Proposta de implementação do algoritmo SHA-256 em Hardware

Carlos E. B. Santos Júnior [1], Marcelo A. C. Fernandes[1]

[1]Departamento de Engenharia de Computação e Automação - DCA
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524 – 59.078-970 – Natal – RN – Brazil

ceduardobsantos@gmail.com, mfernandes@dca.ufrn.br

Abstract. *This paper proposes an implementation for SHA-256 hash algorithm on programmable field gate array (FPGA). The implementation here offers a differentiated approach about other already published papers, with clustered cores executing the SHA-256 algorithm in parallel. The algorithm is used in the date transport of insurance or verifies of the solution, such as HMAC, TLS, and IPSec. Achieving a throughput of about* 101 *Gbps to* 128 *cores in a single FPGA. There is a great possibility of applications for such implementation, which is used as target the FPGA Xilinx Virtex 6 xc6vlx240t-1ff1156.*

Resumo. *Este trabalho propõe uma arquitetura desenvolvida em Field Programmable Gate Array (FPGA) para o algoritmo de hash SHA-256. O SHA-256 é bastante usado em protocolos para transporte de dados seguros ou verificação de integridade, como o HMAC, TLS e IPSec. A implementação proposta neste artigo traz uma abordagem diferenciada em relação a outros trabalhos já publicados, com núcleos agrupados executando o algoritmo SHA-256 em paralelo. Alcançando um throughput de* $0,80847 Gbps$ *para uma único núcleo e em torno de* $101 Gbps$ *para* 128 *núcleos em um único FPGA. Existe uma grande possibilidade de aplicações para tal implementação, a qual utilizou como FPGA alvo o Xilinx Virtex 6 xc6vlx240t-1ff1156.*

1. Introdução

O desenvolvimento de novas metodologias de transmissão de dados e conectividade traz consigo a necessidade de controles relacionados à segurança da informação, a fim de seja garantido a confidencialidade, integridade e disponibilidade das informações trafegadas. Para isso, julga-se interessante ter ferramentas que correspondam a esses aspectos e que possuam um desempenho suficiente para não prejudicar o processamento normal das informações nos equipamentos. Alguns desses controles são a verificação de integridade com o HMAC (*Hash-based Message Authentication Code*), a assinatura digital com o PKI (*Public Key Infrastructure*) e a transmissão de dados por meio do protocolo TLS. Além desses, [Michail et al. 2012] comenta sobre o uso do IPSec e a necessidade de ter soluções rápidas, devido as altas velocidades de conexões de rede, como existe a via fibra óptica que ultrapassa os 30Gbps. Tendo em vista essa demanda, este trabalho tem por objetivo desenvolver

um hardware dedicado para o algoritmo da família SHA-2, o SHA-256, que possui um hash de comprimento fixo em 256 bits e é bastante utilizado nas metodologias citadas (HMAC, PKI, TLS e IPSec), com o objetivo de analisar o desempenho e a área de circuito ocupada com outras publicações. O dispositivo utilizado para validar a proposta foi um FPGA (*Field Programmable Gate Array*), Xilinx Virtex 6 xc6vlx240t-1ff1156.

2. Trabalhos Relacionados

A primeira implementação publicada do algoritmo SHA-256 em FPGA foi por [Ting et al. 2002], que fez uso do kit de desenvolvimento Pilchard com uma FPGA Xilinx Virtex XCV300E-8. Este trabalho provê a utilização de vários registradores de deslocamentos em série, divididos em três blocos para armazenar as variáveis associadas ao algoritmo SHA-256. O último bloco possui oito registradores de 32 bits que armazenam os valores das variáveis hash. Tendo por fim o resultado com um *throughput* de 87 Mbps com um *clock* de 88 MHz, utilizando 1261 *slices* de um FPGA Xilinx. Já o trabalho apresentado em [Sklavos and Koufopavlou 2005], utiliza um FPGA Xilinx Virtex v200pq240, da qual propõe uma arquitetura única para três algoritmos da família SHA-2, o SHA 256, o SHA-384 e o SHA-512. Essa implementação traz um único módulo que executa todas as iterações necessárias para cada algoritmo utilizando o *rolling loop* (laço de repetição). A saída desse módulo interage com os valores das constantes armazenados em memórias ROM e por fim possuem um módulo para o armazenamento do código hash, que para gerar o do SHA-256 necessita de 65 *clocks*. Os resultados para a implementação somente do SHA-256 foi de 2384 CLB (Blocos Lógicos Configuráveis), 74 MHz de frequência máxima e 291 Mbps de *throughput*.

