POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Progettazione di un acceleratore hardware per cifratura autenticata mediante sintesi ad alto livello

Progetto di Ingegneria Informatica

Grazzani Davide, Rogora Matteo, Zaffiretti Stefano

Advisor:
Prof. Zoni Davide

Co-advisors:
Galimberti Andrea

Academic year:
2021-2022

Estratto: Gli schemi di cifratura autenticata garantiscono confidenzialità, integrità e autenticità dei dati trasmessi. AES-GCM è attualmente lo schema di cifratura autenticata più diffuso. La necessità di implementare comunicazioni cifrate tra gli svariati dispositivi che compongono il cosiddetto Internet delle cose (o IoT, Internet of Things), come elettrodomestici, sensori intelligenti e tag RFID, ha portato alla definizione di crittosistemi "lightweight", adatti cioè anche a dispositivi con ridotte prestazioni e consumi energetici. ACORN e Ascon sono i due crittosistemi "lightweight" per cifratura autenticata che sono risultati finalisti nella competizione CAESAR, avente l'obiettivo di identificare nuovi schemi alternativi a AES-GCM.
Supportare in modo efficace, mediante un acceleratore hardware dedicato, l'esecuzione di un crittosistema su un dispositivo IoT permette di garantire la sicurezza richiesta dall'uso di dati sensibili, critici e privati mantenendo allo stesso tempo una qualità del servizio e delle prestazioni soddisfacenti per l'utente finale.
La sintesi ad alto livello traduce la specifica di un sistema digitale, descritta in un linguaggio di programmazione quale C, nella sua descrizione register transfer level (RTL), senza richiedere conoscenze avanzate di progettazione hardware. La descrizione RTL ottenuta può quindi essere sintetizzata su FPGA o ASIC.

Key-words: Sintesi ad alto livello (HLS), Sicurezza, Cifratura autenticata

1. Introduzione

Ascon[1] appartiene a una famiglia di algoritmi di cifratura autenticata progettati per essere a bassa dissipazione di potenza e di facile implementazione, al contempo garantendo le contromisure necessarie ad eventuali attacchi side-channel. È stato selezionato come scelta primaria durante la competizione CAESAR e ora sta competendo come finalista nel NIST Lightweight Cryptography competition (2019–).
Vitis™ HLS[2] è uno strumento di sintesi ad alto livello che consente di cablare, nella struttura logica RTL e in blocchi RAM/DSP, funzioni in C, C++ e OpenCL™.
Nel corso del nostro progetto, avendo come punto di partenza l'implementazione in C di Ascon, ci siamo occupati della sua sintesi in HLS, dell'analisi dei risultato ottenuto e dell'ottimizzazione del risultato RTL. In ordine di valutare la bontà del risultato in HLS abbiamo utilizzato come riferimento l'implementazione "ad hoc" in VHDL. Nella prossima sezione ci occuperemo di spiegare più nel dettaglio lo scopo del nostro lavoro, con riferimento alla metodologia di sviluppo, le fasi di implementazione e l'approccio con cui abbiamo affrontato le limitazioni intrinseche di HLS.

2. Scopo del Progetto

Lo scopo del nostro progetto è stato quello di, avendo come punto di partenza l'implementazione in linguaggio C del crittosistema Ascon, convertirlo con lo strumento Vitis HLS in un'implementazione di sintesi ad alto livello. Abbiamo utilizzato la versione Vitis HLS 2021.2 approcciandoci al suo funzionamento, perlomeno inizialmente, come una black box.
Una volta ottenuto, in fase di cosimulazione, un risultato che soddisfacesse le specifiche dell'implementazione software in C ci siamo occupati di analizzare qualitativamente il risultato ottenuto, confrontandolo con l'implementazione hardware VHDL di Ascon.
Infine, sia sfruttando funzionalità proprie dello strumento Vitis HLS, come le direttive di sintesi, sia modificando opportunamente l'implementazione in C, abbiamo cercato quanto più possibile di avvicinarci alle performance registrate dall'implementazione hardware in VHDL. Nella prossima sezione percorreremo le principali fasi di sviluppo del progetto.

