



POLITECNICO
MILANO 1863

Progetto Ingegneria Informatica

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

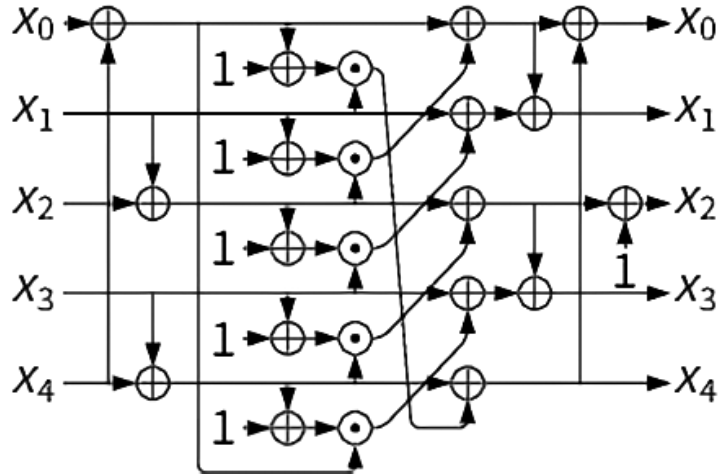
Introduzione



Ascon: Cos'è? Perché è interessante? Cosa offre?

Vitis HLS: Cos'è? Perché è interessante? Cosa offre?