O trabalho desenvolvido por [Michail et al. 2012] tem como foco a alta performance do algoritmo SHA-256. Para isso, utiliza uma estrutura central em quatro partes, organizado em *pipeline*. No artigo faz-se uso de diversas técnicas de otimização como *retiming*, *precomputation* e *loop unrolling*, a fim de melhorar o *throughput* e a relação *throughput*/área. Com isso, [Michail et al. 2012] conseguiu 4 valores de hash em 32 *clocks*, sendo 8 *clocks* para cada parte do pipeline. Utilizou-se FPGAs Xilinx Virtex 5 e 6, tendo os melhores resultados no Virtex 6 com 172MHz e $11,008$ Gbps (*throughput*), usando 1831 *slices*. No trabalho de [García et al. 2014] é proposto uma solução do SHA-256 compacta focada para dispositivos móveis. Essa opera com a reutilização de módulos, tendo como principal componente uma Unidade Lógica Aritmética de 4 entradas (*Hash ALU*). Desse modo, para gerar um hash foi necessário a utilização de 280 *clocks*, porém com o uso de apenas 139 *slices* e 527 LUTs (*LookUp Tables*), a uma frequência de $64,45$ MHz obteve-se um *throughput* de $117,85$Mbps, tendo como FPGA alvo uma Xilinx Virtex 5.

Já em [Padhi and Chaudhari 2017] é criado uma arquitetura com registradores em *pipeline*, semelhante a de [Michail et al. 2012], utilizando o Xilinx Virtex-4, dividido em duas partes (Expansor e Compressor). Com esta abordagem é alcançada uma frequência máxima de $170,75$ MHz e um *throughput* de $1344,98$Mbps. O artigo de [binti Suhaili and Watanabe 2017] tem como objetivo fornecer uma implementação de hardware de alta velocidade para o algoritmo SHA-256, para isso faz duas implementações que chama de SHA-256 e SHA-256 *unfolding*. Com a utilização

de seis módulos, semelhante a proposta apresentada aqui. Porém com o diferencial das entradas dos módulos e com o arranjo desses no FPGA, pois na implementação *unfolding* de [binti Suhaili and Watanabe 2017] usa-se apenas 32 clocks, com 1215 LUTs, 871 registradores e 2429, 52 Mbps de *throughput* utilizando o FPGA Arria II Gx da Altera.

3. Descrição do Algoritmo SHA-256

O SHA-256 é um algoritmo de *hash* pertencente à família SHA-2, a qual compreende por mais três algoritmos, o SHA-224, o SHA-384 e o SHA-512, tendo como principal diferença entre eles a quantidade de iterações dentro do *loop* principal e o comprimento em bits do código hash, sendo a parte decimal do nome referente a esse quantitativo. A família SHA-2 é definida pela *Federal Information Processing Standards Publications* (FIPS PUBS) 180-4 [NIST 2015].

O SHA-256 opera com uma mensagem de tamanho qualquer na entrada, definida por $\mathbf{m}_i$, (de comprimento K_i bits). Segundo o [NIST 2015], essa mensagem, necessita ter um comprimento divisível por 512 após dois processos de extensão da mensagem. O primeiro, conhecido por *padding* ou inserção do preenchimento, é formado pela junção do número binário 1 (um) ao final da mensagem $\mathbf{m}_i$ (na parte menos significativa) e por meio do preenchimento de L bits 0 (zeros) até alcançar o comprimento de 448 bits, tratado aqui por $\mathbf{p}_i$, logo $\mathbf{K}_i + \mathbf{p}_i = 448 mod 512$. O segundo processo é a junção dos 64 bits restantes por meio da representação binária T do tamanho de $\mathbf{m}_i$ em bits, mostrado neste trabalho por $\mathbf{v}_i$. Desse modo, é possível verificar que $\mathbf{z}_i = [\mathbf{m}_i\, \mathbf{p}_i\, \mathbf{v}_i]$.

