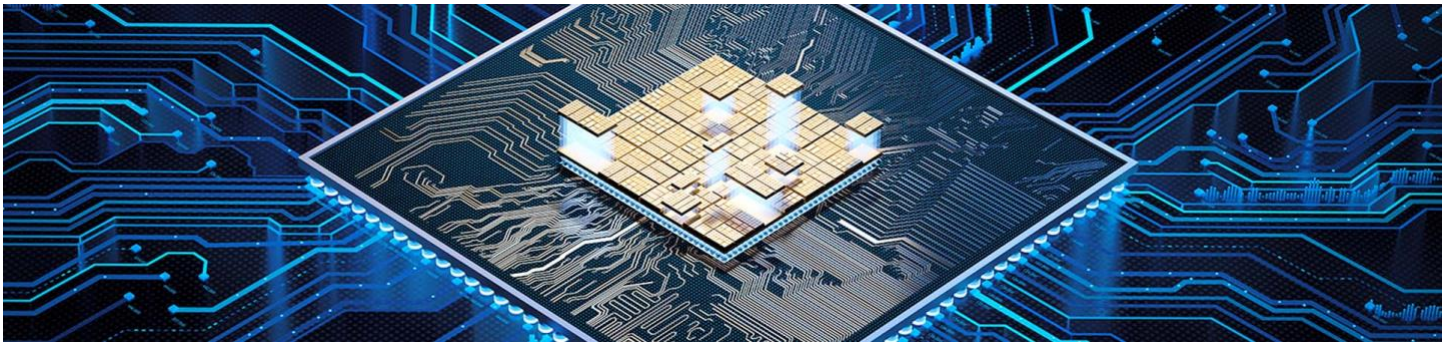
HYPER MISO SERIAL PERIPHERAL INTERFACE

Niranjan Anant Bhat – Sr. Solution Engineer Electronics, Altair / Tejas P Karanji – Electronic System Design Team, Altair / October 2025



Introduction

Buffering high-resolution images with minimal latency is a cornerstone of real-time vision systems, facilitating efficient data transfer from buffers to processors in demanding applications such as autonomous vehicles, drones, robotics, Internet of Things (IoT), and industrial automation. Yet traditional protocols struggle to meet these demands. Inter-Integrated Circuit (I²C) lacks bandwidth; Controller Area Network (CAN) is robust but slow, and Serial Peripheral Interface (SPI), though faster, is constrained by its single-data-line architecture. High-speed protocols, such as Mobile Industry Processor Interface (MIPI) - Camera Serial Interface 2(CSI-2), introduce significant complexity, cost, and power overhead, making them less suitable for lightweight embedded systems.

This paper introduces an alternative approach, leveraging a Hyper-Master in Slave Out (MISO) SPI architecture that enables parallel data reception across multiple lines, as shown in Fig. 1. This increases throughput without raising the SPI clock frequency, delivering a simple, scalable, and efficient solution for high-speed image and sensor data transmission in resource-constrained environments. Altair® DSim™ performed RTL simulation and verification, while Altair® Silicon Debug Tools™ supported schematic visualization and signal-level debugging.

Challenges

Bandwidth Limitations in Traditional Serial Protocols

Standard protocols like I²C, CAN, and single-line SPI lack the bandwidth required for the real-time transmission of high-resolution image data, especially in latency-sensitive applications such as autonomous systems and industrial vision systems.

Balancing Throughput with Simplicity

High-speed interfaces like MIPI-CSI-2 offer the necessary bandwidth but come at the cost of increased complexity, power consumption, and design overhead, making them less suitable for lightweight or low-power embedded systems.

SPI Architecture Constraints

While SPI is simple and faster than I²C or CAN, its single-data-line structure inherently limits throughput, making it suboptimal for high-speed streaming of sensor or image data.

Low-Latency Data Reception in Embedded Environments

Achieving high-speed, low-latency data reception without escalating system complexity or increasing clock frequency is challenging in embedded systems with tight performance, cost, and power constraints.

Problem formulation

Designing a Scalable Parallel Data Interface

Developing a Hyper MISO SPI architecture introduces new challenges in synchronization, timing alignment, and data reconstruction on the master side while maintaining compatibility with existing SPI control mechanisms.

Efficient RTL Simulation and Verification

Implementing and verifying a Hyper MISO SPI protocol required precise RTL modeling and rigorous simulation. Altair® DSim™ ensured timing correctness and functional reliability.

Debugging Parallel Data Streams

Multi-line, parallel SPI data reception demanded advanced signal-level debugging tools. Altair® Silicon Debug Tools™ were essential for visualizing data paths as shown in Fig. 2, validating synchronization and debugging edge cases during transmission and reconstruction.

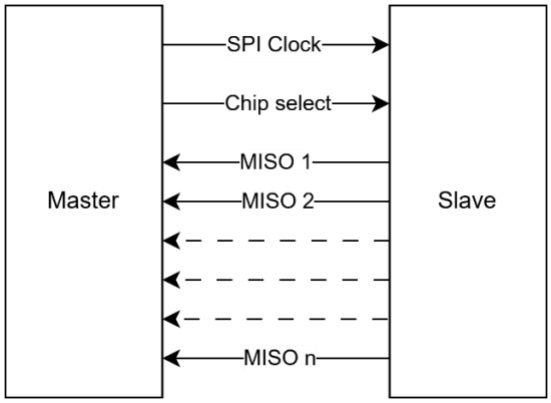


Fig. 1. Hyper MISO SPI protocol structure

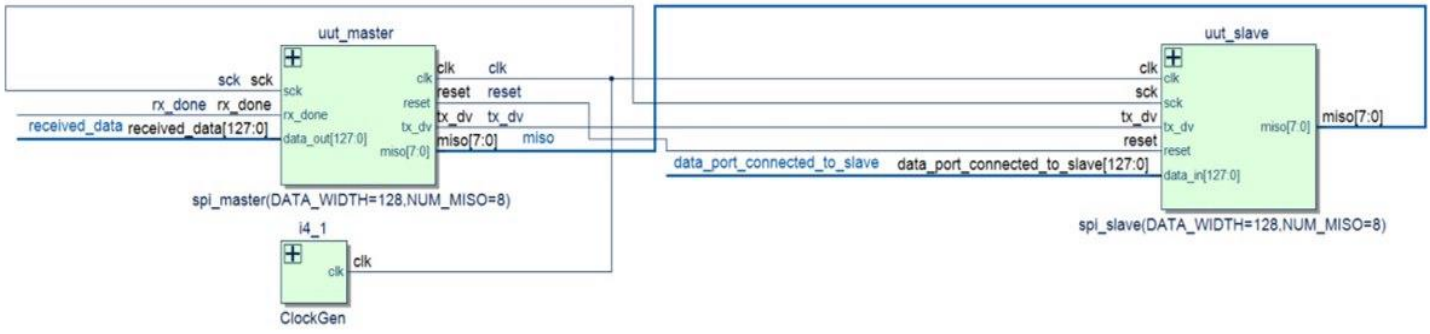


Fig. 2. 8-MISO SPI protocol structure viewed through Altair® Silicon Debug Tools™

