Department of Electrical & Computer Engineering

Summer - 2024/2025

ENCS3310 Advanced Digital Systems Design

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

Prepared by:

Saifeddin Saleh 1231474

Omar Shobaki 1230329

Instructor: Dr. Ayman Hroub

Date: 12/8/2025

Abstract

This project involves the design and verification of a simplified single-channel Direct Memory Access (DMA) controller for transferring data from a UART peripheral to memory without continuous CPU involvement. Implemented in Verilog using a finite state machine, the controller operates in idle, read, and write states, with parameterized data and address widths for flexibility. A simplified UART model provides buffered byte access, and a basic synchronous memory model stores the transferred data. A comprehensive testbench verifies functionality across various transfer sizes and start addresses, confirming correct operation, timing, and completion signaling for efficient embedded data transfers.

Table of contents

UART

The simplified UART used in this project abstracts away traditional serial line behavior to streamline integration with the DMA controller. Instead of handling bit-level transmission and reception, it provides buffered byte access through a small internal memory preloaded with fixed data. When the DMA controller asserts the read enable signal, the UART outputs the next byte after one clock cycle, ensuring predictable timing. This approach eliminates the complexities of baud rate generation and framing, focusing solely on delivering ready-to-use parallel data for efficient transfer to the memory module.

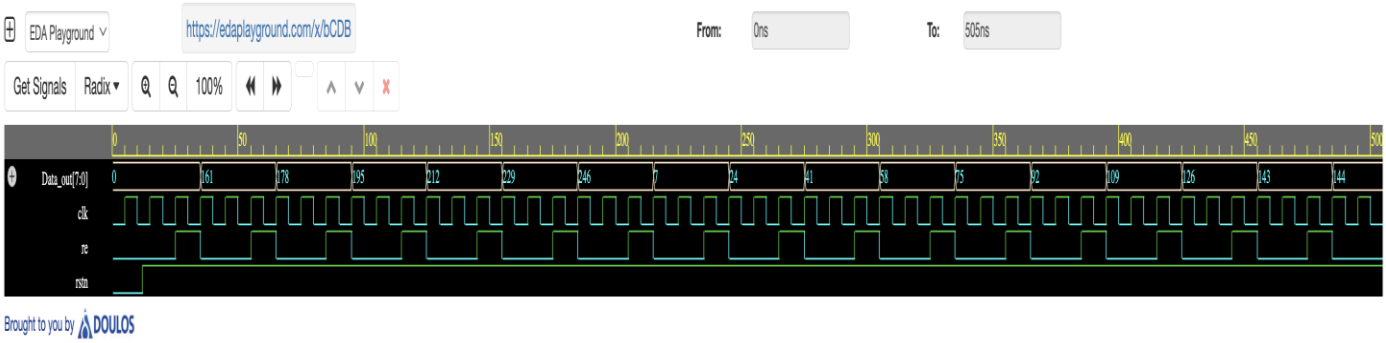
UART verilog code

```
//Saifeddin Saleh 1231474
//Omar Shobaki 1230329
module UART(clk, rstn, re, Data_out);
  input clk, rstn, re;
  output reg [7:0] Data_out;
  reg [7:0] buffer [15:0];
  reg [3:0] index;
  reg re_delay;
  always @(posedge clk or negedge rstn) begin
    if(!rstn)begin
      index <= 0;
      Data_out <= 0;
      re_delay <= 0;
      buffer[0] <= 8'hA1;
      buffer[1] <= 8'hB2;
      buffer[2] <= 8'hC3;
      buffer[3] <= 8'hD4;
      buffer[4] <= 8'hE5;
      buffer[5] <= 8'hF6;
      buffer[6] <= 8'h07;
      buffer[7] <= 8'h18;
      buffer[8] <= 8'h29;
      buffer[9] <= 8'h3A;
      buffer[10] <= 8'h4B;
      buffer[11] <= 8'h5C;
      buffer[12] <= 8'h6D;
      buffer[13] <= 8'h7E;
      buffer[14] <= 8'h8F;
      buffer[15] <= 8'h90;
    end
    else begin
      re_delay <= re;
      if (re_delay) begin
        Data_out <= buffer[index];
        index <= index+1;
      end
    end
  end
endmodule

```

UART testbench

```
//Saifeddin Saleh 1231474
//Omar Shobaki 1230329
module tb_UART;
reg clk,rstn,re;
wire[7:0] Data_out;
UART uart(.clk(clk),.rstn(rstn),.re(re),.Data_out(Data_out));
always #5 clk=~clk;
initial begin
 $dumpfile("uart.vcd");
 $dumpvars(0,tb_UART);
 clk=0;rstn=0;re=0;
 #12 rstn=1;
 @(posedge clk);
 repeat(16) begin
  @(posedge clk);re=1;
  @(posedge clk);re=0;
  $display("Data_out = 0x%0h",Data_out);
  @(posedge clk);
 end
 #10 $finish;
end
endmodule
```

UART waveform



As we can see, the preloaded data was transfered after one clock cycle since there's a delay.

