EOS 期末報告

系級: 電機四

組員: 陳冠維、林庭毅、陳祥均、羅豐祥

1 AES(解密未實現)

暫存器分配(14 個暫存器、32-bits data width)

1.1 系統控制信號

- clk : 系統時鐘（設計頻率：200MHz，週期5ns）
- reset : 重置信號（高電平有效，同步重置）
- start : 啟動加密/解密操作（正邊緣脈衝觸發）
- done : 操作完成信號（高電平表示完成）

1.2 模式控制

- mode : 模式選擇信號
  - 1 = 加密模式（Encryption）
  - 0 = 解密模式（Decryption）

1.3 資料載入控制

- load : 載入控制信號
  - 需要**兩個時鐘週期**完成128位元資料載入
  - 第一週期：load 0→1 觸發高64位元載入
  - 第二週期：load 1→0 觸發低64位元載入

1.4 資料介面

- key[63:0] : 64位元密鑰輸入
  - 總共需載入兩次完成128位元密鑰
  - 載入順序：先高位[127:64]，後低位[63:0]
- data_in[63:0] : 64位元明文/密文輸入
  - 總共需載入兩次完成128位元資料
  - 載入順序：先高位[127:64]，後低位[63:0]
- data_out[127:0] : 128位元輸出資料
  - 一次性輸出完整的128位元結果

1.5 載入時序示例

週期1: load=1, key[63:0]=key[127:64], data_in[63:0]=plaintext[127:64]

週期2: load=0, key[63:0]=key[63:0],   data_in[63:0]=plaintext[63:0]

週期3+: load=0, start=1（開始運算）

1.6 加密&解密過程（10回合，以加密為例）

1. **初始回合密鑰加法**（Round 0）
    - AddRoundKey：明文⊕初始密鑰
2. **主要回合**（Round 1~9）
    - SubBytes: S-box位元組替換
    - ShiftRows: 行位移轉換
    - MixColumns: 混合行運算
    - AddRoundKey: 回合密鑰加法
3. **最終回合**（Round 10）
    - SubBytes: S-box位元組替換
    - ShiftRows: 行位移轉換
    - AddRoundKey: 最終回合密鑰加法（跳過MixColumns）

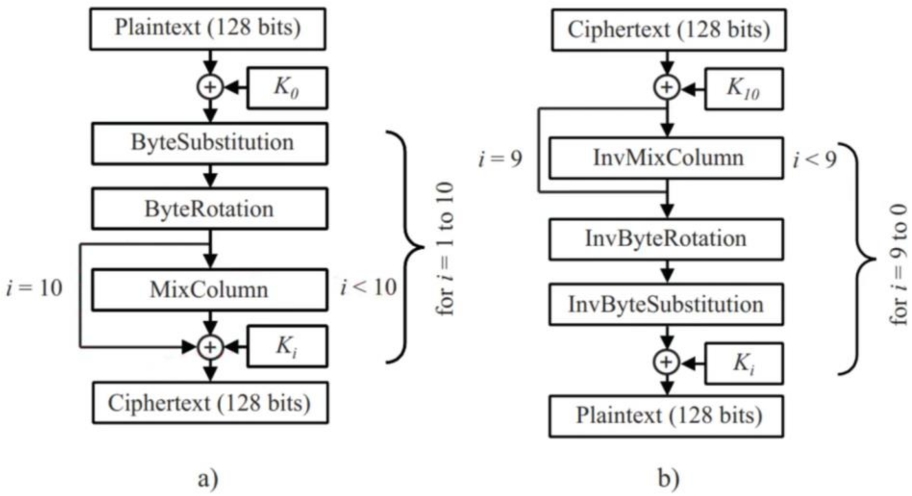


圖1.1.1 加密與解密流程圖

1.7 解密失敗原因分析

最關鍵原因--load存在控制競爭

1. **數據載入和密鑰擴展的時序不同步：**當我在輸入時序加入peridoc_clock時，每次解密的輸出都不同。
2. **解密時的密鑰調度順序可能有誤：**

   加密時：

- 數據在輪 1-9 需要 MixColumns ✓（代碼正確）

- 數據在輪 10 不需要 MixColumns ✓（代碼正確）
- 密鑰永遠不需要 MixColumns ✓（代碼正確）

解密時（等效解密算法）：

- 數據在解密輪 1 不需要 InvMixColumns ✓（代碼正確）
- 數據在解密輪 2-10 需要 InvMixColumns ✓（代碼正確）
- 密鑰在使用輪 1-9 的密鑰時需要 InvMixColumns
    - 但代碼在 round_cnt = 1（使用輪 9 密鑰）時沒有處理 ✕
    - 代碼在 round_cnt = 10（使用輪 0 密鑰）時正確地沒有處理 ✓

3. MixColumns 在解密流程中的應用時機不正確

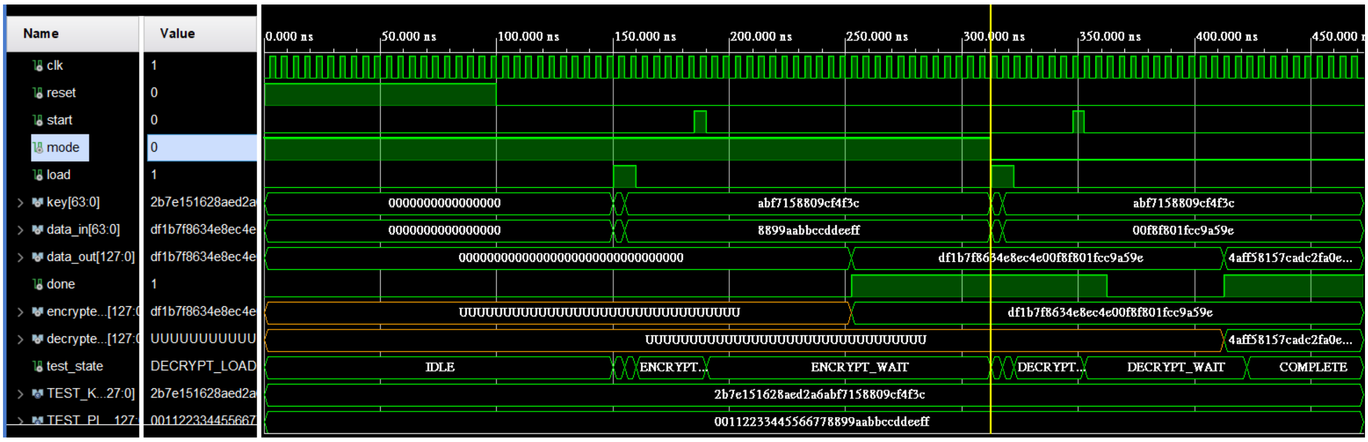


圖1.1.2 testbench時序圖(黃線以左(mode 1 加密)，黃線以右(mode 1 解密))

```
key_select_0 <= new_key0_d1 when (mode = '1') else
                mixcol_key_0 when(mode = '0' and round_cnt > 1 and round_cnt < 10) else
                new_key0_d1;
key_select_1 <= new_key1_d1 when (mode = '1') else
                mixcol_key_1 when(mode = '0' and round_cnt > 1 and round_cnt < 10) else
                new_key1_d1;
key_select_2 <= new_key2_d1 when (mode = '1') else
                mixcol_key_2 when(mode = '0' and round_cnt > 1 and round_cnt < 10) else
                new_key2_d1;
key_select_3 <= new_key3_d1 when (mode = '1') else
                mixcol_key_3 when(mode = '0' and round_cnt > 1 and round_cnt < 10) else
                new_key3_d1;
done <= done_d2;
```

