Alpha Technology

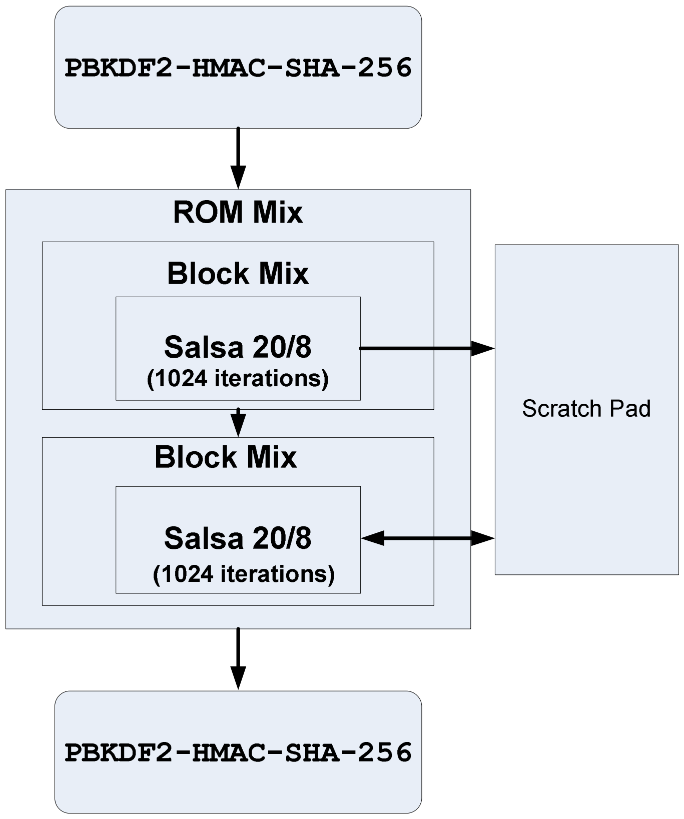
Implementation and Analysis of Scrypt Algorithm in FPGA (Proof of concept)

www.alpha-t.net
info@alpha-t.net

Alpha Technology (INT) LTD
Manchester, England

Call Flow

Scrypt algorithm, consists of 2048 iterations of the blockmix function. Each iteration of blockmix transforms 128 bytes of input into 128 bytes of output through the use of Salsa20/8 function. Salsa20/8 transforms 64 bytes input into 64 bytes output, so it must be run twice for each blockmix. The blockmix function also performs an xor operation and rearranges the bytes to get its output.



Scrypt has two parts. The first part, consisting of 1024 iterations of blockmix, produces 128kB (one megabit) of data, starting from a seed value. Each blockmix produces 128 bytes, and the result is written to memory and also used as the input to the next iteration of the blockmix function. The result therefore is 1024x128 bytes, 128kB. The second set of 1024 iterations is used to reprocess some of the data. The second set of iterations occur in an unpredictable order: the next block to be processed is determined from the result of processing the current block. The reprocessed block then overwrites its previous location.

Sandwiched around all this are two iterations of SHA-256, one used to generate the initial seed, the other used to generate a final hash value from the final state of the 128kB of working data.

