專題名稱：《低功耗感測控制模擬器（簡易版）》期末作業

**🧠 一、專題概念與背景介紹**

在物聯網（IoT）設備中，許多感測器（如溫度、濕度、光照）不需要一直啟動，為了**節省電力與系統資源**，我們會讓這些感測器\*\*「週期性啟動」\*\*。

本專題即是用 Verilog 撰寫一個 FSM 模擬這種行為，模擬一個裝置，每經過一定時間就讓感測器開啟一下（用 data\_current=1 表示開啟），其餘時間為關閉狀態（data\_current=0）。

### 💡 ****二、創新點****

1. **模擬感測器啟動與關閉行為**（資料流節能控制）
2. 可結合未來真實的 UART 傳輸、藍牙啟動等行為
3. 可自訂「感測開啟模式」：例如 2個時脈開、6個時脈關（節能策略）
4. 可套用在智慧農業、智慧居家節電裝置設計報告中

🔧 **三、FSM 模組程式碼**

🔧《FSM\_low\_power\_sensor.v》

// 低功耗感測控制模組

// 低功耗感測控制模組

module FSM\_low\_power\_sensor (

input clk, // 時脈輸入

input rst\_n, // 非同步低態有效重置信號

output reg data\_current // 感測器是否啟動：1=啟動，0=關閉

);

reg [3:0] state\_current, state\_next; // 目前狀態與下一狀態

reg data\_next; // 下一個狀態的感測器狀態

// 狀態編碼

localparam

IDLE1 = 4'd0, // 閒置狀態 1

IDLE2 = 4'd1, // 閒置狀態 2

IDLE3 = 4'd2, // 閒置狀態 3

IDLE4 = 4'd3, // 閒置狀態 4

IDLE5 = 4'd4, // 閒置狀態 5

SENSE1 = 4'd5, // 感測狀態 1

SENSE2 = 4'd6; // 感測狀態 2

// 同步狀態更新

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) // 當重置信號有效時，設置狀態為 IDLE1

state\_current <= IDLE1;

else

state\_current <= state\_next; // 更新狀態

end

// 組合邏輯決定下一狀態與輸出

always @\* begin

case (state\_current)

IDLE1: begin state\_next = IDLE2; data\_next = 1'b0; end // 持續閒置

IDLE2: begin state\_next = IDLE3; data\_next = 1'b0; end

IDLE3: begin state\_next = IDLE4; data\_next = 1'b0; end

IDLE4: begin state\_next = IDLE5; data\_next = 1'b0; end

IDLE5: begin state\_next = SENSE1; data\_next = 1'b0; end // 進入感測狀態

SENSE1: begin state\_next = SENSE2; data\_next = 1'b1; end // 啟動感測器

SENSE2: begin state\_next = IDLE1; data\_next = 1'b1; end // 返回閒置狀態

default: begin state\_next = IDLE1; data\_next = 1'b0; end

endcase

end

// 資料輸出暫存器

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n)

data\_current <= 1'b0; // 重置時，感測器關閉

else

data\_current <= data\_next; // 更新感測器狀態

end

endmodule

🔧《FSM\_low\_power\_sensor\_tb.v》

`timescale 1ns/1ns

`define clk\_period 20 // 20ns 時脈週期

module FSM\_low\_power\_sensor\_tb;

reg clk; // 時脈信號

reg rst\_n; // 非同步重置信號

wire data\_current; // 感測器狀態輸出

// 實例化 DUT（Device Under Test）

FSM\_low\_power\_sensor uut (

.clk(clk),

.rst\_n(rst\_n),

.data\_current(data\_current)

);

// 時脈產生器

initial begin

clk = 0;

forever #(`clk\_period/2) clk = ~clk; // 產生週期性時脈信號

end

// 測試流程

initial begin

// 初始化並進行重置

rst\_n = 0;

#100; // 保持重置狀態一段時間

rst\_n = 1;

// 模擬運行一段時間後停止

#3000;

$stop;

