International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

An Event-Triggered DMA Mini Controller Using Low-Power SoC Application

1st Bandi Doss, 2ndA. Sai Teja, 3rdB. Raja Sri, 4thM. Suresh, 5thT. Anusha, 6thCh. Vinay

1, 2, 3, 4, 5, 6 Department of Electronics and Communication Engineering, CMR Technical Campus, Hyderabad, India

Emails: dasalways4u@gmail.com, saitejaa3103@gmail.com, rajuprinty123@gmail.com, megavathsuresh27@gmail.com, thatkarlaanusha48@gmail.com, chilukani.vinay@gmail.com

---

Abstract - In modern embedded architectures and System-on-Chip (SoC) platforms, transferring data between memory blocks and peripheral interfaces is one of the most frequent operations. However, in many practical scenarios—such as wireless sensor nodes, biomedical monitors, and IoT devices—data is not generated continuously. Instead, it arrives sporadically in response to external real-world triggers. Keeping a conventional direct memory access controller continuously running or polling peripheral status registers during these idle intervals causes unnecessary clock switching and wastes precious power.

To overcome this issue, this project presents the design, implementation, and verification of a compact, Event-Triggered Direct Memory Access (DMA) Mini Controller tailored for low-power SoC environments. The controller stays in a low-power quiescent state and begins moving data only after receiving an external hardware event signal (event_in). The transfer parameters—including the 32-bit source starting address, the 32-bit destination starting address, and the 16-bit transfer size—are fully programmable. A Finite State Machine (FSM) coordinates the internal operations, transitioning cleanly between idle, active transfer, and completion stages.

The controller was modeled in Verilog HDL and thoroughly evaluated using the AMD Vivado Design Suite through RTL elaboration, behavioral simulation, and power estimation. The design was verified using a representative test case with a source starting address of 0x1000, a destination starting address of 0x2000, and a transfer length of 0x0004 words. The simulation waveforms confirm that the controller remains inactive prior to the event, turns active immediately upon trigger assertion, increments source and destination memory pointers accurately on each beat, and asserts a transfer-done flag upon finishing. Power analysis in Vivado reported an estimated total on-chip power consumption of 0.122 W on the target FPGA platform. This lightweight architecture offers an effective, easily extensible hardware building block for event-driven, power-conscious digital systems.

Key Words: DMA controller, Event-triggered architecture, Low-power SoC, Verilog HDL, Finite State Machine (FSM), Vivado simulation.

1. INTRODUCTION

Modern embedded systems and System-on-Chip (SoC) architectures are required to interface with an ever-growing variety of hardware peripherals, including analog-to-digital converters (ADCs), digital sensors, serial communication modules (such as UART, SPI, and I2C), and external memory banks. A central challenge in these systems is moving large blocks of incoming or processed data from one memory location to another as quickly and efficiently as possible.

In early microcontrollers and simple processing units, data movement was handled through Programmed Input/Output (PIO). In PIO, the central processing unit (CPU) reads each data byte or word from a source register into an internal accumulator, and then writes it out to a destination address using load and store instructions inside a software loop. While this works fine for small, occasional transfers, it becomes a severe bottleneck when handling larger data streams. The processor spends precious clock cycles executing routine memory fetch-store cycles instead of executing core application algorithms.

Direct Memory Access (DMA) was introduced to resolve this issue by offloading bulk data transfers to dedicated hardware. Once configured with the source address, destination address, and data length, a DMA controller gains direct access to the system bus and coordinates data movement independently, interrupting the CPU only when the entire block transfer has finished.

Even though conventional DMA relieves the processor, standard DMA architectures in low-power systems face another hurdle: **determining when to begin a transfer without burning energy.** In battery-operated IoT and edge-computing devices, peripherals produce data only when an external physical event occurs (such as an accelerometer sensing vibration or a sensor detecting a threshold change). If the DMA subsystem stays active waiting for data, or if the CPU must wake up repeatedly just to check status flags and issue a start command, the system consumes unnecessary dynamic power ($P_{\text{dynamic}} \propto \alpha C V^2 f$).

