0.1 Architettura

Il Carry Save è un addizionatore in grado di effettuare la somma tra 3 stringhe di bit di lunghezza n. In particolare, tale macchina è formata da:

1. blocchi *carry save*, ossia dei full adder che si occupano di sommare 3 bit dello stesso peso delle tre stringhe;

2. blocchi *full adder* che sommano al risultato del CSL_i il riporto uscente dal blocco CSL_{i-1} e dal full adder precedente.

L'architettura della macchina è osservabile in fig.1. In particolare, si noti come la seconda catena di full adder vada a formare un *ripple carry adder*. Per semplificarne l'implementazione possiamo dunque sostituire a tale schema quello in fig.2, formato da un *carry save logic* (serie di carry save block) e un RCA.

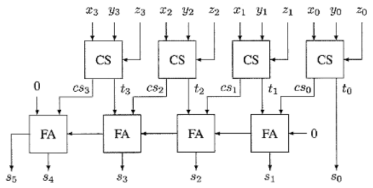

Figure 1: Architettura del Carry Save.

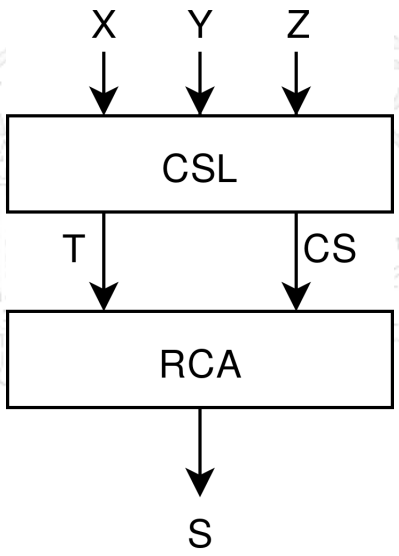

Figure 2: Architettura del Carry Save con CSB e RCA.

0.2 Implementazione

0.2.1 Carry Save Logic

Per l'implementazione della Carry Save Logic, ossia della catena di Carry Save Block, si è utilizzata una descrizione di tipo structural. La generazione dei singoli blocchi full adder è stata realizzata come segue:

```
T <= T_temp;
CS <= CS_temp;
carry_save_blocks: for i in 0 to (width-1) generate
  carry_save_block: full_adder port map (
    x => X(i),
    y => Y(i),
    c_in => Z(i),
    s => T_temp(i),
    c_out => CS_temp(i)
  );
end generate carry_saves;
```

Codice Componente 1: Generazione dei Carry Save Block.

L'implementazione completa è consultabile qui: carry_save_logic.vhd

0.2.2 Carry Save

Per realizzare il Carry Save, è stata una descrizione di tipo structural con un parametro generico *width* per definire la il numero di bit dei tre operandi. Data *width*, la somma sarà espressa su *width*+2 bit. Oltre al componente Carry Save Logic già descritto, si è fatto uso di un Ripple Carry Adder per sommare CS con i valori di T (shiftati a destra) e ottenere le cifre più significative del risultato S:

```
S(0) <= T(0);
A <= '0' & T(width-1 downto 1);
RCA: ripple_carry_adder GENERIC MAP (
  width => width )
  X => A,
  Y => CS,
  c_in => '0',
  S => S(width downto 1),
  c_out => S(width+1)
);
```

Codice Componente 2: Utilizzo del Ripple Carry Adder nel Carry Save.

L'implementazione completa del Carry Save è consultabile qui: carry_save.vhd

0.3 Simulazione e sintesi

0.3.1 Simulazione

Per tale componente è stata effettuata una simulazione behavioural, durante la quale sono stati fatti variare i tre operandi da sommare. I risultati ottenuti sono osservabili in fig.3.

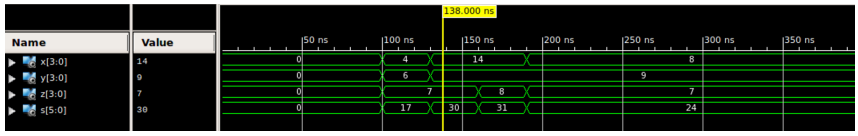

Figure 3: Simulazione behavioural del Carry Save.

0.3.2 Sintesi

Si è proceduto infine alla sintesi del componente utilizzando diversi valori di lunghezza delle stringhe tramite manipolazione del parametro generico *width*. Come per gli addizionatori precedenti, sono stati ottenuti il *numero di slices* e *minimum period* in funzione del numero di bit per valutare le prestazioni di tale macchina. I risultati sono riportati in fig.6.

