0.1 Architettura

Il Carry Select può essere inteso come un'estensione del RCA al fine di migliorarne le prestazioni in caso di numero di bit elevato. Dal momento che le prestazioni del RCA peggiorano linearmente all'aumentare dl numero di bit, si può pensare di suddividere la carry chain in P blocchi, ciascuno dei quali lavora su M bit delle stringhe in ingresso. L'architettura del componente è raffigurata in fig.1.

In particolare, si noti come ogni blocco (ad eccezione del primo) sia composto da due RCA e due multiplexer: gli RCA sommano gli stessi M bit del blocco corrispondente, ma si distinguino per il valore di *c_ in*, ossia il riporto in ingresso. In base al valore effettivo del riporto in ingresso, calcolato allo stadio precedente, i due multiplexer selezioneranno i bit dell'uscita S e il *c_ out* dell'RCA corrispondente. In questo modo, siamo in grado di calcolare contemporaneamente gli M bit in uscita di ogni blocco P: il ritardo complessivo sarà dunque pari a quello di un unico RCA di M bit + quello delle P-1 coppie di multiplexer da pilotare, ossia $T_{CSEL} = M*T_{FA} + (P-1)*T_{MUX}$. Tale vantaggio in tempo denota tuttavia un enorme svantaggio in termini di area occupata, notevolmente superiore a quella di un normale RCA con un'unica carry chain di M*P blocchi.

0.2 Implementazione

0.2.1 Carry Select Block

Per la realizzazione del componente Carry Select, si è dapprima proceduto alla realizzazione di un *Carry Select Block*, ossia dei blocchi che andranno a formare il Carry Select. L'interfaccia di questo componente è la seguente:

```
1  entity carry_elect_block is
2    generic (
3      width : NATURAL := 4
4    );  port (
5      A : in STD_LOGIC_VECTOR ((width-1) downto 0);
6      B : in STD_LOGIC_VECTOR ((width-1) downto 0);
7      c_in : in STD_LOGIC;
8      S : out STD_LOGIC_VECTOR ((width-1) downto 0);
9      c_out : out STD_LOGIC);
10 end carry_select_cell;
```

Codice Componente 1: Interfaccia di un Carry Select Block.

Tale componente, come visto prima, è formato da due RCA e due multiplexer 2-1. I due addizionatori si occuperanno di sommare le due stringhe in ingresso A e B (di lunghezza generica *width*) con *c_ in* pari, rispettivamente, a 0 e 1. In base al valore *c_ in* effettivo in ingresso al blocco, i due multiplexer sceglieranno quali dei due valori *S* e *c_ out,* calcolati dai due RCA, riportare in uscita. L'implementazione completa, realizzata tramite descrizione structural, è consultabile qui: carry_select_block.vhd