Para uma compreensão geral do algoritmo estudado foi descrito em pseudo-código todos os passos para gerar um código hash SHA-256, o qual está exposto no Algoritmo 1. Os passos citados acima, expansão de $\mathbf{m}_i$, estão mostrados entre as linhas 1 e 5. Outro passo gerado é a divisão de $\mathbf{m}_i$ em L_i blocos de 512 bits, definido na etapa *DivisãoMensagem* (linha 8 do Algoritmo 1), sendo cada bloco armazenado em um vetor $\mathbf{b}_j$, o qual é dividido em 16 palavras, $\mathbf{u}_j$, de 32 bits cada, caracterizado como

$$\mathbf{b}_j = \left[\begin{array}{cccc} \mathbf{u}_j[0] & \mathbf{u}_j[1] & \ldots & \mathbf{u}_j[15] \end{array} \right], \tag{1}$$

Logo após essa etapa, há a de inicialização das variáveis do código hash (linha 6), são oito no total, cada uma de 32 bits as quais juntas formarão a mensagem de saída gerada posteriormente a todos os passos do algoritmo, delineada aqui por $\mathbf{h}_i$ e caracterizada como número *hash*, de tamanho fixo $C = 256$ bits, e expresso como

$$\mathbf{h}_i = \left[\begin{array}{cccccccc} \mathbf{ha} & \mathbf{hb} & \mathbf{hc} & \mathbf{hd} & \mathbf{he} & \mathbf{hf} & \mathbf{hg} & \mathbf{hh} \end{array} \right]. \tag{2}$$

Porém a cada n iteração das 64 relativas ao algoritmo SHA-256 é preciso alterar outras variáveis hash, tratadas aqui por

$$\mathbf{H}(n) = \left[\begin{array}{ccccccc} \mathbf{A}(n) & \mathbf{B}(n) & \mathbf{C}(n) & \mathbf{D}(n) & \mathbf{E}(n) & \mathbf{F}(n) & \mathbf{G}(n) & \mathbf{H}(n) \end{array} \right], \tag{3}$$

onde a cada palavra dessa é atribuído um número obtido mediante o cálculo dos primeiros trinta e dois bits das partes fracionárias das raízes quadradas dos oito primeiros números primos [NIST 2015].

O laço de repetição (*loop*) da linha 11 do Algoritmo 1 calcula as funções lógicas relacionadas a expansão das 16 palavras iniciais da mensagem após o pre-processamento, $\mathbf{z}_i$, para 64 palavras, que a partir dessa operação será tratada como $\mathbf{w}(n)$. A FIPS 180-4 [NIST 2015] trata a fase desse *loop* como preprocessamento do hash, que além de calcular o $\mathbf{w}(n)$ é preciso computar os valores de $\mathbf{s0}(n)$ e $\mathbf{s1}(n)$, definidos pelas linhas 12 e 13 do Algoritmo 1 e expressos por,

$$\mathbf{s0}(n) = \text{rr}(w(n-15), 7) \oplus \text{rr}(w(n-15), 18) \oplus \text{rs}(w(n-15), 3), \tag{4}$$

$$\mathbf{s1}(n) = \text{rr}(w(n-2), 17) \oplus \text{rr}(w(n-2), 19) \oplus \text{rs}(w(n-2), 10), \tag{5}$$

nessas equações o $\oplus$ é a operação *ou exclusivo* bit a bit e o $\text{rr}(\mathbf{r}, \mathbf{s})$ simboliza a função *rightrotate*, expressa como

$$\text{rr}(\mathbf{r}, \mathbf{s}) = (\mathbf{r} \gg \mathbf{s}) \vee (\mathbf{r} \ll (32 - \mathbf{s})), \tag{6}$$

onde $\vee$, $\ll$ e $\gg$ são operações de *OR* e de deslocamento (*shift*) bit a bit a esquerda e a direita, respectivamente. Já o $\text{rs}(\mathbf{r}, \mathbf{s})$ é o *shift* bit a bit à direita sem rotacionar.