3. Svolgimento del Progetto

Abbiamo utilizzato come riferimento per l'implementazione in C di Ascon quella fornita dai loro programmatori sulla repository Github[3], in particolare la versione 128v12/ref. Riguardo all'implementazione VHDL abbiamo utilizzato come riferimento quella fornita su Github[4] dagli sviluppatori di Ascon, in particolare la v1.
La realizzazione del nostro progetto si è svolta in tre fasi principali. La fase di sintesi ha riguardato l'analisi del comportamento del codice C di Ascon per poter verificare il suo corretto funzionamento durante la fase di cosimulazione in Vitis HLS. Come testbench di riferimento per questa fase abbiamo utilizzato quella fornita direttamente dagli sviluppatori e scritta anch'essa in C. Lo strumento Vitis HLS infatti permette di serializzare con un flusso RTL sia il codice sorgente dell'algoritmo che verificare il suo comportamento con testbench scritte nei linguaggi supportati dall'ambiente di sviluppo, in questo caso il C.
Essendo lo strumento HLS designato specificatamente per sintetizzare acceleratori hardware partendo da implementazioni di algoritmi in C si è rivelato particolarmente adatto per il crittosistema Ascon e una volta ottenuto un flusso HLS funzionante e corretto ci siamo dedicati alla fase di ottimizzazione e di analisi prestazionale.
Riguardo l'ottimizzazione è stato possibile sfruttando sia funzionalità intrinseche all'ambiente di sviluppo, quali le direttive che applicando modifiche al codice C. Infine ci siamo occupati di analizzare il risultato ottenuto, mettendolo in confronto con il comportamento e le prestazioni dell'implementazione puramente software in C e pertanto eseguita su processori generic purpose, sia con l'implementazione di design puramente hardware realizzata in VHDL. Parleremo nel dettaglio di questi risultati in una sezione apposita.

3.1. Sintesi e Cosimulazione

La principale problematica che abbiamo riscontrato nell'esecuzione della cosimulazione è stata l'incompatibilità dello strumento Vitis HLS di serializzare l'utilizzo dei puntatori.
Infatti abbiamo dovuto ridefinire le interfacce dei moduli di cifratura e decifratura passando dall'utilizzo di puntatori generici (`unsigned char*`) all'utilizzo di vettori di lunghezza definita(`unsigned char[n]`), poiché Vitis HLS, non potendo gestire la memoria in maniera dinamica, ha accesso solamente alla memoria specificata nel momento della sintesi, che nel caso di puntatori generici comprende solamente la variabile puntata, mentre nel caso di vettori di lunghezza esplicita comprende l'intero vettore.
Questa è stata la principale limitazione che abbiamo riscontrato durante l'utilizzo di questo strumento di sintesi, poiché C è un linguaggio di programmazione che usufruisce intensivamente di puntatori e memoria dinamica.
Riguardo ad altre modifiche al codice C abbiamo ridefinito i puntatori all'interno dell'interfaccia con la direttiva `volatile`, per evitare problemi noti di inconsistenza nel caso di letture multiple dallo stesso puntatore.
La definizione dello stato interno è stata modificata da una `struct {unsigned long[5]}` ad una più semplice `unsigned long[5]`, sebbene questo non fosse un passo strettamente necessario, garantisce correttezza in caso di casting dello stato ad una differente rappresentazione e rende più compatto il codice in alcune sezioni (es. definizione delle permutazioni).
Ottenuto un flusso sintetizzabile, abbiamo proseguito nella verifica della sua correttezza.
Per fare ciò abbiamo diviso in segmenti l'esecuzione dell'algoritmo e inserito un'interfaccia di controllo, in corrispondenza delle funzioni di debug del codice originale. Questo permette di copiare lo stato interno della funzione al termine di un determinato segmento di codice, in questo modo abbiamo confrontato gli stati interni dell'implementazione C e della sintesi in diversi momenti dell'esecuzione per trovare eventuali incongruenze.
Per effettuare il confronto abbiamo attuato delle modifiche alla testbench fornitasi dagli sviluppatori di Ascon, creando due interfacce per la funzione di encrypt (una sintetizzata nel flusso HLS, l'altra implementata in codice C) ed aggiungendo ai controlli standard della testbench un ulteriore controllo sugli stati interni delle funzioni.

