HW 7: Designing a custom IP for Merge Operation with Xilinx Fifo Generator

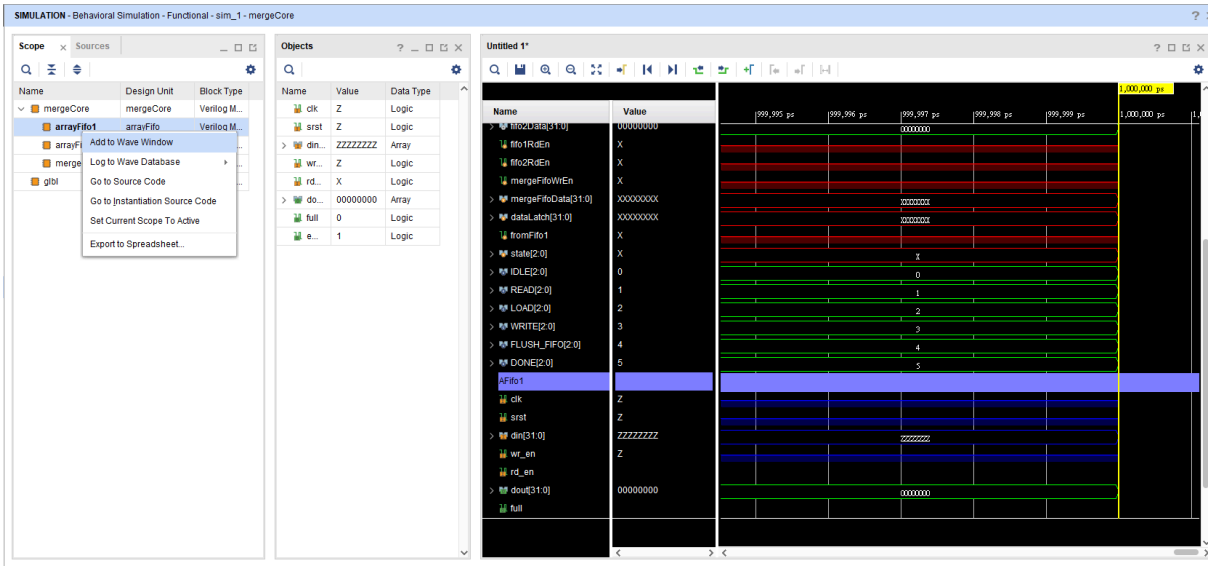
資訊所P76134082 陳冠言、醫資所Q56134102 王宇軒

Result

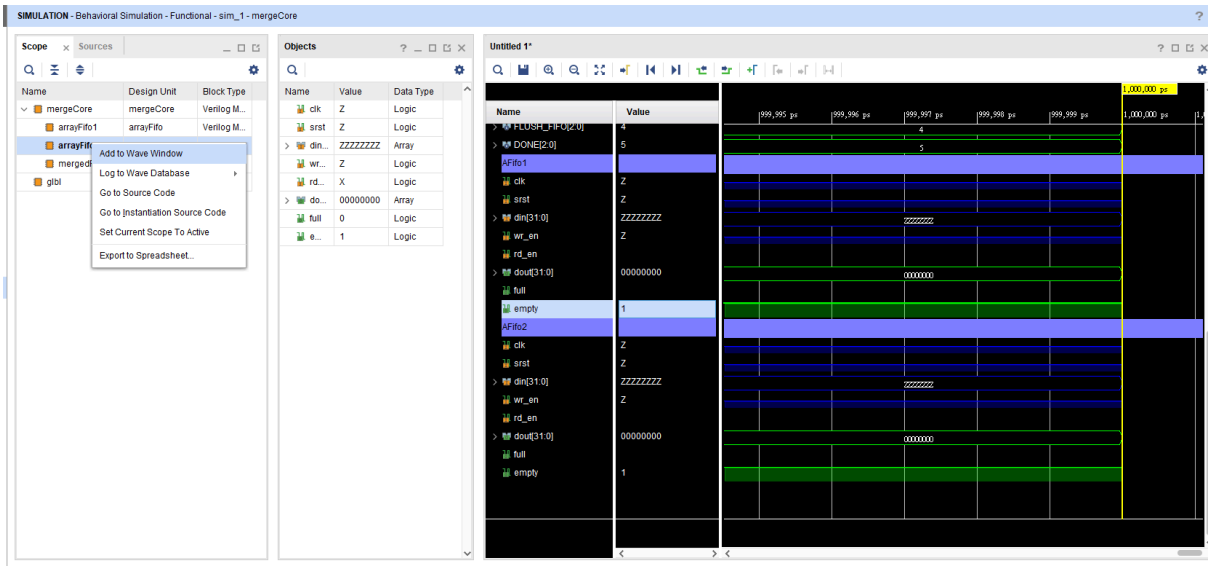
本次作業主要是將 Merge Operation 的硬體邏輯包裝成 Vivado 的自訂 IP，並透過 AXI4 介面與處理器整合，實現兩個已排序陣列合併為一個排序陣列（由大到小）的功能。系統整合後，處理器可將資料寫入 IP，啟動合併運算，並於運算完成後讀回合併結果。透過 SDK 撰寫驗證程式，並結合 Tera Term 進行序列埠通訊，驗證硬體與軟體整合運作正確。

