Implementazione di una calcolatrice binaria in VHDL per FPGA Basys 3

*Abstract*—Questo articolo presenta l'implementazione in VHDL di una calcolatrice che utilizza l'aritmetica binaria su un FPGA Basys 3. La calcolatrice sviluppata è in grado di eseguire operazioni di somma, sottrazione, moltiplicazione e divisione, riflettendo le operazioni fondamentali comuni nei moderni calcolatori digitali. Si interagisce con la calcolatrice implementata tramite gli switch, i pulsanti incorporati nella scheda FPGA e due cavetti, mentre si possono leggere le cifre digitate, così come i risultati dei calcoli, tramite i display a sette segmenti, anch’essi compresi nella scheda Basys 3. L'obiettivo principale di questa ricerca è comprendere in profondità il funzionamento di un dispositivo di calcolo così essenziale, che tutti hanno utilizzato almeno una volta nella vita consapevolmente e assai più volte inconsapevolmente all’interno dei sistemi elettronici contemporanei. Ad oggi le calcolatrici binarie sono presenti in tutti i moderni sistemi di calcolo e sono estremamente ottimizzate, il nostro scopo non è quindi ottimizzare un sistema già ampiamente affermato, ma bensì svelare i meccanismi al suo interno. L'articolo illustra il processo di progettazione e l'implementazione pratica di questa calcolatrice binaria su FPGA Basys 3, evidenziando le sfide affrontate durante lo sviluppo e le soluzioni adottate di conseguenza.

Keywords—calcolatrice, aritmetica binaria, sommatore, sottrattore, moltiplicatore, divisore

# Introduzione

Il progetto di una calcolatrice binaria implementata su FPGA si inserisce nel contesto dell'ingegneria elettronica digitale, esplorando il fondamentale ruolo dei circuiti logici e dell'aritmetica binaria. Questo lavoro non solo si concentra sullo sviluppo di un dispositivo di base, ma evidenzia l'importanza cruciale di tali concetti nell'ecosistema più ampio dei calcolatori moderni, inclusi ambiti come l'Internet of Things (IoT) e l'evoluzione delle tecnologie informatiche. Non si limita quindi alla realizzazione di un dispositivo tascabile funzionante per risolvere le operazioni elementari (somma, differenza e prodotto), utilizzabile alla scuola primaria, ma si propone anche come possibile versione di implementazione di una ALU semplificata rispetto alle odierne concorrenti.

Il nostro progetto è difatti un’implementazione di parte delle moderne Arithmetic Logic Unit (ALU), o un’espansione delle prime ALU, che eseguivano le sole operazioni aritmetiche di somma e differenza. Con la diminuzione delle dimensioni dei transistor, accelerata ulteriormente dall’invenzione dei circuiti integrati, come predetto dalla legge di Moore, fu possibile implementare ALU sempre più grandi e complesse sui microprocessori. Ad oggi le ALU possono eseguire: operazioni aritmetiche, quali somma, sottrazione, moltiplicazione, divisione, incremento e decremento; operazioni logiche, come AND, OR, NOT, XOR, shift a sinistra o a destra e rotazioni; confronti di maggioranza, minoranza o uguaglianza; operazioni specializzate, come ad esempio operazioni di conteggio ed operazioni di mascheramento. Le prestazioni delle ALU dipendono da diversi fattori, tra cui la lunghezza dei bit su cui operano, la frequenza del clock del processore e l’architettura del processore su cui essa è integrata. In generale, le moderne ALU permettono, grazie all’implementazione di registri a scorrimento e dell’operazione di moltiplicazione binaria integrata, di eseguire in un solo ciclo di clock operazioni che avrebbero richiesto molti più cicli con le vecchie architetture, raggiungendo il miliardo di operazioni eseguite al secondo[1]. Infine, recenti ricerche hanno dimostrato la possibilità di implementare ALU biocompatibili, che rivoluzioneranno i sistemi embedded che abbiamo conosciuti fino ad oggi[2].

Il contributo apportato dal nostro progetto rispetto allo stato dell’arte ovviamente non migliora le incredibili prestazioni delle ALU commerciali, avendo utilizzato un dispositivo non creato appositamente per svolgere queste operazioni. Tuttavia, l’innovazione portata da questo progetto si basa sugli algoritmi che implementano le operazioni tramite l’aritmetica binaria, difatti:

* Sono stati creati un sommatore e un sottrattore che implementano le operazioni in binario, tramite l’utilizzo di sotto blocchi più semplici;
* È stato implementato un moltiplicatore che sfrutta la logica della moltiplicazione binaria in colonna, creando un algoritmo innovativo;
* È stata trovata un’implementazione completamente nuova di un divisore che si basa sulla logica della divisione binaria in colonna;
* Inoltre, è stato possibile comunicare con la calcolatrice tramite un protocollo di scrittura dei dati in input creato appositamente da noi;
* Infine, sono stati rappresentati i numeri inseriti e i risultati ottenuti tramite i display a sette segmenti integrati nella scheda FPGA Basys 3.

