

Iterated addition

Oprîtoiu Flavius
flavius.opritoiu@cs.upt.ro

September 18, 2023

Introduction

Objectives:

- ▶ Construct structures for multi-operand addition

Iterated algorithms' hardware implementation implies two distinct phases:

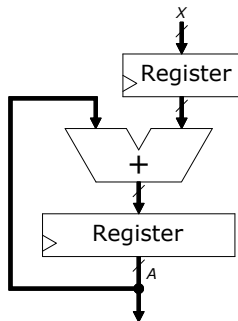
- constructing the state-related part, which stores the algorithm's state from the current to the next iteration
- implement the data processing part, which updates the algorithm's state from one iteration to the other

Sequential multi-operand addition

Each clock cycle a new operand is delivered on input register, X .
The following algorithm calculates the result of a multi-operand addition and stores it in accumulator A :

```
1:  $A \leftarrow 0$   
2: loop  
3:    $A \leftarrow A + X$   
4: end loop
```

having the following hardware implementation:



Solved problem

Datapath of a cryptographic application

Exercise: Construct the datapath of an architecture for the 256-bit Secure Hash Algorithm 2 (SHA-2) algorithm (see [FIPS15], section 5.1.1).

Solution: The unit receives at input 512-bit blocks which it process sequentially in order to determine the hash associated with the received message. The hash result is delivered at the output as a 256-bit binary vector.

The block processing implies the following operations:

- ▶ *Message schedule:* extends the 16 words of a received block to 64 words
- ▶ *Compression function:* takes one word from the *message schedule* and updates, in 64 iteration, variables a , b , c , d , e , f , g and h
- ▶ *Hash update:* adds to the current hash value, split in 8 words, the values of the a to h variables

Solved problem (contd.)

Message schedule

The 512-bit block is split in 16 32-bit words: M_0, M_1, \dots, M_{15} , with M_0 representing the most significant 32-bit of the block and, M_{15} representing the least significant 32-bits.

For 64 iterations, in each clock cycle, the *message schedule* constructs a new word in the least significant position (the first constructed word comes after M_{15} , and this first generated word will be followed by the next one etc.). In each iteration, the most significant word, M_0 , is delivered at the output.

At any given moment only 16 words are required for construction of the next word. As a consequence, the new word will occupy the least significant position (M_{15}) all the other more significant words being shifted on the immediate, more significant, position (M_{15} will move to M_{14} , M_{14} to M_{13} , ..., M_1 to M_0).

Solved problem (contd.)

Message schedule

Message schedule is formally described by the algorithm below:

Input: Block BLK ▷ BLK is split into 16 32-bit words
Output: Word M_0 on 32-bit ▷ Delivers M_0 in each iteration

```
1: procedure MESSAGE_SCHEDULE( $BLK$ )  
2:    $M_0 \leftarrow BLK[511 : 480]$  ▷ Initialize the 16 words,  $M_i$   
3:    $M_1 \leftarrow BLK[479 : 448]$   
4:   ...  
5:    $M_{14} \leftarrow BLK[63 : 32]$   
6:    $M_{15} \leftarrow BLK[31 : 0]$   
7:   for  $i = 0$  to 63 do ▷ Construct a new word and update the 16 stored words  
8:      $NEW\_WORD \leftarrow \sigma_1(M_{14}) + M_9 + \sigma_0(M_1) + M_0$   
9:      $M_0 \leftarrow M_1$   
10:     $M_1 \leftarrow M_2$   
11:    ...  
12:     $M_{14} \leftarrow M_{15}$   
13:     $M_{15} \leftarrow NEW\_WORD$   
14:   end for  
15: end procedure
```

The addition operator, $+$, in this slide and the next to come is performed (mod 2^{32})

Solved problem (contd.)

Message schedule

Functions $\sigma_0(\alpha)$ and $\sigma_1(\beta)$ are defined bellow:

$$\sigma_0(\alpha) = RtRotate(\alpha, 7) \oplus RtRotate(\alpha, 18) \oplus RtShift(\alpha, 3)$$

$$\sigma_1(\beta) = RtRotate(\beta, 17) \oplus RtRotate(\beta, 19) \oplus RtShift(\beta, 10)$$

where: $RtRotate(x, p)$ rotates word x to the right by p positions;
 $RtShift(x, p)$ shifts word x to the right by p bits (adding 0s to msb) and \oplus denotes the EXCLUSIV-OR operator

For implementing these operators, one can use Verilog functions:

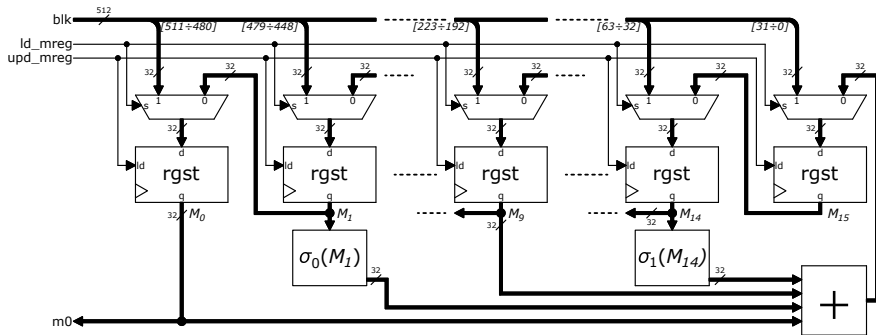
```
1 function [31:0] RtRotate (input [31:0] x, input [4:0] p);
2     reg [63:0] tmp;
3     begin
4         tmp = {x, x} >> p;
5         RtRotate = tmp[31:0];
6     end
7 endfunction
```

This function is called by: $RtRotate(\alpha, 7)$

Solved problem (contd.)