<RUN SIMULATION>

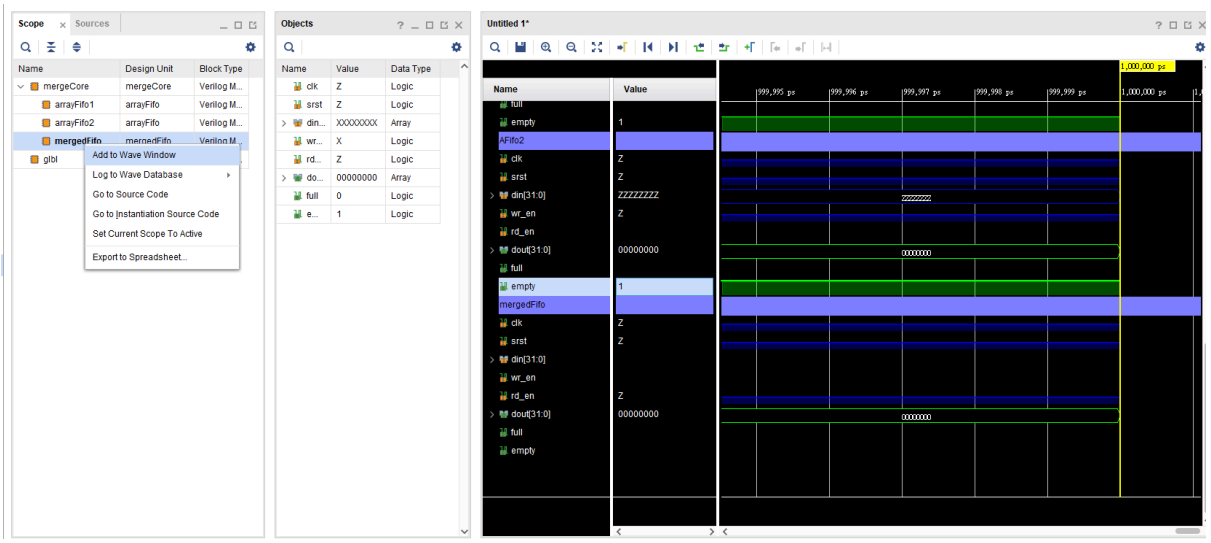
AFifo1, arrayFifo1 add to new window

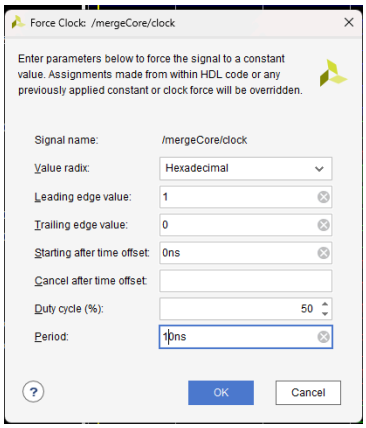
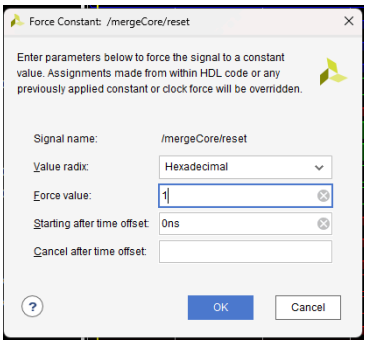
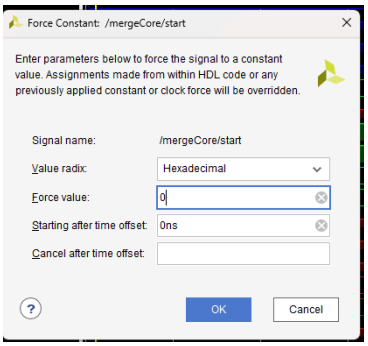
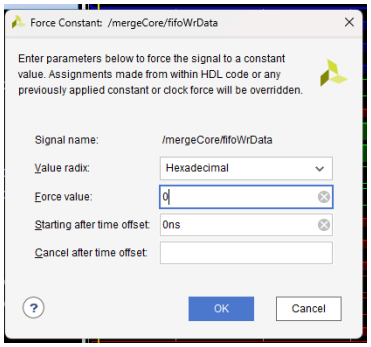
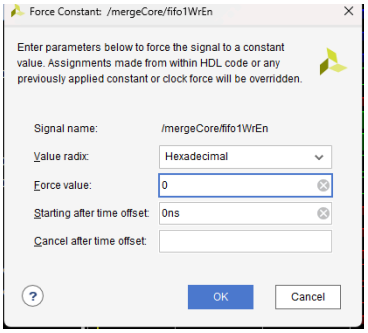
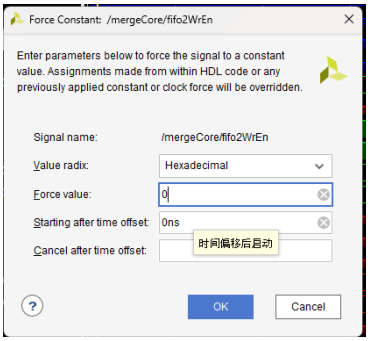