圖1.1.2 錯誤模塊

2 DES

2.1 暫存器配置

- **實際暫存器數量**: 16個暫存器（slv_reg0 到 slv_reg15）
- **資料寬度**: 32位元（不是128位元）
- **時鐘**: S_AXI_ACLK（AXI系統時鐘）
- **重置**: S_AXI_ARESETN（低電平有效，異步重置）

2.2. 加密演算法

- **實際演算法**: **DES**（Data Encryption Standard）
- **資料寬度**: 64位元（不是128位元AES）
- **密鑰長度**: 64位元（不是128位元）

2.3. 暫存器分配

功能	**64-bit 寄存器對應**	**低位（bits [31:0]）**	**高位（bits [63:32]）**
金鑰（Key）	key_std = {slv_reg3, slv_reg2}	slv_reg2	slv_reg3
輸入資料（Plaintext／Ciphertext）	data_bus_std = {slv_reg1, slv_reg0}	slv_reg0	slv_reg1
輸出結果（Result）	寄存器寫回：slv_reg5 + slv_reg6	slv_reg5 （只讀）	slv_reg6 （只讀）

2.4. 操作流程

1. 寫入64位元輸入資料到 slv_reg0（低32位）和 slv_reg1（高32位）
2. 寫入64位元密鑰到 slv_reg2（低32位）和 slv_reg3（高32位）
3. 設定 slv_reg4[1] 選擇加密/解密模式
4. 設定 slv_reg4[0] = 1 啟動操作
5. 等待 slv_reg7[0] = 1 表示操作完成
6. 從 slv_reg5 和 slv_reg6 讀取64位元輸出結果

2.5. DES演算法特點

- **回合數**: 16回合（AES是10回合）
- **區塊大小**: 64位元
- **密鑰長度**: 64位元（實際有效的只有56位元）

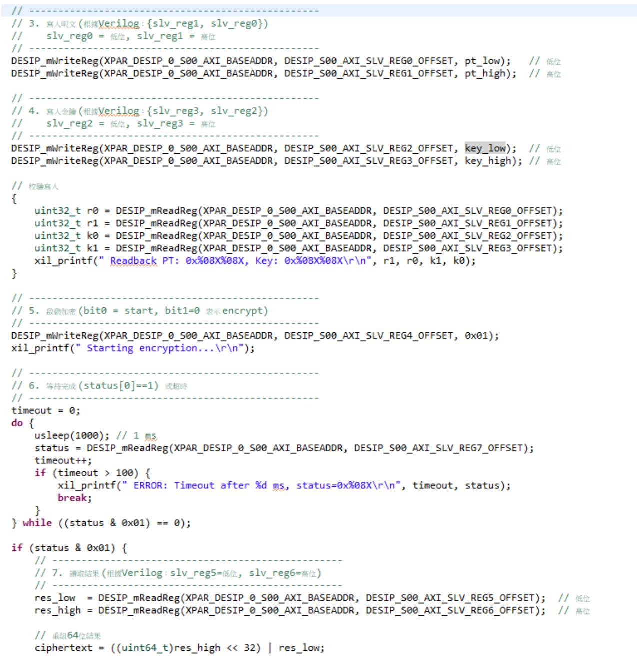
2.6. 位元順序轉換

代碼中有大量的位元順序轉換，將標準的［63:0］格式轉換為DES模組使用的［1:64］格式，其中：

- ［1:64］格式：位元1為LSB，位元64為MSB
- ［63:0］格式：位元0為LSB，位元63為MSB

2.7. 測試

這段測試程式碼的邏輯是針對一個加密模組進行驗證的，主要透過 AXI 寄存器介面與硬體模組進行溝通。首先寫入明文資料到對應的暫存器（slv_reg0），再寫入金鑰到另一組暫存器（slv_reg2 和 slv_reg3）。接著讀取這些值以確認資料已正確寫入。之後，透過寫入控制寄存器來觸發加密操作（bit0=1 表示啟動加密）。啟動後進入輪詢機制，不斷檢查狀態寄存器的 bit0 是否變為 1，表示加密完成；若逾時則跳出。完成後讀取結果寄存器（slv_reg5 與 slv_reg6）以取得加密後的密文，並將低位與高位組合成一個 64-bit 的結果。整體邏輯是「設定→啟動→等待→讀取結果」，用來驗證加密硬體模組的正確性與功能。



```c
// ------------------------------------------------
// 3. 寫入明文 (根據Verilog：{slv_reg1, slv_reg0})
//    slv_reg0 = 低位, slv_reg1 = 高位
// ------------------------------------------------
DESIP_mWriteReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG0_OFFSET, pt_low);   // 低位
DESIP_mWriteReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG1_OFFSET, pt_high);  // 高位

// ------------------------------------------------
// 4. 寫入金鑰 (根據Verilog：{slv_reg3, slv_reg2})
//    slv_reg2 = 低位, slv_reg3 = 高位
// ------------------------------------------------
DESIP_mWriteReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG2_OFFSET, key_low);  // 低位
DESIP_mWriteReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG3_OFFSET, key_high); // 高位

// 校驗寫入
{
    uint32_t r0 = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG0_OFFSET);
    uint32_t r1 = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG1_OFFSET);
    uint32_t k0 = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG2_OFFSET);
    uint32_t k1 = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG3_OFFSET);
    xil_printf(" Readback PT: 0x%08X%08X, Key: 0x%08X%08X\r\n", r1, r0, k1, k0);
}

// ------------------------------------------------
// 5. 啟動加密 (bit0 = start, bit1=0 表示 encrypt)
// ------------------------------------------------
DESIP_mWriteReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG4_OFFSET, 0x01);
xil_printf(" Starting encryption...\r\n");

// ------------------------------------------------
// 6. 等待完成 (status[0]==1) 或超時
// ------------------------------------------------
timeout = 0;
do {
    usleep(1000); // 1 ms
    status = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG7_OFFSET);
    timeout++;
    if (timeout > 100) {
        xil_printf(" ERROR: Timeout after %d ms, status=0x%08X\r\n", timeout, status);
        break;
    }
} while ((status & 0x01) == 0);

if (status & 0x01) {
    // ------------------------------------------------
    // 7. 讀取結果 (根據Verilog：slv_reg5=低位, slv_reg6=高位)
    // ------------------------------------------------
    res_low  = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG5_OFFSET);  // 低位
    res_high = DESIP_mReadReg(XPAR_DESIP_0_S00_AXI_BASEADDR, DESIP_S00_AXI_SLV_REG6_OFFSET);  // 高位

    // 重組64位結果
    ciphertext = ((uint64_t)res_high << 32) | res_low;
```

圖2.7.1 測試代碼

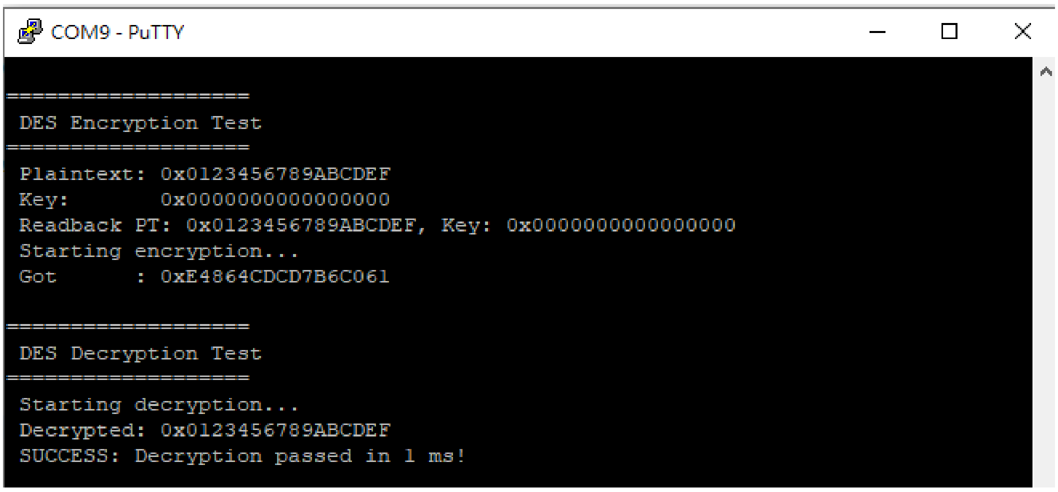


圖2.7.2 測試結果輸出

3. GCD

3.1 暫存器配置

- **實際暫存器數量**: 4個暫存器 (slv_reg0 到 slv_reg3)
- **資料寬度**: 32位元
- **時鐘**: S_AXI_ACLK (AXI系統時鐘)
- **重置**: S_AXI_ARESETN (低電平有效，異步重置)