The primary goal of this B.Tech project is to bridge this gap by designing a lightweight, Event-Triggered DMA Mini Controller. The controller operates autonomously on a hardware-level trigger. It stays in a low-activity idle state until an external trigger pulse (event_in) arrives. Once triggered, it reads the pre-configured source address, destination address, and transfer size registers, executes the multi-cycle transfer burst, updates its internal address pointers, asserts status flags, and returns to idle.

By managing all operations with a clean, synthesizable Finite State Machine (FSM), the controller avoids complex bus negotiation overhead while offering reliable, predictable timing. The complete design has been written in Verilog HDL and validated using AMD Vivado.

2. EXISTING SYSTEM

To understand the benefits of the proposed architecture, we first examine how standard memory transfer mechanisms operate in conventional SoC setups.

In basic embedded systems without dedicated hardware controllers, the central processor handles all data transfers directly:

- The CPU polls peripheral status registers or services an interrupt.
- It fetches data from the source peripheral address into a core register.
- It writes that data into the destination memory buffer.
- It increments the memory pointers and decrements a loop counter.
- It repeats these steps until all bytes have been moved.

This approach wastes CPU cycles and increases software latency. The CPU cannot enter a low-power sleep state while transferring data, which shortens battery life in portable embedded designs.

its idle state during periods in which no transfer is requested.

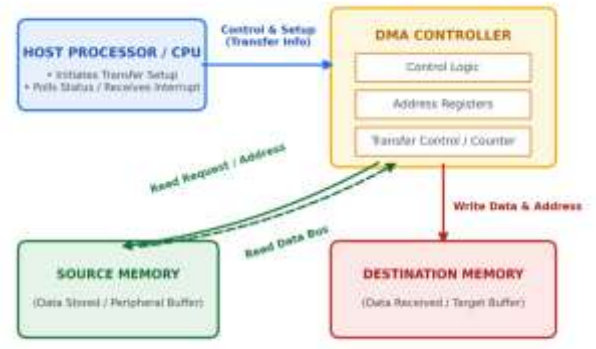


Fig -1: Existing System Architecture – Basic DMA Data Transfer

3. PROPOSED SYSTEM

The proposed system is an Event-Triggered DMA Mini Controller designed to deliver autonomous, power-efficient data movement for small-to-medium SoC applications. Instead of relying on CPU software polling or continuous hardware activation, this controller remains inactive until an external digital pulse appears on the event_in line.

finally to the completion state. During the transfer, the current source and destination addresses are maintained internally and updated as the operation progresses.

Two status signals are provided by the controller:

- **dma_active** — indicates that the DMA operation is in progress.
- **dma_done** — indicates that the requested transfer has reached completion.

The important design idea is that the transfer section is activated according to the event rather than simply remaining busy while the clock is running. This supports the low-power objective of the project by avoiding unnecessary transfer activity during the waiting period.

CORE FUNCTIONAL BLOCKS

As shown in the block diagram above, the controller consists of several dedicated internal units:

- **Event Detection Unit:** Monitors the external event_in line. It detects trigger pulses generated by peripherals—such as an ADC conversion-ready signal, a GPIO level change, or a sensor threshold alert—and synchronizes the event with the controller's clock domain.
- **Control Unit (FSM):** The central decision-making block. It orchestrates the controller through defined states: initialization, waiting for events, reading configuration registers, executing data movement, asserting handshake lines, and returning to idle.
- **Address Generation Unit (AGU):** Contains the dedicated 32-bit registers for src_addr and dst_addr. When an event occurs, it loads the starting addresses (src_addr_start and dst_addr_start) and increments these memory pointers sequentially after each successful word transfer.
- **Transfer Counter:** A dedicated down-counter initialized with the programmed transfer_size. It decrements after every transferred beat to track how many items have been moved, signaling the FSM when the requested block length is satisfied.
- **Data Transfer Unit:** Manages the datapath across the system bus (such as an AMBA AHB or AXI interconnect), coordinating the read cycle from source memory and the write cycle to destination memory.