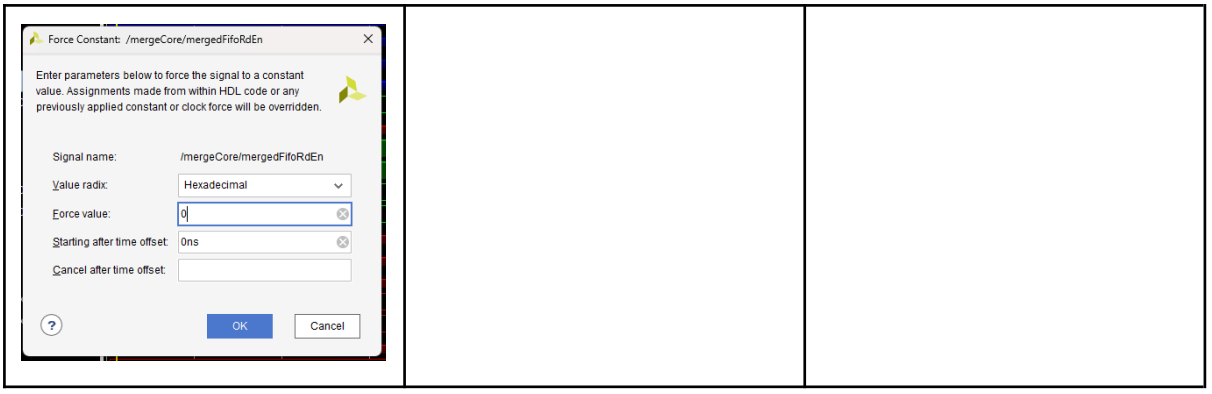
AFifo2, arrayFifo2 add to new window

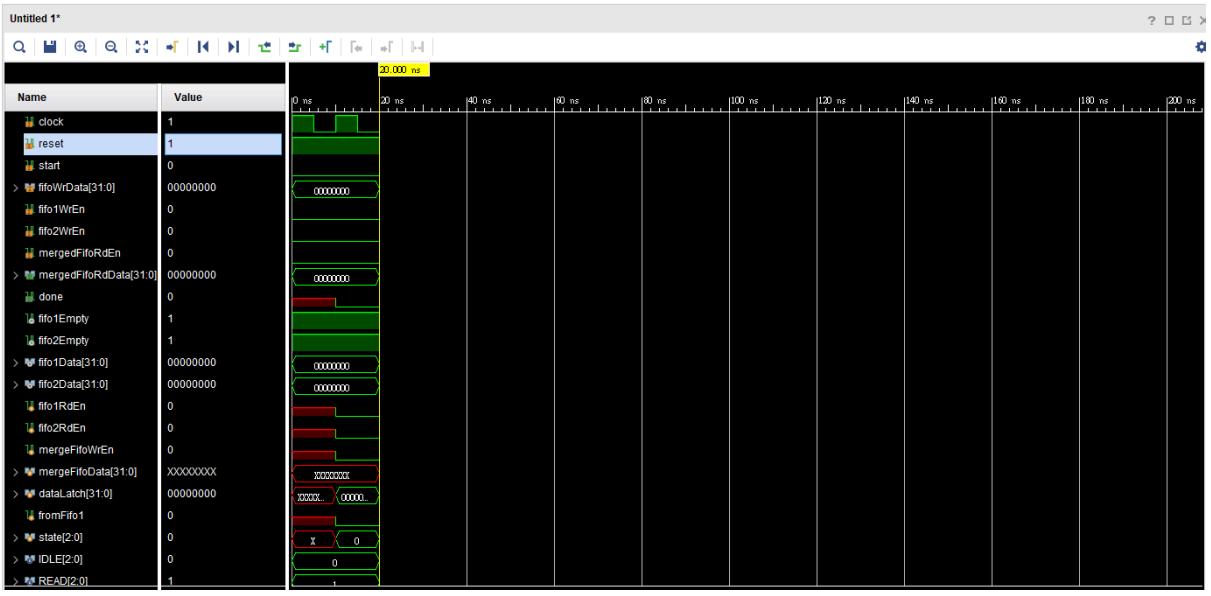


mergedFifo, mergedFifo add to new window

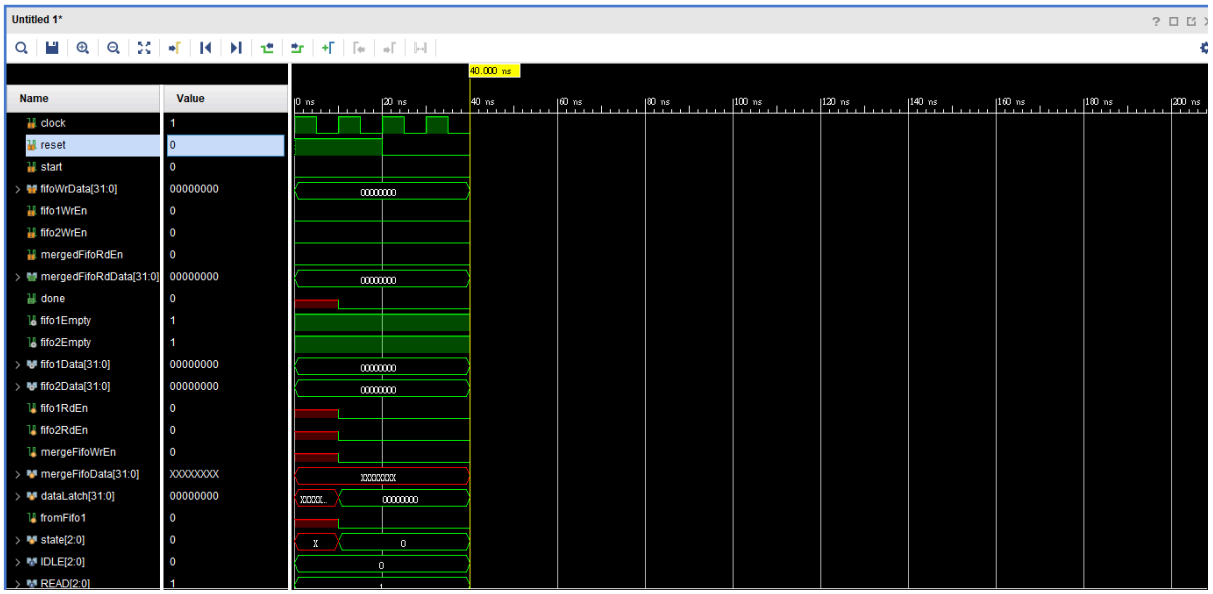


clock setting	reset setting	start setting
		
