BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENCS3310, ADVANCED DIGITAL SYSTEMS DESIGN

Course project

Design and Verification of a Simplified Single-Channel DMA Controller for UART to- Memory Data Transfer

----

Prepared by: Talin Mohammad Omran Bayatneh – 1211305

Miassar Johar Fouad shamla – 1210519

Instructor: Dr. Ayman Hroub

Section: 1

Date: 13/8/2025

1. Abstract

In this project, we designed and tested a simple Direct Memory Access (DMA) controller that moves data from a UART module to a memory module without needing the CPU to handle each transfer. The system was built using Verilog HDL and tested with Synopsys VCS. It includes three main parts: a UART model that provides data, a memory model to store it, and a DMA controller that controls the process using a finite state machine (FSM). We created testbenches to check the normal operation and also tested unusual cases to make sure the design is reliable. The simulation results show that the DMA controller works correctly and can transfer data efficiently, helping reduce the CPU's workload.

Table of Contents

Table of figures

List of tables

2. Introduction

In digital systems, moving data quickly and efficiently is important, especially when dealing with peripherals like UART. Normally, the CPU would handle this transfer, but that can slow down the whole system. To solve this, we use a Direct Memory Access (DMA) controller, which can move data directly between devices and memory without needing the CPU for every step.

In this project, we built a DMA controller in Verilog to transfer data from a UART module into a memory module automatically. The UART works as our data source, holding preloaded values, and the memory acts as the storage. The DMA controller connects the two, reading data from the UART and writing it into memory in sequence.

We used a Finite State Machine (FSM) to control the process, making it easy to follow each step from reading data, to writing it, to preparing for the next transfer. This keeps the design organized and reliable.

Finally, we tested the whole system with a testbench that checks normal operation as well as unusual cases, making sure the DMA works as expected. The result is a clear example of how FSM-based design can make data transfer faster and free the CPU for other tasks.

3. System Design

3.1 Overall Architecture

Our DMA system is built in a modular way, meaning it’s made of smaller parts that each have a clear job. The main parts are:

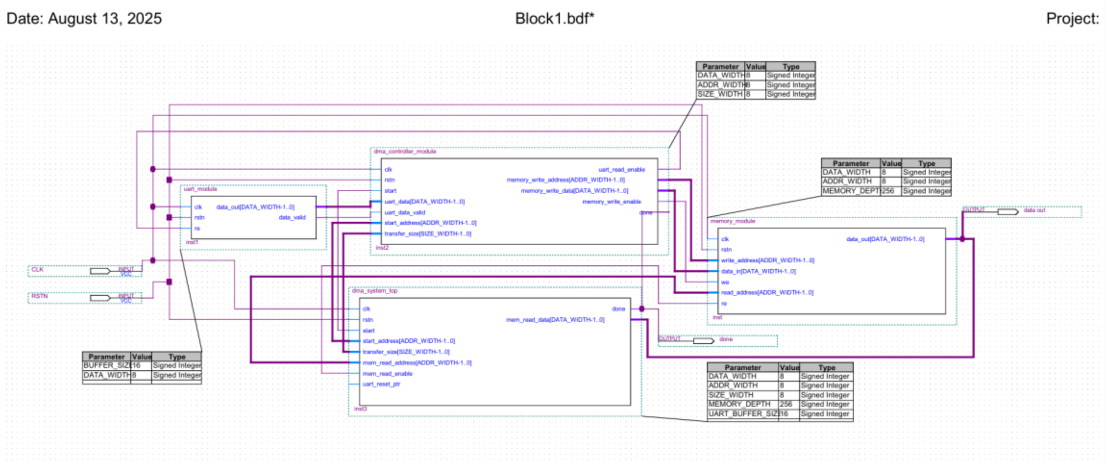
→ **UART Model:** acts like a data source, simulating a UART device that already has data stored.

→ **Memory Model:** works as the data destination, storing whatever the DMA sends.

→ **DMA Controller:** moves data from UART to memory without needing the CPU.