Alpha Technology

The Salsa20/8 core Function

```
#define R(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
void salsa208_word_specification(uint32 out[16],uint32 in[16])
{
int i;
uint32 x[16];

for (i = 0;i < 16;++i) x[i] = in[i]; // one cycle

for (i = 8;i > 0;i -= 2) {

//upper half//
// 4 cycles
x[ 4] ^= R(x[ 0]+x[12], 7); x[ 8] ^= R(x[ 4]+x[ 0], 9);
x[12] ^= R(x[ 8]+x[ 4],13); x[ 0] ^= R(x[12]+x[ 8],18);

// 4 cycles
x[ 9] ^= R(x[ 5]+x[ 1], 7); x[13] ^= R(x[ 9]+x[ 5], 9);
x[ 1] ^= R(x[13]+x[ 9],13); x[ 5] ^= R(x[ 1]+x[13],18);

// 4 cycles
x[14] ^= R(x[10]+x[ 6], 7); x[ 2] ^= R(x[14]+x[10], 9);
x[ 6] ^= R(x[ 2]+x[14],13); x[10] ^= R(x[ 6]+x[ 2],18);

// 4 cycles
x[ 3] ^= R(x[15]+x[11], 7); x[ 7] ^= R(x[ 3]+x[15], 9);
x[11] ^= R(x[ 7]+x[ 3],13); x[15] ^= R(x[11]+x[ 7],18);

//lower half//
// 4 cycles
x[ 1] ^= R(x[ 0]+x[ 3], 7); x[ 2] ^= R(x[ 1]+x[ 0], 9);
x[ 3] ^= R(x[ 2]+x[ 1],13); x[ 0] ^= R(x[ 3]+x[ 2],18);

// 4 cycles
x[ 6] ^= R(x[ 5]+x[ 4], 7); x[ 7] ^= R(x[ 6]+x[ 5], 9);
x[ 4] ^= R(x[ 7]+x[ 6],13); x[ 5] ^= R(x[ 4]+x[ 7],18);

// 4 cycles
x[11] ^= R(x[10]+x[ 9], 7); x[ 8] ^= R(x[11]+x[10], 9);
x[ 9] ^= R(x[ 8]+x[11],13); x[10] ^= R(x[ 9]+x[ 8],18);

// 4 cycles
x[12] ^= R(x[15]+x[14], 7); x[13] ^= R(x[12]+x[15], 9);
x[14] ^= R(x[13]+x[12],13); x[15] ^= R(x[14]+x[13],18);

}

for (i = 0;i < 16;++i) out[i] = x[i] + in[i];//one cycle
}
```

One salsa consists of 34 cycles

- One cycle to read
- Inside for loop can be divided into two lower half can operate only when upper half operation is finished. Within upper half we have four sections which can function parallel and each section takes 4 cycles. So for one iteration of inner loop it takes 8 cycles (4 for upper and 4 for lower section each). Loop will have 4 iterations (4 x 8 = 32 cycles)
- One cycle for last loop.

Block Mix Algorithm

Parameters:
r :Block size parameter

Input
B[0], ..., B[2 * r - 1] :Input vector of 2 * r 64-octet blocks.

Output:
B'[0], ..., B'[2 * r - 1] :Output vector of 2 * r 64-octet blocks.

Steps:

```
1. X = B[2 * r - 1]

2. for i = 0 to 2 * r - 1 do
   T = X xor B[i]
   X = Salsa (T)
   Y[i] = X
   end for

3. B' = (Y[0], Y[2], ..., Y[2 * r - 2],
         Y[1], Y[3], ..., Y[2 * r - 1])
```

ROM Mix Algorithm

Input:

r Block size parameter.
B Input octet vector of length 128 * r octets.
N CPU/Memory cost parameter, must be larger than 1,a power of 2 and less than 2^(128 * r / 8).

Output:

B' Output octet vector of length 128 * r octets.

Steps:

```
1.  X = B

2.  for i = 0 to N - 1 do
    V[i] = X
    X = Block Mix (X)
    end for

3.  for i = 0 to N - 1 do
     j = Integerify (X) mod N where Integerify (B[0] ... B[2 * r - 1]) is defined as the
                          result of interpreting B[2 * r - 1] as a little-endian integer.
     T = X xor V[j]
     X = =Block Mix (T)
    end for

4. B' = X
```

The Scrypt Algorithm

Input:

P : Passphrase, an octet string.
S : Salt, an octet string.
r : Block size parameter.
N : CPU/Memory cost parameter, must be larger than 1, a power of 2 and less than 2^(128 * r / 8).
P : Parallelization parameter, a positive integer less than or equal to ((2^32-1) * hLen) / MFLen where hLen is 32 and MFlen is 128 * r. dkLen : Intended output length in octets of the derived key; a positive integer less than or equal to (2^32 - 1) * hLen where hLen is 32.

Output:

DK : Derived key, of length dkLen octets.