fifoWrData setting	**fifo1WrEn setting**	**fifo2WrEn setting**
		
mergedFifoRdEn setting		





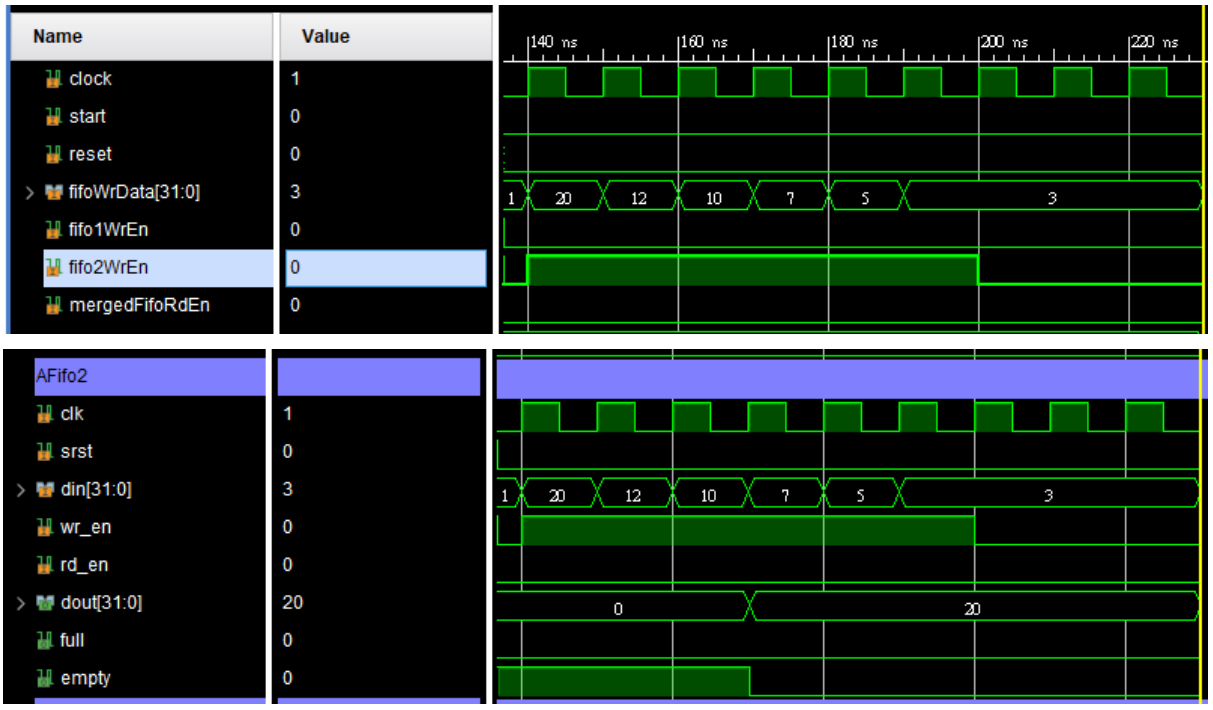
reset force constant = 0



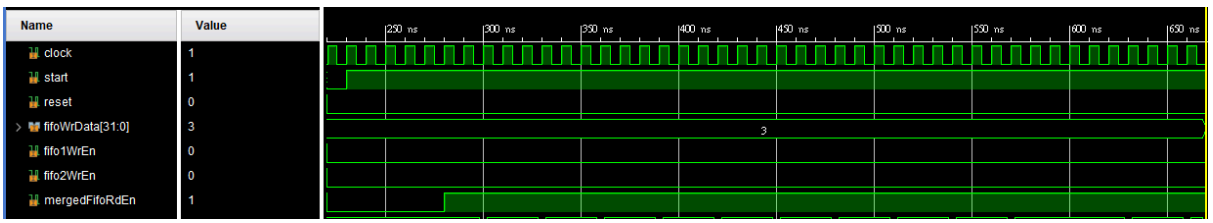
fifo1WrEn = 1

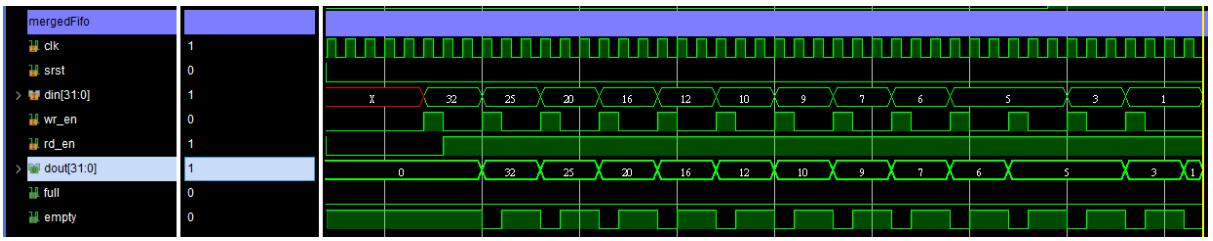