All three parts are connected inside a Top-Level Module. This top module sends the right signals between the modules, makes sure the timing is correct, and handles reset and clock connections.

3.2 System Block Diagram


Date: August 13, 2025
Block1.bdf*
Project:
dma_controller_module
clk
rstn
start
uart_data[DATA_WIDTH-1..0]
uart_data_valid
start_address[ADDR_WIDTH-1..0]
transfer_size[SIZE_WIDTH-1..0]
uart_read_enable
memory_write_address[ADDR_WIDTH-1..0]
memory_write_data[DATA_WIDTH-1..0]
memory_write_enable
done
inst2
uart_module
clk
rstn
re
data_out[DATA_WIDTH-1..0]
data_valid
inst1
memory_module
clk
rstn
write_address[ADDR_WIDTH-1..0]
data_in[DATA_WIDTH-1..0]
we
read_address[ADDR_WIDTH-1..0]
re
data_out[DATA_WIDTH-1..0]
inst
dma_system_top
clk
rstn
start
start_address[ADDR_WIDTH-1..0]
transfer_size[SIZE_WIDTH-1..0]
mem_read_address[ADDR_WIDTH-1..0]
mem_read_enable
uart_reset_ptr
done
mem_read_data[DATA_WIDTH-1..0]
inst3
CLK
RSTN
INPUT
VCC
OUTPUT
data out
done

Parameter	Value	Type
DATA_WIDTH	8	Signed Integer
ADDR_WIDTH	8	Signed Integer
SIZE_WIDTH	8	Signed Integer

Parameter	Value	Type
DATA_WIDTH	8	Signed Integer
ADDR_WIDTH	8	Signed Integer
MEMORY_DEPT	256	Signed Integer

Parameter	Value	Type
BUFFER_SIZE	16	Signed Integer
DATA_WIDTH	8	Signed Integer

Parameter	Value	Type
DATA_WIDTH	8	Signed Integer
ADDR_WIDTH	8	Signed Integer
SIZE_WIDTH	8	Signed Integer
MEMORY_DEPTH	256	Signed Integer
UART_BUFFER_SIZ	16	Signed Integer

Figure 1: System Block Diagram

3.3 Modules Description

3.3.1 UART Module

This module is a simplified version of a UART device, made for testing. It doesn’t actually send data over a serial line instead of that it stores a small 16-byte buffer that already has test data (we add ***“ADVANCE DIGITAL”***).

→How the module works:

- When the DMA asks for data (read_enable), it sends out the next byte in the buffer.

- It has a signal to tell the DMA when the data is ready.

- It keeps track of where it is in the buffer with a pointer.

- It can reset the pointer so we can test multiple times.

3.3.2 Memory Module

This module is where the DMA stores data it receives from the UART.

→How the module works:

- Can store 256 bytes (8 bits per address).

- Has separate ports for reading and writing so both can happen without interfering.

- The DMA writes to it using an address, the data byte, and a write enable signal.

- We can also read from it separately to check if the data transfer worked.

- It clears all stored data if reset.

3.3.3 DMA Controller Module

This module is the main control unit. It decides when to read from the UART and when to write to memory. It uses a Finite State Machine (FSM) to control the process step by step.

→ How the module works:

- Starts when it sees a start signal.

- Counts how many bytes to transfer.

- Reads from the UART only when data is ready.

- Writes to memory and updates the address each time.

- Stops when all bytes are transferred and sends a done signal.

- Handles special cases like if the transfer size is 0.

3.4 Finite State Machine Design

→ has three states:

- IDLE: waiting for the start signal. If transfer size is zero, it finishes immediately.

- READ UART: asks UART for the next byte and waits until it's ready.

- WRITE MEMORY: writes the byte to memory, updates counters, and decides if we're done or need to get more data.

→ The FSM makes sure data is always read and written in the right order, without losing any bytes.