Na linha 14 tem-se a função $\mathbf{w}(n)$, a qual é utilizada para expandir a mensagem, $\mathbf{w}(n)$, composta por 16 palavras (32 bits cada) em 64, adicionando mais 48 palavras, conforme a Equação 7.

$$\mathbf{w}(n) = \mathbf{w}(n-16) + \mathbf{s0}(n) + \mathbf{w}(n-16) + \mathbf{s1}(n) \tag{7}$$

Já no *loop* da linha 16 do Algoritmo 1 ocorre as funções relacionadas ao processamento do hash, conforme RFC 4634 e FIPS 180-4 [NIST 2015]. Para cada n-ésima iteração de cada j-ésimo bloco, $\mathbf{b}_j(n)$ é feito o cálculo das funções lógicas, **S1**, **S0**, **Ch** e **Maj** a partir dos valores das variáveis *hash* $\mathbf{A}(n)$, $\mathbf{B}(n)$, $\mathbf{C}(n)$ e $\mathbf{E}(n)$, $\mathbf{F}(n)$, $\mathbf{G}(n)$, conforme descritas pelas Equações 8 a 11.

$$\mathbf{S1}(n) = \text{rr}(\mathbf{E}(n-1), 6) \oplus \text{rr}(\mathbf{E}(n-1), 11) \oplus \text{rr}(\mathbf{E}(n-1), 25), \tag{8}$$

$$\mathbf{Ch}(n) = (\mathbf{E}(n-1) \wedge \mathbf{F}(n-1)) \oplus \ (\neg\mathbf{E}(n-1) \wedge \mathbf{G}(n-1)), \tag{9}$$

$$\mathbf{S0}(n) = \text{rr}(\mathbf{A}(n-1), 2) \oplus \text{rr}(\mathbf{A}(n-1), 13) \oplus \text{rr}(\mathbf{A}(n-1), 22), \tag{10}$$

$$\mathbf{Maj}(n) = (\mathbf{A}(n-1) \wedge \mathbf{B}(n-1)) \oplus (\mathbf{A}(n-1) \wedge \mathbf{C}(n-1)) \oplus (\mathbf{B}(n-1) \wedge \mathbf{C}(n-1)), \tag{11}$$

onde $\neg$ e $\wedge$ são os operadores de negação e *AND* bit a bit, respectivamente.

Após essa etapa o valor das variáveis $\mathbf{A}(n)$ a $\mathbf{H}(n)$ é atualizado, de acordo com a linha 21 do Algoritmo 1. A atualização variáveis de *hash* é expressa como

$$\mathbf{H}(n) = \mathbf{G}(n-1), \tag{12}$$

$$\mathbf{G}(n) = \mathbf{F}(n-1), \tag{13}$$

$$\mathbf{F}(n) = \mathbf{E}(n-1), \tag{14}$$

$$\mathbf{E}(n) = \mathbf{D}(n-1) + \mathbf{Temp1}(n-1), \tag{15}$$

$$\mathbf{D}(n) = \mathbf{C}(n-1), \tag{16}$$

Algoritmo 1 SHA-256 para cada i-ésima mensagem $\mathbf{m}_i$

$\mathbf{z}_i \leftarrow [\mathbf{m}_i]$
$\mathbf{p}_i \leftarrow \mathit{GeraçãoPreenchimento}(K_i)$
$\mathbf{z}_i \leftarrow [\mathbf{m}_i\, \mathbf{p}_i]$
$\mathbf{v}_i \leftarrow \mathit{GeraçãoComprimento}(K_i)$
$\mathbf{z}_i \leftarrow [\mathbf{m}_i\, \mathbf{p}_i\, \mathbf{v}_i]$
$\mathbf{h_i} \leftarrow \mathit{InicializaçãoHash}(\,)$
para $j \leftarrow 0$ **até** $L_i - 1$ **faça**
  $\mathbf{b}_j \leftarrow \mathit{DivisãoMensagem}(\mathbf{z}_i)$
  $n \leftarrow -1$
  $\mathbf{H}(n) \leftarrow \mathit{InicializaVariáveisHash}(\,)$
  para $n \leftarrow 0$ **até** 63 **faça**
    $\mathbf{s0}(n) \leftarrow \mathit{CálculoFunções0}(n, \mathbf{b}_j)$
    $\mathbf{s1}(n) \leftarrow \mathit{CálculoFunções1}(n, \mathbf{b}_j)$
    $\mathbf{w}(n) \leftarrow \mathit{CálculoFunçãoW}(n, \mathbf{b}_j, s0, s1)$
  fim para
  para $n \leftarrow 0$ **até** 63 **faça**
    $\mathbf{S1}(n) \leftarrow \mathit{CálculoFunçãoS1}(n, \mathbf{E}(n))$
    $\mathbf{S0}(n) \leftarrow \mathit{CálculoFunçãoS0}(n, \mathbf{A}(n))$
    $\mathbf{maj}(n) \leftarrow \mathit{CálculoFunçãoMaj}(n, \mathbf{A}(n), \mathbf{B}(n), \mathbf{C}(n))$
    $\mathbf{Ch}(n) \leftarrow \mathit{CálculoFunçãoCh}(n, \mathbf{E}(n), \mathbf{F}(n), \mathbf{G}(n))$
    $\mathbf{H}(n) \leftarrow \mathit{AtualizaVariáveisHash}(\mathbf{H}(n))$
  fim para
  $\mathbf{h}_i \leftarrow \mathit{AtualizaHash}(\mathbf{H}(n))$
fim para