Status Handshake Signals:

- **dma_active:** Driven HIGH to tell external bus arbiters and the host processor that a transfer is underway.
- **dma_done:** Pulses HIGH once the transfer finishes, providing an interrupt or status flag to the host system.

By decoupling the transfer trigger from software polling and relying on dedicated hardware state logic, the proposed controller keeps dynamic power to a minimum during the waiting state while ensuring rapid response times when events occur.

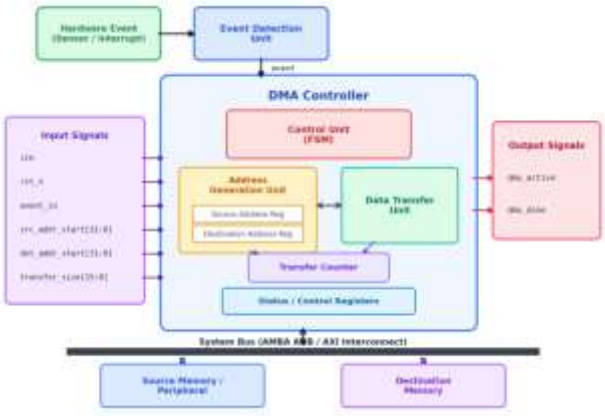


Fig -2: Proposed System Block Diagram – Event-Triggered DMA Controller

4. SYSTEM ARCHITECTURE

The mini controller uses a compact, clearly defined interface. It relies on standard single-bit control lines along with parameterized address and data buses.

The main input signals used in the design are:

Table -1: Main Input Signals of the Mini Controller

Signal	width	purpose
clk	1 bit	Provides the clock for the controller
rst_n	1 bit	Resets the controller
event_in	1 bit	Requests the DMA operation
src_addr_start	32 bit	Starting source address
dst_addr_start	32 bit	Starting destination address
transfer_size	16 bit	Specifies transfer length

Detailed Subsystem Descriptions

A. Event Input (event_in)

The event_in pin is the primary trigger for the mini controller. In typical embedded systems, this pin connects directly to an external interrupt, a comparator output, or a sensor's data-ready pin. While event_in remains LOW, the controller stays completely idle. Once a valid trigger is detected, the controller leaves its waiting state and initiates the transfer.

B. Source Address Register (src_addr_start)

This 32-bit input sets the starting read address. In an SoC system, this address points either to a static RAM buffer or to the output data register of a peripheral (such as an ADC sample buffer or a SPI receive FIFO). When the controller activates, it latches this value into an internal working register (src_addr) and increments it after each read cycle.

C. Destination Address Register (dst_addr_start)

This 32-bit input sets the target write address in memory (for instance, a circular buffer or a dedicated array in SRAM). The controller latches this value into its internal working register (dst_addr) and increments it sequentially alongside the source address pointer.

D. Transfer Size Register (transfer_size)

The 16-bit transfer_size register defines the length of the data burst, supporting block sizes up to $2^{16} - 1 = 65{,}535$ words. This register initializes an internal transfer counter, which decrements on each word transfer. When the counter reaches zero, the controller terminates the transfer cycle.

E. FSM Controller

The Finite State Machine (FSM) is the core control block of the design. Rather than relying on continuous combinational logic or complex multi-threaded polling, the FSM isolates each phase of execution into distinct states. This architecture simplifies design verification, prevents race conditions, and eliminates glitch-induced switching activity.

F. Status and Handshake Flags

To keep the host CPU and surrounding SoC blocks informed without software overhead, the controller provides two dedicated status outputs:

- **dma_active:** Stays HIGH for the entire duration of the transfer, letting the system bus arbiter grant priority to DMA transactions.
- **dma_done:** Asserts as soon as the final word is written, notifying the CPU that new data is ready in memory.