3.2. 計算演算法

- **實際演算法**: **GCD** (Greatest Common Divisor - 最大公因數)
- **輸入資料寬度**: 8位元 (使用32位元暫存器的低8位)
- **輸出資料寬度**: 8位元
- **計算方法**: 歐幾里得演算法

3.3. 暫存器分配

功能	暫存器	位元範圍	說明
輸入數值 X	slv_reg0	[7:0]	第一個輸入數值 (僅使用低8位)
輸入數值 Y	slv_reg1	[7:0]	第二個輸入數值 (僅使用低8位)
控制暫存器	slv_reg2	[0]	啟動信號 (寫入1啟動計算)
輸出結果	slv_reg3	[7:0]	GCD計算結果 **(只讀)**

3.4. 操作流程

1. 寫入第一個8位元數值到 slv_reg0[7:0]
2. 寫入第二個8位元數值到 slv_reg1[7:0]
3. 設定 slv_reg2[0] = 1 啟動GCD計算
4. 系統自動檢測啟動脈衝並開始計算
5. 等待計算完成 (通常在1ms內完成)
6. 從 slv_reg3[7:0] 讀取8位元GCD結果

3.5. GCD演算法特點

- **演算法類型**: 歐幾里得演算法 (Euclidean Algorithm)
- **計算複雜度**: O(log(min(x,y)))
- **硬體實現**: 迭代式設計，節省硬體資源
- **計算時間**: 根據測試結果，所有計算皆在2ms內完成，但仍需增加狀態否則結果只會取回0

3.6. 脈衝檢測機制

代碼中實現了完整的脈衝檢測系統，我透過兩變數 start_pulse, done_pulse 去檢測開始與結束計算的過程，確保計算的正確啟動和結束。

```
// 生成GCD核心需要的高電平復位信號
assign rst_gcd = ~S_AXI_ARESETN;

// 邊緣檢測邏輯
assign start_pulse = slv_reg2[0] & ~start_prev;   // 檢測start的上升邊緣
assign done_pulse = done_i & ~done_prev;          // 檢測done的上升邊緣
```

圖3.6.1 脈衝檢測機制代碼

3.7. 狀態管理

透過更改gcdip.vhd (line 141)，將傳出訊號done (脈衝波)，作為計算完成的狀態訊號，並在實例化gcd計算加入done端口得到狀態 (gcdip_v1_0_S00_AXI.v line 460)

```
130     OUT_REG: regis port map(
131                 rst         =>  rst,
132                 clk         =>  clk,
133                 load        =>  enable,
134                 input       =>  xsub,
135                     output  =>  result
136                     );
137
138     d_o <= result;
139
140     --傳出完成狀態
141     done <= enable;
```

圖3.7.1狀態輸出代碼結構

- start_prev: 儲存前一個時鐘週期的啟動狀態，用於邊緣檢測
- done_prev: 儲存前一個時鐘週期的完成狀態，用於邊緣檢測
- reg_start_i: 內部啟動信號，在檢測到啟動脈衝時設為高電平，完成時清除

3.8. 測試結果範例

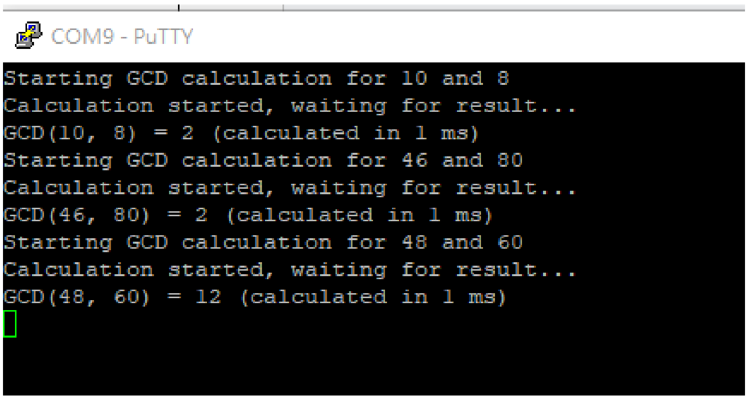


圖3.8.1 gcd測試結果輸出

4. 電路整合

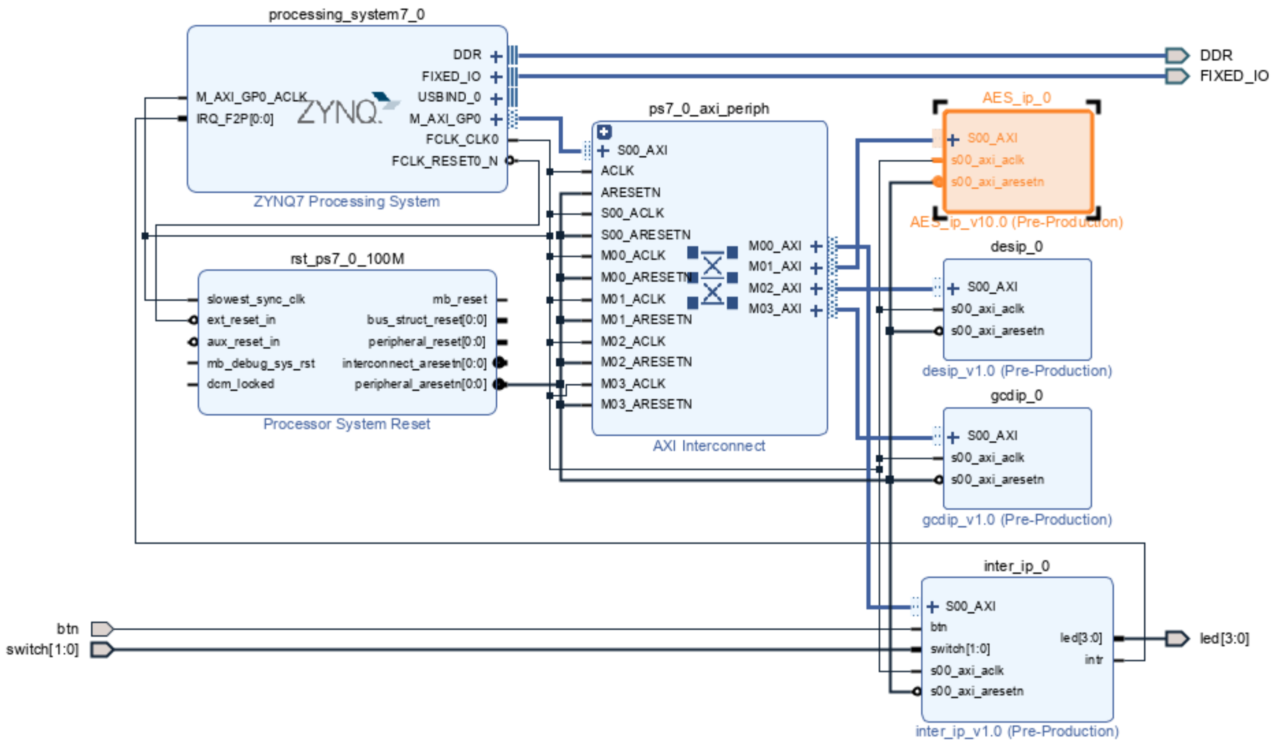


圖4.1.1 完整電路輸出

4.1 處理器系統 與 AXI 匯流排連接

使用 ZYNQ7 Processing System 做為主控核心，透過 M_AXI_GP0 匯流排與 AXI Interconnect 相連，負責與各個自訂 IP 模組進行 AXI4-Lite 的通訊。並透過 ps7_0_axi_periph（AXI Interconnect），將 PS 與三個 IP 使用 AXI-Lite Slave 通道（S00_AXI）進行對接。包含AES_ip_0（加密模組）、desip_0（可能為 DES 加密模組）、gcdip_0（最大公因數運算模組）：