end

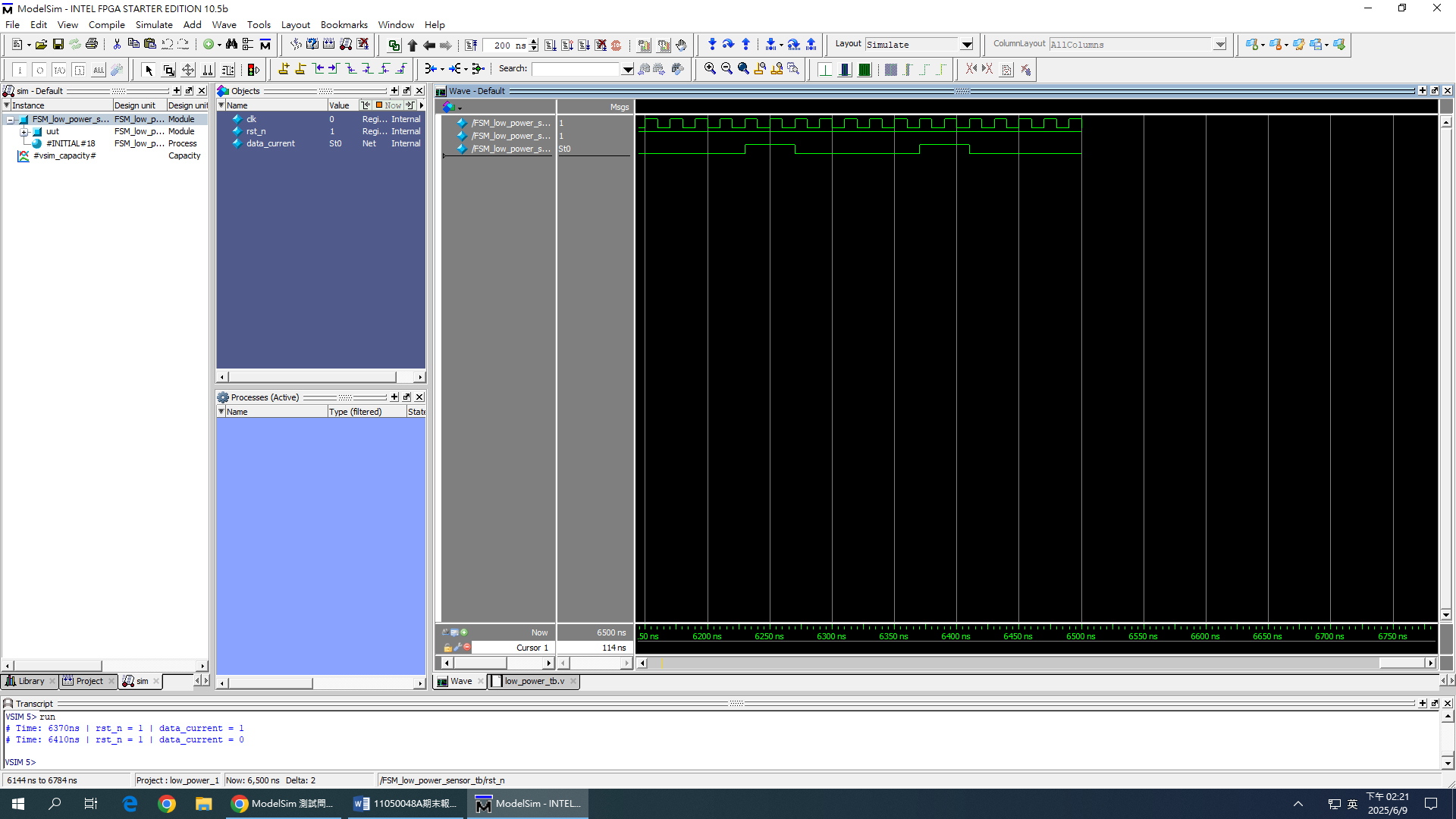
// 顯示測試訊息，方便觀察輸出值

initial begin

$monitor("Time: %0dns | rst\_n = %b | data\_current = %b", $time, rst\_n, data\_current);

end

endmodule



「週期性開啟感測器（FSM 控制 data\_current = 1，其餘時間為 0）」的確能 **達成省電效果**，下面是原因分析：

**是否比較省電？**

**🔍 為什麼這樣會省電？**

| **機制** | **說明** | **對功耗的影響** |
| --- | --- | --- |
| **週期性開關感測器** | 感測器不是持續工作，而是「定時才啟動」收集資料，例如每 8 個時脈只開 2 個時脈。 | 感測器在關閉狀態時不消耗或幾乎不消耗電力，大幅降低功耗。 |
| **FSM 控制啟動時機** | 使用有限狀態機（FSM）可精準控制感測器開啟的時間與週期，避免不必要的運作。 | 控制邏輯簡單，不需持續處理資料，節省 MCU 運算與感測器功耗。 |
| **數位訊號控制（data\_current）** | 透過 Verilog 控制 signal（如 data\_current = 1/0），只在需要時才啟用 sensor。 | 有效降低整體系統的活動率（activity rate），間接降低電流消耗。 |

**🔋 範例（假設情境）：**

* 感測器持續開啟功耗：10mW
* 使用 FSM 控制：每 10 單位時間只開 2 單位
* ➜ 平均功耗降為：

210×10mW=2mW\frac{2}{10} \times 10mW = 2mW102​×10mW=2mW

👉 省下 **80% 電力**

**💡 小總結：**

你的 FSM 週期性控制設計，屬於**Duty Cycling** 的節電策略，這在：

* 智慧農業
* 居家自動化
* 電池驅動裝置（例如感測節點）

中都**非常常見且有效**，是實作節電裝置的正確方向。