5. WORKING OF THE CONTROLLER

The controller's operation follows a clear sequential cycle, from initial reset to event detection, burst execution, and completion.

After the required transfer operation is completed, the controller generates the dma_done indication. It then returns to the idle condition and waits for the next event.

The overall sequence can be represented as:

Reset → Idle → Event Detection → Transfer → Completion → Idle

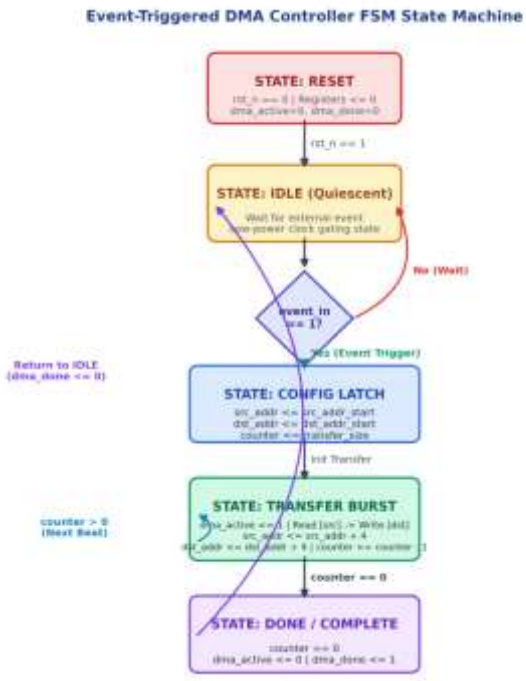


Fig -3: Controller FSM State Machine and Operational Flowchart

OPERATIONAL SEQUENCE

- **Power-On and Reset State:** When rst_n is asserted LOW, all internal registers, working address pointers, transfer counters, and status outputs are cleared to zero.
- **Quiescent Idle State:** Once rst_n goes HIGH, the controller enters the IDLE state. During this time, the clock feeds only the minimal state-holding registers, and the datapath registers remain static. The outputs dma_active and dma_done both stay LOW.
- **Event Detection and Configuration Latching:** The controller continuously monitors event_in. When a pulse appears, the FSM advances to the setup phase. It latches src_addr_start into the internal register src_addr, latches dst_addr_start into dst_addr, and loads transfer_size into the internal down-counter.
- **Data Transfer Execution:** The FSM asserts dma_active = 1 and enters the TRANSFER state. In each transfer cycle:
  - A data word is read from src_addr and written to dst_addr.
  - Both src_addr and dst_addr are incremented to point to the next memory locations.

– The transfer counter decrements by one.

- **Completion Handshake:** When the internal counter reaches zero, all requested words have been transferred. The FSM deasserts dma_active and asserts dma_done = 1 to signal completion to the system.
- **Return to Idle:** On the following clock cycle, dma_done is cleared back to 0, and the controller returns to the IDLE state, ready for the next external event trigger.

ALGORITHM:

The algorithm used in the project can be summarized as follows:

1. Initialize the controller using reset.
2. Remain in the idle state.
3. Monitor event_in.
4. Detect the transfer request.
5. Load the source starting address.
6. Load the destination starting address.
7. Use transfer_size to control the required transfer length.
8. Activate dma_active.
9. Perform the transfer and update the current addresses.
10. Check for transfer completion.
11. Assert dma_done.
12. Return to the idle state.