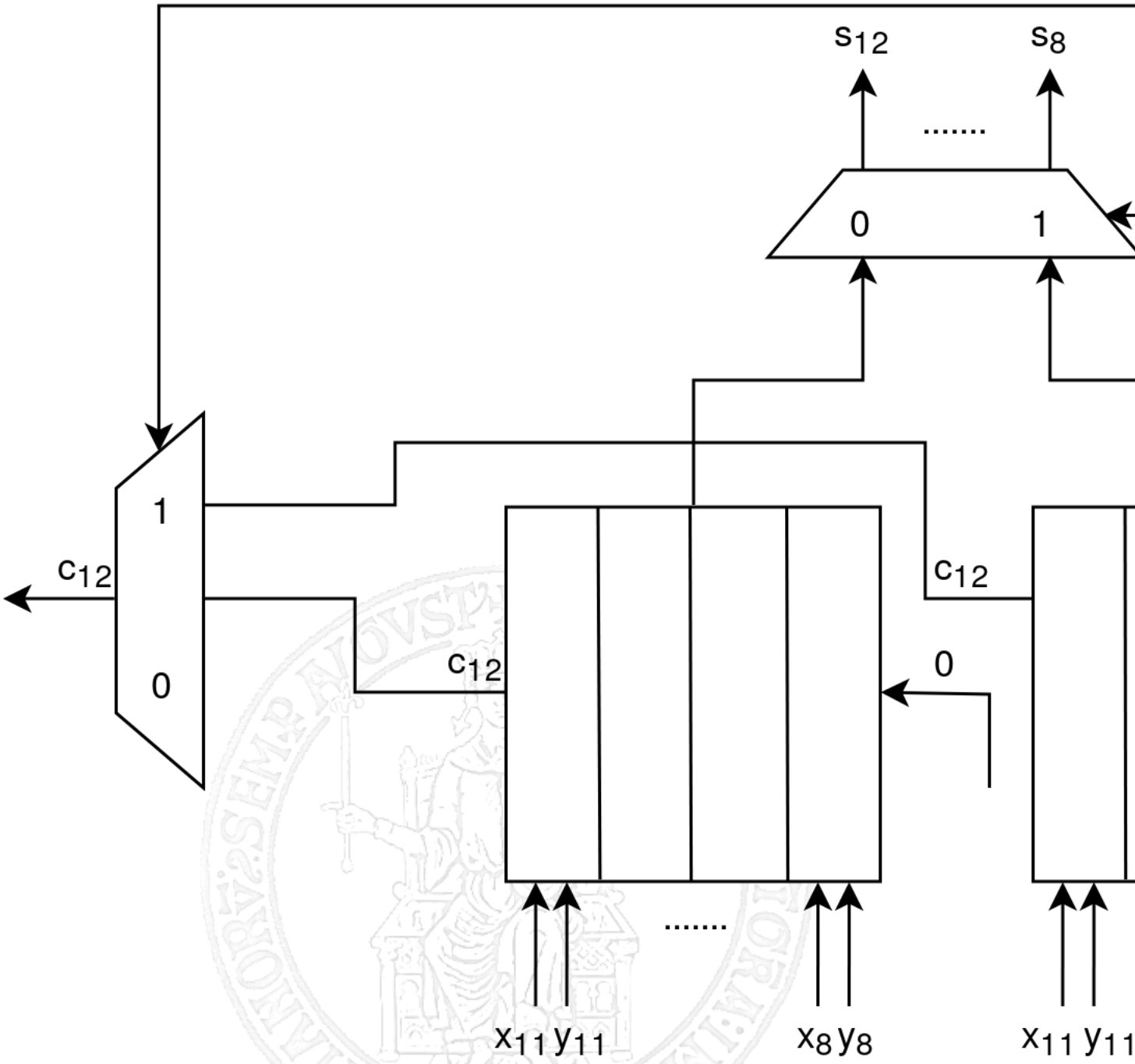

Figure 1: Architettura del Carry Select.

0.2.2 Carry Select

Anche per realizzare il Carry Select è stata utilizzata una descrizione di tipo structural. Prima di tutto, si sono utilizzati due parametri generici: P, ossia il numero di blocchi della catena, ed M, ossia il numero di bit sommato da ciascun blocco. Dopodiché, per quanto concerne l'architettura, sono stati generati l'RCA iniziale e P-1 blocchi restanti per formare la carry chain del sommatore:

```
rca: rippleCarry_adder port map(
  X => A((M-1) downto 0),
  Y => B((M-1) downto 0),
  c_in => internal_carry(0),
  S => S_TEMP((M-1) downto 0),
  c_out => internal_carry(1)
);

blocks: for i in 1 to P-1 generate
  csel_block: carry_select_block port map (
    A => A ((((i+1)*M)-1) downto(i*M)),
    B => B ((((i+1)*M)-1) downto (i*M)),
    c_in  => internal_carry(i),
    S => S_TEMP((((i+1)*M)-1) downto (i*M)),
    c_ou => internal_carry(i+1)
  );
end generate blocks;
```

Codice Componente 2: Generazione dei blocchi del Carry Select.

L'implementazione completa è consultabile qui: carry_select.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 due operandi da sommare. I risultati ottenuti sono osservabili in fig.2.

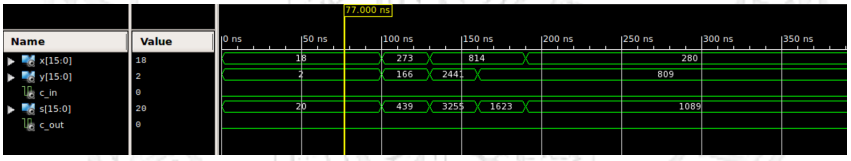

Figure 2: Simulazione behavioural del Carry Save.

[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, 19) (9, 58) (16, 99) (36, 228) (64,387) (121, 827) ; [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, 1.429) (9, 2.057) (16, 2.475) (36, 2.977) (64, 3.618) (121, 4.194) ;

Figure 3: Grafici dei risultati ottenuti post-sintesi in funzione del numero di bit.

0.3.2 Sintesi

0.3.2.1 Scelta di P ed M

Si è proceduto infine alla sintesi del componente. Per la scelta di P ed M, sono state fatte le seguenti considerazioni: data la formula e $T_{CSEL} = M * T_{FA} + (P-1) * T_{MUX}$, è possibile ottimizzare tale quantità scegliendo $M = \sqrt{N * \frac{T_{FA}}{T_{MUX}}}$ e $P = \frac{N}{M}$.

Si è deciso quindi di fissare $N = M * P = 32$ e di sintetizzare il componente per valutare le prestazioni temporali del circuito. Per ottenere il risultato ottimale, dati $T_{FA} = 0.893$ ns e $T_{MUX} = 0.889$ ns, possiamo trovare $M \simeq \sqrt{N} = 8$ e $P = 8$. Per effettuare una dimostrazione dell'efficacia di tale formula, sono riportati in tabella i *minimum period* di funzionamento del Carry Select ottenuti al variare di M e P (con N fissato pari a 32):

P=1, M=64	3.690 ns
P=2, M=32	3.760 ns
P=4, M=16	3.753 ns
P=8, M=8	3.618 ns
P=16, M=4	3.685 ns
P=32, M=2	3.690 ns
P=64, M=1	3.690 ns

E' possibile osservare come la scelta dei valori $M = 8$ e $P = 8$ ci restituisca i risultati ottimali in termini di tempi del circuito.

0.3.2.2 Prestazioni all'aumentare del numero di bit

Si è proceduto infine alla sintesi del componente utilizzando diversi valori N e scegliendo M e P tramite la formula vista precedentemente. Sono stati quindi 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.3.

Si noti come non sono risultate particolari differenze rispetto all'utilizzo di un normale RCA: ciò è probabilmente dovuto al fatto che il tool di sintesi ottimizza il circuito e fa pieno utilizzo della matrice di interconnessione tra gli slices per migliorare le prestazioni del circuito.