Introduzione: Ascon



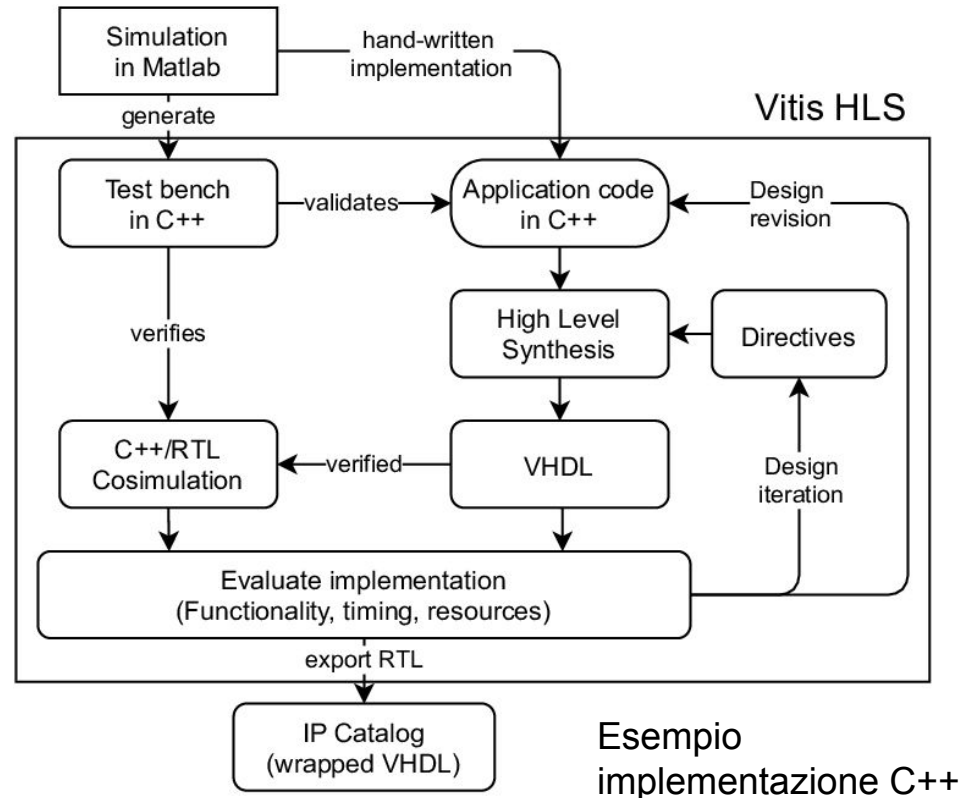
Ascon's S Box

Ascon is a family of **authenticated encryption** and **hashing** algorithms designed to be lightweight and easy to implement, even with added countermeasures against side-channel attacks.

```
// state words x0..x4 (uint64), temporary variables t0..t4 (uint64)
x0 ^= x4;    x4 ^= x3;    x2 ^= x1;
t0  = x0;    t1  = x1;    t2  = x2;    t3  = x3;    t4  = x4;
t0 =~ t0;    t1 =~ t1;    t2 =~ t2;    t3 =~ t3;    t4 =~ t4;
t0 &= x1;    t1 &= x2;    t2 &= x3;    t3 &= x4;    t4 &= x0;
x0 ^= t1;    x1 ^= t2;    x2 ^= t3;    x3 ^= t4;    x4 ^= t0;
x1 ^= x0;    x0 ^= x4;    x3 ^= x2;    x2 =~ x2;
```

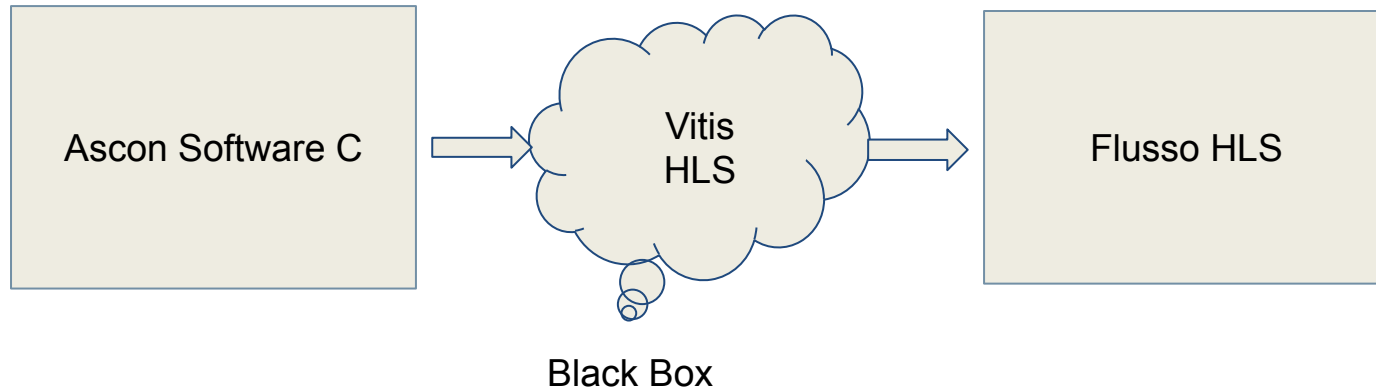
Introduzione: Vitis HLS

Vitis HLS è uno strumento di sintesi, sviluppato dalla Xilinx, che consente di sintetizzare un flusso RTL ad alto livello partendo da un algoritmo scritto in C, C++ o Open CL.