Implementation details

The proposed **Hyper MISO SPI protocol** introduces architectural and logic-level enhancements compared to solutions presented in the reference *[1],* optimized for **simulation-based image data buffering** in lightweight embedded systems. The RTL design and testbench are developed in **SystemVerilog.** The architecture of the proposed solution is depicted in Figs. 3 and 4.

Key aspects of proposed implementation in comparison to [1]:

- **Data Line Complexity and Speed:** While [1] uses 8 channels selected via a chip select (CS), each having 16 MISO lines operating at 5 Mbps, the proposed design adopts a single channel with 8 MISO lines operating at 25 MHz. This approach reduces complexity and area, making the design well-suited for high-speed, low-power embedded applications.

- **Transfer Data Valid Signal**: A custom tx_dv trigger signal was implemented to explicitly initiate data transfers, enabling more precise control compared to the conventional CS-based triggering method in [1].

- **Synchronized Data Reception**: Dedicated synchronization logic was implemented to achieve accurate reception across all 8 MISO lines using a shared SPI clock, ensuring precise clock-domain alignment and enhanced timing reliability compared to [1].

- **Functional test**: The testbench developed in this paper enables image data transfer over parallel MISO lines, with reconstruction performed within the simulation environment. This approach provides a more targeted and verifiable framework compared to [1], which does not use any automated data verification. In contrast, the proposed framework validates the transmission by comparing the input and output images to ensure they are identical.

SPI Master Module

The SPI master module orchestrates communication by generating the SPI Clock (`sck`) and managing synchronized data capture from the slave, as shown in Fig. 3. A key aspect in this design is its capability to ingest multiple bits per clock cycle via parallel MISO lines, significantly improving data throughput, similar to the implementation in [1]. The master comprises the following elements:

- **Control Logic**: A bit counter module tracks the total number of bits received and controls the duration of active transmission. It ensures the correct alignment and completion of data capture by comparing the current bit index with the expected payload size and gates the `sck` generation after one full transmission cycle of n bits is completed.

- **SPI Clock Generation**: The master derives the SPI clock (`sck`) by dividing the overall system clock (`clk`) and toggles it using an edge-controlled finite state machine. This ensures consistent bit timing and synchronization between master and slave modules.

- **Reversed MISO Handling**: To address alignment challenges inherent in parallel bit collection, the design includes a logic block that reverses the bit order of the incoming MISO lines using a generate construct. This ensures that the parallel bits are correctly aligned before being committed to the shift register.

- **Data Processing**: An n-bit parametric parallel shift register is implemented to accommodate high-throughput applications. The register is designed to shift data from multiple MISO lines simultaneously on the rising edge of `sck`, effectively capturing bits from all the MISO lines per cycle, depending on the MISO width.

- **Output Processing**: Upon completion of the data transfer, the shift register contents are transferred to a data output register, which is then exposed to the system bus or application logic.

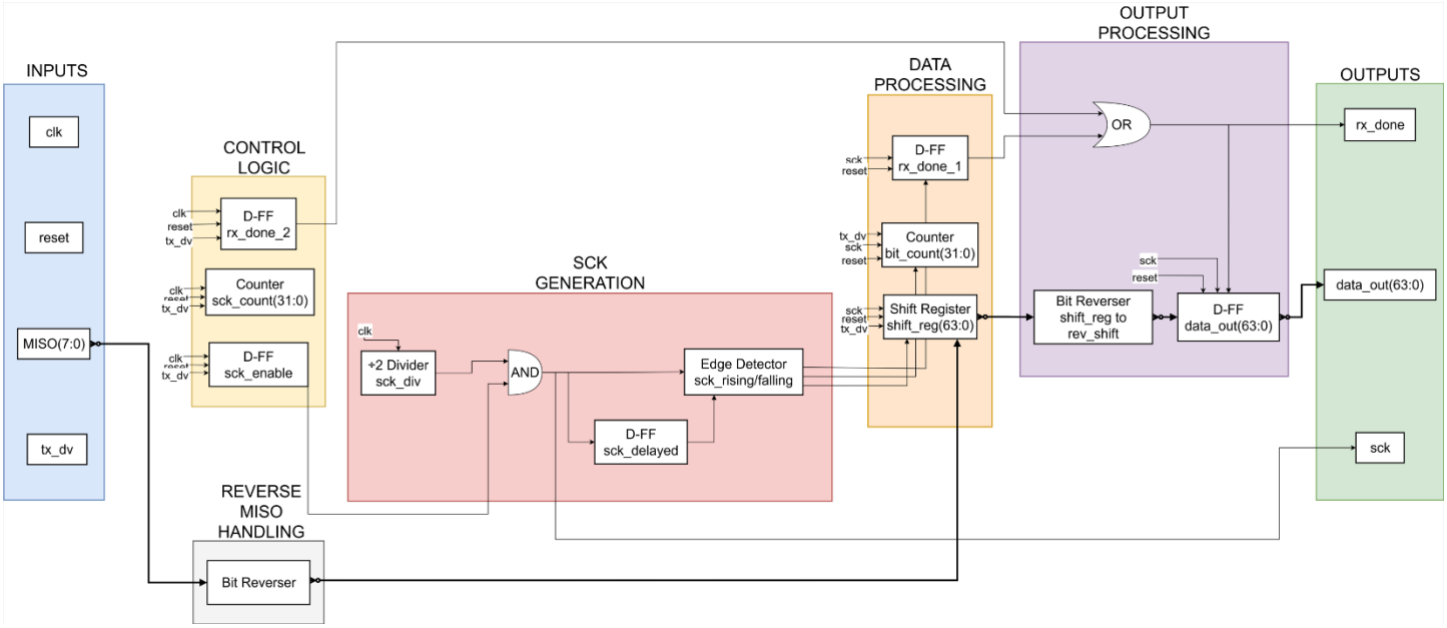


Fig. 3 Hyper MISO SPI Master Architecture

SPI Slave Module

The slave module is enhanced to support multi-bit data transmission via multiple MISO lines, operating in sync with the master's clock signal. The design ensures flexibility and reusability across various data widths and system configurations, as illustrated in Fig. 4.