$$\mathbf{C}(n) = \mathbf{B}(n-1), \tag{17}$$

$$\mathbf{B}(n) = \mathbf{A}(n-1) \tag{18}$$

e

$$\mathbf{A}(n) = \mathbf{Temp1}(n-1) + \mathbf{Temp2}(n-1) \tag{19}$$

nos quais,

$$\mathbf{Temp1}(n) = \mathbf{H}(n-1) + \mathbf{s1}(n-1) + \mathbf{Ch}(n-1) + \mathbf{K}(n-1) + \mathbf{w}(n-1), \tag{20}$$

$$\mathbf{Temp2}(n) = \mathbf{S0}(n-1) + \mathbf{Maj}(n-1) \tag{21}$$

e $\mathbf{K}(n)$ é um vetor com os primeiros 32 bits das partes fracionárias das raízes cúbicas dos primeiros 64 números primos [NIST 2015].

A finalização do algoritmo com o valor final do código hash, 256 bits, é dado após as 64 iterações e por meio da soma das variáveis hash (de $\mathbf{A}$(n) a $\mathbf{H}$(n) com os valores iniciais do hash, armazenados inicialmente no vetor $\mathbf{h}_i$. Como exemplo a variável $\mathbf{A}$, que se repete para todas as demais variáveis hash, $\mathbf{ha} = \mathbf{A}(63) + \mathbf{ha}$.

4. Implementação do SHA-256 no Hardware Reconfigurável

A Figura 1 detalha uma única implementação do algoritmo SHA-256 em hardware utilizando uma representação em *Register Transfer Level* (RTL). Com base na figura

pode-se observar o direcionamento dos sinais (ou variáveis) entre os componentes de *datapath* e os registradores RA, RB, RC, RD, RE, RF, RG e RH.

O início do fluxo de sinais ocorre mediante entrada da i-ésima mensagem $\mathbf{m}_i$ no módulo chamado INIT, o qual é incumbido pelas funcionalidades apresentadas entre linhas 1 e 6 do Algoritmo 1. O módulo DM (linha 8 do Algoritmo 1), executa a função da Divisão de Mensagem, que divide a mensagem em blocos, $\mathbf{b}_j$, conforme a Equação 1, e desse o dividem em 16 palavras, $\mathbf{u}_j$, de 32 bits cada, as quais são as entradas para o módulo GW. Esse que atua na expansão da mensagem $\mathbf{w}(n)$ para 64 palavras, expressos pela Equação 7. Outra entrada tanto para esse módulo quanto para o GK (armazena os valores do vetor $\mathbf{K}(n)$) é a saída do módulo CN, um contador de 6 bits (amplitude de 0 a 63), referente as linhas 11 e 16 do Algoritmo 1. Já o contador CJ atua no controle do laço descrito pela linha 7 do Algoritmo 1.

