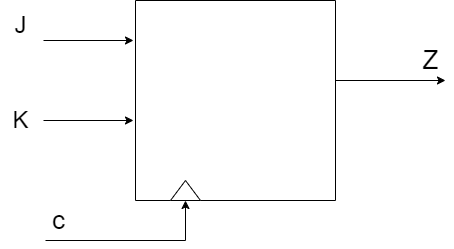
# Progetto di Un Flip Flop di tipo JK - DDR ovvero sincronizzato sia sul fronte di salita che di discesa del clock

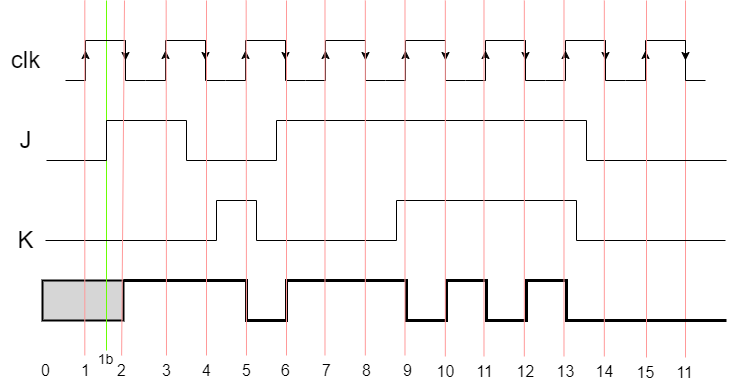
Individuare i segnali coinvolti:

Visto dall’esterno il sistema si presenta dotato di 3 ingressi e di un’uscita:  
- due segnali a livelli *J e K* che controllano la commutazione del FlipFlop  
- un segnale *clk* i cui fronti di salita/discesa scandiscono gli istanti nei quali il FF potrebbe commutare



.

Descrizione sintetica   
Comprendere pienamente il funzionamento del dispositivo, meglio se attraverso un esempio che evidenzi un ipotetico segnale di stimoli (composto dai segnali J, K, clk ) e la risposta attesa:



1. All’istante 0, ovvero appena acceso il circuito, il dispositivo si mette in uno stato casuale ignoto all’utilizzatore (in genere contrassegnato con ‘X’).
2. All’istante 1: viene rilevato un fronte di clock, ma essendo entrambi i segnali J e K bassi non si altera l’uscita (che comunque rimane sconosciuta).

All’istante 1b si alza il segnale J, ma il FF non reagisce in quanto al di fuori dell’istante di campionamento

1. All’istante 2: in presenza del fronte di discesa del clock il sistema trova i segnali di controllo J=1 e K=0 e si mette nello stato alto.
2. All’istante 3: idem
3. All’istante 4: J=0 e K=0 quindi il sistema mantiene lo stato precedente
4. All’istante 5: J=0 e K=1 quindi il FF si resetta portando l’uscita a livello basso.
5. All’istante 6: J=1 e K=0 il FF riporta l’uscita allo stato alto
6. All’istante 7 e 8: idem
7. All’istante 9: J=1 e K=1 il FF commuta il uso stato e si porta a livello basso
8. Agli istanti tra 10 e 13 J=1 e K=1, pertanto ogni volta il FF cambia di stato rispetto lo stato precedente
9. Agli istanti tra 14 e 16 J=0 e K=0, il FF mantiene lo stato precedente

Evidenziare i meccanismi di funzionameto

Se volessimo analizzare le condizioni che portano il Flip - Flop ad attivarsi/disattivarsi evidenziando quali siano le sequenze di segnali atte a creare una commutazione, ci troveremmo di fronte a molteplici sequenze tutte egualmente valide: ad esempio considerando la commutazione da 0 a 1 e ordinando i tre segnali di ingresso con *JKc* dovremmo considerare come valide allo scopo tutte le seguenti sequenze di attivazione:

JKc: 100 🡪 101

JKc: 101 🡪 100

JKc: 110 🡪 111

JKc: 111 🡪 110

E similmente per le sequenze di “disattivazione”:

JKc: 010 🡪 011

JKc: 011 🡪 010

JKc: 110 🡪 111

JKc: 111 🡪 110

Realizzare ora una macchina asincrona in grado di riconoscere tutte queste sequenze richiederebbe di primo acchito una macchina a 10 stati che poi difficilmente potrebbe essere semplificata.