- **Data Loading**: Data is loaded onto the input shift register upon the rising edge of the tx_dv signal.
- **Control logic**: A local counter tracks the number of bits transmitted, ensuring that only valid data is output during an active transmission window, and ends transmission when all the bits are transmitted.
- **Active Transmission Flag**: An internal flag signals whether the slave is currently engaged in a transaction. This flag is used to gate the output logic and prevent erroneous transmissions.
- **Shift Register**: A configurable-width shift register holds the outgoing data to be transmitted. It shifts data out on the **falling edge** of sck to maintain timing symmetry with the master's data capture on the rising edge. This implementation uses Mode 3 of the SPI. [2]
- **MISO Bit Assignment**: The outgoing data bits are mapped to individual MISO lines using a parameterized generate block. This facilitates clean scalability when increasing the number of MISO lines or adjusting data width.

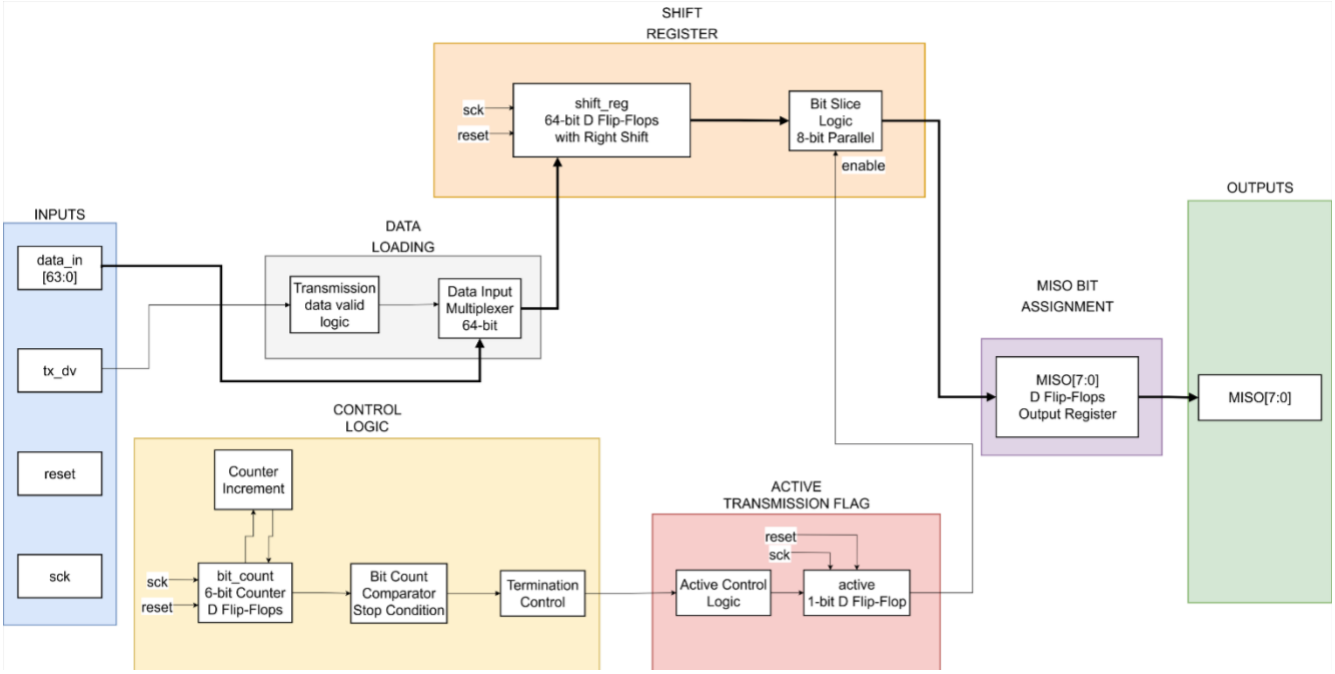


Fig. 4 Hyper MISO SPI Slave Architecture

Testing with Images: Writing a Testbench

A comprehensive testbench was developed to validate the functionality and correctness of the Hyper MISO SPI architecture. The testbench simulates image data transmission between the SPI master and slave modules, verifying the integrity and timing of the received output across parallel MISO lines. This simulation is critical for confirming the design's suitability for lightweight embedded applications, where efficient and reliable data transfer is essential.

Testbench Architecture

The testbench is architected to mimic real-world image transmission scenarios, particularly targeting sensor communication applications with critical high-throughput and synchronized data exchange. It incorporates essential components such as file handling, signal generation, and synchronization monitoring to ensure comprehensive Hyper MISO SPI protocol verification. Configurable parameters like `DATA_WIDTH` and `NUM_MISO` control the granularity of data transmission and the number of MISO lines utilized during simulation. These parameters are not only central to the testbench configuration. Still, they are also propagated to both the SPI master and slave modules, where they dictate register sizes, shift register widths, and bus dimensions to ensure consistent and scalable operation. Signal declarations include essential lines such as the system clock, SPI clock, active transmission flag (`tx_dv`), image data input (`image_data`), received data (`received_data`), and control flags such as reset and `rx_done`. The MISO bus is implemented as a multi-bit wire to enable concurrent data transmission across multiple lines, aligning with the parallel nature of the proposed protocol. File I/O is handled through binary files that emulate image data streams; file descriptors such as `input_bin` and `output_bin` are initialized using `$fopen` and managed with `$fclose`, while file names are dynamically passed using `$value$plusargs` to allow flexible reuse across simulation runs.