Il resto dell’articolo è organizzato come segue. Nella Sezione II approfondiremo gli articoli correlati al nostro lavoro. Spiegheremo poi il setup di misura e la tecnologia utilizzata nella Sezione III. Descriveremo poi l’hardware nella Sezione IV e il software sviluppato nel progetto nella Sezione V. Nella Sezione VI descriveremo i risultati sperimentali. Infine, nella Sezione VII discuteremo le conclusioni dell’articolo.

# Articoli Correlati

La ricerca nell'implementazione di calcolatrici binarie attraverso Verilog HDL e FPGA ha conosciuto una notevole crescita, come dimostrato da una serie di studi recenti.

Il primo di questi articoli che citiamo focalizza l'attenzione sul design di una ALU a 4 bit implementata su Basys 3, che permette di diminuire il consumo di potenza e mantenere la velocità. Questa offre funzionalità come somma, sottrazione, moltiplicazione e varie operazioni logiche, ma non implementa la divisione[3].

Ulteriori ricerche si concentrano sull'utilizzo della logica reversibile nell'architettura di una ALU. Questo approccio innovativo, basato sulla reversibilità logica, promette di migliorare l'efficienza energetica, mantenendo la velocità di esecuzione delle operazioni, introducendo nuovi paradigmi nella progettazione di circuiti di calcolo[4].

Un altro studio si concentra sull'ottimizzazione dell'uso del segnale di clock nei circuiti digitali, particolarmente nella riduzione della dissipazione di potenza dinamica. Esso approfondisce la tecnica di clock gating, mirando a disabilitare i moduli inattivi in determinati periodi operativi per limitare la dissipazione di potenza. Attraverso simulazioni sull'hardware FPGA, analizza diverse tecniche di clock gating basate su VHDL per un'ALU a 16 bit con 13 istruzioni, esaminando l'impatto sulla dissipazione di potenza su frequenze operative da 100 MHz a 500 MHz[5].

Un ulteriore articolo presenta una ALU per il calcolo in virgola mobile a precisione singola, ottimizzata per operazioni di addizione, sottrazione, moltiplicazione e divisione. La struttura proposta è basata sull'implementazione di algoritmi specifici: un sommatore carry look-ahead per l'addizione e la sottrazione, un codificatore di Booth modificato per la moltiplicazione e l'algoritmo di Goldschmidt per la divisione. La soluzione implementata permette di aumentare la velocità e ridurre il consumo di potenza rispetto alle ALU comuni[6].

L’ultimo articolo che citiamo propone il design di una calcolatrice a 8 bit utilizzando Verilog HDL. Quest’ultimo si concentra sulla gestione di operazioni matematiche come addizione, sottrazione, moltiplicazione, divisione, funzioni di elevamento a potenza, radice quadrata e fattoriale su numeri a otto cifre. Sebbene sia efficace, la calcolatrice implementata genera solo risultati interi a otto cifre e potrebbe quindi necessitare di miglioramenti futuri per supportare precisioni decimali e operazioni matematiche più complesse[7].

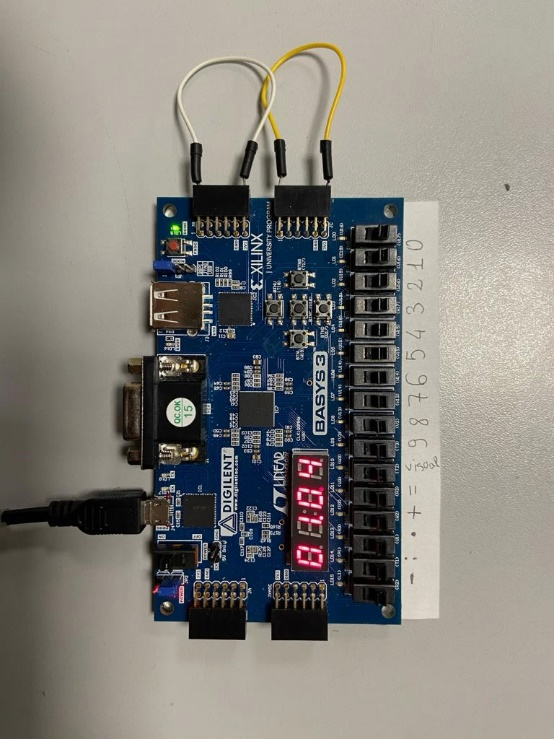
Inizio modulo

Questa varietà di approcci e contributi evidenzia l'importanza del VHDL e delle FPGA nella creazione di calcolatrici binarie efficienti e indica chiaramente la direzione verso una maggiore integrazione di funzionalità avanzate per affrontare sfide matematiche, e non solo, sempre più intricate.