Va però sottolineato che identificare ogni singola possibile sequenza NON è sempre il metodo più idoneo per procedere. Nelle FSM gli stati servono appunto a mantenere una memoria grazie alla quale poter prendere la decisione corretta quando vi verificano particolari condizioni. Ritorniamo quindi ad analizzare nuovamente il funzionamento della macchina per comprendere meglio cosa essa debba memorizzare per poter attuare correttamente la transizione desiderata e il suo funzionamento può essere riassunto nelle seguenti istanze:

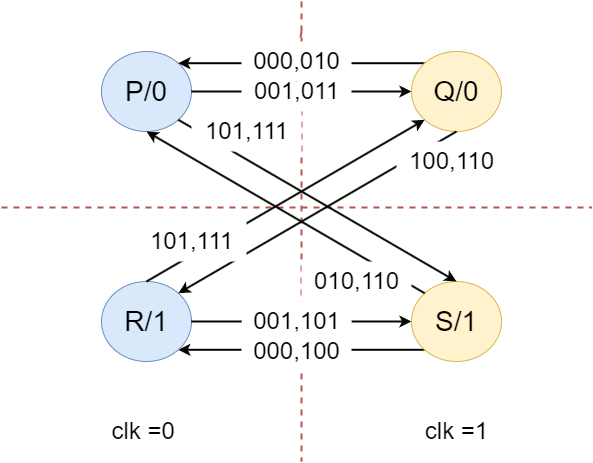
1. Una eventuale transizione avviene SOLAMENTE quando si presenta un fronte di clock (salita o discesa)
2. Mentre il clock è stabile, qualunque modifica dei segnali J e K non altera l’uscita
3. Lontano dalle transizioni del clock la macchina deve mantenere stabile l’uscita allo stato 0 oppure allo stato 1

Pertanto chiediamoci cosa deve essere in grado di memorizzare la macchina per poter agire correttamente:

* La macchina deve memorizzare lo stato dell’uscita (0 o 1)
* La macchina deve tenere memoria di quale era il livello del clock nei momenti precedenti al momento attuale e solo quando rileva un’alterazione di quest’ultimo (fronte di salita o di discesa) può scegliere (in base agli ingressi J e K presenti in quel momento) se portare l’uscita in uno stato diverso oppure se mantenerla allo stato attuale.

Per mantenere questa informazione basta pertanto realizzare quattro stati che memorizzino relativamente sia il livello del clock negli istanti precedenti sia lo stato dell’uscita e che effettui la transizione da uno stato all’altro solo se rileva un’alterazione al livello del clock.

Una bozza secondo Moore della macchina può essere descritta col seguente grafo dove i tre valori sui rami nell’ordine rappresentano J,K e c. Si noti che:



* Gli stati P e Q sono relativi all’uscita bassa mentre R ed S sono relativi all’uscita alta
* Gli stati P ed R vengono raggiunti quando il clock è basso mentre Q ed S vengono raggiunti quando il clock diventa alto
* I passaggi di stato avvengono solo ed esclusivamente quando il clock cambia di livello per cui si passa sempre da uno stato posto a sinistra nel grafo ad uno stato posto a destra e viceversa e non vi è alcun passaggio tra stati posti dallo stesso lato del grafo
* Si deve solamente scegliere se lo stato su cui “approdare” sia quello con uscita alta o quello con uscita bassa e questo lo si effettua in base al livello degli ingressi J e K
  + Se J=0 e K=0 ci si sposta in orizzontale (stessa uscita)
  + Se J=1 e K=1 ci si sposta in diagonale (uscita alternata)
  + Se J=0 e K=1 ci si sposta verso lo stato con uscita bassa
  + Se J=1 e K=0 cisi sposta verso lo stato con uscita alta
* Tutte le restanti combinazioni degli ingressi (non scritte nel grafo di cui sopra per semplicità) mantengono la macchina nel medesimo stato

Descrizione completa attraverso Tavola di Huffman

La macchina sopra descritta può essere forse espressa in modo più chiaro attraverso la tavola di Huffman



Si può iniziare evidenziando i 4 stati e tutte le possibili condizioni degli ingressi.   
In Verde sono evidenziati gli stati stabili.