Module Instantiations

The testbench instantiates the SPI master and slave modules with appropriate bindings and parameters, as shown in Fig. 2.:

- SPI Slave Module: Connected to the `clk`, `sck`, and `reset` signals, the slave receives input data through `data_port_connected_to_slave` and transmits it over the MISO lines. Transmission is gated using the `tx_dv` signal.
- SPI Master Module: The master receives the MISO lines and the shared `sck` and `clk`, processes the incoming bits, and stores the result in `received_data`. The `rx_done` signal indicates completion of data reception.

Clock Generation and Control

System Clock: A 50 MHz system clock is generated using an always block that toggles `clk` every 10 ns, simulating a 20 ns period, and propagated to the slave and master modules.

Functional Tasks

To modularize testbench behavior, the following tasks are defined:

- ***`send_tx_dv:`*** Toggles the `tx_dv` signal high and low with controlled delays, indicating a valid data packet is ready for transmission by the slave.
- ***`send_reset`*****:** Asserts and de-asserts the `reset` signal, ensuring the master and slave modules initialize to known states before the simulation begins.

Simulation Flow (*`Initial`* Block)

The simulation is orchestrated through a structured `initial` block, which performs the following operations:

- **File Handling**: Using the `$value$plusargs` compiler directive, the testbench reads the input image file name in binary format and opens both input and output files for reading and writing.
- **Module Initialization**: The `send_reset` task is invoked to reset both modules and clear all registers. This step guarantees clean startup behavior across all registers and counters.
- **Data Transmission Loop**: The input image is read in `DATA_WIDTH` sized chunks. Each chunk is loaded into `data_port_connected_to_slave` and transmitted by asserting `tx_dv`. After each transmission, the master waits for `rx_done` to signal reception completion, and the received data is written to the output file.
- **Cleanup**: Once the image file is fully transmitted and received, both input and output files are closed, marking the end of the simulation.

This image-based testbench provides a realistic and practical framework for validating the Hyper MISO SPI protocol. It confirms the system's accuracy in handling high-throughput data streams and ensures alignment and synchronization across multiple data lanes, crucial features for deployment in performance-sensitive embedded applications.

Data synchronization

The signaling behavior of the proposed Hyper MISO SPI protocol aligns closely with conventional SPI timing schemes, with specific modifications to streamline frame control and synchronization. In standard SPI implementations, the slave select(*SPI_SS[x]*) signal is toggled between successive transfers to guarantee a high-to-low transition that marks the beginning of a new frame. The slave select output polarity is inverted to become active-high. In an idle state, the slave select (*SPI_SS[x]*) signal is kept low. Data is available on the clock cycle following the slave select (*SPI_SS[x]*) assertion. Both the master and slave modules conventionally sample data bits into their respective shift registers on the falling edge of the SPI clock (*SPI_clk*), with the latched data becoming valid on the subsequent rising edge. Furthermore, the Data Output Enable (*SPI_DOE_N*) signal is asserted (active low) throughout the transfer window to control the output drivers on the MISO lines. [3]

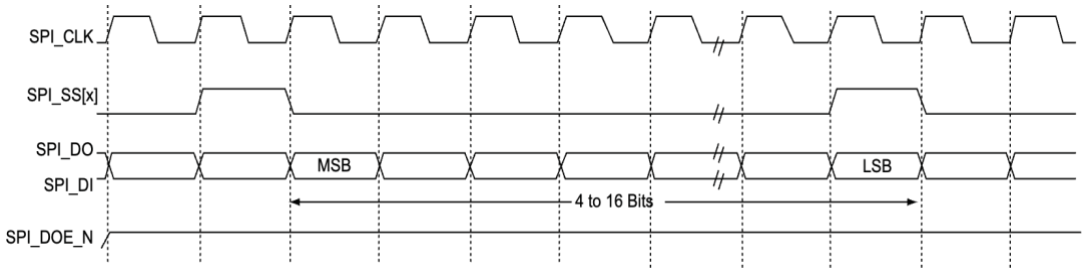


Fig. 5 Texas Instruments (TI) Synchronous Serial Multiple Frame Transfer. Reproduced from [3]

In the proposed architecture, the transmission data valid (*tx_dv*) signal serves as a substitute for the traditional slave select signal, functioning in active-low mode with normal polarity. The rising edge of *tx_dv* serves as the frame synchronization event, triggering the SPI master to initiate clock generation and the slave module to begin data transmission. As in conventional protocols, both master and slave perform data sampling on the falling edge of the SPI clock and latch the received data on the rising edge, preserving timing symmetry and ensuring reliable data alignment. Without proper synchronization between the master and slave modules, SPI communication can become misaligned, leading to incorrect data being received. If a tx_dv signal does not mark the start of a transmission, the master and slave shift registers may begin shifting data at different times. This can cause the master to sample invalid bits or miss the first few bits of a frame entirely. Over time, even minor timing mismatches can accumulate, especially in high-speed or multi-bit parallel systems, resulting in persistent data corruption. Additionally, in this system, where the SPI clock is gated or dynamically controlled, a lack of a clear frame boundary can lead to ambiguity in where one transmission ends and the next begins. The *tx_dv* signal ensures that both the master and slave start each transmission in sync, clearing any previous state and allowing accurate, frame-aligned data transfer. This design choice retains compatibility with SPI timing conventions while reducing signal overhead and control complexity.

Image to binary conversion

A preprocessing step is employed to convert standard image formats into a binary representation suitable for simulation to evaluate the performance and correctness of the Hyper MISO-based SPI communication protocol in handling large, structured datasets, such as image files. This conversion is handled via a Python script built upon the Pillow library [4], which allows flexible image manipulation and pixel-level access. The script begins by opening the source image and converting it to a standardized 32-bit RGBA format, ensuring that each pixel is represented using four 8-bit unsigned integers: Red, Green, Blue, and Alpha. This guarantees uniformity regardless of the original image format or color depth and can handle images with transparent backgrounds as well.