Come ci si aspettava, il circuito occupa molta più area rispetto alle sue controparti con solo due operandi. Tuttavia, i tempi di elaborazione risultano molto simili, grazie alle capacità di ottimizzazione del tool di sintesi.

0.4 Approfondimento: confronto con RCA a tre operandi

Per poter effettuare una valutazione delle prestazioni del Carry Save, si è deciso di realizzare un altro componente per la somma di tre operandi. In particolare, tale componente è stato realizzato mediante l'utilizzo di due Ripple Carry Adder in cascata. L'architettura del componente è visibile in fig.5.

L'implementazione dell'RCA a tre operandi, effettuata tramite descrizione structural, è consultabile qui: rca_tre_operandi.vhd

[group style=group size=2 by 1,horizontal sep=2cm, yticklabel style=font=, xticklabel style=font=] [legend style=font= , anchor=north, at=(0.70,0.16), xmin=0,xmax=128, ymin = 0, ymax = 1250, grid=major, width=0.45 height=,xlabel= Numero di bit, ylabel=Numero di slice] coordinates (0,0) (4, 29) (8, 68) (16, 143) (32, 305) (64,605) (128, 1228) ; [legend style=anchor=north, at=(0.50,0.95), xmin=0,xmax=128, ymin = 0, ymax = 5, grid=major, width=0.45height=, xlabel= Numero di bit, ylabel=Minimum period (ns)] coordinates (0,0) (4, 1.870) (8, 2.404) (16, 2.974) (32, 3.501) (64, 4.307) (128, 4.747) ;

Figure 4: Grafici dei risultati ottenuti post-sintesi in funzione del numero di bit.

Si è poi sintetizzato il componente seguendo le stesse procedure del Carry Save. Nei grafici riportati in fig.??, è possibile osservare le differenze tra le prestazioni dei due.

Come previsto, le prestazioni del Carry Save sono molto migliori, specialmente all'aumentare del numero di bit. Grazie all'utilizzo dei blocchi carry save, infatti, i riporti vengono calcolati contemporaneamente in tutti gli stadi, ottimizzando i tempi del circuito rispetto al caso con gli RCA. Anche in termini di area il Carry Save è migliore, in quanto utilizza un full adder in meno per il bit più significativo del risultato.

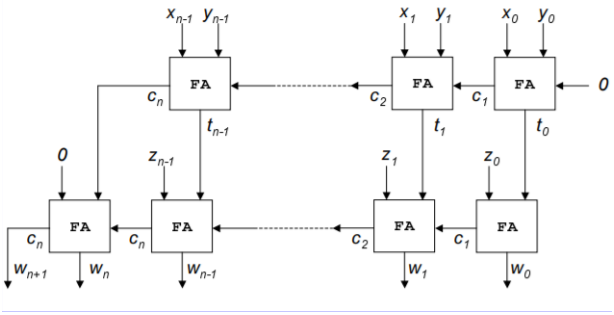

Figure 5: Architettura dell'RCA a tre operandi.

[group style=group size=2 by 1,horizontal sep=2cm, yticklabel style=font=, xticklabel style=font=] [legend style=font= , anchor=south, at=(0.77,0.01), xmin=0,xmax=128, ymin = 0, ymax = 1600, grid=major, width=0.45 height=,xlabel= Numero di bit, ylabel=Numero di slice] coordinates (0,0) (4, 29) (8, 68) (16, 143) (32, 305) (64,605) (128, 1228) ; coordinates (0,0) (4, 32) (8, 69) (16, 143) (32, 343) (64,777) (128, 1521) ; CS RCA [legend style=font= , anchor=south, at=(0.77,0.01), xmin=0,xmax=128, ymin = 0, ymax = 7.5, grid=major, width=0.45height=, xlabel= Numero di bit, ylabel=Minimum period (ns)] coordinates (0,0) (4, 1.870) (8, 2.404) (16, 2.974) (32, 3.501) (64, 4.307) (128, 4.747) ; coordinates (0,0) (4, 1.782) (8, 2.804) (16, 4.040) (32, 4.871) (64, 5.715) (128, 7.117) ; CS RCA

Figure 6: Grafici dei risultati ottenuti post-sintesi in funzione del numero di bit.