* La parte a sinistra della tabella è relativa ad un clock basso, la parte di destra è relativa al clock alto
* Nella parte sinistra vi sono due stati stabili P ed R (rispettivamente con uscita a livello basso ed altro) e similmente nella parte destra con gli stati Q ed S
* Gli stati P,Q,R,S rimangono stabili per qualsiasi modifica dei segnali J e K, vi è un passaggio di stato solamente quando c’è una variazione del clock (un salto dalla parte sinistra alla parte destra della tabella)
* Ad ogni fronte positivo del clock vi sarà una transizione dalla parte sinistra della tabella alla parte destra, mentre per ogni fronte negativo vi sarà una transizione dalla parte destra alla parte sinistra

Si procede ora a riempire la tabella con tutte le transizioni possibili



Ad esempio: Ci si trova nello stato P con ingressi J=0 e K=0 (e ovviamente con clock=0). Nel momento in cui quest’ultimo dovesse passare a livello 1 (fronte di salita) si deve scegliere se portarsi nello stato Q (con uscita 0) o nello stato S (con uscita 1). La risposta corretta è ovviamente Q. Allo stesso modo si possono riempire tutte le caselle relative al fronte di salita del clock partendo da P quando sono presenti tutte le altre combinazioni di ingressi:

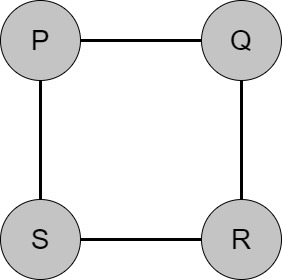


Le transizioni dagli stati posti a destra della tabella a quelli posti a sinistra invece sono verificate durante i fronti di discesa del clock, che nel nostro caso specifico sono da considerassi anch’essi istanti di sincronismo e perciò a loro volta destinati da alterare il valore dell’uscita in base ai valori presenti sugli ingressi J e K.



Scelta di un’opportuna codifica degli Stati

Si può notare agevolmente attraverso il grafo delle transizioni che per evitare corse critiche basta che in fase di codifica si garantisca l’adiacenza delle seguenti coppie di stati: PQ, QR, RS, SP.



Quindi una possibile codifica degli stati che utilizzi solo due bit potrebbe essere ad esempio

P: 00  
Q: 01  
R: 11  
S: 10

Tavola di Flusso e realizzazione

Utilizzando la codifica scelta la tavola di flusso del sistema diventa:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| clk | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | Z |
| Y1Y2\JK | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |  |
| 00 | 00 | 00 | 00 | 00 | 01 | 01 | 10 | 10 | 0 |
| 01 | 00 | 00 | 11 | 11 | 01 | 01 | 01 | 01 | 0 |
| 11 | 11 | 11 | 11 | 11 | 10 | 01 | 01 | 10 | 1 |
| 10 | 11 | 00 | 00 | 11 | 10 | 10 | 10 | 10 | 1 |

Il che ci porta ad individuare le seguenti tavole di Karnaugh nella ricerca delle funzioni relative alle due variabili di stato:

Per la variabile y1’:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| clk | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y1Y2\JK | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 01 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

Per la variabile y2’:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| clk | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y1Y2\JK | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 10 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Per l’uscita Z:

|  |  |
| --- | --- |
| y1y2 | Z |
| 00 | 0 |
| 01 | 0 |
| 11 | 1 |
| 10 | 1 |

Analizzando le suddette tavole si può notare che con la codifica adottata quando il clock è basso (parte sinistra) entrambe le variabili Y1 e Y2 forniscono la medesima uscita, mentre quando il clock è alto (parte destra) sono una l’inversa dell’altra. Questo può essere sfruttato per una semplificazione congiunta del circuito nella forma (secondo Shannon)

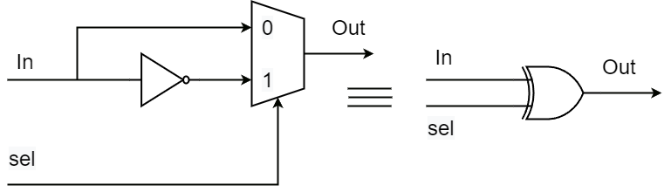
Ove:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| clk | 0 | 0 | 0 | 0 |
| Y1Y2\JK | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 0 | 0 | 1 |

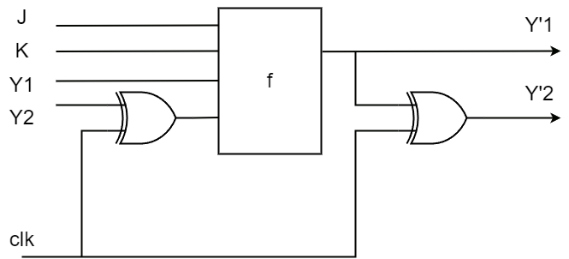
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| clk | 1 | 1 | 1 | 1 |
| Y1Y2\JK | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 0 | 0 | 1 |
| 10 | 1 | 1 | 1 | 1 |