Fig. 6. Input Image

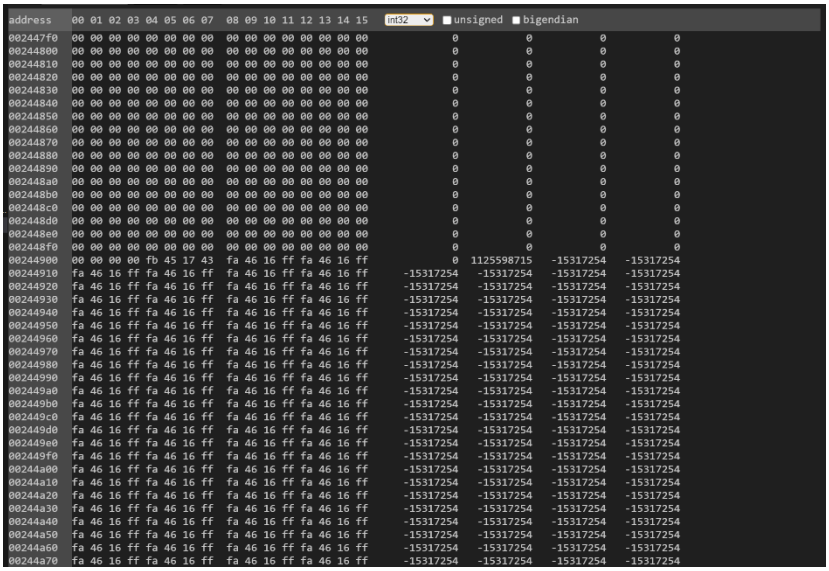


Fig.7. Image converted to binary

The image dimensions (width and height) are critical for reconstruction, and thus, they are encoded and written as the first 8 bytes of the binary file using Python's `struct.pack()` function with two 32-bit integers. This header enables the testbench to understand the layout of the image and calculate the total number of pixels expected. Subsequently, each pixel in the image is traversed in row-major order and packed as four consecutive bytes (R, G, B, A) to form a continuous data stream. The result is a compact binary file beginning with a resolution descriptor followed by a linear sequence of 32-bit pixel values, as shown in Figs. 6 and 7.

This binary output serves as the input data source in the SystemVerilog testbench. During simulation, the testbench reads the binary file. The pixel data is then read in chunks matching the `DATA_WIDTH` parameter of the SPI slave module and injected into the `data_port_connected_to_slave` input of the slave, after which data transmission can be tested.

Once the transmission of each pixel block is completed and acknowledged via the `rx_done` signal, the testbench writes the received data into a new binary file, replicating the original format. This output can later be converted back into an image for visual comparison or compared using the testbench, enabling pixel-by-pixel verification of transmission accuracy. This process validates the SPI protocol's functional correctness and stress-tests its ability to handle high-volume, real-time data in embedded imaging applications. Moreover, using images of varying resolution and complexity, developers can assess performance characteristics such as latency, throughput, and signal synchronization across multiple MISO lines under realistic data loads.

Results

The proposed Hyper MISO-based fast SPI communication protocol was thoroughly evaluated using a SystemVerilog testbench designed to emulate real-world sensor data transmission, particularly for high-resolution images. The results demonstrate functional correctness and significant performance scalability in terms of the number of MISO lines and transmission width.

Functional Correctness

An end-to-end test was performed to verify the integrity of data transmission, where an image was converted into a binary stream, transmitted via the SPI interface, and reconstructed on the receiving side. The data received was compared to the source data on a bit-by-bit basis, and the results confirmed perfect alignment without any mismatches. This reconstruction validates the accuracy of the Hyper MISO SPI protocol. The synchronized edge-triggering design is implemented in both master and slave modules. Altair® Silicon Debug Tools™ facilitated enhanced analysis by overlaying simulation-time signal values directly onto the schematic, thereby enabling more intuitive correlation between waveform behavior and schematic, as in Fig. 8.

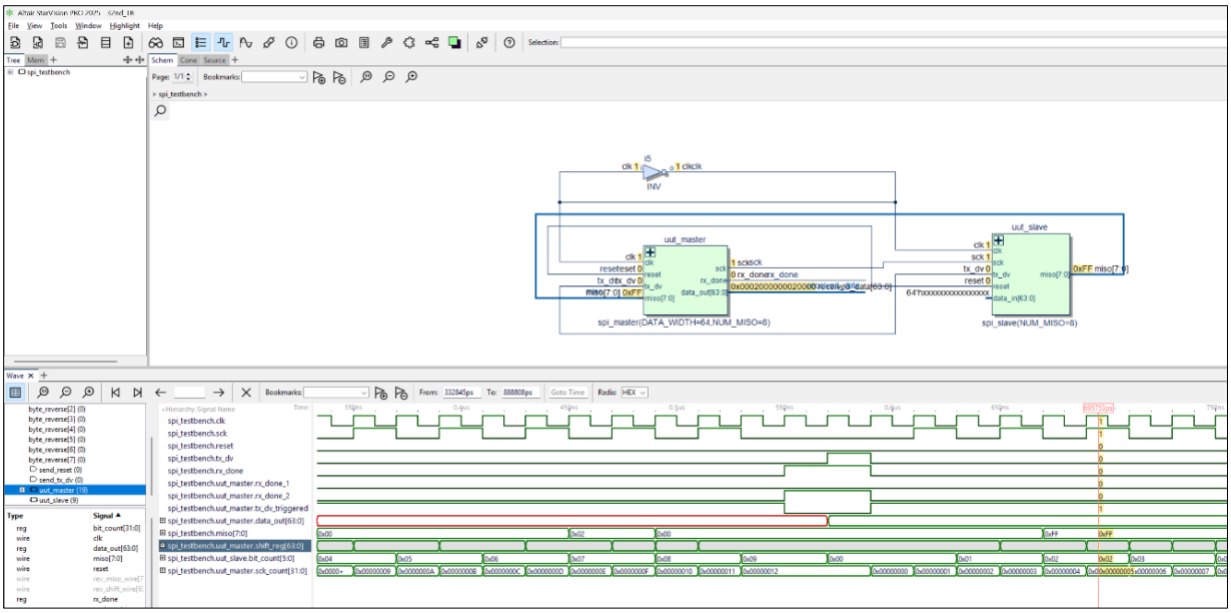


Fig.8 Intuitive correlation between waveform behavior and schematic - Altair® Silicon Debug Tools™

Simulation Validation

Simulations were conducted using **Altair® DSim™**, ensuring accurate timing and logic verification. One of the stress tests involved transmitting a large image of size **10,000 × 10,000 pixels** (approximately 100 megapixels), processed as a continuous stream of 32-bit RGBA pixel values. The system successfully handled this large dataset without timing violations or data loss, affirming the design's capability to manage high-volume transmissions over extended durations without losing synchronization.