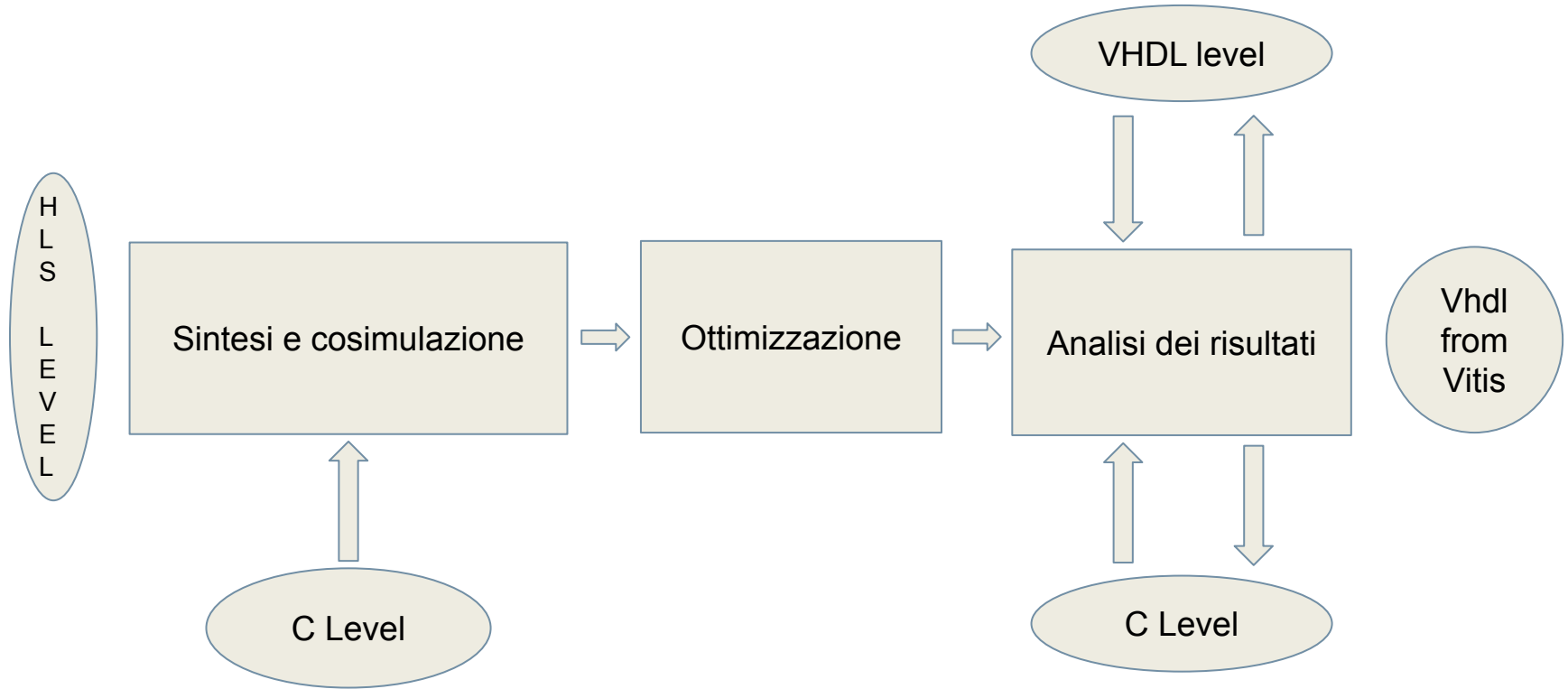


Scopo del Progetto

Lo scopo del nostro progetto è stato quindi, avendo come punto di partenza l'implementazione software in C di Ascon, quello di generare un flusso HLS valido.



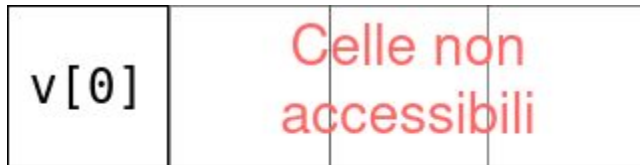
Svolgimento del Progetto



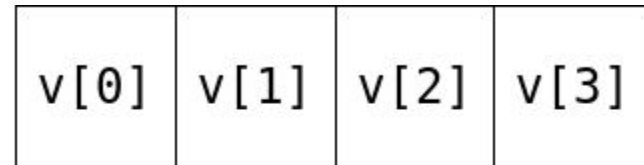
Sintesi

Per poter sintetizzare il codice C, a questo devono essere applicate alcune modifiche, al fine di rispettare le limitazioni intrinseche della sintesi hardware:

- Eliminare ogni dinamicità di memoria, esplicitando al momento della sintesi le dimensioni dei vettori di interfaccia.



Riferimento a 4 char tramite
`unsigned char* v;`



Riferimento a 4 char tramite
`unsigned char v[4];`



Sintesi

Per poter sintetizzare il codice C, a questo devono essere applicate alcune modifiche, al fine di rispettare le limitazioni intrinseche della sintesi hardware:

- Definire i puntatori dell'interfaccia specificando la direttiva `volatile`, per garantire consistenza in caso di letture consecutive.

Sintesi

Per poter sintetizzare il codice C, a questo devono essere applicate alcune modifiche, al fine di rispettare le limitazioni intrinseche della sintesi hardware:

- Ridefinizione dello stato per assicurare la correttezza dei cast e rendere il codice visivamente più compatto.

```
typedef struct {  
    unsigned long x[5];  
}state;
```

```
unsigned long state[5];
```



Tipi di ottimizzazione

L'ottimizzazione di diversi aspetti dell'implementazione finale si può ottenere usando due diversi approcci:

- Modifiche strutturali al codice.
- Applicazione di direttive di ottimizzazione di Vitis HLS in maniera mirata.