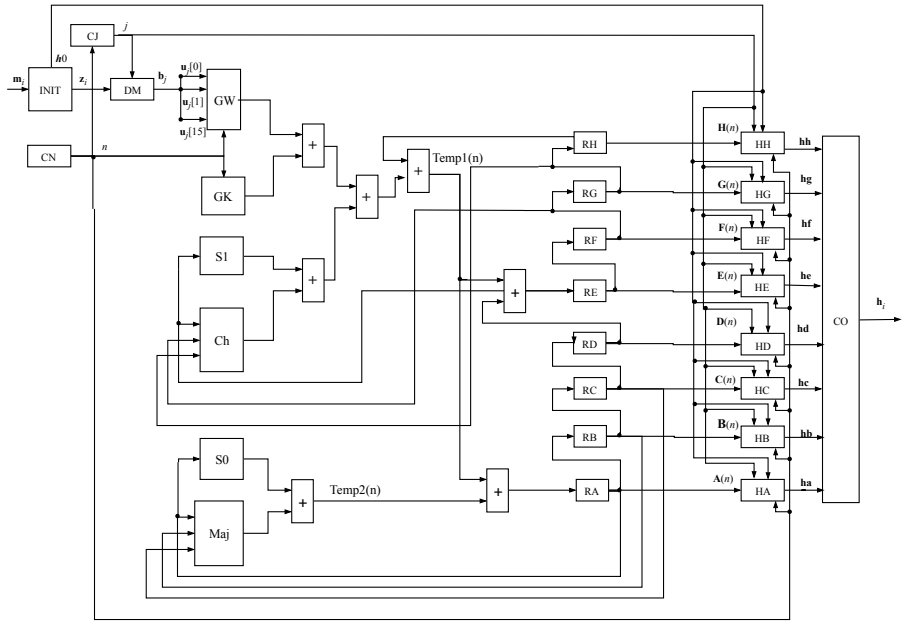

Figura 1. Arquitetura geral da implementação em hardware do SHA-256 proposto.

Os módulos S1, Ch, S0 e Maj são as implementações das Equações 8, 9, 10 e 11, os quais utilizam portas lógicas com circuitos específicos de 32 bits.

Os valores de $\mathbf{Temp1}(n)$ e $\mathbf{Temp2}(n)$ são os resultados da soma dos demais módulos (Equações 20 e 21). Inclusive para o $\mathbf{Temp1}(n)$ faz-se necessário o valor do registrador RH, o qual armazena o valor referente a $\mathbf{H}(n)$ do vetor das variáveis hashes, descrito pela Equação 3 (esses valores encontram-se disponíveis no [NIST 2015]). O que ocorre com os demais registradores e a cada clock (iteração de n) os valores armazenados são atualizados entre si, conforme preconiza as Equações 12 a 19. Essa etapa de atualização das variáveis hashes é executada na linha 21 do Algoritmo 1.

Por conseguinte, ao final das 64 iterações do laço de repetição em n (linha 16 do Algoritmo 1) as partes que constitui o código hash, **ha**, **hb**, **hc**, **hd**, **he**, **hf**, **hg** e **hh** (ver Equação 2), são atualizadas pelos módulos HA, HB, HC, HD, HE, HF, HG e HH, respectivamente. Esta etapa é executada na linha 23 do Algoritmo

1. Por fim, o módulo CO têm a função de concatenar os 8 barramentos de 32 bits constituídos pelos sinais **ha**, **hb**, **hc**, **hd**, **he**, **hf**, **hg** e **hh** e produzir um sinal serial com o código hash $\mathbf{h}_i$.

5. Análises e Resultados

A Tabela 1 exibe os resultados alcançados após o processo de síntese da proposta descrita na seção 4 no FPGA Xilinx Virtex 6 xc6vlx240t-11156, o qual necessita de 65 clocks para gerar o código hash, semelhante ao [Padhi and Chaudhari 2017]. Cada estrutura implementada, relativa a Figura 1, é chamada de núcleo, assim a coluna com o título NN da Tabela 1 refere-se a quantidade de núcleos implementados em um único FPGA. A coluna seguinte, NR, expõe o número de registradores utilizados e a terceira coluna, PR, apresenta a porcentagem de registradores usados em comparação ao total disponível no FPGA alvo (301.440). A quarta e quinta coluna, chamadas de NLUT e PLUT, representam o quantitativo de LUTs usadas em cada implementação e a porcentagem de LUTs utilizadas em relação às disponíveis (150.720). Já a sexta e a sétima coluna exibem os resultados da taxa de amostragem, T_s e o *throughput*, R_s, respectivamente.

Tabela 1. Resultados associados a ocupação, taxa de amostragem e throughput para vários núcleos do SHA-256.