Message schedule

The datapath component that implements the *message schedule* is depicted in the figure below:



Note: module *rgst* is available [▶ here](#)

Solved problem (contd.)

Compression function and hash update

The hash result, on 256-bit, is formed of 8 32-bit words: H_0 , H_1 , H_2 , H_3 , H_4 , H_5 , H_6 and H_7 , with H_0 being the most significant and H_7 the least significant.

Compression function uses 8 32-bit variables: a , b , c , d , e , f , g and h . The 8 variables are initialized to the values of words H_0 , ..., H_7 of the current hash result. Afterwards, along 64 iterations, variables a to h are updated based on their current value, the value of word M_0 delivered by the *message schedule* and the value of a round constant, $K(i)$.

At the end of the 64 iterations, the hash result is *updated* by adding to each of the 8 words H_0 to H_7 the values of the variables a to h .

The compression function followed by the hash update is performed for each new received block.

Solved problem (contd.)

Compression function and hash update

Input: Message blocks are received

Output: Hash result words $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$

1: **procedure** SHA256

2: InitializeHashResultWords()

3: **do**

4: $a \leftarrow H_0$

5: $b \leftarrow H_1$

6: ...

7: $h \leftarrow H_7$

8: **for** $i = 0$ **to** 63 **do**

9: $T_1 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K(i) + M_0$ ▷ M_0 from expression

10: $T_2 \leftarrow \Sigma_0(a) + Maj(a, b, c)$ ▷ of T_1 is delivered by the

11: $h \leftarrow g; g \leftarrow f; f \leftarrow e$ ▷ *message scheduler* which,

12: $e \leftarrow d + T_1$ ▷ since performing the same

13: $d \leftarrow c; c \leftarrow b; b \leftarrow a$ ▷ number of iterations (64), can

14: $a \leftarrow T_1 + T_2$ ▷ operate in parallel with this loop

15: **end for**

16: $H_0 \leftarrow H_0 + a$

17: $H_1 \leftarrow H_1 + b$

18: ...

19: $H_7 \leftarrow H_7 + h$

20: **while not** *last block*

21: **end procedure**

Solved problem (contd.)

Compression function and hash update

The SHA-256 algorithm uses the following functions:

$$\Sigma_0(x) = \text{RtRotate}(x, 2) \oplus \text{RtRotate}(x, 13) \oplus \text{RtRotate}(x, 22)$$

$$\Sigma_1(x) = \text{RtRotate}(x, 6) \oplus \text{RtRotate}(x, 11) \oplus \text{RtRotate}(x, 25)$$

$$\text{Ch}(x, y, z) = (x \text{ and } y) \quad \oplus \quad ((\text{not } x) \text{ and } z)$$

$$\text{Maj}(x, y, z) = (x \text{ and } y) \quad \oplus \quad (x \text{ and } z) \quad \oplus \quad (y \text{ and } z)$$

The above and and not operators are bit-wise (operate on vectors, at the individual bit level). Constants $K(i)$, indexed by current iteration, i , are specified by the standard ([FIPS15], section 4.2.2):

i	$K(i)$
0	32'h428a2f98
1	32'h71374491
2	32'hb5c0fbcf
...
63	32'hc67178f2

Solved problem (contd.)

Compression function and hash update

The 8 words of the hash result, $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$, are initialized to values specified by the standard ([FIPS15], section 5.3.3):

Output: Initialize the hash result words $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$

1: **procedure** INITIALIZEHASHRESULTWORDS

2: $H_0 \leftarrow 32'h6a09e667$

3: $H_1 \leftarrow 32'hbb67ae85$

4: $H_2 \leftarrow 32'h3c6ef372$

5: $H_3 \leftarrow 32'ha54ff53a$

6: $H_4 \leftarrow 32'h510e527f$

7: $H_5 \leftarrow 32'h9b05688c$

8: $H_6 \leftarrow 32'h1f83d9ab$

9: $H_7 \leftarrow 32'h5be0cd19$

10: **end procedure**

References

[FIPS15] National Institute of Standards and Technology, “FIPS PUB 180-4: Secure Hash Standard,” Gaithersburg, MD 20899-8900, USA, Tech. Rep., Aug. 2015. [Online]. Available: <http://dx.doi.org/10.6028/NIST.FIPS.180-4> (Last accessed 06/04/2016).