3.2. Ottimizzazione

Il risultato iniziale offerto da Vitis HLS è necessario di revisione e ottimizzazione. Al fine di perfezionare il flusso, è possibile agire in due modi diversi. Applicando delle direttive embedded nella sintesi del flusso HLS, mediante lo strumento di sintesi, o modificando l'architettura software in modo tale che si avvicini di più in una forma che sia sintetizzabile in maniera più efficiente dallo strumento Vitis HLS.
Ci occuperemo invece di mostrare le analisi quantitative degli esiti dell'ottimizzazione del flusso HLS nella prossima sezione.

3.2.1. Direttive Vitis HLS

Vitis HLS consente di eseguire la sintesi con delle configurazioni specifiche, chiamate direttive, in grado di modificare sensibilmente diversi aspetti implementativi del nostro FPGA finale. Esistono diverse direttive, qui di seguito verranno riportate quelle impiegate. Obbiettivo principale e quindi condiviso tra tutte queste direttive è quello di ridurre l'area oppure le latenze.

Nome direttiva	Breve Spiegazione
Inline	Rimozione della gerarchia della funzione a questo livello. Viene utilizzata per implementare un'ottimizzazione logica in grado di rimuovere gli overhead causati da una chiamata a funzione. Ne conseguono migliori latenze
Pipeline	Riduzione del tempo di inizializzazione permettendo l'esecuzione in contemporanea di operazioni. Direttiva applicabile a loop o funzioni
DataFlow	Permette esecuzioni concorrenti a livello di task. Utile per ottimizzare throughput e/o latenza

Table 1: Direttive

3.2.2. Modifiche Codice C

Per quanto concerne l'ottimizzazione tramite modifiche effettuate al codice C, illustriamo in seguito una rapida spiegazione del tipo di modifica e perché abbiamo ritenuto necessario effettuarla. Abbiamo:

- Ridotto il numero dei pin I/O utilizzati ridefinendo tutte le variabili di interfaccia, sia in input che in output, ad 8 bit rispetto agli originali 64 bit (derivanti dall'ottimizzazione ottenuta su architetture a 64 bit).
- Implementato interfacce dati come vettori di parole a 32 bit anziché 8 bit per velocizzare le operazioni di caricamento e scrittura (le quali risultavano eccessivamente pesanti nello scheduler e non ottimizzabili in quanto non parallelizzabili), poi scartate poichè il numero di pin I/O richiesto avrebbe superato la quantità di pin fornita dall'FPGA usato come riferimento nell'implementazione ad hoc in VHDL.
- Ridefinito le permutazioni in maniera ricorsiva al fine di ridurre l'area occupata, al costo di perdere in prestazioni. Questa operazione si è dimostrata inconcludente in quanto tra i risultati post implementazione delle due versioni non risulta alcuna differenza.

4. Analisi dei risultati ottenuti

Nella prima tabella vediamo riassunti complessivamente i risultati delle ottimizzazioni effettuate al flusso HLS, mostrando sia le ottimizzazioni effettuate al codice C che le direttive Vitis implementate.
Questa tabella condensa tutti i risultati che abbiamo ottenuto nelle analisi prestazionali e delle diverse implementazioni e ottimizzazioni, ed è il nostro punto di partenza per più significative analisi e report dettagliati.
Con la Table 3 infatti filtriamo il contenuto di questa tabella "complessiva" iniziando ad estrapolare considerazioni più pratiche di performance ed efficienza.