# Setup di Misura

Nel nostro progetto abbiamo utilizzato una scheda FPGA Basys 3 con AMD Artix 7[8]. Questa è una scheda di sviluppo progettata per essere usata esclusivamente con Vivado Design Suite, il software che abbiamo utilizzato per scrivere questo progetto in VHDL.

Per inserire le cifre degli operandi e le operazioni scelte sono stati utilizzati tutti gli switch già presenti sulla scheda Basys 3, così come è stato utilizzato un pulsante integrato nella scheda per eseguire l’operazione di reset. Infine, abbiamo utilizzato due cavetti per pilotare: la visualizzazione sui display a sette segmenti dell’input o dell’output; la scelta tra visualizzazione del quoziente o del resto, nel caso in cui si scelga la divisione. Il setup di misura per il nostro progetto è quindi formato quasi solo dalla scheda FPGA Basys 3, che abbiamo affiancato da una comoda legenda utile agli utenti, come è visibile in Fig. 1.



**Fig. 1.** Setup di misura con FPGA Basys 3 e legenda che spiega l’utilità degli switch.

# Hardware Implementato

Il progetto di questa calcolatrice può essere scomposto in vari sotto blocchi hardware, come è possibile osservare in Fig. 2.

Il primo blocco che incontriamo è l’*input\_keyboard*, grazie al quale andiamo a leggere e visualizzare sui display a sette segmenti gli operandi che andremo ad utilizzare. Da questo blocco preleviamo i due operandi e l’operazione scelta, andando ad aggiungere i bit necessari per uniformarli allo standard richiesto dai blocchi di ciascuna operazione. A questo punto troviamo i quattro blocchi delle operazioni, ovvero un *sommatore*, un *moltiplicatore*, un *sottrattore* e un *divisore*. Infine, selezioniamo il risultato desiderato in base all’operazione scelta dall’utente tramite un multiplexer e l’andiamo a rappresentare sui display a sette segmenti tramite il modulo *decoder\_display*.

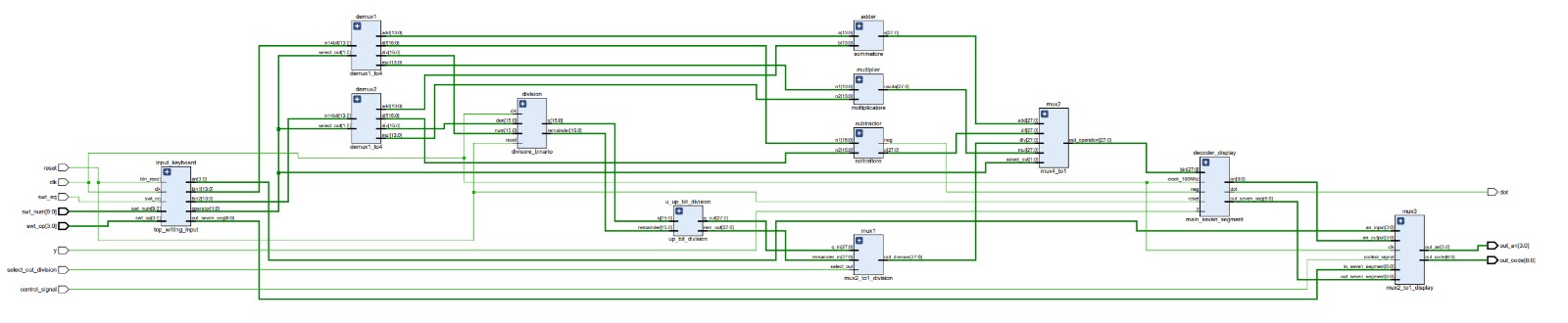
# Software Sviluppato

Il software che abbiamo implementato si divide in vari macro blocchi: un lettore di input, un visualizzatore sui display a sette segmenti e un’unità di calcolo binario.

## Lettore di input

Il primo blocco che possiamo individuare è quello che permette di leggere i numeri da prendere come operandi e l’operazione da eseguire.