Table 1: FSM State Transition Table

Current State	Next State	Condition	Action
IDLE	IDLE	!start OR size = 0	wait / complete immediately
IDLE	READ_UART	start && size > 0	init transfer params
READ_UART	READ_UART	!uart_data_valid	Keep waiting for data
READ_UART	WRITE_MEMORY	uart_data_valid	buffer data
WRITE_MEMORY	IDLE	byte_counter >= size-1	complete transfer, assert done
WRITE_MEMORY	READ_UART	byte_counter < size-1	continue transfer

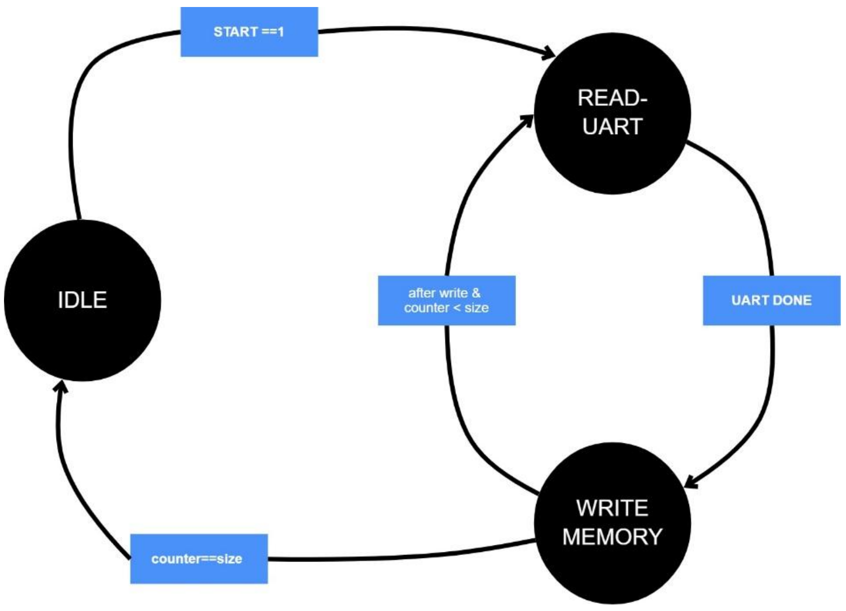


Figure 2: FSM diagram

4. Implementation

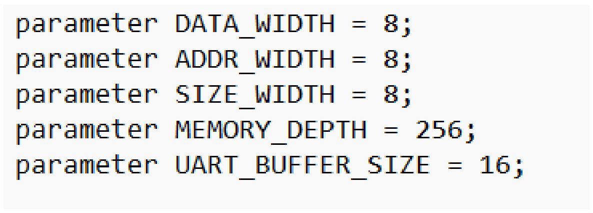
4.1 Programming Language

→ The DMA system was implemented in Verilog HDL (Hardware Description Language), which is IEEE 1364 compliant and widely used for designing and verifying digital circuits.

→ The design and verification were carried out using EDA. Also, Quartus was used to create and visualize the block diagrams of the system.

4.2 System Parameters and Configuration

The design is fully parameterized, making it flexible and reusable for different system sizes.



```
parameter DATA_WIDTH = 8;
parameter ADDR_WIDTH = 8;
parameter SIZE_WIDTH = 8;
parameter MEMORY_DEPTH = 256;
parameter UART_BUFFER_SIZE = 16;
```

Figure 3: system parameters

→ DATA_WIDTH = 8, Byte-oriented data transfers, common in microcontrollers.

→ ADDR_WIDTH = 8, Supports 256 memory locations (2^8).

→ SIZE_WIDTH = 8, Allows transfers of up to 255 bytes.

→ MEMORY_DEPTH = 256, Matches address width capacity.

→ UART_BUFFER_SIZE = 16, Depth of UART’s internal test buffer.

5. Simulation & Testing

→ The DMA system was verified using a hierarchical, bottom-up testing approach.

- First, each module (UART, Memory, DMA Controller) was tested individually to confirm correct functionality.

- After module-level verification, the full system integration was tested to ensure proper interaction between all components.

→ All testbenches shared a common structure, including:

- **Clock Generation**: 100 MHz clock with a 10 ns period.