Solution index		C code alterations	Description	Vitis Directives	Latency (cycles)			Synthesis (clock in ns)				Implementation (clock in ns)			
					Avg	Max	Min	LUT	FF	Target	Reach	LUT	FF	Target	Reach
a		VHDL reference	VHDL implementation	-	100	-	-	-	-	-	-	1459	666	10	10
b		Standard version	Debloat of uselsess ports (baseline)	-	120	146	94	9213	3654	15	15	4590	3438	10	8.5
c	1	Standard version	Directives on permutations.h	INLINE RECOURSIVE	129	160	99	42969	5259	10	3	11361	4984	10	7.5
	2			(P6, P8, P12)	253	307	198	43265	12473	3	3	13252	10393	3	2.9
d		Standard version	Directives on round.h	PIPELINE (ROR)	147	181	113	11492	3420	13	13	3701	3147	10	8.4
e		Pipelined permutations	Recursive permutation definition (P12 → P8 → P6)	-	120	146	94	9213	3654	15	14	4590	3438	10	8.5
f		Standard version	Directives on round.h	DATAFLOW (ROUND)	403	510	293	9908	7302	13.5	13.5	6352	7174	10	6.4
g	1	Standard version	Directives on permutations.h	DATAFLOW	157	196	120	50401	18665	10	3	23275	21029	10	7.9
	2			(P6, P8, P12)	297	344	231	50823	22821	3	3	23375	21029	10	7.9
h	1	Standard version	Directives on permutations.h	PIPELINE	130	161	99	43052	6671	10	3	11057	6399	10	7.4
	2			(P6, P8, P12)	297	350	231	43723	19057	3	3	11057	6399	3	7.4
i	1	Standard version	Directives on permutations.h	PIPELINE	124	153	97	43081	6219	10	3	11407	5947	10	9
	2			INLINE RECURSIVE (P6, P8, P12)	251	303	199	43504	13903	3	3	11407	5947	3	9

Table 2: Riepilogo risultati ottimizzazione HLS

	FPGA Area	Time (ns)	Performance (W/s)	Performance (%)	Efficiency	Efficiency (%)	Perfomance	Balanced	Efficiency	Total
a	2125	1000	1000000	73,37	470.59	100	79.70	86.69	94.67	86.69
b	8028	1020	980392.16	71,93	122.12	25.95	62.74	48.94	35.15	48.94
c.1	16345	967.5	1033591.73	75,83	63.36	13.44	63.36	56.12	25.92	56.12
c.2	23645	733.7	1362954.89	100	57.64	12.25	82.45	56.12	29.80	56.12
d	6848	1234.8	809847.75	59.42	118.26	25.13	52.56	42.27	31.99	42.27
e	8028	1020	980392.16	71.93	122.12	25.95	62.74	48.94	35.15	48.94
f	13526	2579.2	387717.12	28.45	28.66	6.09	23.98	17.27	10.56	17.27
g.1	44304	1240.3	806256.55	59.16	18.20	3.87	48.10	31.51	14.92	31.51
g.2	44404	2346.3	426202.96	31.27	9.60	2.04	25.42	16.66	7.89	16.66
h.1	17456	962	1039501.04	76.27	59.55	12.65	63.55	44.46	25.38	44.46
h.2	17456	2197.8	455000.46	33.38	26.07	5.54	27.81	19.46	11.11	19.46
i.1	17354	1116	896057.35	65.74	51.63	10.97	54.79	38.36	21.93	38.36
i.2	17354	2259	442673.75	32.48	25.51	5.42	27.07	18.95	10.83	18.95

Table 3: Tabella riassuntiva delle implementazioni hardware

Questa tabella contiene dati rielaborati in grado di offrire visione di performance ed efficienza: sia singolarmente, con un valore numerico, ma anche in contesto offrendo la possibilità di confrontare direttamente implementazioni una con l'altra.
Gli istogrammi mostrati nelle pagine successive rappresentano, in maniera più dettagliata, gli esiti dell'ottimizzazione tenendo in considerazione diversi criteri di analisi (performance, efficienza...).
Nota: con Efficienza pura intendiamo Performance/Area, con Perfomance intendiamo Words/second.