Il modulo fondamentale che svolge questa operazione è *read\_input*, che inizialmente salva il primo numero, fino a che vengono immesse quattro cifre decimali o anche prima, se viene selezionata un’operazione. Dopo aver selezionato quale operando si intende usare, il modulo inizia a salvare il secondo numero, fino a un massimo di quattro cifre decimali o fino all’input dell’uguale. Tutte le cifre, così come le operazioni e l’uguale, vanno selezionati alzando e poi abbassando uno switch. Nel momento in cui si riabbassano



**Fig. 2.** Blocchi hardware del progetto realizzato rappresentati tramite lo schema RTL.

gli switch, questi potranno essere nuovamente utilizzati per riconoscere l’input successivo. Questo è possibile grazie a delle variabili di controllo che impediscono di leggere più volte al programma lo stesso input selezionato in realtà dall’utente una sola volta. L’operazione di reset, che permette di tornare al momento in cui iniziano ad essere salvati gli input, può invece essere richiamata tramite l’utilizzo del pulsante BTNL integrato nella scheda Basys 3. Il modulo read\_input viene poi richiamato da *input\_on\_display*, che converte le cifre salvate dalla codifica iniziale alla codifica binaria e le fa visualizzare sui display a sette segmenti, previo collegamento del filo giallo a 3.3V.

## Visualizzatore sui display a sette segmenti

Il secondo blocco individuabile è quello che permette di leggere le cifre digitate e gli output delle elaborazioni sui display a sette segmenti.

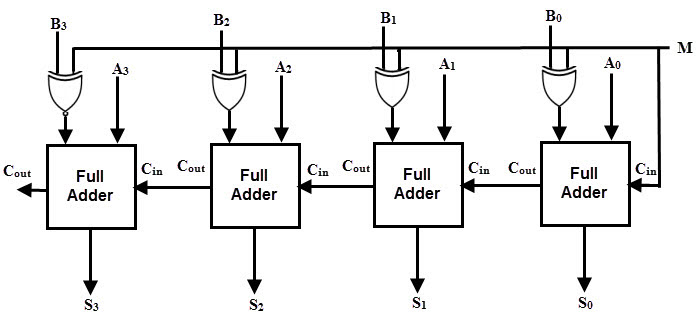
Per visualizzare i numeri sui 4 display a sette segmenti messi a disposizione dalla scheda Basys 3 si è implementato un codificatore. Per apprendere al meglio l’architettura sviluppata è utile tenere in considerazione che la frequenza operativa (100MHz) della scheda Basys 3 è diversa dalla frequenza con la quale i display a 7 segmenti vengono aggiornati (tra i 62.5 e i 1000 Hz). Di conseguenza il primo blocco sviluppato, *display\_ctrl*, è stato quello che permette la diminuzione della frequenza di clock. Esso è costituito da due ingressi, ovvero la frequenza di clock del dispositivo Basys 3 e un segnale di reset. In uscita troviamo un segnale di controllo a 1kHz, grazie al quale è possibile scrivere sui singoli display ogni 2,5 ms. C’è poi un blocco che permette di passare dalla rappresentazione binaria alle singole cifre decimali, il *bin\_to\_7seg*. Il terzo blocco, *seven\_segment\_decoder*, è invece il codificatore vero e proprio, che prende in ingresso dei vettori di bit da rappresentare e in uscita restituisce una codifica per ogni cifra del numero intero. L’ultimo blocco, *main\_seven\_segment,* prende i due segnali di uscita dei due precedenti blocchi e li combina opportunamente, in modo da fornire una corretta scrittura del numero da visualizzare attraverso dei segnali che pilotano la codifica della singola cifra e la sua corretta scrittura nei display. Ricordiamo che nel caso in cui il filo giallo sia collegato a 3.3V si può leggere l’input, mentre collegandolo a terra si può visualizzare l’output. Infine, è utile notare che è stato aggiunto un segnale che permette di visualizzare il risultato visionando le 4 cifre più o meno significative, in base alla posizione dello switch in posizione 0, in quanto, a differenza dell’input, questo può raggiungere anche le 8 cifre decimali). Grazie invece al collegamento del filo bianco è possibile selezionare la visualizzazione del quoziente o del resto nel caso in cui si sia selezionata come operazione la divisione.

## Unità di calcolo binaria

Infine, il terzo e più importante macro blocco è quello che racchiude l’implementazione vera e propria delle operazioni di calcolo in aritmetica binaria. Quest’ultimo si compone di un sommatore, un sottrattore, un moltiplicatore e un divisore.