4.2 Reset 與時脈控制

- 利用 Processor System Reset 模組產生各模組所需的 reset 訊號（s00_axi_aresetn）與時脈（s00_axi_aclk），確保所有模組同步啟動。

- 系統時脈由 Zynq PS 提供，100MHz 或其他頻率來源。

4.3 中斷整合

- 你將中斷來源 inter_ip_0（整合了按鍵與開關的輸入模組）連接到 PS 的 IRQ（IRQ_F2P[0:0]），讓 ARM 核心能接收來自 PL 端的中斷事件。

- 此模組同時處理 btn、switch 輸入與 led 控制，形成一個用於除錯與互動的界面。

5. Standalone

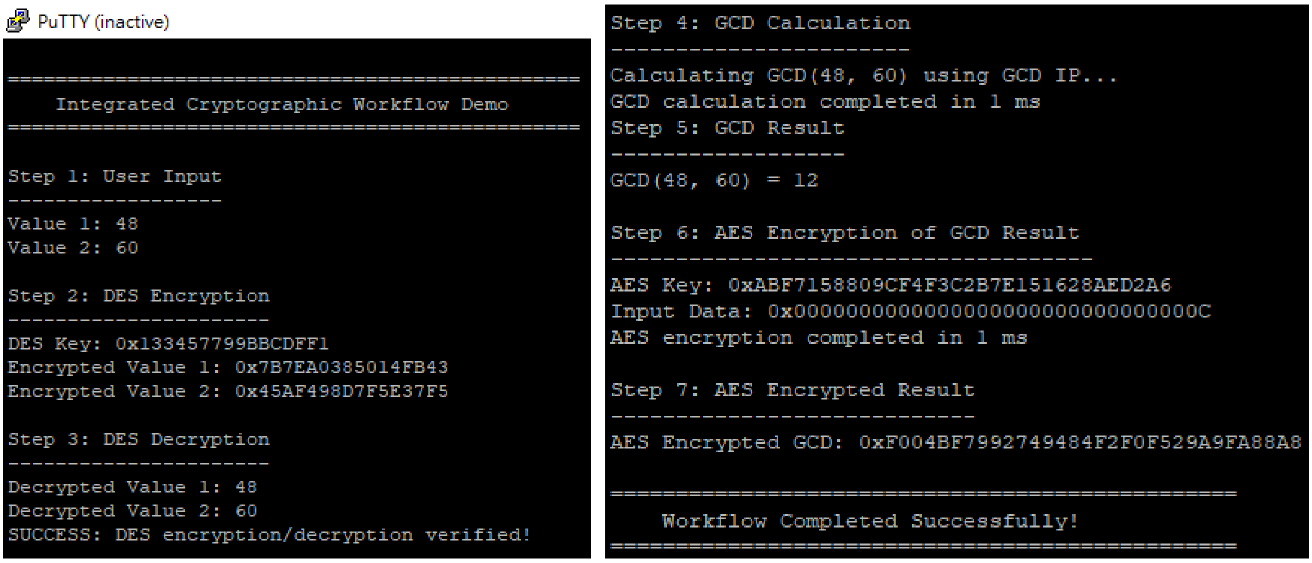


圖5.1.1 Standalone結果輸出

整個加密流程如下：

- **DES加解密**：輸入明文和產生的密文對照，顯示加密有效。
- **GCD計算結果**：計算完成的時間與GCD結果（12）。
- **AES加密GCD結果**：明確展示AES加密輸入值與輸出的加密後密文。

5.1 流程步驟：

1. **使用者輸入**：使用者提供兩個整數（例如48和60）。
2. **DES加密**：透過DES硬體IP，以64位元金鑰（如0x133457799BBCDFF1）將輸入值進行加密。
3. **DES解密**：使用DES硬體IP解密取出的資料，並以解密數據進行後續計算。
4. **GCD計算**：計算最大公因數（GCD），如輸入值48與60，則計算結果為12。
5. **AES加密GCD結果**：將GCD計算結果透過AES硬體IP（128位元金鑰，如0xABF7158809CF4F3C2B7E151628AED2A6）進行加密。
6. **結果輸出**：系統完成AES加密後輸出最終密文，如0xF004BBF7992749484F2F0F529A9FA8A8。

5.2 關鍵程式碼片段

```
// DES加密函數範例
uint64_t des_encrypt(uint64_t plaintext, uint64_t key) {
    // 設定明文與金鑰
    DESIP_mWriteReg(BASEADDR, REG_PLAINTEXT_LOW, (uint32_t)(plaintext & 0xFFFFFFFF));
    DESIP_mWriteReg(BASEADDR, REG_PLAINTEXT_HIGH, (uint32_t)(plaintext >> 32));
    DESIP_mWriteReg(BASEADDR, REG_KEY_LOW, (uint32_t)(key & 0xFFFFFFFF));
    DESIP_mWriteReg(BASEADDR, REG_KEY_HIGH, (uint32_t)(key >> 32));

    // 啟動DES加密
    DESIP_mWriteReg(BASEADDR, REG_CONTROL, START_ENCRYPTION);

    // 等待完成
    while(!(DESIP_mReadReg(BASEADDR, REG_STATUS) & STATUS_DONE));

    // 取得密文
    uint64_t ciphertext = ((uint64_t)DESIP_mReadReg(BASEADDR, REG_RESULT_HIGH) << 32)
                        | DESIP_mReadReg(BASEADDR, REG_RESULT_LOW);

    return ciphertext;
}
```