Steps:

```
1. B[0] || B[1] || ... || B[p - 1] = PBKDF2-HMAC-SHA256 (P, S, 1, p * 128 * r)

2. for i = 0 to p - 1 do
    B[i] = ROM Mix (r, B[i], N)
   end for

3. DK = PBKDF2-HMAC-SHA256 (P, B[0] || B[1] || ... || B[p - 1],1, dkLen)
```

Performance Analysis (Theoretical – Based on Optimized algorithm)

Loop1: 1024 *(2*salsa time)

Salsa is called two times during first 1024 iterations of loop1
1024 *(2*salsa time) = 1024 *(2*34) = **69632 Cycles**

Loop2 :1024*(fetching latency from scratch pad+ (2* salsa time))

Salsa is called two times during first 1024 iterations of loop 2
Considering a 256 bus width for RAM fetching latency (minimum) = 2 cycles
1024*(fetching latency from scratch pad+ (2* salsa time)) = 1024*(2+2*34)
= **71680 Cycles**

Total cycles for ROM Mix = 69632 Cycles + 73728 Cycles = 141312 Cycles

If one cycle = 10ns (100 MHz)

Time for 1 ROM Mix = 1413.2 us
ie, without the over head of PBKDF2-HMAC-SHA-256 theoretical Hash rate will be **700 Hash/s.**

One Cycle	Frequency	Hash rate
10ns	100 MHz	700 Hash/s
5ns	200MHz	1.4 KHash/s
3.3ns	300MHz	2.15 KHash/s
2.5ns	400 MHz	2.8 KHash/s

Implementation Result

- **Target device - xc7z020-2clg484 (on ZedBoard Zynq Development and Evaluation Kit)**

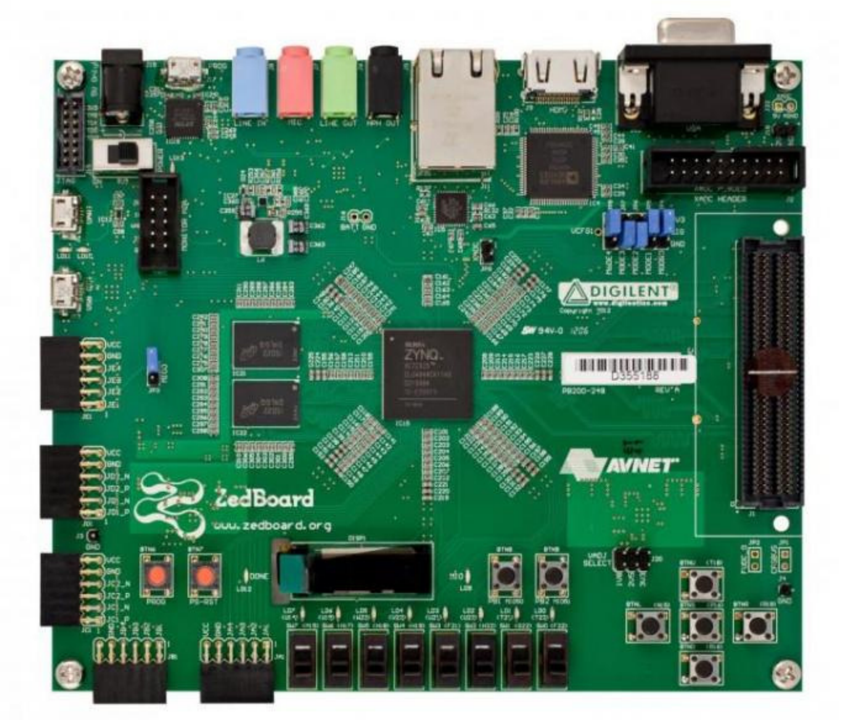


Figure : ZedBoard Zynq Development and Evaluation Kit

Resource utilization summary

Number of BSCANs	1 out of 4 25%
Number of BUFGs	3 out of 32 9%
Number of External IOB33s	6 out of 200 3%
Number of LOCed IOB33s	6 out of 6 100%
Number of MMCME2_ADVs	1 out of 4 25%
Number of RAMB18E1s	21 out of 280 7%
Number of RAMB36E1s	55 out of 140 39%
Number of Slices	7156 out of 13300 53%
Number of Slice Registers	18042 out of 106400 16%
Number used as Flip Flops	18041
Number used as Latches	1
Number used as LatchThrus	0
Number of Slice LUTS	20271 out of 53200 38%
Number of Slice LUT-Flip Flop pairs	23480 out of 53200 44%

Performance analysis:

Clock used by algorithm	: 120 MHz (8.33 ns)
No of clock pulses from start to done Used by algorithm	: 562870
Latency	: **4.69 ms**

A 32 bit counter was implemented in the design to count the number of clock pulses used by algorithm to produce one complete Hash.