Memory

The memory module in this project is a simple synchronous storage unit designed to hold the data transferred from the UART by the DMA controller. It supports parameterized address and data widths, allowing flexibility in size and word length. Data is written to the specified address when

the write enable signal is asserted and is stored for later access. A separate read interface allows retrieving data from any memory location when the read enable signal is active. This straightforward design ensures predictable, cycle-accurate behavior, making it well-suited for simulation and functional verification of the DMA transfer process.

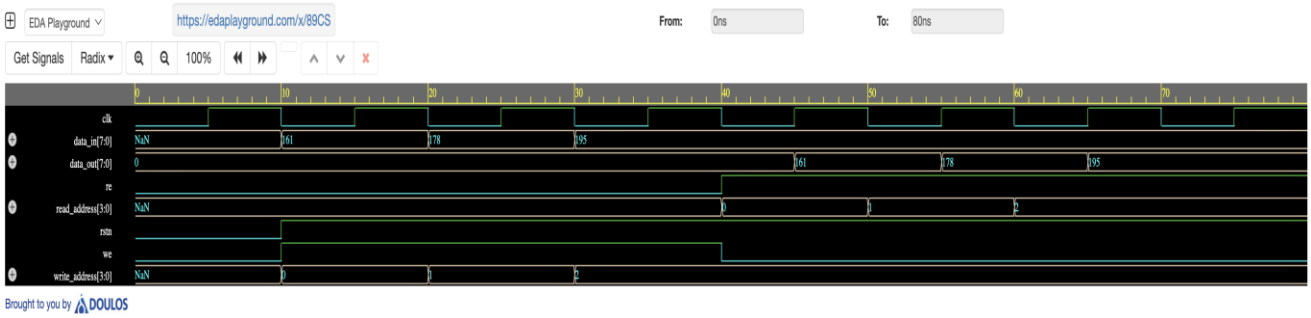
Memory verilog code

```
// Saifeddin Saleh 1231474
// Omar Shobaki 1230329
module memory(clk, rstn, write_address, data_in, we, read_address, data_out, re);
  input clk, rstn, we, re;
  input [7:0] data_in;
  input [3:0]  write_address, read_address;
  output reg [7:0] data_out;
  reg [7:0] memor [15:0];
  always @(posedge clk or negedge rstn)begin
    if(~rstn) begin
      data_out <= 8'b0;
    end
    else begin
      if(we) memor[write_address] <= data_in;
    if(re) data_out <= memor[read_address];
        end
    end
endmodule


```

Memory testbench

```
// Saifeddin Saleh 1231474
// Omar Shobaki 1230329
module tb_memory;
reg clk,rstn,we,re;
reg [7:0]data_in;
reg [3:0]write_address,read_address;
wire[7:0]data_out;
memory m(clk,rstn,write_address,data_in,we,read_address,data_out,re);
always #5 clk=~clk;
initial begin
 $dumpfile("mem.vcd");
 $dumpvars(0,tb_memory);
 clk=0;rstn=0;we=0;re=0;
 #10 rstn=1;
 we=1;
 write_address=0;
 data_in=8'hA1;#10;
 write_address=1;
 data_in=8'hB2;#10;
 write_address=2;
 data_in=8'hC3;#10;
 we=0;re=1;
 read_address=0;#10;
 $display("addr0=%h",data_out);
 read_address=1;#10;
 $display("addr1=%h",data_out);
 read_address=2;#10;
 $display("addr2=%h",data_out);
 #10 $finish;
end
endmodule

```

Memory waveform



As we can see, the data_in is being trasfered to data_out, which is how a memory works.

DMA controller

The DMA controller in this project is a single-channel hardware unit designed to autonomously transfer data from the UART module to memory without continuous CPU supervision. Implemented in Verilog using a finite state machine, it operates in sequential states to request data from the UART, wait for its availability, and write it into the target memory location. It uses internal counters to track the number of bytes left to transfer and automatically increments the memory write address during the process. Once the configured transfer size is completed, the controller asserts a done signal to notify the CPU, enabling efficient and streamlined data movement.

