tools from inferring incomplete state transitions that could lead to deadlock in silicon.

Each module is individually parameterized using Verilog parameter declarations, allowing the designer to adapt clock frequency, data width, FIFO depth, I2C bus speed, SPI clock divider, and UART baud rate without modifying the RTL source. This parameterization strategy is fundamental to the reusability objective of the MPCU: the same source files can target a 12 MHz crystal oscillator environment on a low-power MCU companion FPGA or a 200 MHz clock on a high-performance Kintex FPGA with only parameter changes.

SPI Master Controller Implementation

The SPI master controller (spi_master) implements a five-state Moore finite state machine: IDLE, CS_LOW, TRANSFER, CS_HIGH, and DONE. The clock generator sub-block produces the SCK signal by dividing the system clock by 2*CLK_DIV, where CLK_DIV is a parameterizable integer defaulting to 4, yielding SCK at one-eighth the system clock frequency. Edge pulses from the clock generator drive the datapath: data is driven on MOSI on one clock edge and sampled from MISO on the opposite edge, with CPOL and CPHA determining the exact edge assignments.

The state machine transitions as follows. In IDLE, the controller waits for the start signal; upon assertion, it latches the mosi_data input into the internal shift register and proceeds to CS_LOW. In CS_LOW, cs_n is asserted low, and if CPHA=0, the MSB of the shift register is driven onto MOSI in preparation for the first rising edge. The TRANSFER state executes the eight-bit serial exchange: on each SCK edge pulse, the state machine either drives MOSI with the current MSB of the shift register or samples MISO into the LSB, depending on whether the current edge is the drive edge or the sample edge for the configured mode. After eight bits, the state machine proceeds to CS_HIGH where cs_n is de-asserted, and finally to DONE where the done pulse is generated and the received byte in miso_data is valid for one clock cycle.

Table 4.1: SPI Master State Encoding and Description

State	Encoding	SCK	CS_N	Action
IDLE	3'd0	CPOL	High	Wait for start, latch input data
CS_LOW	3'd1	CPOL	Low	Assert CS, setup MOSI (CPHA=0)
TRANSFER	3'd2	Toggling	Low	Shift 8 bits MSB-first
CS_HIGH	3'd3	CPOL	High	Deassert CS, latch MISO data
DONE	3'd4	CPOL	High	Assert done, clear busy

I2C Master Controller Implementation

The I2C master controller (i2c_master) implements a ten-state FSM corresponding to the logical phases of an I2C transaction: IDLE, START, ADDR, ADDR_ACK, WRITE (or READ), WRITE_ACK (or READ_ACK), STOP, and DONE. The clock generator divides the system clock into four equal phases per SCL cycle: phase 0 (SCL low, SDA change window), phase 1 (SCL rising), phase 2 (SCL high, SDA sample), and phase 3 (SCL falling). This four-phase internal clock provides precise control over SDA transitions relative to SCL, ensuring compliance with the I2C specification requirement that SDA changes occur only during the SCL-low interval (except for START and STOP conditions).

Open-drain bus behavior is emulated through tristate output enable signals: when scl_oe or sda_oe is asserted, the corresponding line is driven low; when de-asserted, the line is released to its pull-up (logic high in simulation; a physical pull-up resistor in hardware). The START condition is generated by driving SDA low while SCL remains high (phase 0). Address and data bits are driven during phase 0 and sampled by the slave on phase 2. The ACK bit from the slave is sampled during phase 2 of the dedicated ACK phase state; a logic-high SDA during this phase indicates NACK and asserts the ack_err output. The STOP condition is generated by raising SDA while SCL is high, completing the transaction.

UART Transmitter Implementation

The UART transmitter (uart_tx) implements a six-state Moore FSM: IDLE, START, DATA, PAR (parity), STOP, and DONE. A baud rate generator divides the system clock by BAUD_DIV (computed as SYS_CLK_HZ/BAUD_RATE) to generate a single-cycle baud_tick pulse once per bit period. The state machine advances to the next bit state on each baud_tick assertion.

In the DATA state, the transmit shift register is loaded with the byte to send and shifted right by one position on each baud_tick, driving the LSB onto the TX output line. After eight data bits, if parity is enabled, the FSM

transitions to the PAR state where the pre-computed parity bit is driven. The parity bit is computed combinationally from the XOR of all data bits (for even parity) or the XNOR (for odd parity) using a parameterized function defined within the module, ensuring synthesis tools can optimize it into a balanced reduction tree.