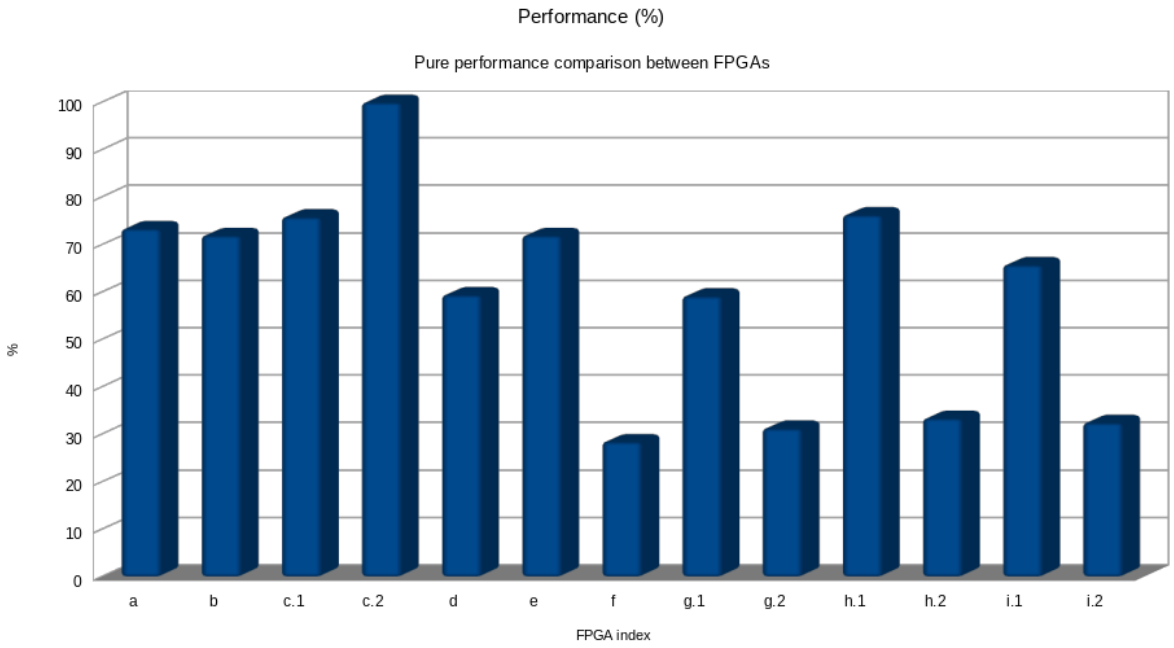

Figure 1: Pure Performance

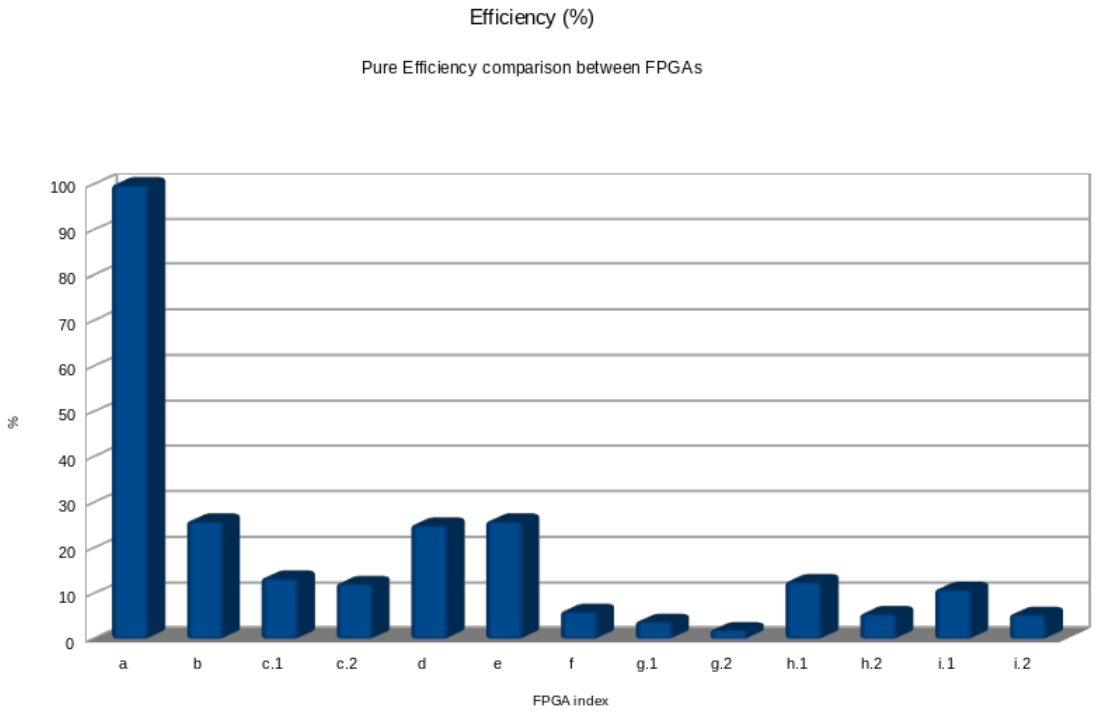

Figure 2: Pure Efficiency

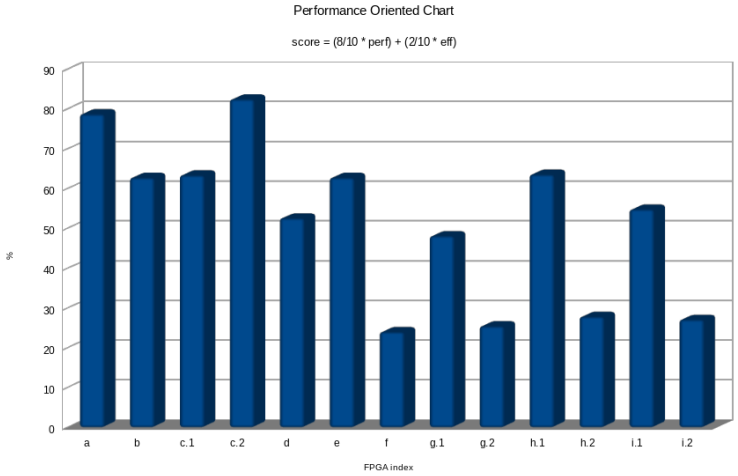

Figure 3: Performance oriented

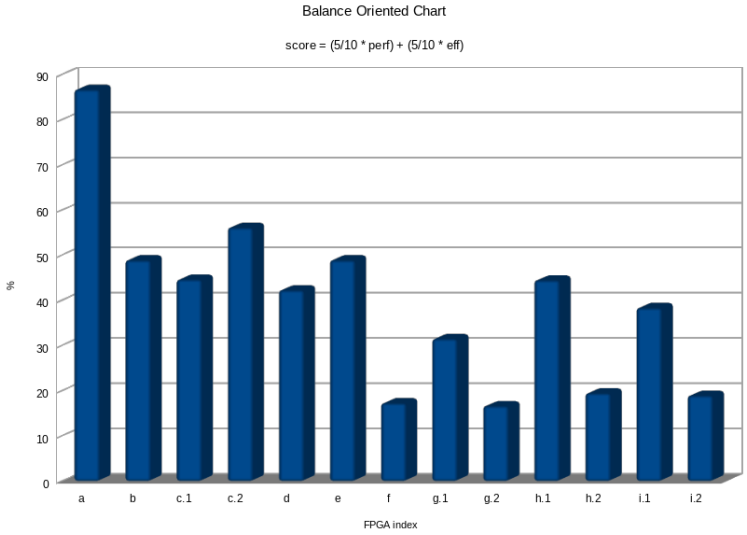

Figure 4: Bilanciato

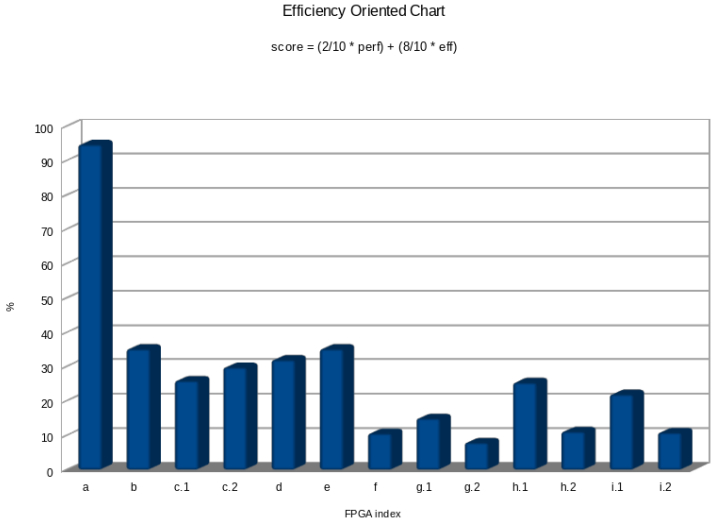

Figure 5: Efficiency oriented

Le informazioni fornite da questi istogrammi ci permettono di affermare che:

- l'implementazione standard VHDL (presa da GitHub) è la seconda migliore in termini di pura performance ed è molto più che ottima in termini di efficienza (circa