Throughput and Performance

The communication system was tested under various configurations by altering the number of MISO lines and the width of data transmitted per cycle. With an SPI clock frequency of 25 MHz, the data rates are listed in Table 1. These results confirm that throughput scales linearly with the number of MISO lines and the data width, demonstrating the effectiveness of parallelism in improving communication efficiency. The ability to sustain nearly 145 Mbps in a simulation environment highlights the suitability of the design for high-throughput, low-latency embedded applications such as real-time image acquisition from multi-pixel sensor arrays.

Table 1. Simulation configurations for different numbers of MISO lines

Parameter	Single MISO SPI	Quad MISO SPI	Octa MISO SPI
Number of MISO lines	1	4	8
sck frequency (MHz)	25	25	25
Bits per clock cycle	1	4	8
Example image size (Pixels)	640x480	640x480	640x480
Data size (Bits)	9,830,400	9,830,400	9,830,400

Bits / Number of MISO lines	9,830,400	2,457,600	1,228,800
Throughput (Mbps)	18.2	73	145

To determine the SPI communication speed, the SPI output waveform was analyzed using the DSim waveform viewer. The time interval between two consecutive Transmit Data Valid (`tx_dv`) events was measured, representing the duration of known quantity data transfer cycles. By correlating this interval, the effective data rate was calculated for different MISO line SPI cases. This method provides an accurate estimation of throughput based on real-time signal activity.

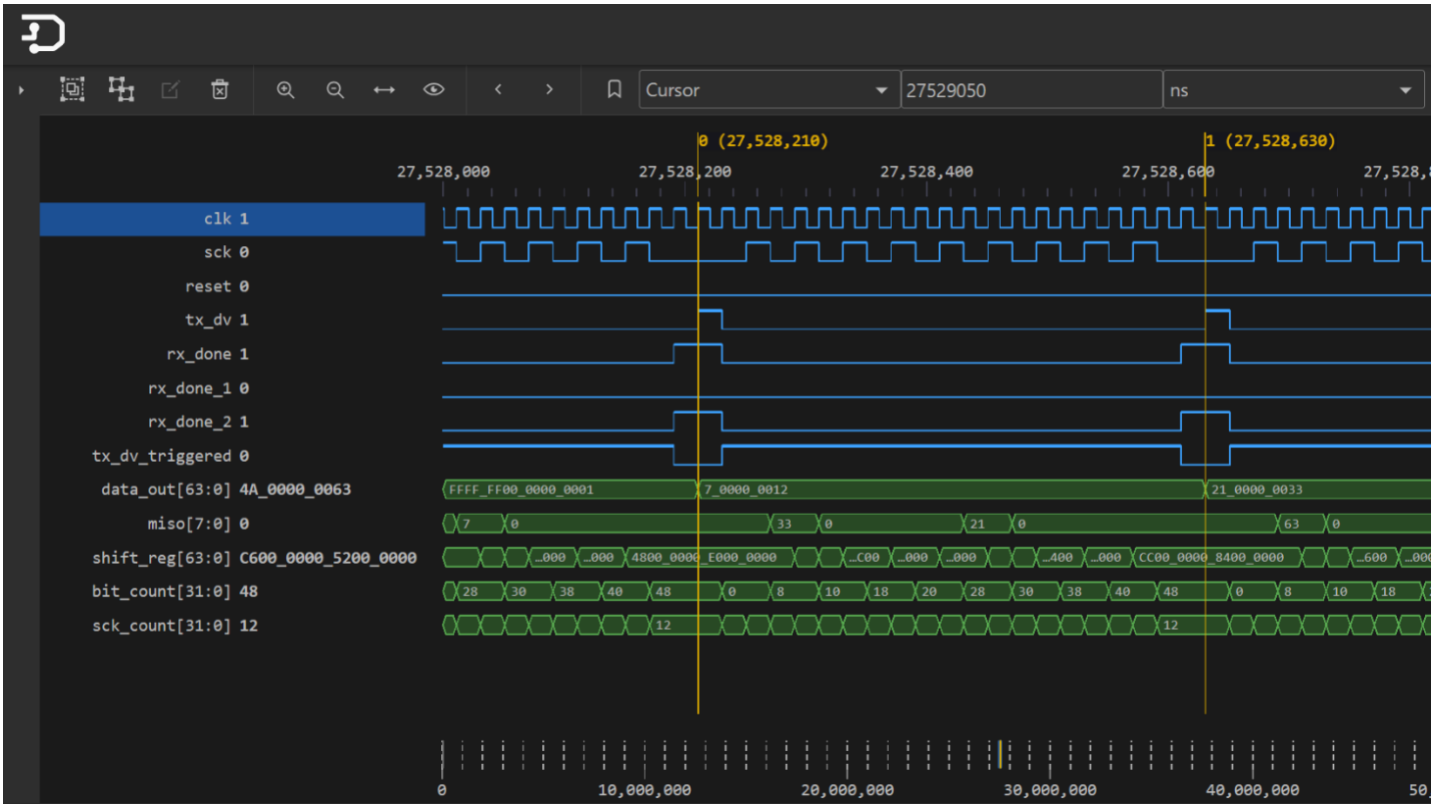


Fig. 9. Octa MISO SPI protocol waveform - Altair® DSim™ waveform viewer

In Fig. 9, marker 0 is placed at the rising edge of the `tx_dv` event at the start of the transmission cycle. Marker 1 is placed at the rising edge of the `tx_dv` event of the next transmission cycle. During one transmission cycle, 64 bits of data are transmitted.

