Design of Chip Security - Spring 2024

Lab01 Report

Name: 王品然

Student ID: 113063572

A. Design architecture and throughput

1. Algorithm

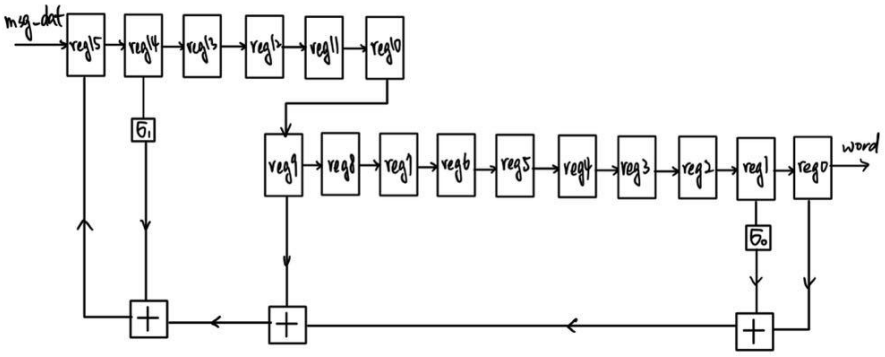
The “Message Schedule” algorithm in SHA2:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

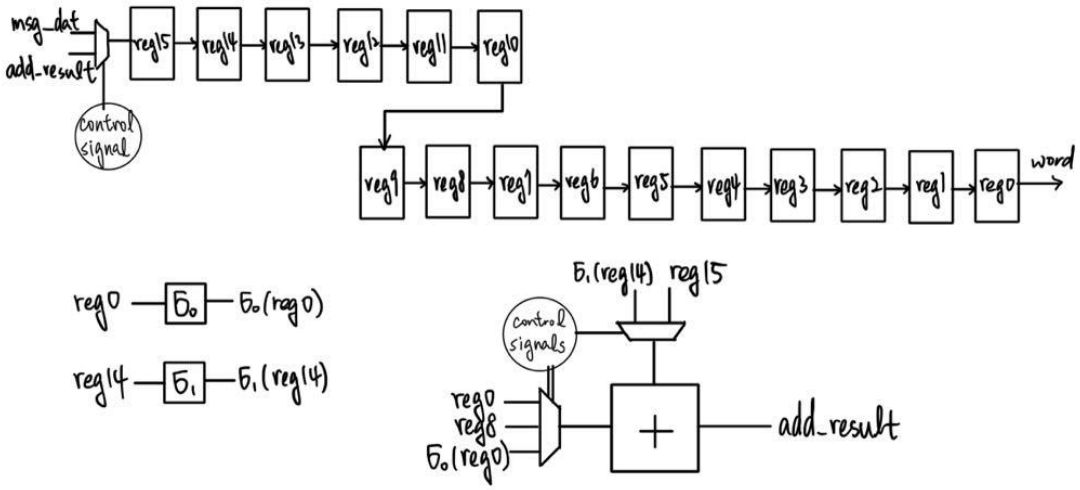
$$\begin{aligned} s_0^{\{256\}}(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \\ s_1^{\{256\}}(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \end{aligned}$$

(note that “$s_0^{\{256\}}$” stands for “$\sigma_0^{\{256\}}$” and “$s_1^{\{256\}}$” stands for “$\sigma_1^{\{256\}}$”)

下圖的 block diagram 可以描述此演算法：



填入一個 message block 之後，即可一直從 reg[0]得到 message schedule 的結果W_t，結構簡單易懂且能夠一個 clock cycle 就能算完一個 output 值，但需要用到三個 32-bit 加法器，較佔面積，不符合這次想要小面積的要求，因此想辦法在 register 數量不變的情況下，只用一個 32-bit 加法器，如下：



這樣簡化的 block diagram 以如下的步驟運算

(1) 填入 16 個 words(一個 message block M^i)到 reg[0]~reg[15]，使得此時 reg[n]暫存M_n^i，也就是W_n。

(2) 將 reg[0]與$\sigma_1(reg[14])$相加(也就是$W_0 + \sigma_1(W_{14})$)並存入 reg[15]，同時 shift reg[n+1] 到 reg[n]，並且 output reg[0](也就是W_0)。

(3) 將 reg[15](暫存著$W_0 + \sigma_1(W_{14})$的結果)與 reg[8](此時存著W_9)相加並存入 reg[15]。

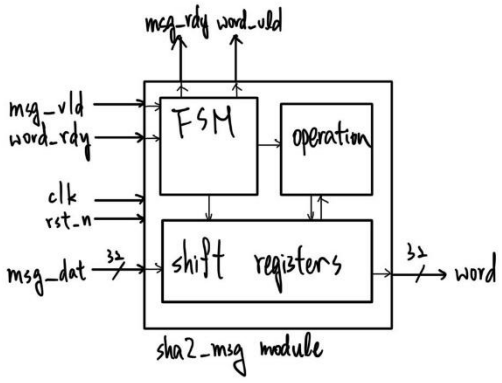
(4) 將 reg[15](暫存著$W_0 + \sigma_1(W_{14}) + W_9$)與$\sigma_0(reg[0])$(此時 reg[0]存著$W_1$)相加得到$W_0 + \sigma_1(W_{14}) + W_9 + \sigma_0(W_1)$也就是演算法的$W_{16}$，並將此存入 reg[15]。

(5) 繼續重複(2)到(4)62 次，再回到(2)完成第 64 次 output。

這樣便完成一個 massage block 的 expanding。這樣的結構下的 throughput 會是 1word per 3 cycle。接下來要加上 flow control 後，即可完成此演算法的電路實現。