il 75% più efficiente della migliore implementata in Vitis HLS) il che rende questa implementazione la migliore e la più consigliabile se si è in cerca un componente perfettamente ottimizzato.
- il sintetizzatore di Vitis HLS fa un buon lavoro out-of-the-box con un'implementazione standard molto vicina in termini di performance a quella originale (GitHub) ed è anche la migliore che abbiamo ottenuto in termini di efficienza.
- Con l'implementazione c.2 siamo riusciti a superare lato prestazioni il design originale che avevamo fissato come obbiettivo.

Nella tabella che segue vengono riportati i risultati ottenuti invece lato software su diverse CPU consumer (dati forniti dagli sviluppatori di Ascon).

Processor	Base clock (GHz)	Clock period (ns)	Clock cycles per byte	Performance (W/s)	Performance (%)
Intel Core i5-6300U	2.4	0.41667	35	4285714.28571	314.44285
ARM1176JZF-S (ARMv6)	1	1	167	374251.497	27.45883
Intel Core i5-4200U	1.6	0.625	49	2040816.51351	149.73469
Cortex-A7 (NEON)	0.55	1.81819	148	232263.51351	17.04117
Amd Ryzen 5600X	3.7	0.27027	21.9	10559360.73059	774.74029
Amd Ryzen 5600X (Optimized compile time)	3.7	0.27027	240.5	961538.46154	70.54807