fifoWrData = 32, 25, 16 , 9, 6, 5, 1

fifo1WrEn = 0
fifo2WrEn = 1
fifoWrData = 20, 12, 10, 7, 5, 3



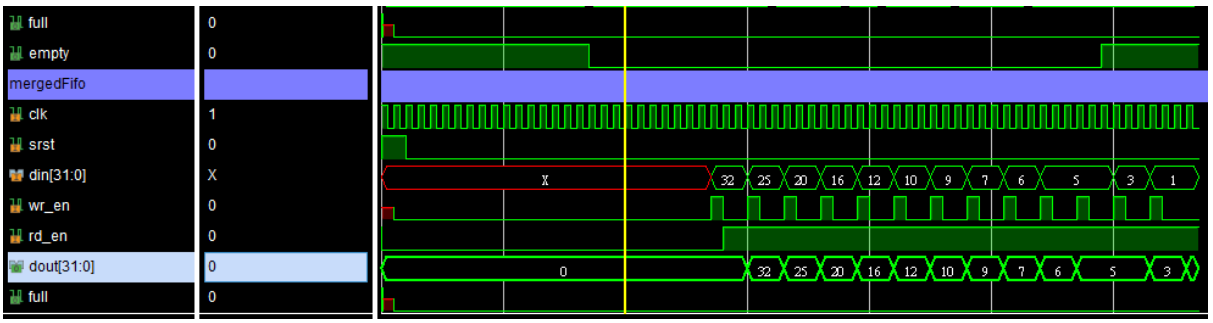
start = 1
mergedFifoRdEn = 1





merge result: mergedData = 32, 25, 20, 16, 12, 10, 9, 7, 6, 5, 5, 3, 1

Total simulation



結果:

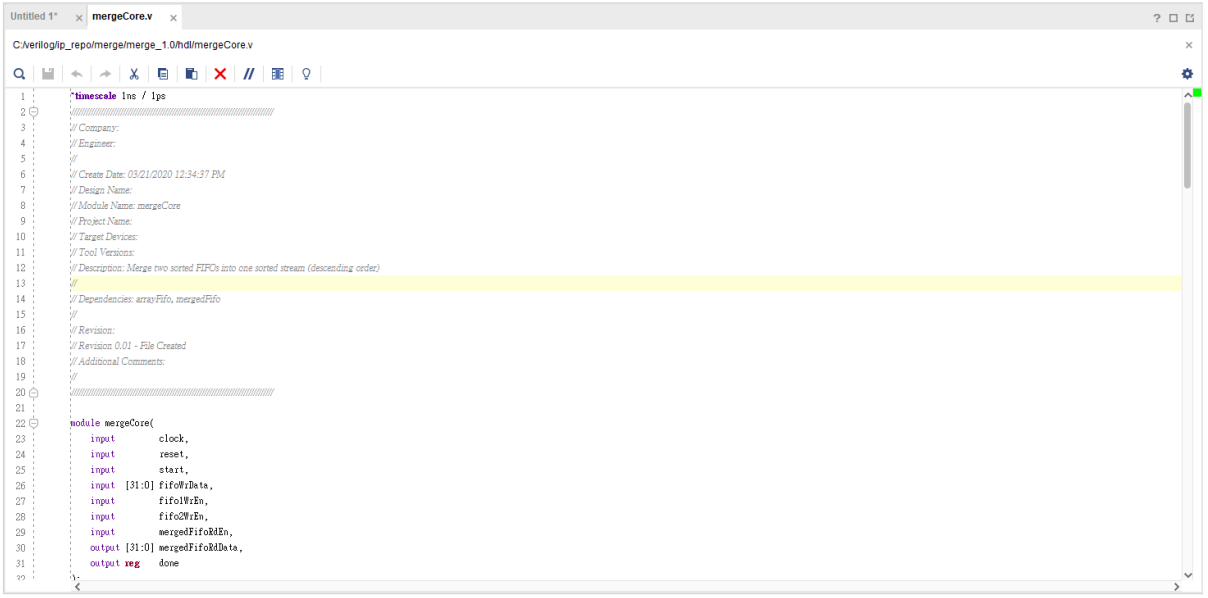
Explain your code

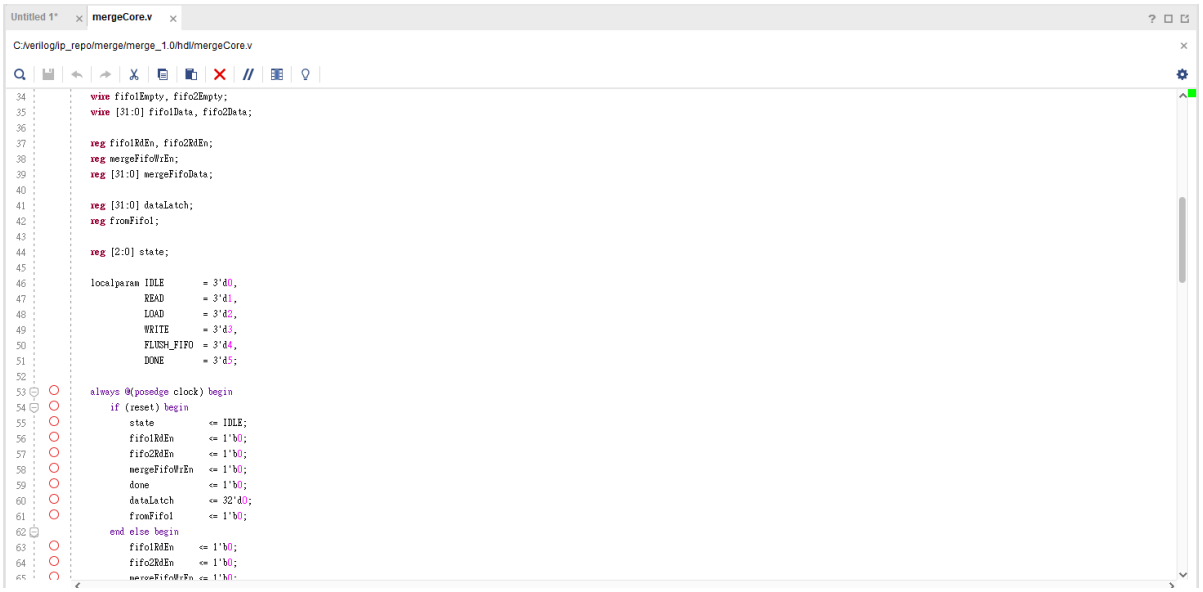
1. FIFO 緩衝區：設計中分別使用兩個 FIFO（arrayFifo1、arrayFifo2）存放輸入的兩組已排序陣列，並用一個 mergedFifo 存放合併後的結果。FIFO 由 Xilinx FIFO Generator IP 產生，確保資料能以先進先出方式處理。
2. 合併邏輯：mergeCore.v 會將兩個已排序（由大到小）的陣列資料，分別寫入兩個 FIFO（arrayFifo1 與 arrayFifo2），並將合併後的結果（同樣由大到小排序）寫入第三個 FIFO（mergedFifo）。整個合併過程由狀態機自動完成，並以 done 訊號通知處理器運算結束，而FSM的狀態說明如下：