# *Sommatore*

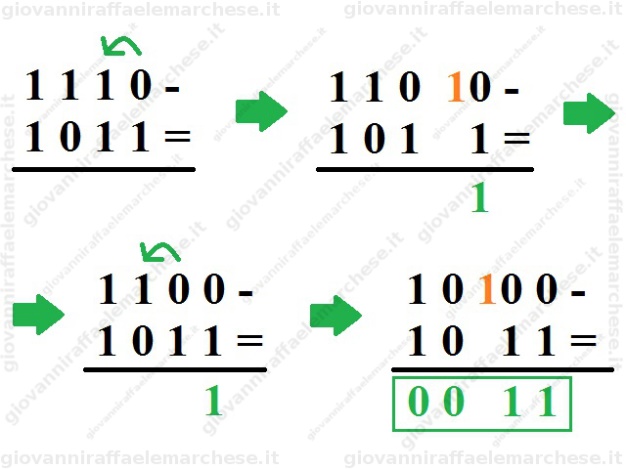
L’architettura del sommatore che abbiamo implementato si compone di due blocchi, che ci permettono di prendere in input due numeri di 14 bit e restituire una somma di 15 bit, utilizzando la stessa logica della somma in colonna tipica dell’aritmetica binaria. Il primo blocco, quello basilare, è chiamato *full\_adder* e prende in input i due bit nella i-esima posizione in colonna dei due numeri da sommare e un eventuale resto che gli va aggiunto. In output restituisce l’i-esimo bit del risultato e un eventuale resto da passare in input al full\_adder successivo. Il blocco superiore è invece il *sommatore*, che si compone di 15 full\_adder che collega in cascata, come è verificabile nella Fig. 3. In input al primo non ho resto e dall’ultimo non avrò alcun riporto.



**Fig. 3.** Sommatore composto da quattro full\_adder in cascata.

# *Sottrattore in modulo*

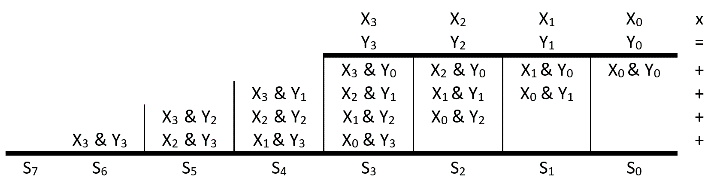
Il sottrattore, analogamente al sommatore, si compone di vari blocchi, che prendono in input due numeri a 16 bit e restituiscono numeri a 16 bit. Anche qui utilizziamo la stessa logica dell’aritmetica binaria, come si può osservare nella Fig. 4. Nel primo blocco, il *full\_subtractor*, abbiamo in input l’i-esimo bit di entrambi i numeri e un eventuale prestito che si è fatto al bit immediatamente meno significativo. In output si hanno un eventuale prestito richiesto al bit subito più significativo e l’i-esimo bit del risultato. Il blocco principale è il *sottrattore\_positivo*, in cui vengono definiti 16 full\_subtractor in cascata, seguendo la stessa logica del sommatore. Infine, abbiamo il blocco del *sottrattore*, in cui viene chiamato il sottrattore\_positivo dopo aver fatto un controllo: se il sottraendo è maggiore del minuendo si invertono i numeri nel richiamare il sottrattore\_positivo e si pone a “1” un flag *neg* che indica che il risultato è negativo. In ogni caso sul display verrà rappresentato solo il modulo del risultato.



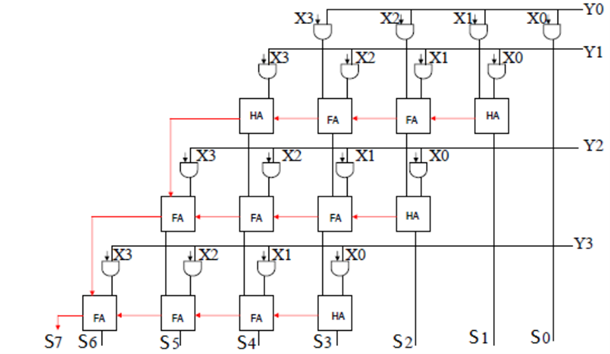
**Fig. 4.** Sottrazione binaria in colonna.