- **Reset Strategy**: Synchronous reset task that initializes all signals and clears states.

- **Signal Monitoring**: $monitor statements to continuously display key signal values.

- **Timeout Protection**: Global simulation timeout to prevent infinite loops in case of design errors.

5.1 Individual Module Testing

5.1.1 UART Module

→ we write a test bench to validate that the UART correctly outputs preloaded data with proper handshaking.

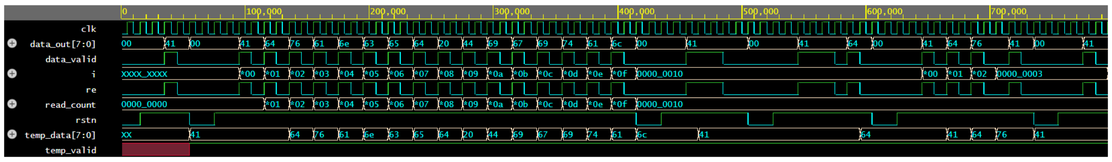


Figure 4: UART waveform in hex

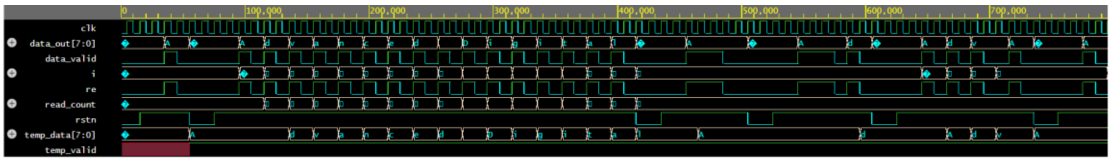


Figure 5: UART waveform in ASCI

→ The waveform confirms that the module operates as intended.

→ After reset, all outputs are cleared and the read pointer is set to the start of the buffer. Each rising edge of the read enable (re) signal produces the next byte from the 16-byte buffer, with data_valid going high for one clock cycle to indicate valid data. The output sequence correctly matches the ASCII codes for the message "Advanced Digital", proving that data is stored and read in the right order. The edge detection logic works properly, ensuring no extra reads occur when re is held high, and the pointer reset function allows reading from the start again when required. All activity is synchronized to the system clock, ensuring stable and consistent behavior.

5.1.2 Memory Module

→ we write a test bench to verify dual-port memory operation for simultaneous reads and writes.

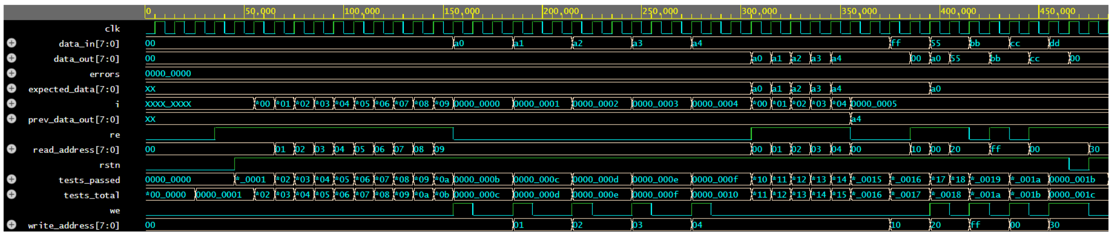


Figure 6: Memory waveform in hex

→ The waveform confirms correct operation across all tested scenarios.

→ After reset, all memory locations output 0x00, verifying proper initialization. Write operations store the expected data values at their respective addresses, and subsequent read-back cycles return the exact values written, demonstrating data integrity. The waveform also shows that read and write operations occur only when their respective enable signals are active, and simultaneous read/write to different addresses functions correctly without interference. Boundary address tests pass without errors, and all operations are synchronized to the clock, ensuring stable and predictable behavior.

5.1.3 DMA Module

→ we write a test bench to confirm finite state machine control for UART-to-memory transfers.