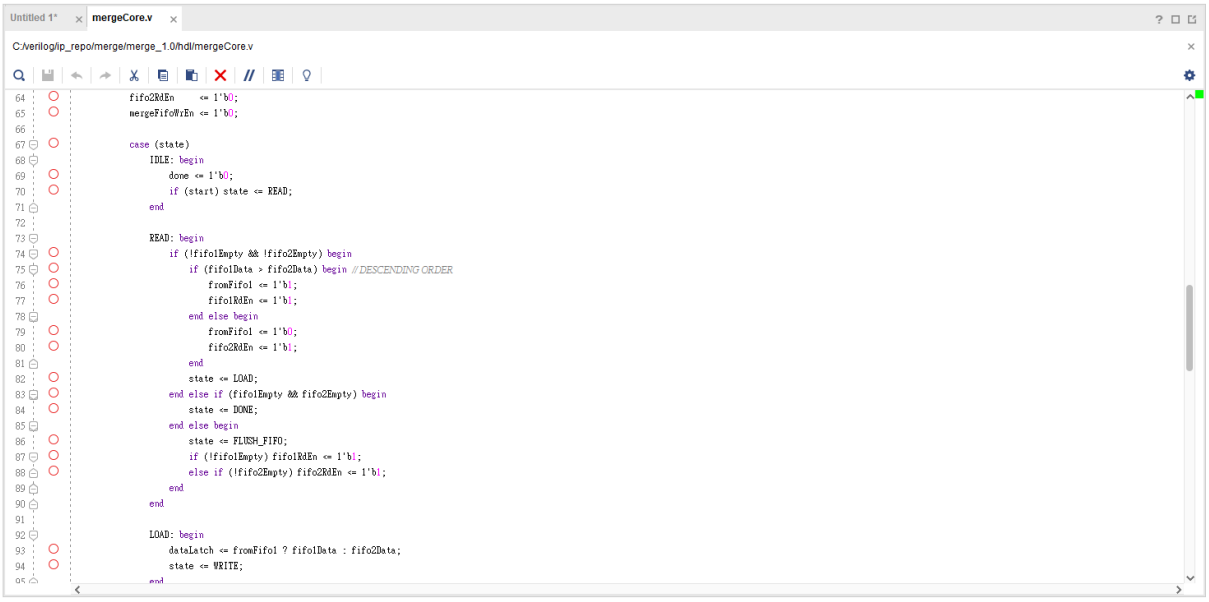
- IDLE：待命狀態，等待 start 訊號。
- READ：同時檢查兩個 FIFO 是否有資料，若都有資料則比較大小，決定從哪個 FIFO 取資料。若有一個 FIFO 已空，進入 FLUSH_FIFO 狀態。
- LOAD：將剛剛比較後選中的資料暫存到 dataLatch。
- WRITE：將暫存資料寫入 mergedFifo。
- FLUSH_FIFO：當其中一個 FIFO 已空，將另一個 FIFO 剩下的資料全部搬到 mergedFifo。
- DONE：合併結束，done 設為 1，等待 start 解除後回到 IDLE。

3. 狀態機設計：合併流程由狀態機控制，包括 Idle（待命）、Compare（比較）、Write（寫入）、Flush（清空剩餘資料）、Done（完成）等狀態。狀態機會根據控制暫存器啟動運算，並於完成後更新狀態暫存器，供處理器查詢2。
4. AXI4 介面：merge_v1_0_S00_AXI.v 作為 AXI wrapper，負責與處理器的資料與控制訊號溝通，包含資料寫入、啟動、狀態查詢與結果讀取等功能。
5. 軟體驗證：SDK 端程式碼依據 register map，將兩組陣列分別寫入對應暫存器，啟動合併，輪詢狀態暫存器，待完成後讀回合併結果，並透過 Tera Term 顯示驗證結果

(程式碼圖片)


Untitled 1*
mergeCore.v
C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v
`timescale 1ns / 1ps
//
// Company:
// Engineer:
//
// Create Date: 03/21/2020 12:34:37 PM
// Design Name:
// Module Name: mergeCore
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Merge two sorted FIFOs into one sorted stream (descending order)
//
// Dependencies: arrayFifo, mergedFifo
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
module mergeCore(
input clock,
input reset,
input start,
input [31:0] fifoWrData,
input fifo1WrEn,
input fifo2WrEn,
input mergedFifoRdEn,
output [31:0] mergedFifoRdData,
output reg done


Untitled 1*
mergeCore.v
C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v
wire fifo1Empty, fifo2Empty;
wire [31:0] fifo1Data, fifo2Data;
reg fifo1RdEn, fifo2RdEn;
reg mergeFifoWrEn;
reg [31:0] mergeFifoData;
reg [31:0] dataLatch;
reg fromFifo1;
reg [2:0] state;
localparam IDLE = 3'd0,
READ = 3'd1,
LOAD = 3'd2,
WRITE = 3'd3,
FLUSH_FIFO = 3'd4,
DONE = 3'd5;
always @(posedge clock) begin
if (reset) begin
state <= IDLE;
fifo1RdEn <= 1'b0;
fifo2RdEn <= 1'b0;
mergeFifoWrEn <= 1'b0;
done <= 1'b0;
dataLatch <= 32'd0;
fromFifo1 <= 1'b0;
end else begin
fifo1RdEn <= 1'b0;
fifo2RdEn <= 1'b0;


