## 0.1 Architettura

L'architettura di tale componente è riportata in fig.1. Si noti come, per ottimizzare i ritardi rispetto al RCA, si calcolano in anticipo i riporti sugli stadi successivi utilizzando le condizioni di propagazione (X and Y) e generazione (X or Y). In particolare, i componenti utilizzati sono:

1. *Propagation/Generation calculator*, che si occupa del calcolo delle condizioni di propagazione ($P_i$) e generazione ($G_i$);
2. *Carry Look Ahead*, che si occupa del calcolo dei riporti. In particolare, $C_{i+1} = G_i + P_i C_i$;
3. *Full adder*, che si occupano del calcolo della somma tra i valori di X e Y e i carry in ingresso.

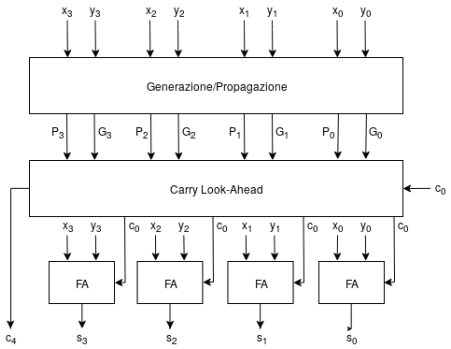


Figure 1: Architettura del Carry Look Ahead.

## 0.2 Implementazione

Si è deciso di implementare il componente *carry_look_ahead_adder* direttamente in forma generica, permettendo di specificare il numero di bit delle stringhe tramite parametro generico *width*. Per l'implementazione, si è utilizzata una descrizione di tipo structural. Di seguito vengono riportati i componenti utilizzati.

### 0.2.1 Propagation/Generation calculator

L'implementazione di questo componente è stata realizzata tramite descrizione dataflow. In particolare, di seguito è riportata la parte di codice relativa alla generazione delle condizioni di propagazione e generazione:

```
1 architecture dataflow of propagation_generation_calculator is
2 begin
3   G <= X and Y;
4   P <= X or Y;
5 end;
```

Codice Componente 1: Descrizione dataflow del Propagation/Generation calculator.

L'implementazione completa è consultabile qui: propagation_generation_calculator.vhd

### 0.2.2 Carry Look Ahead Adder

Per realizzare l'addizionatore, oltre al propagation/generation calculator, si sono utilizzati altri due componenti generati tramite costrutto generate: *carry_look_ahead*, per il calcolo dei riporti, e *full_adders* per la rete di full adder atta al calcolo delle somme:

```
carry_look_ahead: for i in 0 to (width-1) generate
  carry_ahead :
    C(i+1) <= G(i) or (P(i) and C(i));
end generate carry_look_ahead;

full_adders: for i in 0 to (width-1) generate
  fullAdder : full_adder port map (
    x => X(i),
    y => Y(i),
    c_in => C (i),
    s => S_TEMP(i)
  );
end generate full_adders;
```

Codice Componente 2: Generazione del $carry_look_aheadedei full adder$.

Si noti come, all'interno del port map di ciascun full_adder, non siano stati utilizzati i riporti in uscita *c_out*, dal momento che tali riporti sono già stati calcolati precedentemente dal *carry_look_ahead*. L'implementazione completa è consultabile qui: carry_look_ahead_adder.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 e il riporto in ingresso. I risultati ottenuti sono osservabili in fig.2.

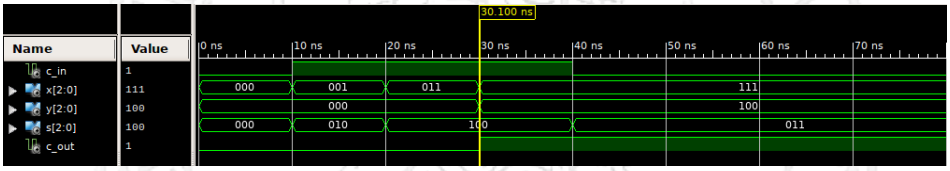


Figure 2: Simulazione behavioural del Carry Look Ahead.

### 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 nel caso del Ripple Carry Adder, sono stati ottenuti il *numero di slices* e *minimum period* (reciproco della massima frequenza di funzionamento) in funzione del numero n di bit per valutare le prestazioni di tale macchina. I risultati sono riportati in fig.3.

[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 = 900, grid=major, width=0.45 height=,xlabel= Numero di bit, ylabel=Numero di slice] coordinates (0,0) (4, 19) (8, 42) (16, 97) (32, 230) (64,432) (128, 849) ; [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) (8, 2.054) (16, 2.480) (32, 2.935) (64, 3.621) (128, 4.134) ;

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

Si osservi come, rispetto ai risultati ottenuti con il Ripple Carry Adder, il circuito occupi un numero di slices (e dunque area) sensibilmente maggiore. Ciò è dovuto al fatto che, introducendo il calcolo dei riporti in parallelo, il numero di componenti è aumentato: tuttavia, tale differenza è apprezzabile solo con numero più elevato di bit (dai 32 in poi). Per quanto riguarda i ritardi, invece, non si registrano particolari differenze rispetto al caso RCA: i periodi minimi risultano pressoché identici. Possiamo dunque concludere che, con l'utilizzo del tool di sintesi per board Nexys 4, non sono particolarmente apprezzabili le differenze tra l'utilizzo di un RCA e di un CLA grazie alle capacità di ottimizzazione del suddetto tool. Si noti infine che, a fronte di sintesi del componente con un numero di bit maggiore di 32, il tool riporta il seguente warning: "*WARNING:Xst:1336 - (*) More than 100% of Device resources are used*". Questo è dovuto al fatto che le risorse necessarie per sintetizzare il componente sulla board non sono sufficienti, e dunque tali risultati sono da considerarsi solo in teoria.