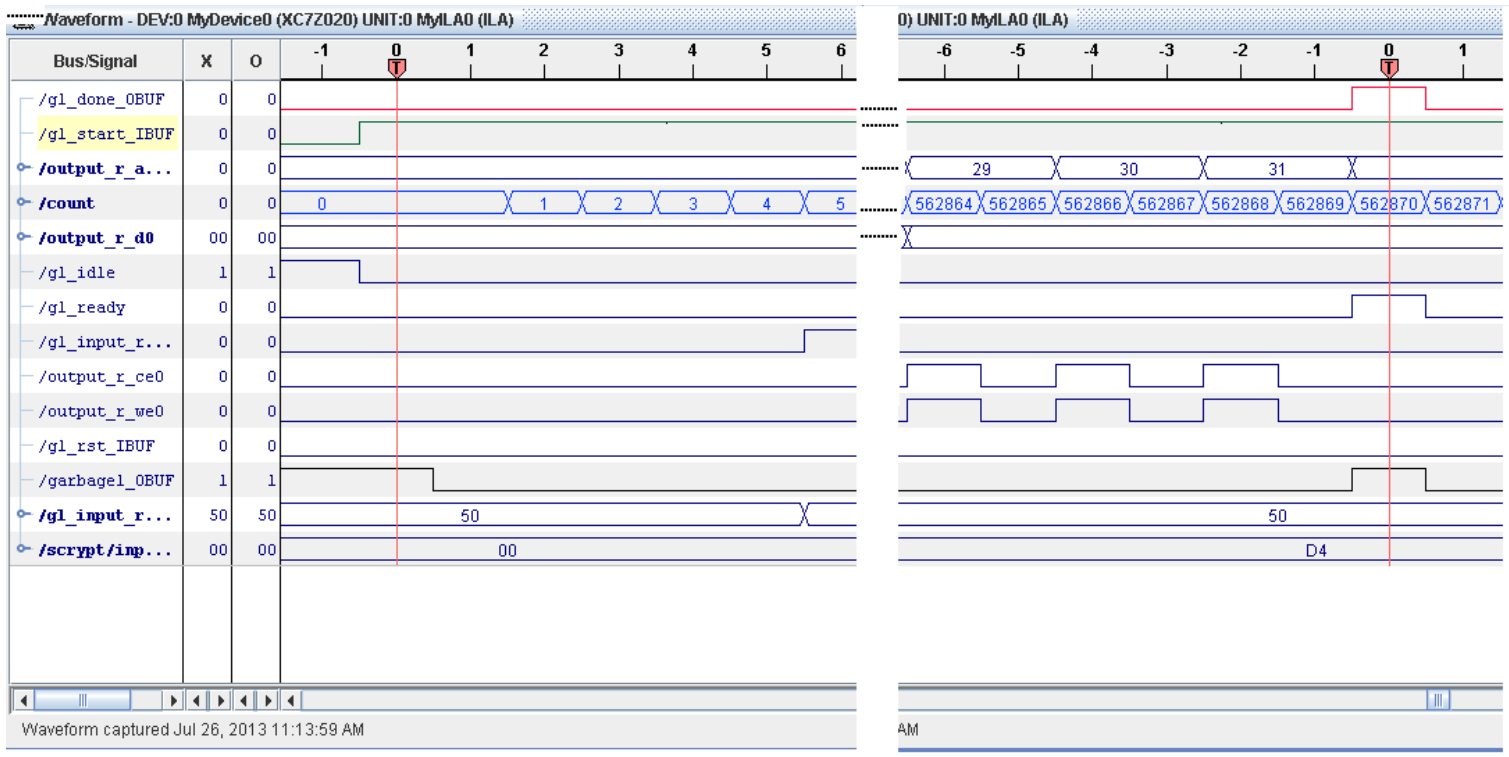


Figure: Chip Scope Snap short (start and done combined)