Modifiche al codice

Alcune rifiniture vengono applicate al codice al fine di migliorare diversi aspetti della sintesi risultante:

- Ridurre la dimensione delle variabili di interfaccia da 64 ad 8 bit per ridurre i pin utilizzati.

Modifiche al codice

Alcune rifiniture vengono applicate al codice al fine di migliorare diversi aspetti della sintesi risultante:

- Implementare vettori a 32 bit nell'interfaccia per velocizzare operazioni di lettura.

Modifica poi rimossa perché richiede un numero eccessivo di pin.

Modifiche al codice

Alcune rifiniture vengono applicate al codice al fine di migliorare diversi aspetti della sintesi risultante:

- Definire ricorsivamente le permutazioni per ridurre i componenti utilizzati, aumentando la latenza.

Questa modifica porta differenze solo a livello di sintesi, nessun cambiamento nell'implementazione finale.

Ottimizzazione

Direttive

Nome direttiva	Breve spiegazione
INLINE	Rimuove gli overhead causati da una chiamata a funzione.
PIPELINE	Scomponi le istruzioni inserendole in una pipeline, miglioramenti notevoli se usato all'interno dei loop.
DATAFLOW	Predisporre un'intera funzione all'inserimento in una pipeline scomponendola in sotto sezioni.

Analisi dei risultati: Dati raccolti

FPGA : xc7k160tfbv484-2

Lunghezza messaggi : 16 byte

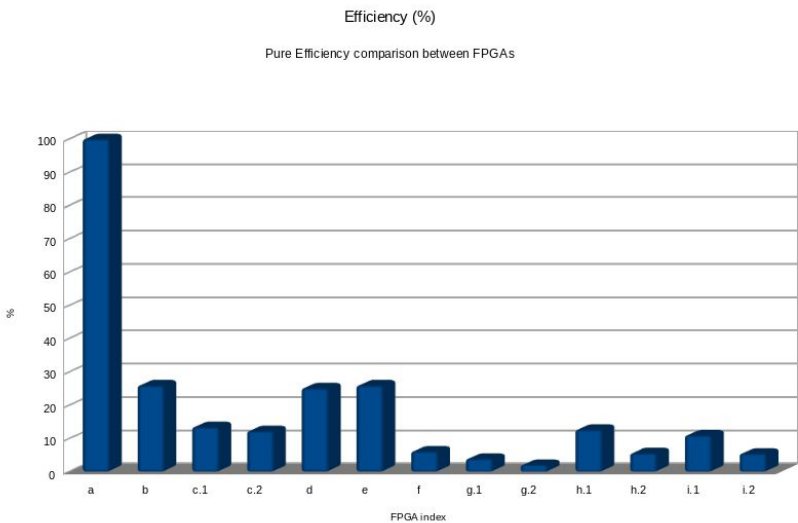
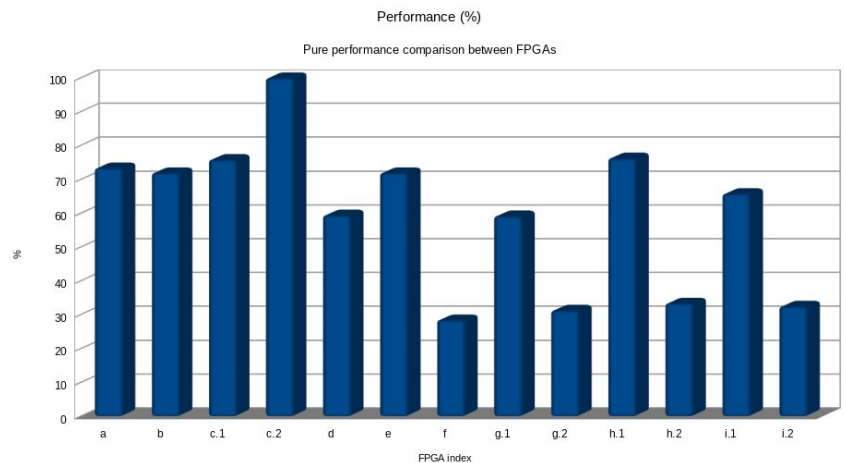
Solution index		C code alterations	Description	Vitis Directives	Latency (cycles)			Synthesis (clock in ns)				Implementation (clock in ns)			
					Avg	Max	Min	LUT	FF	Target	Reach	LUT	FF	Target	Reach
a		VHDL reference	VHDL implementation	-	100	-	-	-	-	-	-	1459	666	10	10
b		Standard version	Debloat of useless ports (baseline)	-	120	146	94	9213	3654	15	15	4590	3438	10	8.5
c	1	Standard version	Directives on permutations.h	INLINE RECURSIVE (P6, P8, P12)	129	160	99	42969	5259	10	3	11361	4984	10	7.5
	2				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 (P6, P8, P12)	157	196	120	50401	18665	10	3	23275	21029	10	7.9
	2				297	344	231	50823	22821	3	3	23375	21029	10	7.9
h	1	Standard version	Directives on permutations.h	PIPELINE (P6, P8, P12)	130	161	99	43052	6671	10	3	11057	6399	10	7.4
	2				297	350	231	43723	19057	3	3	11057	6399	3	7.4
i	1	Standard version	Directives on permutations.h	PIPELINE INLINE RECURSIVE (P6, P8, P12)	124	153	97	43081	6219	10	3	11407	5947	10	9
	2				251	303	199	43504	13903	3	3	11407	5947	3	9