Throughput for the Octa MISO SPI can be calculated as:

$$\text{Throughput} = \frac{\text{number of transmitted bits}}{\text{finish time (marker 1) - start time(marker 0)}} = \frac{64 \text{ bits}}{(27{,}528{,}630 - 27{,}528{,}210) \text{ ns}} = 145 \text{ Mbps}$$

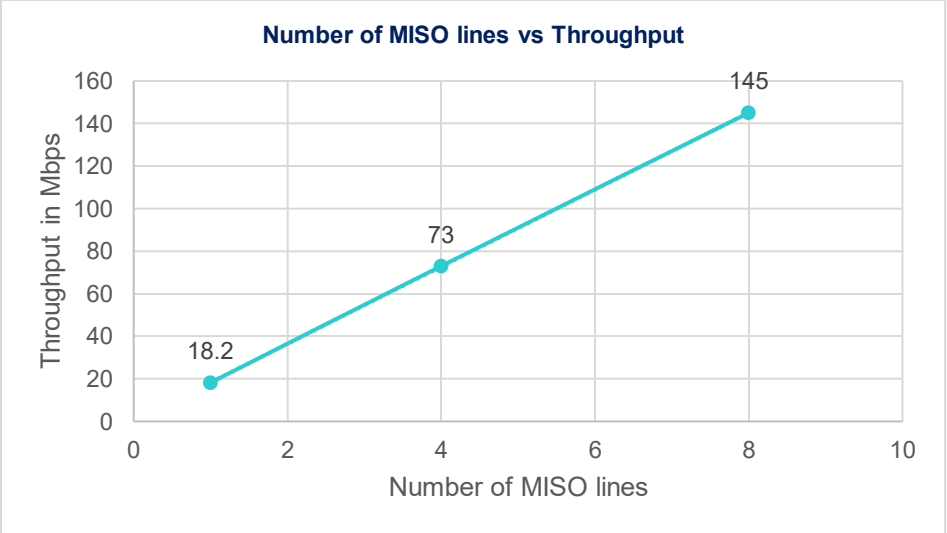


Table 1: Throughput increases nearly linearly with the number of MISO lines, illustrating the scalability of the approach.

Performance vs. Area Tradeoff

To evaluate the efficiency of the proposed Hyper MISO SPI protocol from a hardware implementation perspective, area estimates were extracted for various configurations through logic synthesis for an Application Specific Integrated Circuit (ASIC) using the open-source Yosys synthesis tool. The synthesis was performed using the Nangate 45nm Open Cell Library, which provides a realistic baseline for area modeling in nanometer-scale Complementary Metal Oxide Semiconductor (CMOS) processes. The area was reported in synthesis units (e.g., logic cell equivalents), equivalent to 1 μm² for the Nangate Open Cell Library. These estimates allow for meaningful comparison across different architectures. Throughput figures, previously calculated from simulation results, were paired with these area estimates to analyze the tradeoffs. The results show that increasing the number of MISO lines improves data throughput significantly and incurs a linear to super-linear increase in silicon area. This analysis forms the basis for assessing the scalability and hardware cost-efficiency of the proposed design.

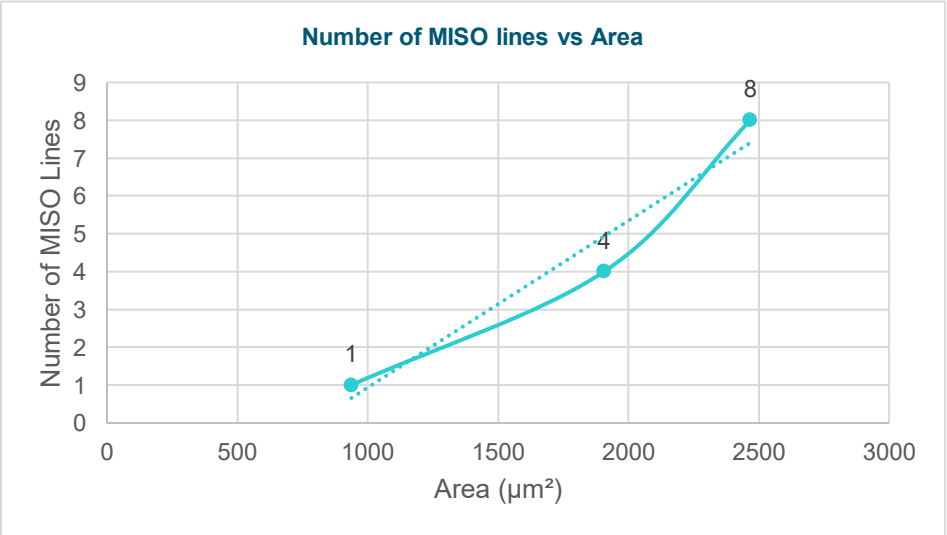


Table 2: Trade-off between increased number of MISO lines and hardware complexity.

For instance, the configuration using **a 1 MISO line with an 8-bit transmission width achieved 18.2 Mbps,** with an area footprint of **936.85 units**. In contrast, the **8 MISO / 64-bit** configuration delivered a much higher throughput of **145 Mbps but** used **1906.68 units**. The **16 MISO** configuration (presumed 128-bit transmission width) scaled up to **2467.68 units**, offering room for further throughput gains, depending on clock frequency and system architecture.

A throughput-to-area ratio (Mbps per unit area) was computed to quantify efficiency. While all configurations exhibited near-linear scaling, the 8 MISO/64-bit setup provided the best compromise between performance and hardware cost, making it the most optimal configuration for high-speed sensors and resource-constrained embedded applications that require high-speed data transmission.

Simulator run-time performance comparison

To assess simulation performance, the SPI master-slave system described in this paper was benchmarked on both **DSim** (a commercial simulator from Altair) and a **widely used open-source simulator**. The test scenario involved transmitting five images of varying sizes, ranging from 512 KB to 100 MB, through the SPI interface. Both simulators were configured to use their **maximum available optimization settings** to ensure a fair comparison. DSim consistently outperformed the open-source simulator in terms of simulation run time, particularly for larger image sizes. This improvement becomes more prominent as the input size scales, highlighting DSim's optimization and event-driven simulation engine as well-suited for large-scale verification tasks. While the open-source simulator performed adequately for small workloads, it exhibited a steep increase in run time beyond 10 MB data transmissions, making it less practical for high-throughput embedded SPI system verification.