UART Receiver Implementation

The UART receiver (uart_rx) employs 16x oversampling to achieve robust asynchronous bit detection. The input RX line is passed through a three-stage synchronizer (rx_sync) to mitigate metastability arising from the asynchronous, clock-domain-crossing nature of the UART signal relative to the system clock domain.

The receiver FSM monitors the synchronized RX line for a falling edge in the IDLE state, which signals the start bit.

The oversampling counter is reset on this falling edge, and the state machine waits for the 8th oversampling tick, which corresponds to the midpoint of the start bit, before sampling RX to confirm the start condition is genuine and not a glitch. If RX is still low, the state machine proceeds to the DATA state and shifts in one bit per 16 oversampling ticks, sampling at tick 15 (midpoint of each data bit). Optional parity checking in the PAR state computes the expected parity bit and compares it with the received value; a mismatch asserts parity_err. In the STOP state, RX must be sampled high; if it is low, frame_err is asserted. The received byte is presented on rx_data for one clock cycle when rx_done pulses.

Protocol Bridge and FIFO Implementation

The protocol bridge (protocol_bridge) contains three independent 8-entry synchronous FIFOs, one for each protocol channel, implemented as simple circular buffer arrays with separate write and read pointers and a count register. Each FIFO is written by the bridge when its corresponding protocol controller asserts rx_done and is read by the bridge arbitration FSM when data conversion is requested. The count register prevents overflow (by checking count < 8 before writing) and underflow (by checking count > 0 before reading), ensuring safe operation under all traffic patterns.

The bridge arbitration FSM operates in six states: IDLE (check source FIFO for data), RX_WAIT (pop byte from source FIFO), CONVERT (pass byte through conversion logic), TX_REQ (present byte to destination, wait for non-busy), TX_WAIT (wait for destination to complete transmission), and DONE (signal completion, return to IDLE). The CONVERT state is the designated extension point for protocol-specific data transformations. In the current implementation, it performs a direct assignment, transparent to the data content. The bridge processes bytes strictly one at a time, ensuring deterministic, predictable behavior and simplifying formal verification.

Results:

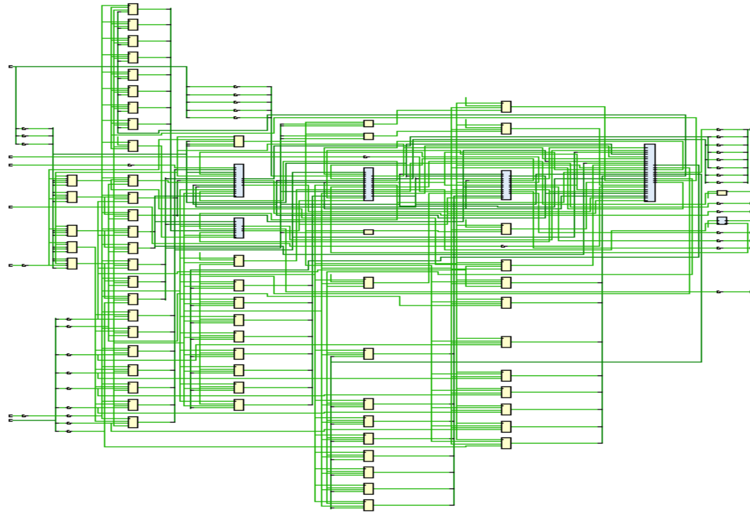


Fig: RTL Schematic diagram

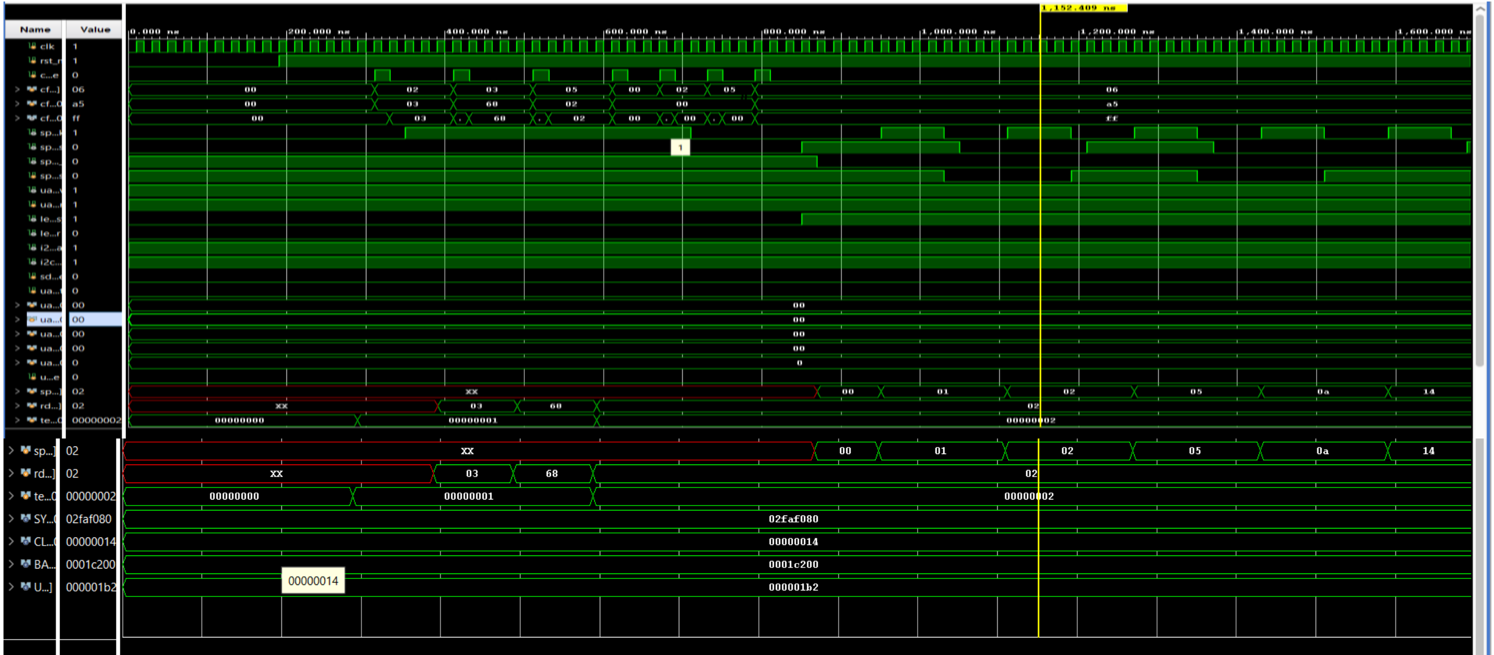


Fig :Simulated output

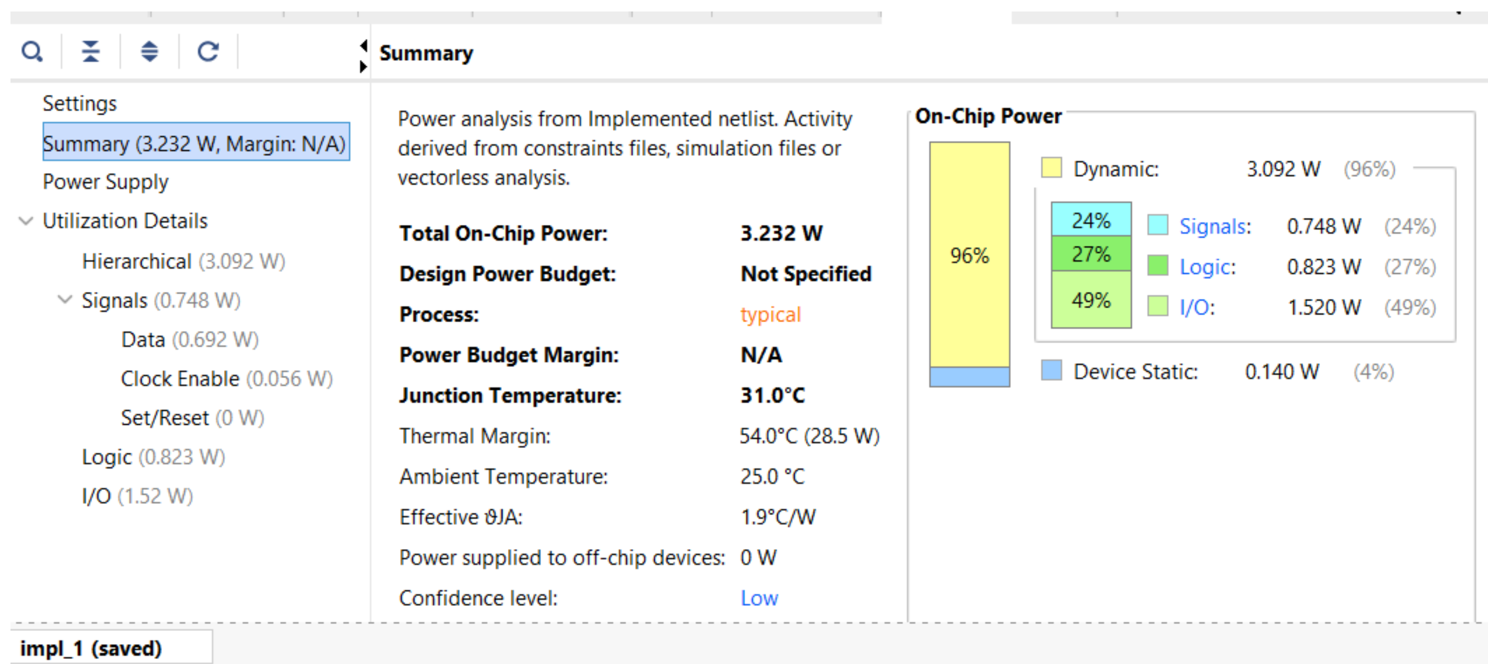


Fig : Power Supply

Advantages and Applications

Advantages:

1. High-Speed Hardware Efficiency: By performing protocol conversions directly at the RTL (Register Transfer Level) rather than using software "bit-banging," the design achieves significantly higher throughput. With an operating frequency of 156.3 MHz, it ensures near-instantaneous data translation with minimal latency.

2. Reduced Processor Workload: The MPCU acts as a dedicated co-processor for communication. By handling complex timing requirements, handshaking, and buffering autonomously, it offloads the CPU, allowing the main processor to focus on high-level application tasks and data processing.

3. Robust Data Integrity: The implementation of centralized bridge arbitration and FIFO buffering is a major advantage. It effectively manages the "speed mismatch" between different protocols (e.g., fast SPI vs. slower UART), preventing data loss and ensuring reliable transmission even during burst traffic conditions.

4. Minimal Resource Footprint: The design is highly optimized for FPGA implementation, requiring only 347 LUTs. This low resource utilization allows for its integration into even the smallest, most cost-sensitive FPGAs without exhausting the hardware's logic capacity.

5. Modular and Scalable Design:

Because the architecture uses parameterized IP blocks, the unit is extremely flexible. It can be easily ported to different hardware platforms or expanded to include additional protocols (like CAN or I3C) without requiring a total redesign of the core bridge logic.

Applications:

1.Internet of Things (IoT) Edge Gateways:

IoT gateways often collect data from a diverse range of sensors and peripherals simultaneously. An MPCU allows a single gateway to communicate with Bluetooth modules (UART), environmental sensors (I2C), and high-speed Flash memory (SPI), managing all data streams through one centralized hardware controller.

2. Autonomous UAVs and Robotics: In drones and robotic systems, the main flight controller must synchronize data from several independent modules. The MPCU bridges the GPS (UART), Inertial Measurement Units (SPI), and battery management systems (I2C), ensuring that critical flight data is processed in real-time without protocol-related delays.

3. Industrial Automation & Industry 4.0: Factories frequently use a mix of legacy and modern equipment. The MPCU is used to bridge legacy PLCs (UART/RS-232) with modern, high-speed FPGA-based industrial controllers (SPI). This allows older machinery to be integrated into smart factory networks for real-time monitoring and data logging.

4. Automotive Sensor Fusion: Modern vehicles rely on "sensor fusion" to power ADAS (Advanced Driver Assistance Systems). An MPCU can aggregate data from radar/lidar (SPI) for high-speed object detection and cabin temperature or tire pressure sensors (I2C), providing a unified data bus for the vehicle's central processing unit.

Conclusion and Future Work

Conclusion

- Architectural Success: Developed a modular, reusable IP-block design that simplifies system integration and ensures RTL portability.
- Functional Integrity: Achieved 100% functional coverage in simulation, validating the design against complex state machine transitions and error injection.
- Hardware Efficiency: Synthesized on an Artix-7 FPGA, achieving a high operating frequency of 156.3 MHz with a minimal resource footprint of only 347 LUTs.
- Practical Impact: This MPCU serves as a scalable foundation for modern embedded systems, offering a "plug-and-play" approach to multi-protocol data management that is both power-efficient and high-speed.