圖5.2.1 Standalone關鍵程式碼片段

6. FreeRTOS

```
=======================================
  FreeRTOS Cryptographic Workflow Demo
=======================================
All resources created successfully
Test cases to process: 5
All tasks created successfully
Starting scheduler...
>>> USER INPUT TASK STARTED <<<

>>> USER INPUT <<<
User entered values: 48, 60
>>> SYSTEM PROCESSING TASK STARTED <<<

=== SYSTEM PROCESSING CYCLE ===
No data in queue to process
>>> STATUS TASK STARTED <<<

[STATUS] Queue items waiting: 0
Encrypted value1: 0x2B3CC3C0573F28BB
Encrypted value2: 0x513BE30CDE56309C
Data queued for processing

>>> USER INPUT <<<
User entered values: 24, 36
Encrypted value1: 0xF4F0434EFECDE1CB
Encrypted value2: 0x6EA236A0A5AF150E
Data queued for processing

[STATUS] Queue items waiting: 2

>>> USER INPUT <<<
User entered values: 100, 150
Encrypted value1: 0x5FF038964F47F00B
Encrypted value2: 0x2569E52F2A144CD7
Queue full! Data lost.

=== SYSTEM PROCESSING CYCLE ===
Processing values: 48, 60

[STATUS] Queue items waiting: 1
Decrypted values: 48, 60
GCD(48, 60) = 12
AES encrypted GCD result: 0xF59C39BD7C802FF6
Processing completed successfully!

>>> USER INPUT <<<
User entered values: 100, 150
Encrypted value1: 0x5FF038964F47F00B
Encrypted value2: 0x2569E52F2A144CD7
Data queued for processing

[STATUS] Queue items waiting: 2

>>> USER INPUT <<<
User entered values: 17, 19
Encrypted value1: 0x2D95CA36844116B4
Encrypted value2: 0xCFDB52B7B81268DD
Queue full! Data lost.
```

```
Encrypted value1: 0x2D95CA36844116B4
Encrypted value2: 0xCFDB52B7B81268DD
Data queued for processing

>>> USER INPUT <<<
User entered values: 84, 126
Encrypted value1: 0xD99E074773C8FBF3
Encrypted value2: 0x9D38F45C072E1069
Queue full! Data lost.

[STATUS] Queue items waiting: 2

>>> USER INPUT <<<
User entered values: 84, 126
Encrypted value1: 0xD99E074773C8FBF3
Encrypted value2: 0x9D38F45C072E1069
Queue full! Data lost.

=== SYSTEM PROCESSING CYCLE ===
Processing values: 100, 150

[STATUS] Queue items waiting: 1
Decrypted values: 100, 150
GCD(100, 150) = 50
AES encrypted GCD result: 0x2D35E8CF292107C9
Processing completed successfully!

>>> USER INPUT <<<
User entered values: 84, 126
Encrypted value1: 0xD99E074773C8FBF3
Encrypted value2: 0x9D38F45C072E1069
Data queued for processing

[STATUS] Queue items waiting: 2
>>> ALL TEST INPUTS COMPLETED <<<

=== SYSTEM PROCESSING CYCLE ===
Processing values: 17, 19

[STATUS] Queue items waiting: 1
Decrypted values: 17, 19
GCD(17, 19) = 1
AES encrypted GCD result: 0x1523A8FF47B1D916
Processing completed successfully!

[STATUS] Queue items waiting: 1

=== SYSTEM PROCESSING CYCLE ===
Processing values: 84, 126

[STATUS] Queue items waiting: 0
Decrypted values: 84, 126
GCD(84, 126) = 42
AES encrypted GCD result: 0xDBB365614C27ED6F
Processing completed successfully!

[STATUS] Queue items waiting: 0

=== SYSTEM PROCESSING CYCLE ===
No data in queue to process
All inputs processed. System processing will continue monitoring.
```

圖6.1.1 FreeRTOS結果輸出

上圖結果說明

顏色	說明	詳細
紅色	佇列等待的元素數量	即時顯示 uxQueueMessagesWaiting(xInputQueue)
黃色	使用者輸入階段	顯示 vUserInputTask 讀取並加密的原始值及密文
綠色	系統計算階段	顯示 vSystemProcessTask 解密後的原始值、GCD 計算過程及 AES 加密結果
藍色	處理完成後資訊	所有測試數據完成後佇列為空，顯示系統進入監控模式

6.1 系統流程概覽 與 同步機制

- **使用者輸入任務 (優先權 2):** 每 4 秒讀取一組測試數據，透過 DES IP 核心加密後，使用‘xQueueSend()‘ 放入佇列。
- **系統處理任務 (優先權 1):** 每 10 秒呼叫‘xQueueReceive()‘ 取出一筆加密數據。使用 DES IP 核心解密，計算 GCD，再透過 AES IP 核心對 GCD 結果加密，最後輸出。
- **狀態監控任務 (優先權 1):** 每 5 秒顯示佇列當前元素數量，使用‘xSemaphoreTake(xPrintMutex)‘ 保護印出操作。
- **佇列：**‘xInputQueue = xQueueCreate(2, sizeof(UserInput_t));‘ 容量 2。

6.2 關鍵程式片段

```c
// 佇列與互斥鎖
QueueHandle_t xInputQueue = xQueueCreate(2, sizeof(UserInput_t));
SemaphoreHandle_t xIPCoreMutex = xSemaphoreCreateMutex();
SemaphoreHandle_t xPrintMutex = xSemaphoreCreateMutex();

// 使用者輸入任務
void vUserInputTask(void *pv) {
    for (int i = 0; i < NUM_TESTS; i++) {
        UserInput_t data = tests[i];
        xSemaphoreTake(xIPCoreMutex, portMAX_DELAY);
        DES_Encrypt(&data);
        xSemaphoreGive(xIPCoreMutex);
        xQueueSend(xInputQueue, &data, portMAX_DELAY);
        vTaskDelay(pdMS_TO_TICKS(4000));
    }
    vTaskDelete(NULL);
}

// 系統處理任務
void vSystemProcessTask(void *pv) {
    UserInput_t data;
    while (1) {
        if (xQueueReceive(xInputQueue, &data, portMAX_DELAY)) {
            xSemaphoreTake(xIPCoreMutex, portMAX_DELAY);
            DES_Decrypt(&data);
            uint32_t gcd = ComputeGCD(data.value1, data.value2);
            AES_Encrypt(&gcd);
            xSemaphoreGive(xIPCoreMutex);
        }
        vTaskDelay(pdMS_TO_TICKS(10000));
    }
}

// 狀態監控任務
void vStatusTask(void *pv) {
    while (1) {
        UBaseType_t count = uxQueueMessagesWaiting(xInputQueue);
        xSemaphoreTake(xPrintMutex, portMAX_DELAY);
        printf("[STATUS] Queue items waiting: %u\n", count);
        xSemaphoreGive(xPrintMutex);
        vTaskDelay(pdMS_TO_TICKS(5000));
    }
}
```

圖6.2.1FreeRTOS關鍵程式

7. Linux Driver

7.1 ip Address

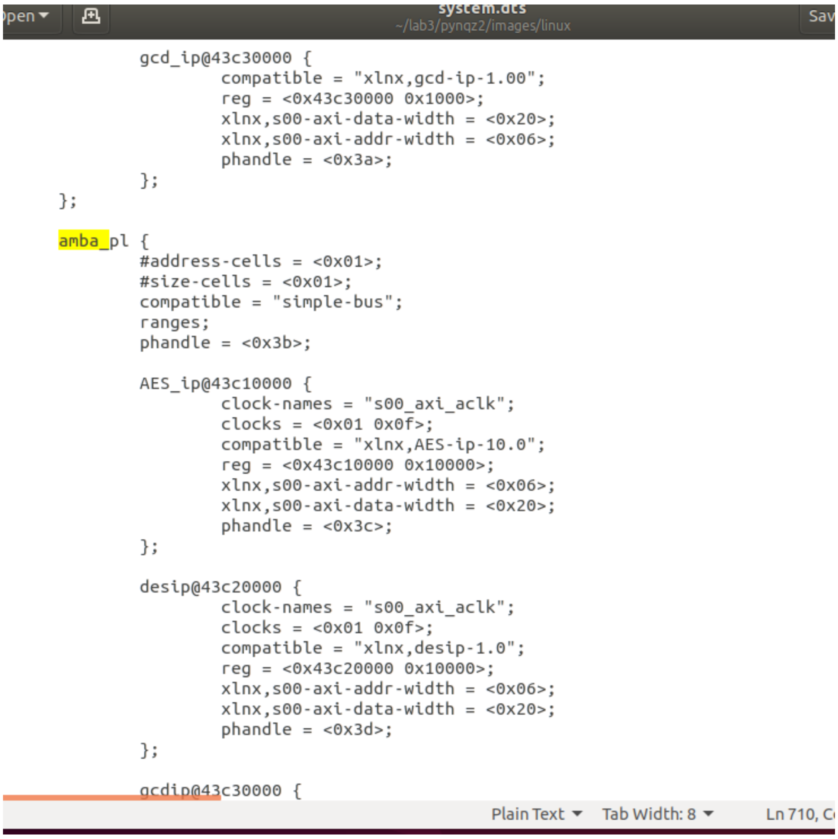
inter_ip_0	0x43c00000	0x43c0ffff	S00_AXI	register
ps7_scugic_0	0xf8f00100	0xf8f001ff	-	register
ps7_ethernet_0	0xe000b000	0xe000bfff	-	register
desip_0	0x43c20000	0x43c2ffff	S00_AXI	register