Che a ben vedere risulta essere uguale ad “f” pur di invertirne la variabile di ingresso y2, ovvero:

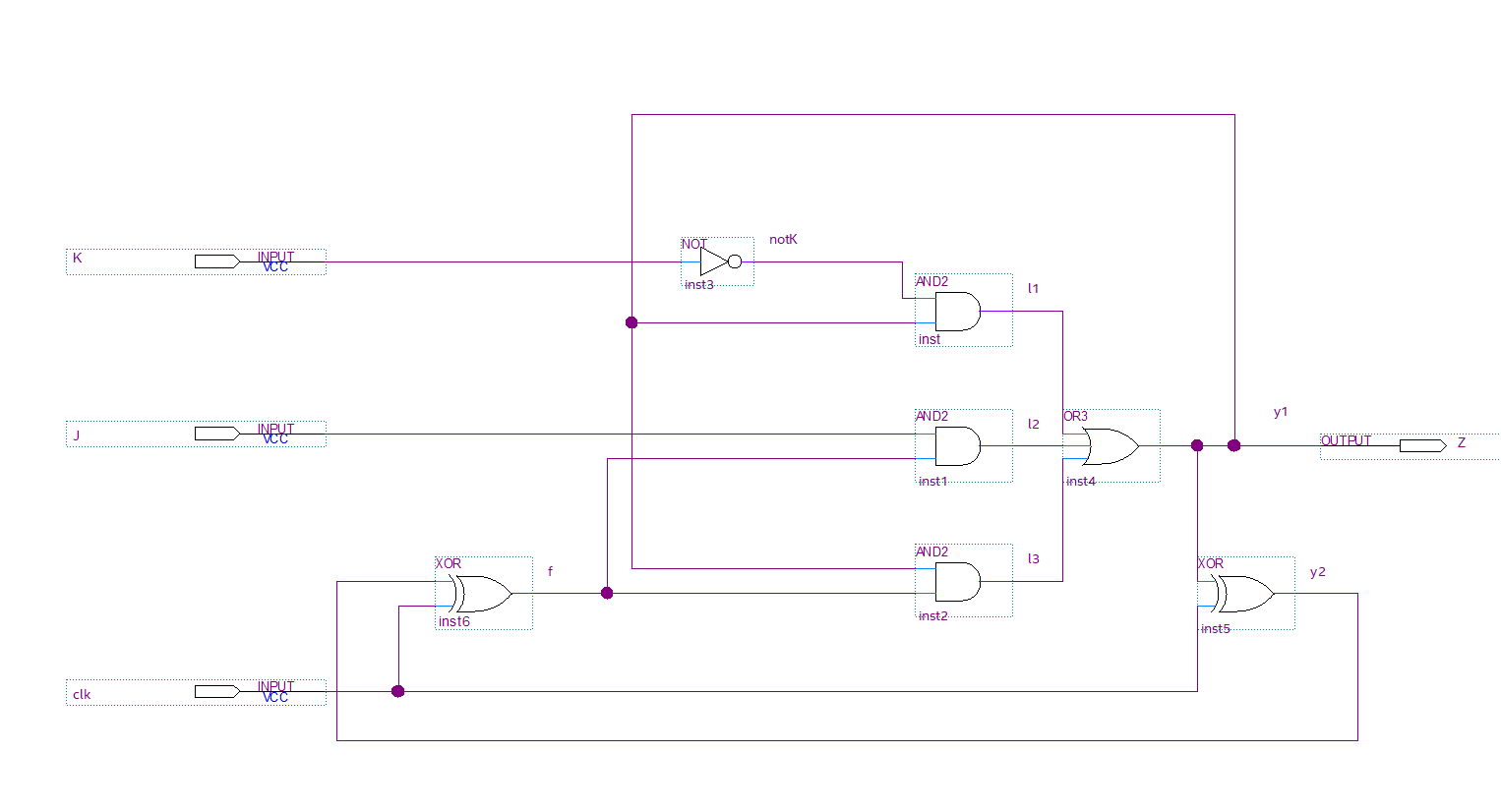
Considerando ora che per invertire (o meno) una variabile in base ad un segnale di controllo si può ricorrere all’impiego di una singola porta XOR:



Uno schema del circuito che potrebbe realizzare dette funzioni potrebbe essere il seguente:



E pertanto il circuito completo, realizzato con porte logiche elementari potrebbe essere:



La cui descrizione in Verilog HDL potrebbe essere la sequente:

module FFJK\_ddr(K,J,clk,Z);

input wire K;

input wire J;

input wire clk;

output wire Z;

wire f;

wire l1;

wire l2;

wire l3;

wire notK;

wire y1;

wire y2;

assign l1 = notK & y1;

assign l2 = J & f;

assign l3 = y1 & f;

assign notK = ~K;

assign y1 = l2 | l3 | l1;

assign y2 = y1 ^ clk;

assign f = y2 ^ clk;

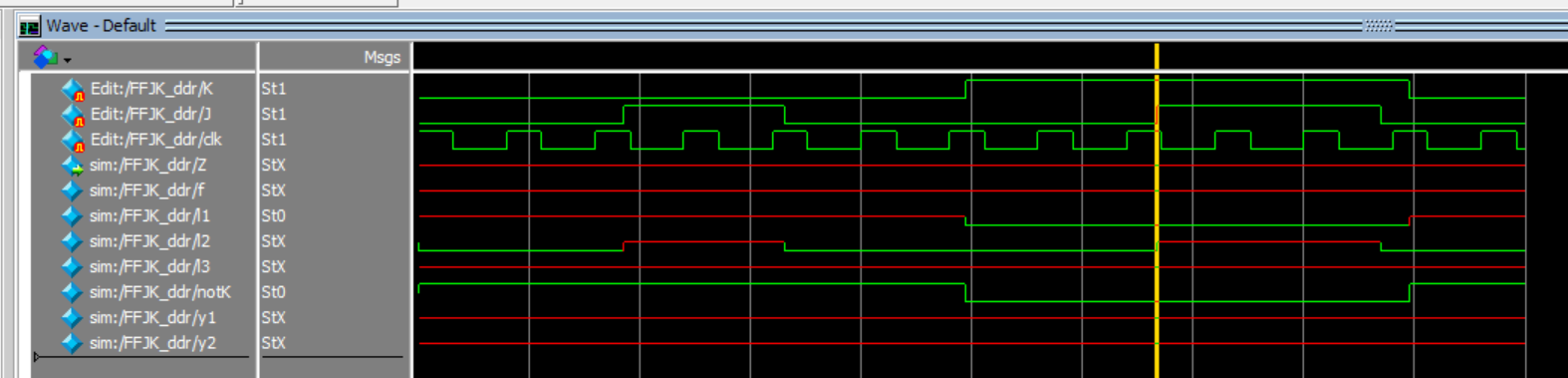
assign Z = y1;

endmodule

Questo circuito per quanto elegante presenta però alcune criticità:

Problematica 1: valori indeterminati:

Appena acceso il circuito può trovarsi in base agli ingressi in uno stato qualsiasi (bistabile). Si noti infatti che in ogni colonna della tabella di Huffman vi sono almeno 2 condizioni di stabilità e non esiste nessuna combinazione degli ingressi in grado di metter il sistema in una condizione nota.



Sebbene da un punto di vista pratico il circuito reale, appena acceso, si porrà in una ben determinata condizione (in modo più o meno aleatorio), in fase di simulazione non vi è modo di prevedere quale sia tale condizione, di conseguenza il simulatore assegnerà ad un primo istante a Y1 ed Y2 due valori indeterminati (X). Essendo essi indeterminati, lo sarà di conseguenza anche la linea “f”, con essa la linea “l3” e quindi nuovamente Y1 ed Y2. Si entra quindi in un in circolo vizioso dal quale non vi è modo di uscire. Pertanto sebbene il circuito reale magari in se non ne necessiti è bene introdurre un segnale di reset asincrono che forzi il circuito in una condizione nota da cui partire.

module FFJK\_ddr(K,J,clk,Res,Z);

input wire K;

input wire J;

input wire clk;

input wire Res;

output wire Z;

wire f;

wire l1;

wire l2;

wire l3;

wire notK;

wire y1,y2;

wire y1r,y2r;

assign l1 = notK & y1r;

assign l2 = J & f;

assign l3 = y1r & f;

assign notK = ~K;

assign y1 = l2 | l3 | l1;

assign y2 = y1 ^ clk;

assign y2r = y2 & ~Res;

assign y1r = y1 & ~Res;