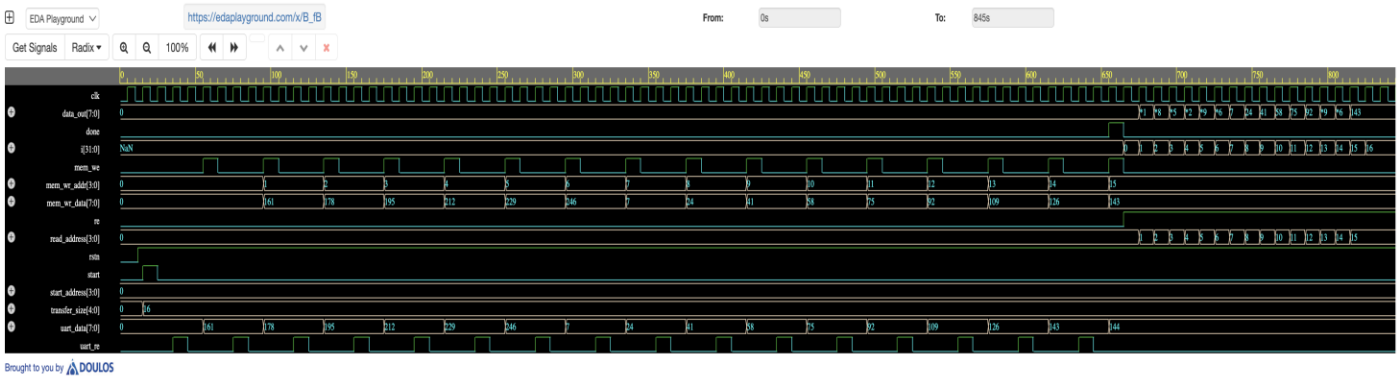
DMA controller verilog code

```
// Saifeddin Saleh 1231474
// Omar Shobaki 1230329
module dma_controller(clk,rstn,start,start_address,transfer_size,done,uart_re,uart_data,mem_we,mem_wr_addr,mem_wr_data);
input clk,rstn,start;
input [3:0]start_address;
input [4:0]transfer_size;
input [7:0]uart_data;
output reg done,uart_re,mem_we;
output reg [3:0]mem_wr_addr;
output reg [7:0]mem_wr_data;
reg [1:0]state,next_state;
reg [4:0]bytes_left;
reg [3:0]addr_ptr;
localparam IDLE=2'b00,READ_UART=2'b01,WRITE_MEM=2'b10;
always@(posedge clk or negedge rstn)begin
 if(!rstn)state<=IDLE;else state<=next_state;
end
always@(posedge clk or negedge rstn)begin
 if(!rstn)begin
  uart_re<=0;
  mem_we<=0;
  done<=0;
  bytes_left<=0;
  addr_ptr<=0;
  mem_wr_addr<=0;
  mem_wr_data<=0;
  next_state<=IDLE;
 end else begin
  uart_re<=0;
  mem_we<=0;
  done<=0;
  case(state)
   IDLE:begin
    if(start)begin
      bytes_left<=transfer_size;
      addr_ptr<=start_address;
      next_state<=READ_UART;
    end
     else next_state<=IDLE;
   end
   READ_UART:begin
    uart_re<=1;
    next_state<=WRITE_MEM;
   end
   WRITE_MEM:begin
    mem_we<=1;
    mem_wr_addr<=addr_ptr;
    mem_wr_data<=uart_data;
    addr_ptr<=addr_ptr+1;
    bytes_left<=bytes_left-1;
    if(bytes_left==1)begin
     done<=1;
     next_state<=IDLE;
    end
     else next_state<=READ_UART;
   end
   default:next_state<=IDLE;
  endcase
 end
end
endmodule

```

DMA controller testbench

```
// Saifeddin Saleh 1231474
// Omar Shobaki 1230329
module tb_dma;
reg clk;
reg rstn;
reg start;
reg [3:0] start_address;
reg [4:0] transfer_size;
wire done;
wire uart_re;
wire mem_we;
wire [3:0] mem_wr_addr;
wire [7:0] mem_wr_data;
wire [7:0] uart_data;
reg re;
reg [3:0] read_address;
wire [7:0] data_out;
integer i;
UART u(clk, rstn, uart_re, uart_data);
memory m(clk, rstn, mem_wr_addr, mem_wr_data, mem_we, read_address, data_out, re);
dma_controller d(clk, rstn, start, start_address, transfer_size, done, uart_re, uart_data, mem_we, mem_wr_addr, mem_wr_data);
always #5 clk = ~clk;
initial begin
    $dumpfile("dma.vcd");
    $dumpvars(0, tb_dma);
    clk = 0;
    rstn = 0;
    start = 0;
    start_address = 0;
    transfer_size = 0;
    re = 0;
    read_address = 0;
    #12 rstn = 1;
    @(posedge clk);
    start_address = 4'd0;
    transfer_size = 5'd16;
    start = 1;
    @(posedge clk);
    start = 0;
    @(posedge clk);
    while (done == 0) @(posedge clk);
    re = 1;
    for (i = 0; i < 16; i = i + 1) begin
        read_address = i;
        @(posedge clk);
        $display("mem[%0d]=0x%02x", i, data_out);
    end
    #20 $finish;
end
endmodule

```

DMA controller wavefrom



Conclusion

The implemented project successfully achieved the design and verification of a simplified single-channel DMA controller capable of transferring data from a UART peripheral to a memory module without continuous CPU involvement. Using a finite state machine, the DMA handled the read–write sequence reliably, asserting uart_re to request data, capturing valid uart_data, and writing it to sequential memory addresses until the transfer size was reached. The UART and memory modules integrated seamlessly with the DMA, and simulation waveforms confirmed correct timing, data integrity, and proper signaling of the done flag upon completion. This design demonstrates the effectiveness of hardware-based data movement for improving system efficiency, reducing CPU load, and ensuring predictable, cycle-accurate operation in embedded systems.