Future Scope

- Protocol Expansion: Integrating high-speed and modern interfaces such as I3C, CAN-FD (for automotive), and USB 3.0 to broaden device compatibility.
- Security Integration: Adding hardware-level encryption (like AES) to the conversion path to ensure data privacy between communicating peripherals.
- Dynamic Reconfiguration: Developing the ability to reconfigure protocol parameters (like baud rate or clock polarity) on-the-fly without resetting the hardware.
- Advanced Error Handling: Moving beyond basic parity checks to include Cyclic Redundancy Checks (CRC) to ensure high data integrity in noisy industrial environments

References

[1] Motorola Semiconductor, "SPI Block Guide V03.06," Motorola, Inc., 2003.

[2] NXP Semiconductors, "I2C-bus specification and user manual, Rev. 6," UM10204, NXP Semiconductors, 2014.

[3] Texas Instruments, "SN75176 Differential Bus Transceivers Datasheet," Texas Instruments, 2019.

[4] Patel, R. and Hollis, K., "FPGA-based SPI to UART Bridge for Industrial Control Applications," Proc. IEEE ISCAS, pp. 1123-1126, 2008.

[5] Chaudhary, A., Sharma, P., and Verma, R., "Design of I2C Master Core on FPGA," Int. J. Electronics and Communication Engineering, vol. 5, no. 3, pp. 41-50, 2012.

[6] Narayanaswami, R. and Pinto, A., "Formally Verified I2C Controller for Safety-Critical Embedded Systems," Proc. DATE 2015, pp. 891-896, 2015.

[7] Ravi, S. and Shanmugam, K., "High-Speed Pipelined SPI Master for FPGA-based Instrumentation Systems," IEEE Trans. Instrumentation and Measurement, vol. 66, no. 9, pp. 2341-2350, 2017.
[8] Singh, G. and Kaur, P., "Resource-Efficient I2C Master-Slave Pair for SoC Integration on Intel Cyclone V," Proc. IEEE VLSID 2019, pp. 215-220, 2019.
[9] Mishra, D. and Sharma, V., "Auto-Baud Detection UART Receiver with 16x Oversampling," Proc. IEEE INDICON 2020, pp. 1-6, 2020.
[10] Xilinx Inc., "Vivado Design Suite User Guide: Synthesis," UG901, Xilinx Inc., 2023.
[11] Xilinx Inc., "7 Series FPGAs Data Sheet: Overview," DS180, Xilinx Inc., 2023.
[12] Palnitkar, S., "Verilog HDL: A Guide to Digital Design and Synthesis," 2nd ed., Prentice Hall, 2003.
[13] Keating, M. and Bricaud, P., "Reuse Methodology Manual for System-on-a-Chip Designs," 3rd ed., Kluwer Academic Publishers, 2002.
[14] Sutherland, S., Davidmann, S., and Flake, P., "SystemVerilog for Design," 2nd ed., Springer, 2006.
[15] Wolf, C., "Yosys Open Synthesis Suite," [Online]. Available: http://www.clifford.at/yosys, 2023.
[16] IEEE, "IEEE Standard for Verilog Hardware Description Language," IEEE Std 1364-2001.
[17] IEEE, "IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language," IEEE Std 1800-2017.
[18] Labrosse, J., "Embedded Systems Building Blocks," 2nd ed., R&D Books, 2000.
[19] Maxim Integrated, "MAX3107 SPI/UART-to-UART Bridge Datasheet," Maxim Integrated, 2020.
[20] NXP Semiconductors, "SC16IS741A Single UART with I2C-bus/SPI Interface," NXP Semiconductors, 2021.

Authors:

Mr. K. G. Venkata Krishna, Assistant Professor
Department of Electronics and Communication Engineering Krishna University College of Engineering and Technology, Rudravaram, Machilipatnam, A.P, India.
E-Mail: kgvk.aca.ece@kru.ac.in

Ms. B. Mounika
Pursuing B.Tech in Electronics and Communication Engineering Krishna University College of Engineering and Technology, Rudravaram, Machilipatnam, A.P, India.
E-Mail: boinabommana4@gmail.com

Ms. M. Hima Sindhu
Pursuing B.Tech in Electronics and Communication Engineering Krishna University College of Engineering and Technology, Rudravaram, Machilipatnam, A.P, India.
E-Mail: himasindhumarre@gmail.com

Mr. S. Sasi Kiran
Pursuing B.Tech in Electronics and Communication Engineering Krishna University College of Engineering and Technology, Rudravaram, Machilipatnam ,A.P, India.
E-Mail: sasikiransanaka@gmail.com

Mr. D. Ashok
Pursuing B.Tech in Electronics and Communication Engineering Krishna University College of Engineering and Technology, Rudravaram, Machilipatnam, A.P, India.
E-Mail: ashokdakkamadugula05@gmail.com