NN	NR	PR (%)	NLUT	PLUT (%)	T_s (ns)	R_s (Gbps)
1	494	0,16	6.077	4,03	9,743	0,80847
16	982	0,32	12.132	8,04	9,990	12,61569
64	3.982	1,32	48.438	32,14	9,981	50,50827
128	5.473	1,81	66.632	44,21	9, 975	101,0773

Ainda na Tabela 1 é possível verificar, que a quantidade de registradores e LUTs usados na implementação para um núcleo e oito são os mesmos, mostrando uma completa utilização desses recursos, porém com uma razão de aproximadamente 8 vezes no *throughput*. A proposta utiliza o conceito dos módulos em *loop*, mantendo o hardware simples, porém ampliado a medida que mais núcleos são adicionados. Para isso possui a necessidade de 64 iterações à geração do código hash. Tendo assim, um *throughput* máximo em torno de 101 Gbps empregando 128 núcleos em paralelo em um único FPGA, resultado ainda não observado na literatura. Esse modelo de implementação pode ser usado para gerar hashes de um banco de dados de senhas em texto claro, por exemplo, 128 hashes são gerados em $638,4ns$, resultando em um desempenho em torno de 200 milhões de hashes por segundo (Mhash/s) para senhas até 56 caracteres. Característica para execução de um ataque de força bruta.

Na Tabela 2 é possível verificar uma comparação, em relação ao *throughput* e a quantidade de *Slices*, entre as publicações citadas neste artigo, tendo na primeira coluna a referência e na segunda o modelo do FPGA utilizado. Os resultados mostram que a abordagem proposta aqui consegue um ganho associado ao *throughput*, ou um *speedup*, de 9× para a proposta mais rápida apresentada em [Michail et al. 2012] e mais de 100× para as outras proposta apresentadas nos trabalhos de [Ting et al. 2002], [Sklavos and Koufopavlou 2005], [García et al. 2014] e

[Padhi and Chaudhari 2017]. E uma utilização de 130× maior na quantidade de *Slices* em relação à [García et al. 2014] e $9,9\times$ em relação à [Michail et al. 2012].

Tabela 2. Tabela comparativa entre as publicações relacionadas.

Referência	Modelo do FPGA	*Slices*	$R_s(Gbps)$
[Ting et al. 2002]	Virtex XCV300E-8	1.261	0,087
[Sklavos and Koufopavlou 2005]	Virtex v200pq240	2.384	0,291
[Michail et al. 2012]	Virtex 6	1.831	11,008
[García et al. 2014]	Xilinx Virtex 5	139	0,1178
[Padhi and Chaudhari 2017]	Xilinx Virtex-4	610	1,3449
Proposta deste trabalho	Xilinx Virtex 6	18.168	140,8218

6. Conclusões

Desse modo, esse trabalho apresentou um modelo para implementação em hardware do algoritmo SHA-256, trazendo uma abordagem de reutilização dos blocos para cada iteração. Tendo como possibilidade a utilização de poucos núcleos (para usos em dispositivos com baixo consumo de energia) ou com até 128 para aplicações com vistas a alta performance, a qual chega a um *throughput* de $101,0773$Gbps. Tais resultados são bastante significativos, tendo em vista que ainda não foram observados resultados semelhantes em periódicos publicados.

Referências

binti Suhaili, S. and Watanabe, T. (2017). Design of high-throughput sha-256 hash function based on fpga. In *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, pages 1–6.

García, R., Algredo-Badillo, I., Morales-Sandoval, M., Feregrino-Uribe, C., and Cumplido, R. (2014). A compact fpga-based processor for the secure hash algorithm sha-256. *Computers and Electrical Engineering*, 40(1):194 – 202. 40th-year commemorative issue.

Michail, H. E., Athanasiou, G. S., Kelefouras, V., Theodoridis, G., and Goutis, C. E. (2012). On the exploitation of a high-throughput sha-256 fpga design for hmac. *ACM Trans. Reconfigurable Technol. Syst.*, 5(1):2:1–2:28.

NIST (2015). Secure Hash Standard (SHS). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Padhi, M. and Chaudhari, R. (2017). An optimized pipelined architecture of sha-256 hash function. In *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, pages 1–4.

Sklavos, N. and Koufopavlou, O. (2005). Implementation of the sha-2 hash family standard using fpgas. *The Journal of Supercomputing*, 31(3):227–248.

Ting, K. K., Yuen, S. C. L., Lee, K. H., and Leong, P. H. W. (2002). An fpga based sha-256 processor. In Glesner, M., Zipf, P., and Renovell, M., editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pages 577–585, Berlin, Heidelberg. Springer Berlin Heidelberg.