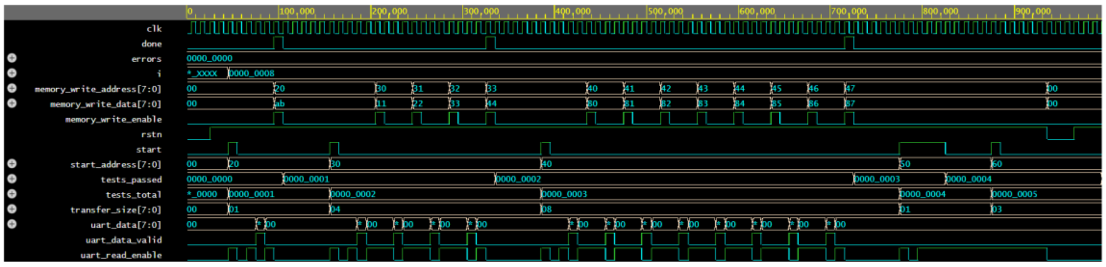


Figure 7 : DMA controller waveform in hex

→ The waveform demonstrates correct and reliable data transfers from the UART module to memory under various test scenarios.

→ In single, multi-byte, and extended transfers, the waveform clearly shows proper handshaking between uart_read_enable and uart_data_valid, followed by aligned memory_write_enable pulses and sequential address increments. Data written to memory matches the data received from UART, confirming transfer integrity. The waveform also verifies robust edge detection for the start signal, preventing unintended retriggers, and shows clean reset recovery during active transfers. All control signals transition synchronously with the clock, and the done signal asserts only after the final byte is written, indicating precise state machine control and successful operation.

5.1.4 Top level Module

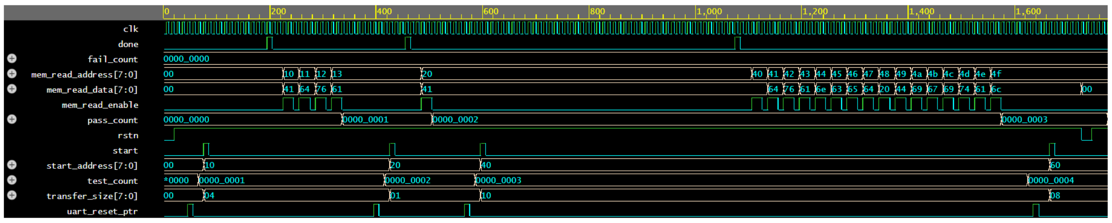


Figure 8: Top level waveform in hex

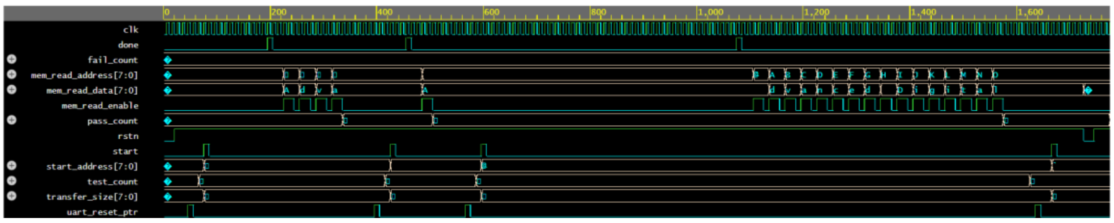
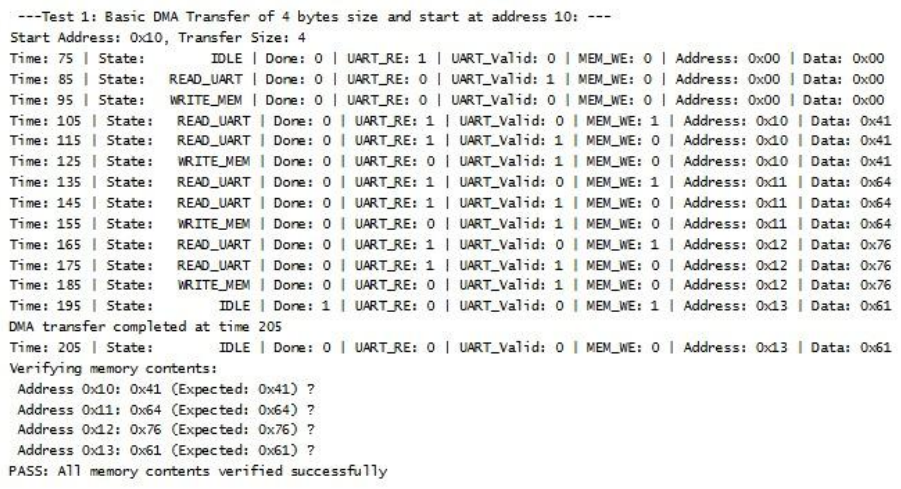