- ❖ Valori di LUT, FF, Max reachable clock presi post implementazione con ottimizzazione Place & Route
- $\text{FPGA area} \rightarrow \text{LUT} + \text{FF}$
- $\text{Time of Computing (per word)} \rightarrow \text{Average Latency} * \text{Max reachable clock}$
- $\text{Performance} \rightarrow 1\text{s} / \text{Time of Computing}$
- $\text{Efficienza} \rightarrow \text{performance} / \text{FPGA area}$

Analisi dei risultati: Estrapolazione dati

	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

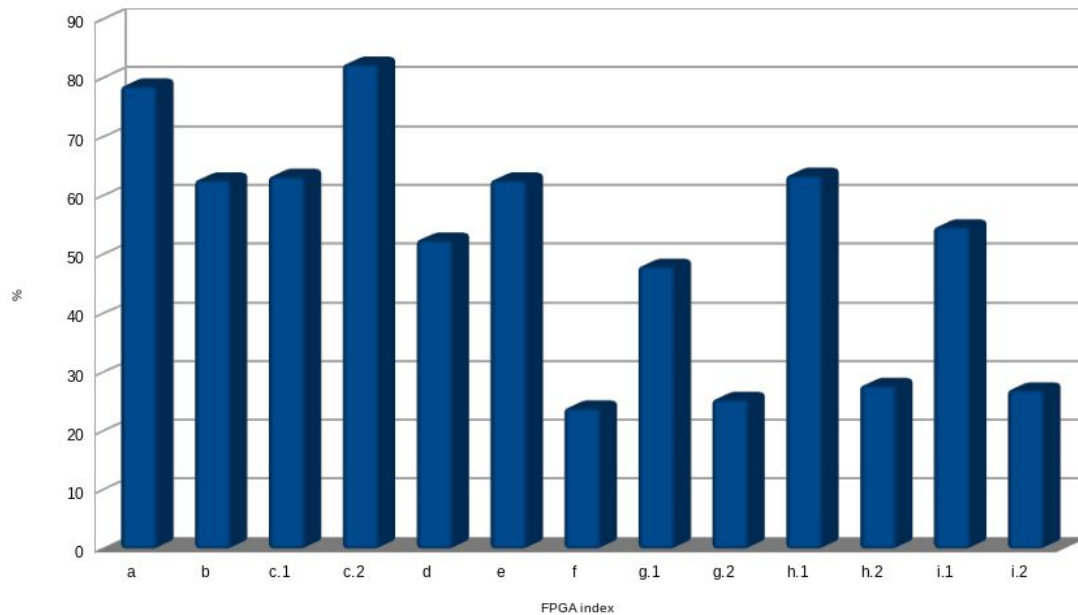
Analisi dei risultati: Performance e Efficienza



Analisi dei risultati: indexing per casi d'uso

Grafico orientato alle performance :