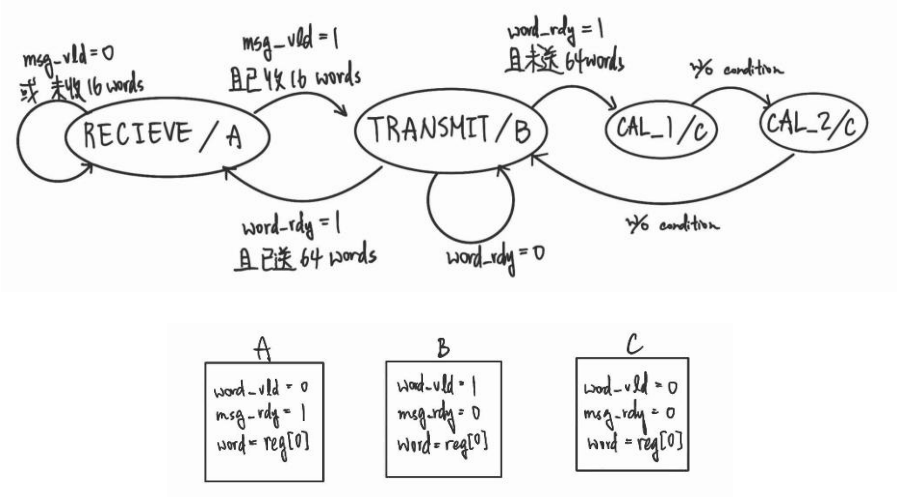
2. sha2_msg module structure

以上述的 block diagram 可以初步建構出 sha2_msg 的架構：



以 FSM 作為 control unit，負責 I/O protocol 和運算的 flow control。

3. FSM



以下解釋各個 state 時做的事(state transition 還有 output 如上圖所示)

RECEIVE: 使 msg_rdy=1，在 msg_vld=1 時，使 reg[15]接收 msg_dat，同時 shift register，並讓計算輸入了幾個 word 的 counter+1。

TRANSMIT: 使 word_vld=1，在 word_rdy=1 時，shift register，並讓計算輸出了幾個 word 的 counter+1，同時運算 reg[0]+$\sigma_1(reg[14])$，並存入 reg[15]。

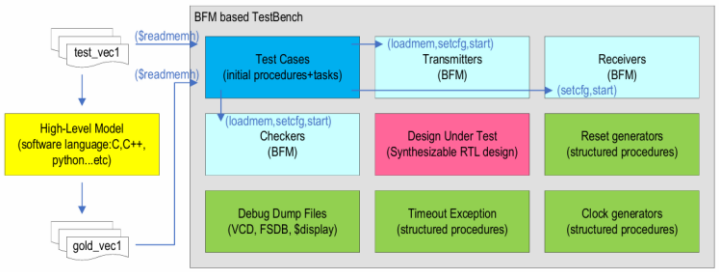
CAL_1: 運算 reg[15]+ reg[8]，並存入 reg[15]。

CAL_2: 運算 reg[15]+ $\sigma_0(reg[0])$，並存入 reg[15]。

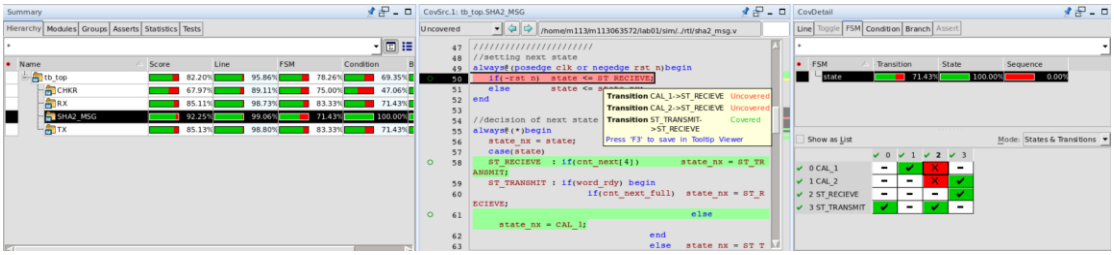
值得注意的一點是，因為這個架構下，事先 input 完一個 block 才要開始 output，因此計算已 input 數量與計算已 output 數量的情況之時間不重疊，所以為了節省面積，可以使兩個功能共用一個 6-bit counter，因此需要在 RECEIVE state 多做一件事，在 counter_next 等於 16 時，將 counter 清零，才能交給 output 端記數。

B. Testbench

這次的 testbench 大多參考 lecture 5 的講義以及所附的例子，使用的是 BFM 的模式，架構如講義中的圖：



High level model 的部分是使用 python 寫“Message Schedule” algorithm 並且使其隨機生成指定長度的 test vector 與其對應之 golden vector，輸出兩份.dat 檔，在 test case 中讀取。驗證的情況包含 msg_vld 和 word_rdy 可能連續或不等間隔拉高，code coverage 的部分，FSM transition 缺了兩個，如下圖：



但其實 uncovered 的狀況不會發生，因為如 lab01 作業內容“rst_n will be active at the beginning of the test.”所說，reset 是在 DUT 還沒接收 data 時就來了，因此其實不會有 CAL_1 或 CAL_2 轉換到 RECEIVE 的狀況發生，除非 reset 在 CAL_1 或 CAL_2 時才來來。至於 line 和 branch 的 coverage 情況則是少了下圖中的那行 default，因此也不重要。