AES_ip_0	0x43c10000	0x43c1ffff	S00_AXI	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
gcdip_0	0x43c30000	0x43c3ffff	S00_AXI	register

7.2 ip table

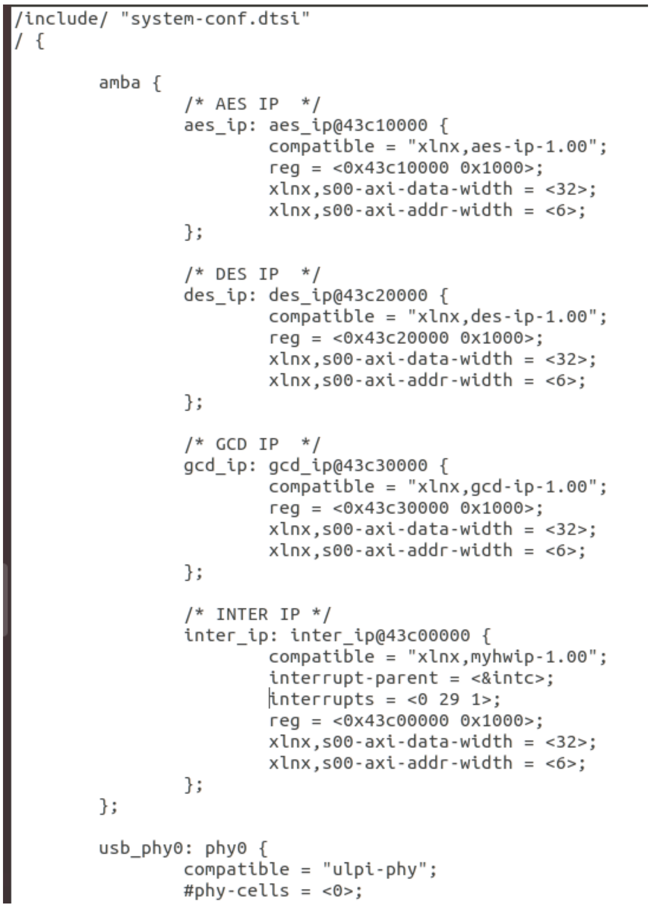
IP名稱	基地址	地址範圍	Device Tree節點
inter_ip_0	0x43c00000	0x43c00000-0x43c00fff	inter_ip@43c00000
AES_ip_0	0x43c10000	0x43c10000-0x43c1ffff	aes_ip@43c10000
desip_0	0x43c20000	0x43c20000-0x43c2ffff	des_ip@43c20000
gcdip_0	0x43c30000	0x43c30000-0x43c3ffff	gcd_ip@43c30000

7.3 device tree compiler




```
		gcd_ip@43c30000 {
			compatible = "xlnx,gcd-ip-1.00";
			reg = <0x43c30000 0x1000>;
			xlnx,s00-axi-data-width = <0x20>;
			xlnx,s00-axi-addr-width = <0x06>;
			phandle = <0x3a>;
		};
	};

	amba_pl {
		#address-cells = <0x01>;
		#size-cells = <0x01>;
		compatible = "simple-bus";
		ranges;
		phandle = <0x3b>;

		AES_ip@43c10000 {
			clock-names = "s00_axi_aclk";
			clocks = <0x01 0x0f>;
			compatible = "xlnx,AES-ip-10.0";
			reg = <0x43c10000 0x10000>;
			xlnx,s00-axi-addr-width = <0x06>;
			xlnx,s00-axi-data-width = <0x20>;
			phandle = <0x3c>;
		};

		desip@43c20000 {
			clock-names = "s00_axi_aclk";
			clocks = <0x01 0x0f>;
			compatible = "xlnx,desip-1.0";
			reg = <0x43c20000 0x10000>;
			xlnx,s00-axi-addr-width = <0x06>;
			xlnx,s00-axi-data-width = <0x20>;
			phandle = <0x3d>;
		};

		gcdip@43c30000 {
```

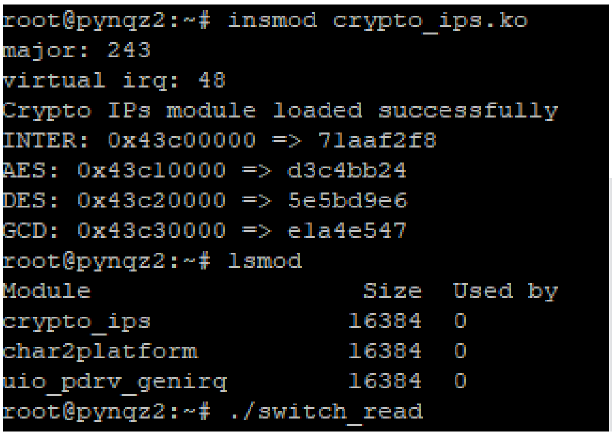
圖7.3.1 system.dts檢查模組匯入



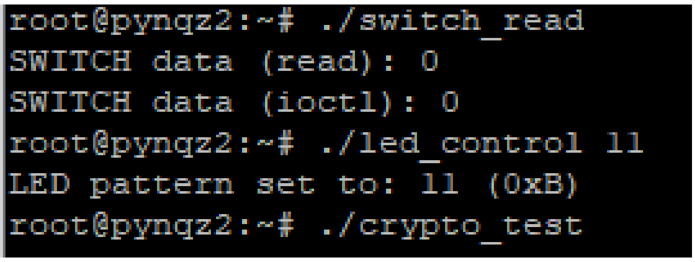
```
/include/ "system-conf.dtsi"
/ {

	amba {
		/* AES IP  */
		aes_ip: aes_ip@43c10000 {
			compatible = "xlnx,aes-ip-1.00";
			reg = <0x43c10000 0x1000>;
			xlnx,s00-axi-data-width = <32>;
			xlnx,s00-axi-addr-width = <6>;
		};

		/* DES IP  */
		des_ip: des_ip@43c20000 {
			compatible = "xlnx,des-ip-1.00";
			reg = <0x43c20000 0x1000>;
			xlnx,s00-axi-data-width = <32>;
			xlnx,s00-axi-addr-width = <6>;
		};

		/* GCD IP  */
		gcd_ip: gcd_ip@43c30000 {
			compatible = "xlnx,gcd-ip-1.00";
			reg = <0x43c30000 0x1000>;
			xlnx,s00-axi-data-width = <32>;
			xlnx,s00-axi-addr-width = <6>;
		};

		/* INTER IP */
		inter_ip: inter_ip@43c00000 {
			compatible = "xlnx,myhwip-1.00";
			interrupt-parent = <&intc>;
			interrupts = <0 29 1>;
			reg = <0x43c00000 0x1000>;
			xlnx,s00-axi-data-width = <32>;
			xlnx,s00-axi-addr-width = <6>;
		};
	};

	usb_phy0: phy0 {
		compatible = "ulpi-phy";
		#phy-cells = <0>;
```