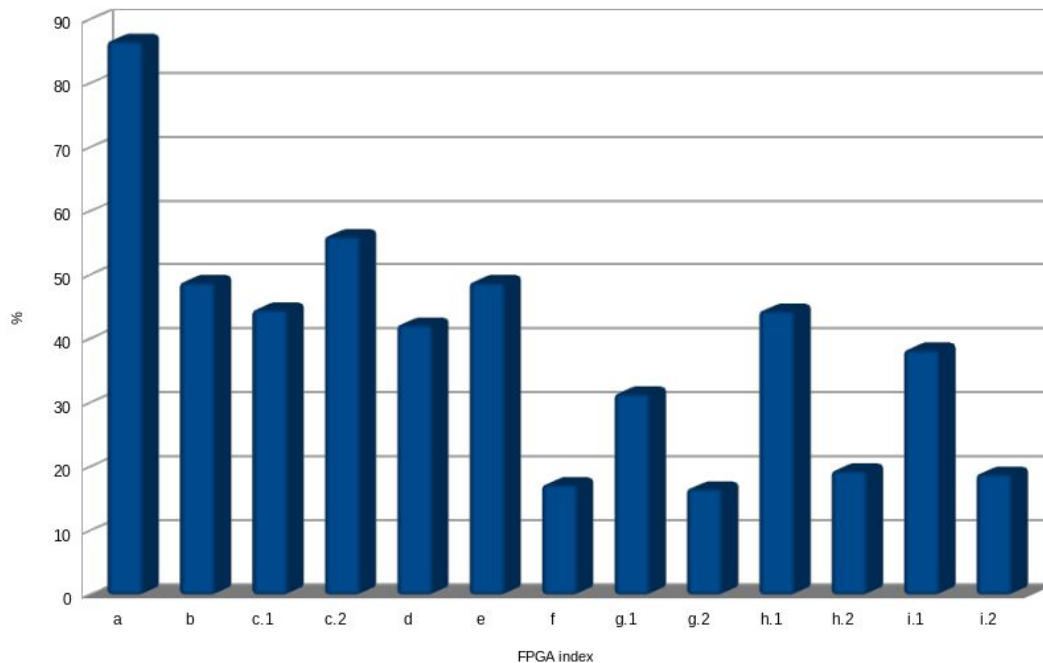
→ $8/10 * \text{performance} + 2/10 * \text{efficienza}$.



Analisi dei risultati: indexing per casi d'uso

Grafico bilanciato :

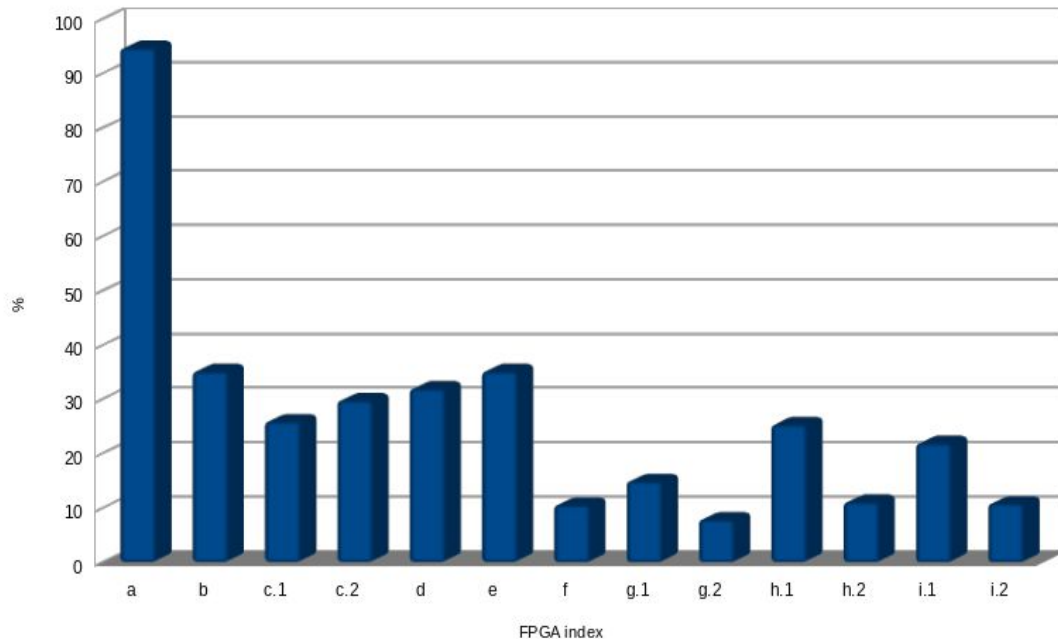
→ $5/10 * \text{performance} + 5/10 * \text{efficienza}$.