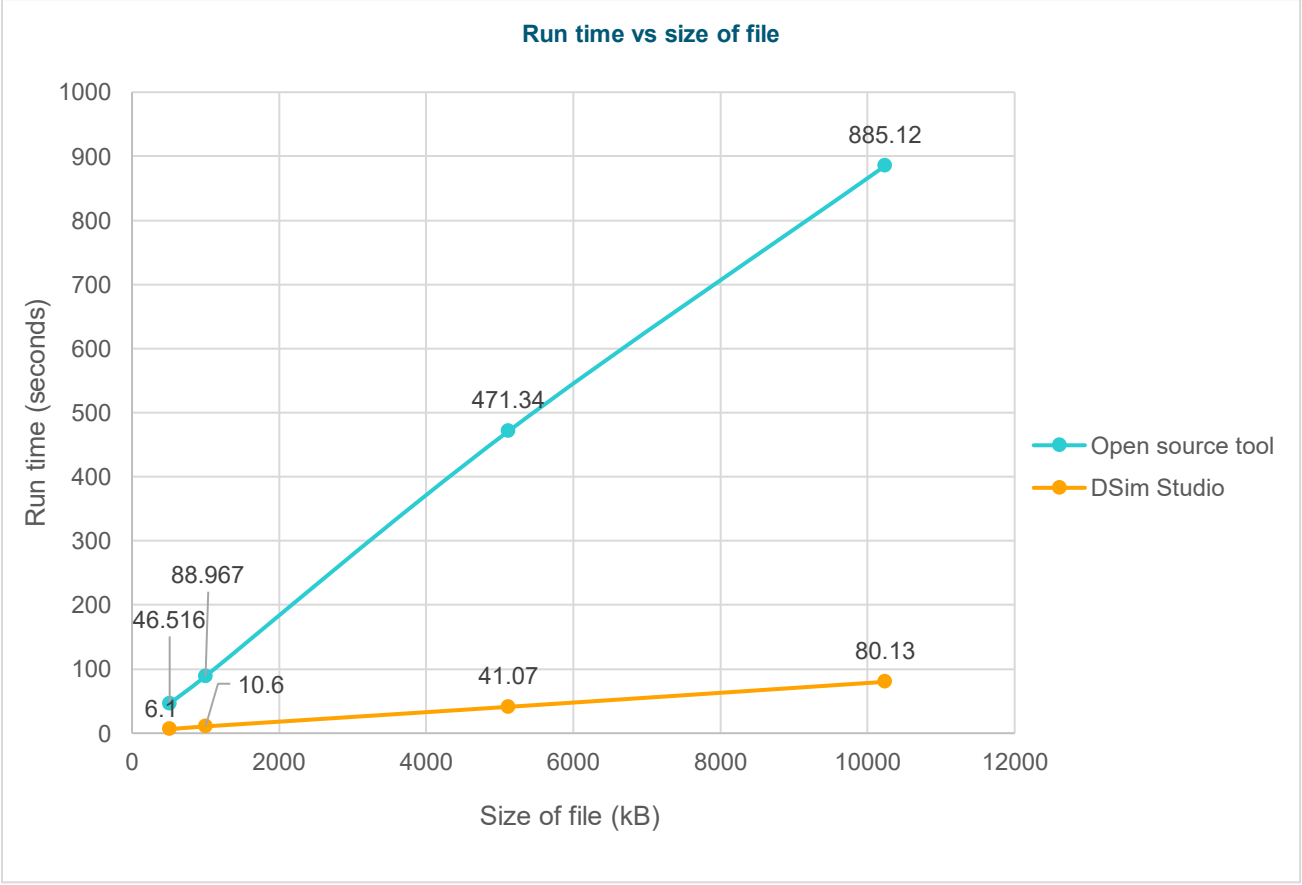


Table 3: Significantly lower execution time of DSim Studio compared to the open-source tool.

Conclusion

This work presents a novel, scalable approach to enhancing SPI throughput using the Hyper MISO architecture tailored for embedded and low-power applications. Through detailed implementation, simulation, and testing using real-world image data as a transmission workload, the design achieved bit-accurate communication, high data throughput, and flexibility across various configurations. The protocol was validated using a SystemVerilog testbench, and image transmission experiments confirmed functional correctness with perfect reconstruction of received data. Performance scaled linearly with the number of MISO lines and bits transmitted per cycle, reaching a maximum of 145 Mbps with 8 MISO lines at 64 bits per transfer. Synthesis results using Yosys and the Nangate 45nm open cell library demonstrated that this performance increase comes at a predictable area cost, enabling quantitative tradeoff analysis between silicon usage and communication efficiency.

To increase SPI throughput, the two primary options are to improve the SPI clock frequency or widen the data path by adding MISO lines. Raising the SPI clock boosts throughput linearly but introduces significant signal integrity issues such as clock skew, reflection, and jitter. It also increases Electro Magnetic Interference / Electro Magnetic Compatibility (EMI/EMC) emissions and challenges the timing closure of both master and slave, especially as frequencies exceed 50 MHz. Conversely, adding more MISO lines (parallel data paths) enables higher throughput at lower clock speeds, reducing high-speed design risks. However, this approach demands careful synchronization between MISO signals at the master, introduces routing complexity, and may require additional logic for bit alignment and data reconstruction.

In practice, a balanced approach yields the best results. By maintaining a moderate SPI clock frequency (e.g., 25–50 MHz) to preserve signal quality, reduce EMI, and simultaneously increasing the number of MISO lines to scale throughput, designers can optimize performance while maintaining robustness. This strategy leverages the advantages of both approaches and avoids the extremes of either, achieving high-speed communication in a manageable and reliable way.

Overall, the proposed Hyper MISO SPI design enables high-bandwidth, low-power communication suitable for next-generation sensor interfaces, edge computing nodes, and bandwidth-sensitive embedded platforms. Future work may explore dynamic MISO scaling, adaptive width negotiation, and ASIC implementation for production-ready systems.

References

[1] D. Z. a. L. L. C. Yang, "Multi-channel high-speed data acquisition system based on improved SPI communication," *Journal of Physics Conference Series,* vol. 2404, 2022.

[2] P. Dhaker, "Introduction to SPI Interface," *ADI Analog Dialogue,* vol. 52, no. Number 3, pp. 49-53, September 2018.

[3] Microchip Technology Inc., "Texas Instruments Synchronous Serial Protocol," in *SmartFusion 2 Microcontroller Subsystem*, 2025, p. 13.2.2.5.

[4] Jeffrey A. Clark and contributors, "Pillow 11.3.0 Documentation," Open source, [Online]. Available: https://pillow.readthedocs.io/en/stable/.

[5] Silicon Integration Initiative Inc, "Nangate Open Cell Library," [Online]. Available: https://github.com/oscc-ip/nangate.

[6] YosysHQ, GmbH, "Yosys Open Synthesis Suite," [Online]. Available: https://yosyshq.net/yosys/.