圖7.3.2 system-conf.dtsi編譯前修改

7.4 komod test result

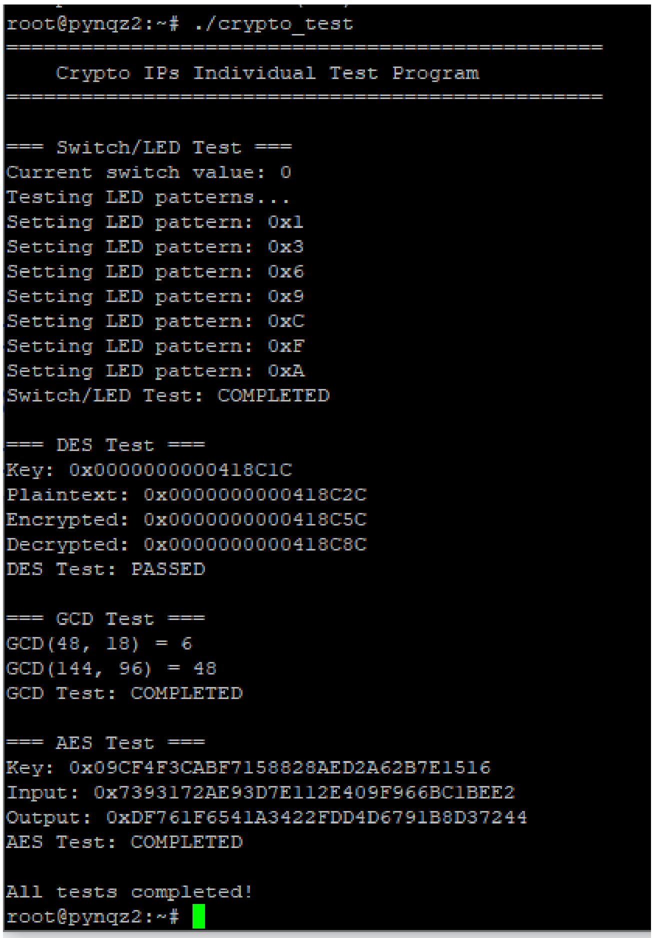


```
root@pynqz2:~# insmod crypto_ips.ko
major: 243
virtual irq: 48
Crypto IPs module loaded successfully
INTER: 0x43c00000 => 7laaf2f8
AES: 0x43c10000 => d3c4bb24
DES: 0x43c20000 => 5e5bd9e6
GCD: 0x43c30000 => ela4e547
root@pynqz2:~# lsmod
Module                  Size  Used by
crypto_ips             16384  0
char2platform          16384  0
uio_pdrv_genirq        16384  0
root@pynqz2:~# ./switch_read
```



```
root@pynqz2:~# ./switch_read
SWITCH data (read): 0
SWITCH data (ioctl): 0
root@pynqz2:~# ./led_control 11
LED pattern set to: 11 (0xB)
root@pynqz2:~# ./crypto_test
```

圖7.4.1 模組載入資訊、switch按鈕值、寫入led燈值



```
root@pynqz2:~# ./crypto_test
================================================
    Crypto IPs Individual Test Program
================================================

=== Switch/LED Test ===
Current switch value: 0
Testing LED patterns...
Setting LED pattern: 0x1
Setting LED pattern: 0x3
Setting LED pattern: 0x6
Setting LED pattern: 0x9
Setting LED pattern: 0xC
Setting LED pattern: 0xF
Setting LED pattern: 0xA
Switch/LED Test: COMPLETED

=== DES Test ===
Key: 0x0000000000418C1C
Plaintext: 0x0000000000418C2C
Encrypted: 0x0000000000418C5C
Decrypted: 0x0000000000418C8C
DES Test: PASSED

=== GCD Test ===
GCD(48, 18) = 6
GCD(144, 96) = 48
GCD Test: COMPLETED

=== AES Test ===
Key: 0x09CF4F3CABF7158828AED2A62B7E1516
Input: 0x7393172AE93D7E112E409F966BC1BEE2
Output: 0xDF761F6541A3422FDD4D6791B8D37244
AES Test: COMPLETED

All tests completed!
root@pynqz2:~#
```

圖7.4.2 ./crypto_test腳位以及功能測試

```
root@pynqz2:~# ./crypto_workflow

==================================================
    Interactive Cryptographic Workstation v1.0
    Supporting DES, GCD, AES with Interrupt Control
==================================================
Crypto device opened successfully

=== Stage 1: Mode Selection ===
Use 2 Switch combination to select operation mode:
SW1 SW0 = Mode
0   0   = Auto Mode (fully automatic execution)
0   1   = Manual Mode (manual confirmation for each step)
1   0   = Debug Mode (show detailed intermediate results)
1   1   = Simple Mode (minimal output for quick testing)
Press Enter to confirm selection

Current selection: SW1=0 SW0=0 = Mode 0 - Auto Mode
Press Enter to confirm...

Mode confirmed: 0

=== Stage 2: Test Case Selection ===
=== Value Input Instructions ===
Use 2 Switch combination to select test case:
SW1 SW0 = Test Case
0   0   = Case 0: Values 12, 8   (simple case)
0   1   = Case 1: Values 48, 18  (medium case)
1   0   = Case 2: Values 144, 96 (complex case)
1   1   = Case 3: Values 255, 85 (max complexity)

Ready for selection...
Current Switch: SW1=0 SW0=0 = Case 0 (Values: 12, 8)
Press Enter to confirm selection...
```

圖7.4.3 ./crypto_workflow(Enter下一個stage)

```
Ready for selection...
Current Switch: SW1=0 SW0=0 = Case 0 (Values: 12, 8)
Press Enter to confirm selection...

Test case confirmed: 0

Test case 0 selected: Value1=12, Value2=8

=== Starting Cryptographic Workflow ===
Processing values: 12 and 8

=== Stage 3: DES Encryption ===
DES encryption completed

=== Stage 4: DES Decryption Verification ===
DES verification SUCCESS: decrypted values 12, 8

=== Stage 5: GCD Calculation ===
GCD(12, 8) = 4

=== Stage 6: AES Encryption of GCD Result ===
AES Encrypted Result: 0xF860C04A7A34CA0450CC1BA8150DED96

=== Stage 7: Workflow Complete ===
All cryptographic operations completed!

Press Enter to restart...
Button interrupt triggered!
Button interrupt triggered!
Button interrupt triggered!
```

圖7.4.4 ./crypto_workflow(完成以及最後push button中斷)

8. Interrupt

- 流程簡介：用戶選擇模式（Switch Button） → 選擇測試資料（Push Button） → DES 加密 → DES 解密 → GCD 計算 → AES 加密
- 用戶模式選擇：

值（Switch Button）	模式	說明
00	Auto	每步驟自動進行
01	Manual	需按按鈕才能進入下一步
10	Debug	輸出完整細節資料
11	Simple	僅顯示主要結果

- 選擇測試資料：

值（Push Button）	測使案例	說明
00	case 0	12,8
01	case 1	48,18
10	case 2	144,96
11	case 3	255,85