# *Moltiplicatore*

Il principio logico che sta alla base della moltiplicazione implementata in questo progetto viene rappresentato nella Fig. 5, in cui troviamo un esempio di una moltiplicazione di due numeri binari a 4 cifre. Questa operazione è composta da due fasi. La prima fase consiste nell’eseguire i vari prodotti parziali tra i due numeri, andando a generare le 4 righe traslate secondo il peso del bit moltiplicato. Nella seconda fase si eseguono poi le somme lungo le 8 colonne per arrivare infine al risultato rappresentato dall’ultimo vettore S. Sfruttando l’algoritmo precedente si è poi implementato il *moltiplicatore*, attraverso l’utilizzo del linguaggio VHDL. Inizialmente è stato implementato a 4 bit, così da testare più facilmente la sua affidabilità e il corretto funzionamento, dopodiché è stato esteso a 14 bit. Al suo interno è possibile trovare un’architettura facilmente modellabile a numeri con più o meno bit, grazie alla sua regolarità. Di fatto costituita da delle semplici porte AND che eseguono i prodotti parziali tra i vari bit e da dei sommatori *full\_adder.* Quest’ultimi sono stati utilizzati per eseguire le varie somme parziali secondo lo schema di funzionamento, visionabile per la moltiplicazione a 4 bit nella Fig. 6.



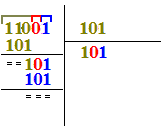
**Fig. 5.** Logica della moltiplicazione binaria in colonna.



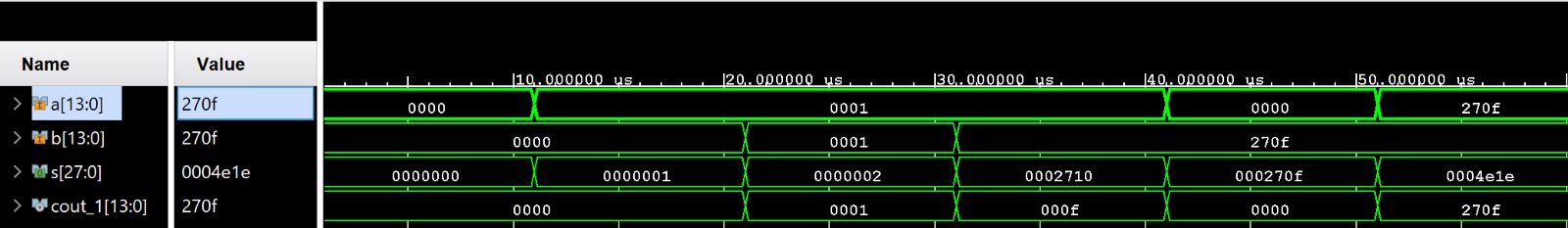
**Fig. 6.** Circuito logico della moltiplicazione binaria a 4 bit.

# *Divisore*

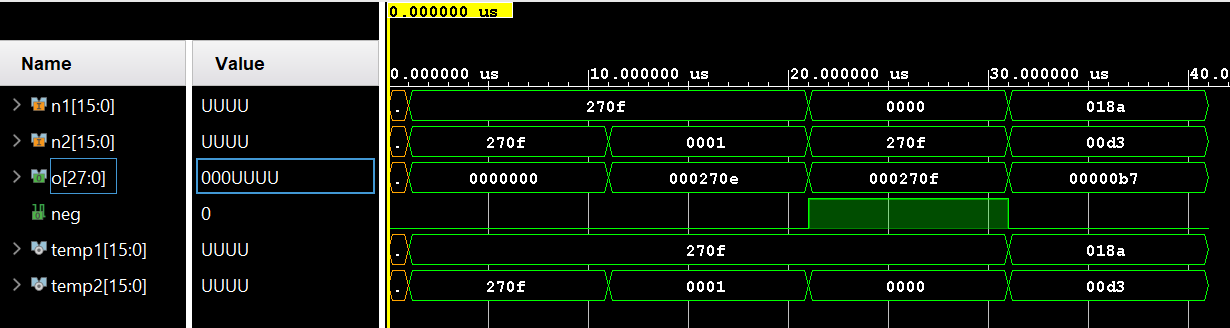
Il *divisore*, a differenza della somma e della sottrazione, si compone di un unico blocco. L’algoritmo che abbiamo implementato si basa sull’aritmetica binaria, ed in particolare su come si svolgono le divisioni in colonna[10]. Quest’ultimo prende in input un divisore e un dividendo da 16 bit e restituisce un quoziente da 16 bit e un resto, anch’esso da massimo 16 bit. Per comprendere appieno come funziona l’algoritmo, lo percorreremo con l’aiuto della Fig. 7. Inizialmente l’algoritmo conta quante sono effettivamente le cifre significative del dividendo e del divisore e nel caso in cui il primo abbia un numero di cifre maggiori si procede con la divisione, altrimenti il quoziente sarà zero e il dividendo coinciderà con il resto. Nel procedere con la divisione, posto che il divisore abbia lunghezza *n* bit, si selezionano le *n* cifre più significative del dividendo, in Fig. 7 quelle verdi, e si salvano in una variabile *temp\_num*. Se temp\_num è maggiore del dividendo, allora si pone il bit più significativo del risultato a 1, altrimenti lo si pone a 0. Se temp\_num era maggiore, lo si riassegna con la differenza tra temp\_num e il divisore. A questo punto si aggiunge a destra di temp\_num il primo bit più significativo del dividendo non ancora incluso in temp\_num, in Fig. 7, stiamo parlando dello 0 rosso. Si verifica adesso se temp\_num è maggiore del divisore e si pone di conseguenza il successivo bit del quoziente a 1 o 0, come già fatto prima. Si continua a ciclare così finché i bit del dividendo sono stati tutti inclusi in temp\_num. A questo punto l’ultima sottrazione tra temp\_num e il divisore corrisponde al resto, mentre il quoziente, traslato a destra del numero opportuno di bit, indica il risultato.