Untitled 1*
mergeCore.v
C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v
fifo2RdEn <= 1'b0;
mergeFifoWrEn <= 1'b0;
case (state)
IDLE: begin
done <= 1'b0;
if (start) state <= READ;
end
READ: begin
if (!fifo1Empty && !fifo2Empty) begin
if (fifo1Data > fifo2Data) begin // DESCENDING ORDER
fromFifo1 <= 1'b1;
fifo1RdEn <= 1'b1;
end else begin
fromFifo1 <= 1'b0;
fifo2RdEn <= 1'b1;
end
state <= LOAD;
end else if (fifo1Empty && fifo2Empty) begin
state <= DONE;
end else begin
state <= FLUSH_FIFO;
if (!fifo1Empty) fifo1RdEn <= 1'b1;
else if (!fifo2Empty) fifo2RdEn <= 1'b1;
end
end
LOAD: begin
dataLatch <= fromFifo1 ? fifo1Data : fifo2Data;
state <= WRITE;

Untitled 1* | mergeCore.v

C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v

```
                LOAD: begin
                    dataLatch <= fromFifo1 ? fifo1Data : fifo2Data;
                    state <= WRITE;
                end

                WRITE: begin
                    mergeFifoWrEn <= 1'b1;
                    mergeFifoData <= dataLatch;
                    state <= READ;
                end

                FLUSH_FIFO: begin
                    if (!fifo1Empty) begin
                        fifo1RdEn     <= 1'b1;
                        dataLatch     <= fifo1Data;
                        fromFifo1     <= 1'b1;
                        state         <= WRITE;
                    end else if (!fifo2Empty) begin
                        fifo2RdEn     <= 1'b1;
                        dataLatch     <= fifo2Data;
                        fromFifo1     <= 1'b0;
                        state         <= WRITE;
                    end else begin
                        state <= DONE;
                    end
                end

                DONE: begin
                    done <= 1'b1;
                    if (!start)
                        state <= IDLE;
```

Untitled 1* | mergeCore.v

C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v

```
                    if (!start)
                        state <= IDLE;
                end
            endcase
        end
    end

    // FIFO instantiations
    arrayFifo arrayFifo1 (
        .clk(clock),
        .srst(reset),
        .din(fifoWrData),
        .wr_en(fifo1WrEn),
        .rd_en(fifo1RdEn),
        .dout(fifo1Data),
        .full(),
        .empty(fifo1Empty)
    );

    arrayFifo arrayFifo2 (
        .clk(clock),
        .srst(reset),
        .din(fifoWrData),
        .wr_en(fifo2WrEn),
        .rd_en(fifo2RdEn),
        .dout(fifo2Data),
        .full(),
        .empty(fifo2Empty)
    );

    mergedFifo mergedFifo (
        .clk(clock),
```

Untitled 1* | mergeCore.v

C:/verilog/ip_repo/merge/merge_1.0/hdl/mergeCore.v

```
        .wr_en(fifo1WrEn),
        .rd_en(fifo1RdEn),
        .dout(fifo1Data),
        .full(),
        .empty(fifo1Empty)
    );

    arrayFifo arrayFifo2 (
        .clk(clock),
        .srst(reset),
        .din(fifoWrData),
        .wr_en(fifo2WrEn),
        .rd_en(fifo2RdEn),
        .dout(fifo2Data),
        .full(),
        .empty(fifo2Empty)
    );

    mergedFifo mergedFifo (
        .clk(clock),
        .srst(reset),
        .din(mergeFifoData),
        .wr_en(mergeFifoWrEn),
        .rd_en(mergedFifoRdEn),
        .dout(mergedFifoRdData),
        .full(),
        .empty()
    );

endmodule
```

Lesson learned

- IP 設計流程熟悉：學會如何從 Verilog 硬體描述語言設計出自訂 IP，並透過 Vivado 工具進行包裝與測試，熟悉 Xilinx IP 開發與整合流程。
- FIFO Generator 應用：實際操作 Xilinx FIFO Generator，理解 FIFO 在資料緩衝與佇列處理的應用與重要性。
- 硬體/軟體協同設計：體會到硬體與軟體協同設計的重要性，從 register map 規劃到 AXI 介面設計，確保處理器與自訂 IP 能順利互動。
- 狀態機與流程控制：深入理解狀態機在硬體運算流程控制的角色，學會設計簡潔且可維護的狀態切換邏輯。
- 系統整合與驗證：從 IP 包裝、Block Design 建置、HDL Wrapper 生成、bitstream 燒錄到 SDK 驗證，完整經歷 FPGA 系統開發與驗證流程。
- 除錯與驗證經驗：學會如何利用 SDK、Tera Term 等工具進行系統除錯與驗證，提升問題排查與解決能力。