Figure 9: Top level waveform in ASCI

→ The waveform illustrates the successful integration and operation of the UART buffer, DMA controller, and memory module.

→ The waveform shows multiple transfer scenarios, starting with small single-byte and 4-byte transfers, followed by a full 16-byte buffer transfer that correctly reproduces the ASCII string "Advanced Digital" in memory. In each case, the start signals trigger the DMA controller, which coordinates UART data reads and sequential memory writes until the done signal indicates completion. The CPU verification phase confirms that transferred data matches the expected values, demonstrating end-to-end data integrity. Additionally, the waveform verifies system robustness by performing a reset during an active transfer, showing that all modules cleanly return to an idle state without errors. Overall, the waveform confirms that the integrated DMA system performs accurate, reliable, and synchronized data transfers across all tested scenarios.

Example of one test at debugging terminal:



```
 ---Test 1: Basic DMA Transfer of 4 bytes size and start at address 10: ---
Start Address: 0x10, Transfer Size: 4
Time: 75 | State:       IDLE | Done: 0 | UART_RE: 1 | UART_Valid: 0 | MEM_WE: 0 | Address: 0x00 | Data: 0x00
Time: 85 | State:  READ_UART | Done: 0 | UART_RE: 0 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x00 | Data: 0x00
Time: 95 | State:  WRITE_MEM | Done: 0 | UART_RE: 0 | UART_Valid: 0 | MEM_WE: 0 | Address: 0x00 | Data: 0x00
Time: 105 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 0 | MEM_WE: 1 | Address: 0x10 | Data: 0x41
Time: 115 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x10 | Data: 0x41
Time: 125 | State:  WRITE_MEM | Done: 0 | UART_RE: 0 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x10 | Data: 0x41
Time: 135 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 0 | MEM_WE: 1 | Address: 0x11 | Data: 0x64
Time: 145 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x11 | Data: 0x64
Time: 155 | State:  WRITE_MEM | Done: 0 | UART_RE: 0 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x11 | Data: 0x64
Time: 165 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 0 | MEM_WE: 1 | Address: 0x12 | Data: 0x76
Time: 175 | State:  READ_UART | Done: 0 | UART_RE: 1 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x12 | Data: 0x76
Time: 185 | State:  WRITE_MEM | Done: 0 | UART_RE: 0 | UART_Valid: 1 | MEM_WE: 0 | Address: 0x12 | Data: 0x76
Time: 195 | State:       IDLE | Done: 1 | UART_RE: 0 | UART_Valid: 0 | MEM_WE: 1 | Address: 0x13 | Data: 0x61
DMA transfer completed at time 205
Time: 205 | State:       IDLE | Done: 0 | UART_RE: 0 | UART_Valid: 0 | MEM_WE: 0 | Address: 0x13 | Data: 0x61
Verifying memory contents:
 Address 0x10: 0x41 (Expected: 0x41) ?
 Address 0x11: 0x64 (Expected: 0x64) ?
 Address 0x12: 0x76 (Expected: 0x76) ?
 Address 0x13: 0x61 (Expected: 0x61) ?
PASS: All memory contents verified successfully
```

6. Conclusion:

The DMA system works as intended, successfully transferring data from the UART buffer to memory with correct timing and data integrity. All normal tests passed, and the system handled reset conditions without errors, proving it is reliable and well-integrated.