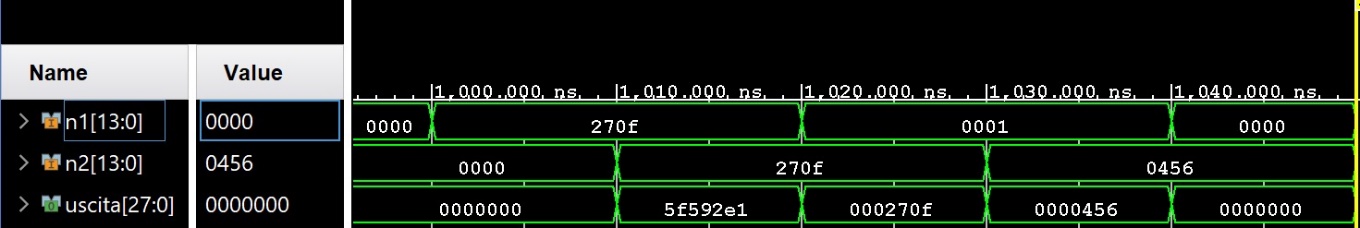
**Fig. 7.** Divisione binaria in colonna.



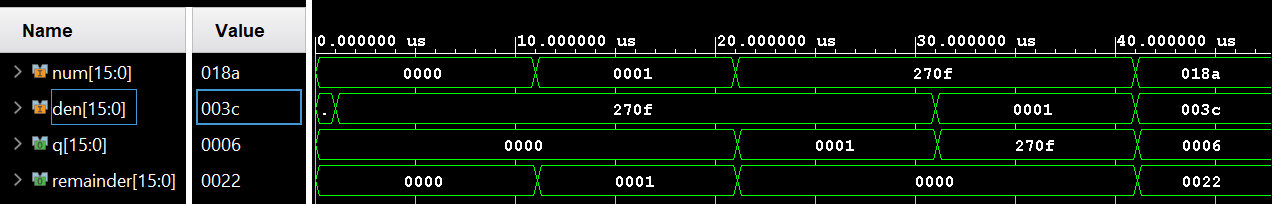
**Fig. 8.** Simulazione comportamentale del blocco sommatore prendendo come input dei valori critici.



**Fig. 9.** Simulazione comportamentale del blocco sottrattore prendendo come input dei valori critici.



**Fig. 10.** Simulazione comportamentale del blocco moltiplicatore prendendo come input dei valori critici.



**Fig. 11.** Simulazione comportamentale del blocco divisore prendendo come input dei valori critici.

# Risultati Sperimentali

I risultati forniti dal nostro progetto sono stati ampliamente verificati già in corso d’opera. Difatti, abbiamo verificato tramite simulazioni comportamentali e funzionali post-sintesi che i risultati ottenuti tramite ogni operazione fossero corretti. Abbiamo fatto questo variando gli input, utilizzando sia valori casuali, che valori critici, come ad esempio i numeri massimi rappresentabili nella somma e nella moltiplicazione, o utilizzando un sottraendo maggiore nel minuendo nella sottrazione, così come usando come dividendo lo zero, o dividendo dei numeri per dei divisori maggiori. Degli esempi si possono trovare nelle Fig. 8, 9, 10 e 11.

Dopo aver implementato anche i protocolli di comunicazione per leggere e visualizzare gli input e gli output con i display a sette segmenti, abbiamo verificato manualmente che gli input corrispondessero alle cifre visualizzate e che, anche mettendo tutto insieme le operazioni restituissero come output i valori corretti, anche nei casi critici.