Analisi dei risultati: indexing per casi d'uso

Grafico orientato all'efficienza:

→ $2/10 * \text{performace} + 8/10 * \text{efficienza}$.



Analisi dei risultati: prima analisi sulle implementazioni

- ❖ implementazione VHDL ottima :
 - seconda in performance
 - prima in efficienza (75 % di vantaggio su tutte le altre)
- ❖ buona prima implementazione di Vitis HLS :
 - prestazioni vicine a VHDL campione
 - migliore implementazione in termini di efficienza

Analisi dei risultati: confronto con CPU

CPU utilizzate :

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

Analisi dei risultati: confronto con CPU

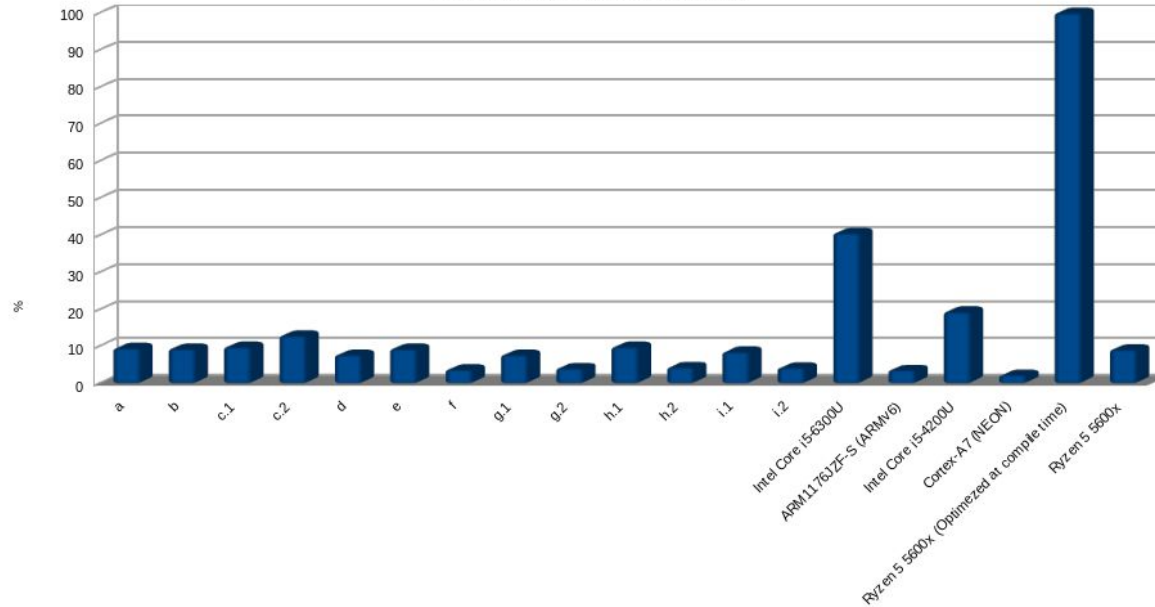


Grafico confronto tra CPU e FPGA










Analisi dei risultati: considerazioni finali e confronto con CPU

- ❖ Le CPU dominano lato performance
 - analisi sui consumi necessari per stabilire eventuale trade-off
- ❖ Le CPU mobile sono molto più lente
 - possibilità di accoppiare processore più vecchio insieme ad un integrato rispetto ad un processore più potente, analisi dei costi necessaria

Esempi di utilizzo :

CCTV camera, componenti di domotica.

Conclusioni

Tipo implementazione	Software C	Flusso HLS	Hardware VHDL
Software Dev ready			
Ad Hoc Implementation			
FPGA ready			



Grazie!