6. IMPLEMENTIATION, VERIFICATION AND POWER ANALYSIS

A. Design Flow and Tool Setup

The complete mini controller was implemented in synthesizable Verilog HDL (IEEE 1364-2001 standard) using the AMD Vivado Design Suite (v2016.2). The development workflow followed standard VLSI design steps:

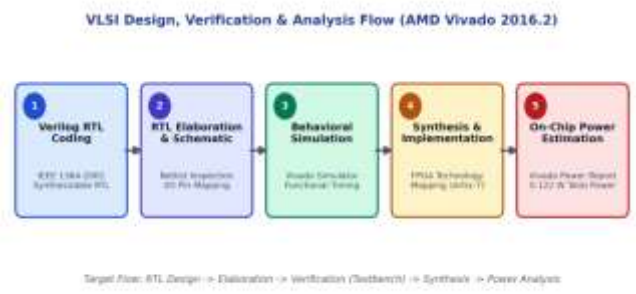


Fig -4: VLSI Implementation and Verification Workflow in AMD Vivado

After the RTL stage, behavioral simulation is carried out to observe how the controller responds to the event input and how the transfer-related signals change with time.

The main signals observed during simulation include:

- Clock and reset
- event_in
- Source address
- Destination address
- Transfer size
- dma_active
- dma_done
- Current source address
- Current destination address

For the supplied test case, the following values are used:

Table -2: Testbench Verification Parameters

Parameter	Test Value
Source start address	0x1000
Destination start address	0x2000
Transfer size	0x0004

The waveform shows the controller waiting before the event. After the event is applied, the DMA becomes active and the address information changes during the transfer. At the end of the operation, the completion signal is generated.

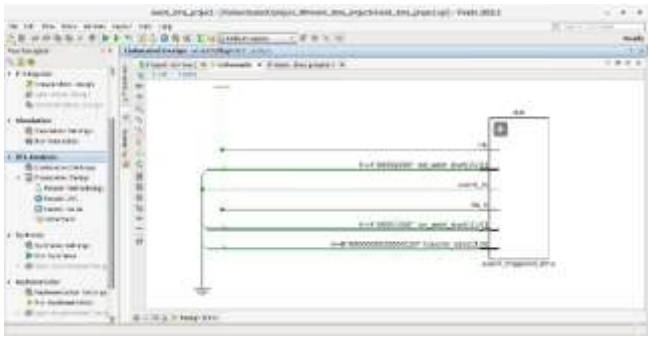


Fig -5: Schematic Diagram of Proposed Model (Vivado RTL Elaboration)

Schematic Explanation:

- This is an Event-Triggered DMA controller.
- clk → provides timing.
- rst_n → resets the DMA.
- event_in → triggers the DMA transfer.
- src_addr_start → starting source address (0x1000).
- dst_addr_start → starting destination address (0x2000).
- transfer_size → tells how much data to transfer.
- dma_active → shows transfer is in progress.
- dma_done → indicates transfer is completed.

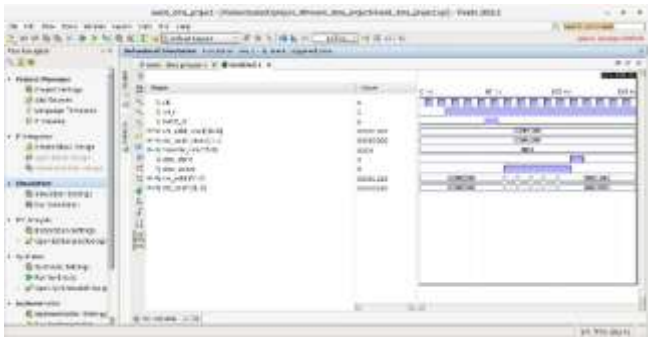


Fig -6: Functional Simulation Waveform (Vivado Behavioral Simulation)

Simulation Explanation:

- Initially, DMA is in reset/idle.
- After reset, it waits for event_in.
- When event_in occurs, dma_active becomes 1.

- Source and destination addresses increase during transfer.
- Finally, dma_done becomes 1, indicating the transfer is complete.