# Conclusioni

In conclusione, abbiamo realizzato una calcolatrice programmandola in VHDL su un FPGA Basys 3, con l’ausilio dei pulsanti e degli switch inclusi nella scheda e di due fili. Siamo riusciti ad implementare tutte le operazioni numeriche fondamentali (somma, sottrazione, moltiplicazione, divisione), partendo da numeri a 16 bit, tramite l’aritmetica binaria, sia riutilizzando degli algoritmi ben noti, sia ideandone altri completamente nuovi e soprattutto funzionanti.

Ripercorrendo i passi progettuali, abbiamo riconosciuto i valori inseriti tramite gli switch (fino a un massimo di quattro cifre decimali) e mostrati sui display a sette segmenti incorporati sulla scheda. Abbiamo poi selezionato l’operazione da attuare tramite gli switch e riconosciuto anche il secondo numero. A questo punto abbiamo effettuato il calcolo tramite gli algoritmi da noi implementati all’interno dell’FPGA e abbiamo mostrato il risultato sui display a sette segmenti, permettendo di cambiare la visualizzazione tramite l’attivazione di uno switch.

Abbiamo verificato tramite numerosi test che il nostro sistema è correttamente funzionante per tutte le operazioni sopra citate e restituisce i valori esatti.

In futuro si potrebbe pensare di aumentare le operazioni implementate, includendo ad esempio l’elevamento a potenza, o anche implementando tutte le altre operazioni svolte dalle moderne ALU.

##### Riferimenti

[1] «Arithmetic logic unit», *Wikipedia*. 27 novembre 2023. Consultato: 8 dicembre 2023. [Online]. Disponibile su: https://en.wikipedia.org/w/index.php?title=Arithmetic\_logic\_unit&oldid=1187157201

[2] C.-L. Lin, T.-Y. Kuo, e W.-X. Li, «Synthesis of control unit for future biocomputer», *J. Biol. Eng.*, vol. 12, p. 14, ago. 2018, doi: 10.1186/s13036-018-0109-4.

[3] A. K. Panigrahi, S. Patra, M. Agrawal, e S. Satapathy, «Design and Implementation of a high speed 4bit ALU using BASYS3 FPGA Board», in *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*, Vellore, India: IEEE, mar. 2019, pp. 1–6. doi: 10.1109/i-PACT44901.2019.8960099.

[4] S. M. Swamynathan e V. Banumathi, «Design and analysis of FPGA based 32 bit ALU using reversible gates», in *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)*, Karur: IEEE, apr. 2017, pp. 1–4. doi: 10.1109/ICEICE.2017.8191959.

[5] N. Khanna e D. K. Mishra, «Clock Gated 16-Bits ALU Design & Implementation on FPGA», in *2018 4th International Conference for Convergence in Technology (I2CT)*, ott. 2018, pp. 1–5. doi: 10.1109/I2CT42659.2018.9057910.

[6] A. Y. N J e A. V R, «FPGA Implementation of a High Speed Efficient Single Precision Floating Point ALU», in *2023 International Conference on Control, Communication and Computing (ICCC)*, Thiruvananthapuram, India: IEEE, mag. 2023, pp. 1–5. doi: 10.1109/ICCC57789.2023.10165441.

[7] S. binti Suhaili, K. J. anak Kumar, N. Julai, M. H. Husin, M. F. M. Sabri, e A. Lit, «Implementation of Verilog HDL in Calculator Design with FPGA Simulation», in *2020 13th International UNIMAS Engineering Conference (EnCon)*, ott. 2020, pp. 1–6. doi: 10.1109/EnCon51501.2020.9299337.

[8] «Basys 3 Reference Manual - Digilent Reference». Consultato: 8 dicembre 2023. [Online]. Disponibile su: https://digilent.com/reference/programmable-logic/Basys 3/reference-manual

[9] «ECO.12150.06 - Tastierino, ECO, 3 x 4, matrice, PC (policarbonato), 20 mA, 24 V». Consultato: 8 dicembre 2023. [Online]. Disponibile su: https://it.farnell.com/eoz/eco-12150-06/tastierino-3x4-matrix-0-02a-24v/dp/1130805

[10] G. Carichino, «Operazioni con i numeri binari», YouMath. Consultato: 8 dicembre 2023. [Online]. Disponibile su: https://www.youmath.it/lezioni/algebra-elementare/lezioni-di-algebra-e-aritmetica-per-scuole-medie/1817-operazioni-tra-numeri-binari.html