Table 4: Tabella riassuntiva ASCON software per messaggi da 16 byte

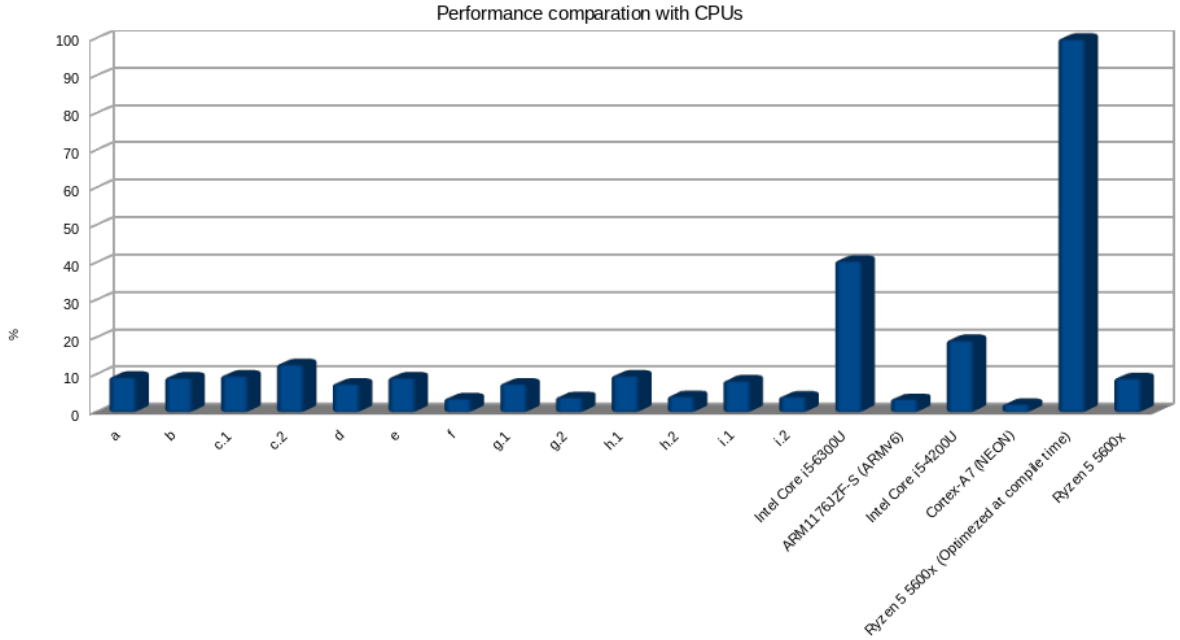

Figure 6: CPU e FPGA a paragone

Analizzando le tabelle precedenti possiamo affermare:

- Le CPU (sia lato desktop che laptop, anche non recenti) dominano sulle prestazioni.
  Un'analisi lato consumi di potenza sarebbe necessaria per stabilire quanto il prediligere il nostro componente sia un trade-off interessante.
  Contestualizzando: un AMD Ryzen 7 5800x ha un picco a 23w durante l'esecuzione di Ascon, i processori generic purpose ad altre prestazioni sono poco adatti in situazioni a bassa dissipazione di potenza.

- Le CPU mobile (seppur datate) sono molto più lente. Il vantaggio di avere un componente dedicato, in contesti in cui si vuole risparmiare più possibile per l'integrato, secondo noi potrebbe offrire il duplice vantaggio di garantire prestazioni elevate tenendo contenuti i costi di realizzazione. Un caso d'uso generale potrebbe essere un dispositivo IoT a ridotta necessità di computazione (magari perché è solo un sensore), ma ha bisogno di poter inviare le proprie informazioni in maniera sicura, come ad esempio: CCTV Camera, termostati intelligenti e componenti di domotica.

5. Conclusioni

Nelle precedenti sezioni abbiamo documentato scopo e svolgimento del progetto, facendo riferimento alle operazioni di ottimizzazione svolte e l'analisi del risultato ottenuto.
Vitis HLS risulta uno strumento utile per un programmatore software interessato a realizzare un acceleratore hardware specifico per un algoritmo, una richiesta sempre più frequente con l'espansione del settore IoT.
Come è emerso dalle nostre analisi prestazionali, infatti, in questo modo, il componente risultante, malgrado non riesca ad eguagliare in prestazioni un componente disegnato ad hoc in realizzazione hardware con VHDL racchiude lo stesso un giusto compromesso tra performance e complessità di realizzazione.

6. Riferimenti

[1] Ascon Lightweight Authenticated Encryption and Hashing `https://ascon.iaik.tugraz.at`
[2] Xilinx Vitis HLS 2021.2 Docs `https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0090-vitis-hls-hub.html`
[3] Repository Github Ascon C `https://github.com/ascon/ascon-c`
[4] Repositoty Github Hardware `https://github.com/ascon/ascon-hardware`