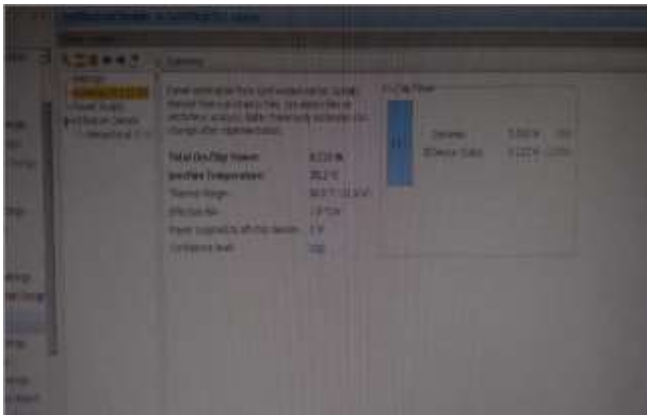


Fig -7: Power Analysis Report (Vivado Synthesized Netlist Estimation)

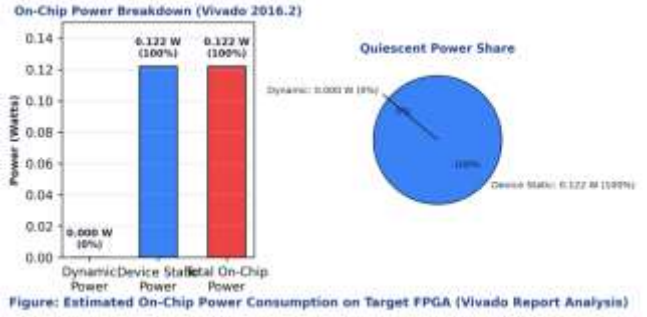


Fig -8: Estimated On-Chip Power Consumption on Target FPGA (Vivado Report Analysis)

Power analysis was performed in AMD Vivado for the tested design configuration. The supplied Vivado result reports approximately:

Total on-chip power = 0.122 W

This value represents the estimated power reported for the particular configuration used during the project evaluation. It should not be treated as a universal power value for every FPGA device or operating condition because power depends on factors such as device selection, clock frequency, switching activity, constraints, and implementation settings.

The purpose of including the power result is to provide an implementation reference for the present design. A direct comparison of power savings would require measurements under the same conditions for a conventional design and the event-triggered design.

REFERENCES

[1] L. Benini, A. Bogliolo, and G. De Micheli, "Dynamic power management of electronic systems," in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1998, pp. 696–702, doi: 10.1145/288548.289120.

[2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," IEEE Journal of Solid-State Circuits, vol. 27, no. 4, pp. 473–484, Apr. 1992, doi: 10.1109/4.126534.

[3] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 8, no. 3, pp. 299–316, Jun. 2000, doi: 10.1109/92.845896.

[4] T. Kitahara, F. Minami, T. Ueda, K. Usami, S. Nishio, M. Murakata, and T. Mitsuhashi, "A clock-gating method for low-power LSI design," in Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), 1998, doi: 10.1109/ASP-DAC.1998.669476.

[5] L. Benini and G. De Micheli, Dynamic Power Management: Design Techniques and CAD Tools. Boston, MA, USA: Kluwer Academic Publishers, 1998, doi: 10.1007/978-1-4615-5455-4.

[6] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, Digital Integrated Circuits: A Design Perspective, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.

[7] Arm Ltd., "AMBA specifications: AXI, AHB, APB, and low-power interfaces." [Online]. Available: https://www.arm.com/architecture/system-architectures/amba/amba-specifications. Accessed: Oct. 14, 2026.

[8] IEEE, IEEE Standard for Verilog Hardware Description Language, IEEE Std 1364-2005, Apr. 2006, doi: 10.1109/IEEESTD.2006.99495.

[9] AMD, Vivado Design Suite User Guide: Power Analysis and Optimization, UG907. [Online]. Available: https://docs.amd.com/r/en-US/ug907-vivado-power-analysis-optimization. Accessed: Oct. 14, 2026.

[10] "A comprehensive survey of clock-gating techniques for low-power VLSI design," in Proceedings of the 2025 9th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Nov. 2025, doi: 10.1109/ICECA66444.2025.11383489.