- LED燈號顯示：

數值	狀態
0001	Idle
0011	Input
0110	DES 工作中
1001	GCD 計算中
1100	AES 加密中
1111	工作完成

```
==================================================
    Interactive Cryptographic Workstation v1.0
    Supporting DES, GCD, AES with Interrupt Control
==================================================
Interrupt system initialized successfully

=== Stage 1: Mode Selection ===
Use 2 Switch combination to select operation mode:
SW1 SW0 = Mode
0   0   = Auto Mode (fully automatic execution)
0   1   = Manual Mode (pushbutton confirmation for each step)
1   0   = Debug Mode (show detailed intermediate results)
1   1   = Simple Mode (minimal output for quick testing)
Press pushbutton to confirm selection

Current selection: SW1=0 SW0=0 = Mode 0 - Auto Mode
Mode confirmed: 0

=== Stage 2: Test Case Selection ===
=== Value Input Instructions ===
Use 2 Switch combination to select test case:
SW1 SW0 = Test Case
0   0   = Case 0: Values 12, 8   (simple case)
0   1   = Case 1: Values 48, 18  (medium case)
1   0   = Case 2: Values 144, 96 (complex case)
1   1   = Case 3: Values 255, 85 (max complexity)

Press pushbutton when ready
Current Switch: SW1=0 SW0=0 = Case 0 (Values: 12, 8)
Current Switch: SW1=0 SW0=1 = Case 1 (Values: 48, 18)
Current Switch: SW1=1 SW0=1 = Case 3 (Values: 255, 85)
Current Switch: SW1=1 SW0=0 = Case 2 (Values: 144, 96)
Test case confirmed: 2

Test case 2 selected: Value1=144, Value2=96

=== Starting Cryptographic Workflow ===
Processing values: 144 and 96

=== Stage 3: DES Encryption ===
DES encryption completed
Encrypted Value 1: 0x23FCA080910FF22F
Encrypted Value 2: 0x827A7B07E8B81FD0

=== Stage 4: DES Decryption Verification ===
DES verification SUCCESS: decrypted values 144, 96
Decrypted Value 1: 0x0000000000000090
Decrypted Value 2: 0x0000000000000060

=== Stage 5: GCD Calculation ===
GCD(144, 96) = 48

=== Stage 6: AES Encryption of GCD Result ===
AES Encrypted Result: 0x3594CC0A8596A295830CED352DFB062F

=== Stage 7: Workflow Complete ===
All cryptographic operations completed!

Press pushbutton to restart...

==================================================
Restarting system...
==================================================
```

```
0   0   = Auto Mode (fully automatic execution)
0   1   = Manual Mode (pushbutton confirmation for each step)
1   0   = Debug Mode (show detailed intermediate results)
1   1   = Simple Mode (minimal output for quick testing)
Press pushbutton to confirm selection

Current selection: SW1=1 SW0=0 = Mode 2 - Debug Mode
Mode confirmed: 2

=== Stage 2: Test Case Selection ===
=== Value Input Instructions ===
Use 2 Switch combination to select test case:
SW1 SW0 = Test Case
0   0   = Case 0: Values 12, 8   (simple case)
0   1   = Case 1: Values 48, 18  (medium case)
1   0   = Case 2: Values 144, 96 (complex case)
1   1   = Case 3: Values 255, 85 (max complexity)

Press pushbutton when ready
Current Switch: SW1=1 SW0=0 = Case 2 (Values: 144, 96)
Button interrupt triggered!
Button interrupt triggered!
Button interrupt triggered!
Test case confirmed: 2

Test case 2 selected: Value1=144, Value2=96

=== Starting Cryptographic Workflow ===
Processing values: 144 and 96

=== Stage 3: DES Encryption ===
DES Key: 0x133457799BBCDFF1
Value1 encrypted: 0x23FCA080910FF22F
Value2 encrypted: 0x827A7B07E8B81FD0
Encrypted Value 1: 0x23FCA080910FF22F
Encrypted Value 2: 0x827A7B07E8B81FD0
Press pushbutton to continue...
Button interrupt triggered!

=== Stage 4: DES Decryption Verification ===
DES verification SUCCESS: decrypted values 144, 96
Decrypted Value 1: 0x0000000000000090
Decrypted Value 2: 0x0000000000000060
Press pushbutton to continue...
Button interrupt triggered!

=== Stage 5: GCD Calculation ===
Calculating GCD(144, 96) using GCD IP...
GCD calculation completed in 1 ms
GCD(144, 96) = 48
Press pushbutton to continue...
Button interrupt triggered!
Button interrupt triggered!
Button interrupt triggered!

=== Stage 6: AES Encryption of GCD Result ===
AES Key: 0xABF7158809CF4F3C2B7E151628AED2A6
Input Data: 0x00000000000000000000000000000030
AES encryption completed in 1 ms
AES Encrypted Result: 0x3594CC0A8596A295830CED352DFB062F

=== Stage 7: Workflow Complete ===
All cryptographic operations completed!
```

```
==================================================
Restarting system...
==================================================

=== Stage 1: Mode Selection ===
Use 2 Switch combination to select operation mode:
SW1 SW0 = Mode
0   0   = Auto Mode (fully automatic execution)
0   1   = Manual Mode (pushbutton confirmation for each step)
1   0   = Debug Mode (show detailed intermediate results)
1   1   = Simple Mode (minimal output for quick testing)
Press pushbutton to confirm selection

Current selection: SW1=1 SW0=1 = Mode 3 - Simple Mode
Current selection: SW1=0 SW0=1 = Mode 1 - Manual Mode
Mode confirmed: 1

=== Stage 2: Test Case Selection ===
=== Value Input Instructions ===
Use 2 Switch combination to select test case:
SW1 SW0 = Test Case
0   0   = Case 0: Values 12, 8   (simple case)
0   1   = Case 1: Values 48, 18  (medium case)
1   0   = Case 2: Values 144, 96 (complex case)
1   1   = Case 3: Values 255, 85 (max complexity)

Press pushbutton when ready
Current Switch: SW1=0 SW0=1 = Case 1 (Values: 48, 18)
Test case confirmed: 1

Test case 1 selected: Value1=48, Value2=18

=== Starting Cryptographic Workflow ===
Processing values: 48 and 18

=== Stage 3: DES Encryption ===
DES encryption completed
Encrypted Value 1: 0x7B7EA0385014FB43
Encrypted Value 2: 0x81447DF6D287CF93
Press pushbutton to continue...

=== Stage 4: DES Decryption Verification ===
DES verification SUCCESS: decrypted values 48, 18
Decrypted Value 1: 0x0000000000000030
Decrypted Value 2: 0x0000000000000012
Press pushbutton to continue...

=== Stage 5: GCD Calculation ===
GCD(48, 18) = 6
Press pushbutton to continue...

=== Stage 6: AES Encryption of GCD Result ===
AES Encrypted Result: 0xE4FB886BC8F045C37D72F5E55BAF4B9C

=== Stage 7: Workflow Complete ===
All cryptographic operations completed!

Press pushbutton to restart...
Press pushbutton to continue...

==================================================
Restarting system...
==================================================
```

```
==================================================
Restarting system...
==================================================

=== Stage 1: Mode Selection ===
Use 2 Switch combination to select operation mode:
SW1 SW0 = Mode
0   0   = Auto Mode (fully automatic execution)
0   1   = Manual Mode (pushbutton confirmation for each step)
1   0   = Debug Mode (show detailed intermediate results)
1   1   = Simple Mode (minimal output for quick testing)
Press pushbutton to confirm selection

Current selection: SW1=1 SW0=0 = Mode 2 - Debug Mode
Current selection: SW1=1 SW0=1 = Mode 3 - Simple Mode
Button interrupt triggered!
Mode confirmed: 3

=== Stage 2: Test Case Selection ===
Test case confirmed: 3

Test case 3 selected: Value1=255, Value2=85

=== Starting Cryptographic Workflow ===
Processing values: 255 and 85

Encrypted Value 1: 0xCF6542BFA603543A
Encrypted Value 2: 0xD793A889F65FE940
DES verification SUCCESS: decrypted values 255, 85
Decrypted Value 1: 0x00000000000000FF
Decrypted Value 2: 0x0000000000000055
GCD(255, 85) = 85
AES Encrypted Result: 0x729AF23F554D12CF3E3033B5457A7005

=== Stage 7: Workflow Complete ===
All cryptographic operations completed!

Press pushbutton to restart...

==================================================
Restarting system...
==================================================
```

圖8.1.1 Interrupt輸出結果

分工表

IP	
AES ip	林庭毅
GCD ip	陳冠維
DES ip	陳冠維、羅豐祥

IP 結合功能	
電路整合	林庭毅
Standalone	陳冠維、陳祥鈞
FreeRTOS	陳冠維、陳祥鈞
Linux_Driver	林庭毅
Interrupt	林庭毅

文書處理	
排版	陳祥鈞
簡報	羅豐祥