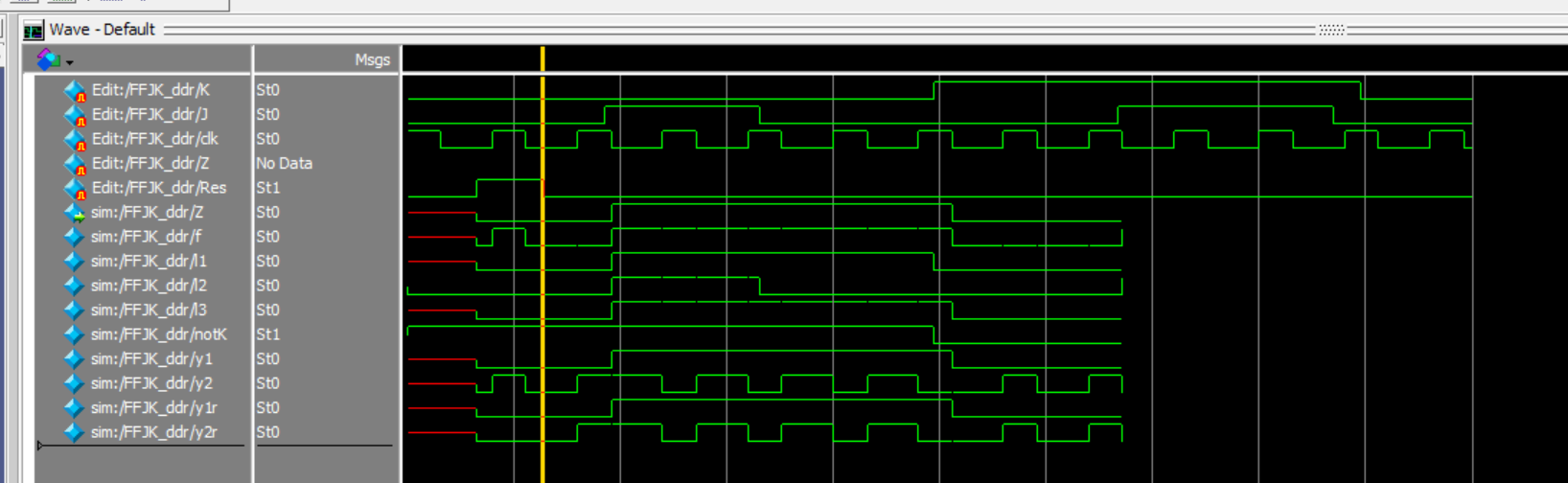
assign f = y2r ^ clk;

assign Z = y1;

endmodule

Problematica 2: TEMPI di RITARDO:

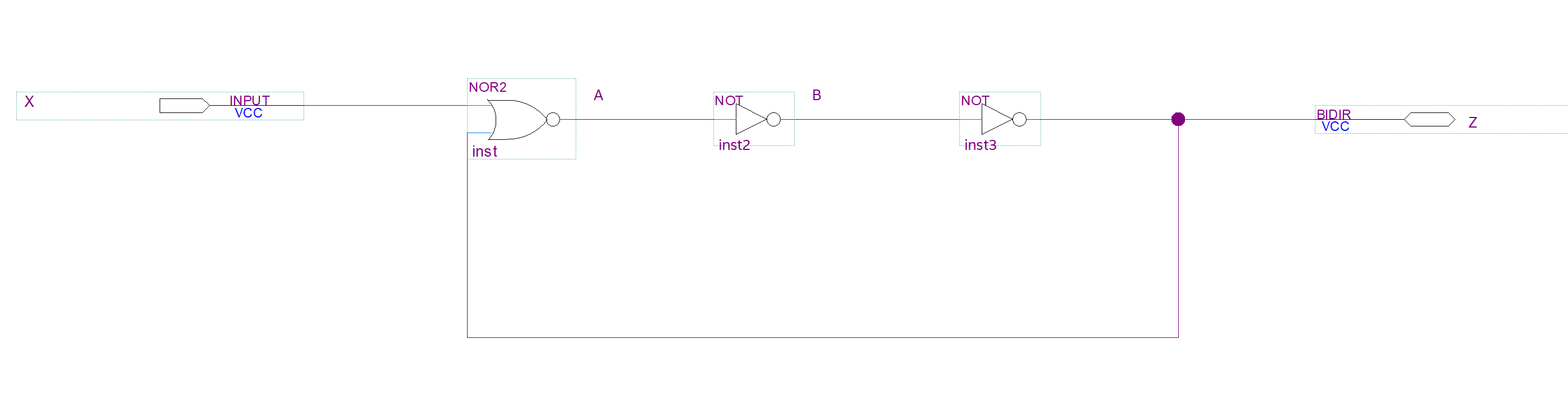
Simulando il suddetto modulo si ottengono i seguenti risultati:

  
  
Il sistema sembra partire correttamente, ma giunto ad un certo punto il sistema si blocca e restituisce l’errore:

# \*\* Error (suppressible): (vsim-3601) Iteration limit 5000 reached at time 672 ps.

Dove il programma di simulazione segnala di essere entrato in un loop infinito dal quale è uscito forzatamente dopo 5000 iterazioni. Detto problema nasce dal fatto che finora abbiamo fatto la simulazione di un sistema del tutto ideale (senza alcun ritardo) dove l’evoluzione del circuito avviene istantaneamente in ogni suo punto.

Un esempio simile, ma sul quale è più facile ragionare potrebbe essere il seguente circuito:



Descritto da

module osc(X,Z);

input wire X;

inout wire Z;

wire a;

wire b;

assign a = ~(X | Z) ;

assign b = ~a;

assign Z = ~b;

endmodule

Fintanto che il segnale di ingresso X è a livello alto la linea A ha un valore noto (pari a 0), così la linea B (pari a 1) e da ultimo l’uscita Z (pari a 0). Ma se si porta a 0 il valore di X, in base ai dati attuali tutte le linee dovrebbero invertire il loro valore, compresa la linea Z che però viene riportata in ingresso e che quindi altera nuovamente i valori, e così via. (A dipende da Z, B dipende da A, Z dipende da B, A dipende da Z ….) peraltro il tutto istantaneamente (ovvero a ritardo nullo). Fintanto che non si introducono i tempi di ritardo delle varie porte ogni simulazione risulta pertanto impossibile.

Problematica 3: Alee Essenziali:

Proviamo pertanto ad introdurre dei tempi di ritardo alle varie porte logiche e per comodità supponiamo che tutte le porte presentino un medesimo ritardo unitario e che le linee di collegamento non abbiano ritardo.

Anche introducendo i tempi di ritardo i risultati non sono soddisfacenti ed il circuito NON funziona correttamente:

assign #1 l1 = notK & y1r;

assign #1 l2 = J & f;

assign #1 l3 = y1r & f;

assign #1 notK = ~K;

assign #1 y1 = l2 | l3 | l1;

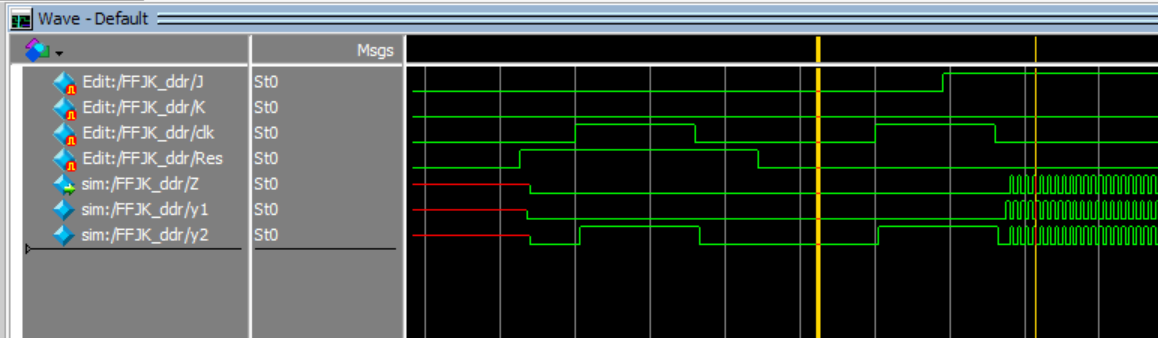
assign #1 y2 = y1 ^ clk;

assign y2r = y2 & ~Res;

assign y1r = y1 & ~Res;

assign #1 f = y2r ^ clk;

assign Z = y1;



In particolare se analizziamo la zona evidenziata tra i cursori gialli (ovvero dopo che un impulso di reset ha portato il circuito in uno stato noto:

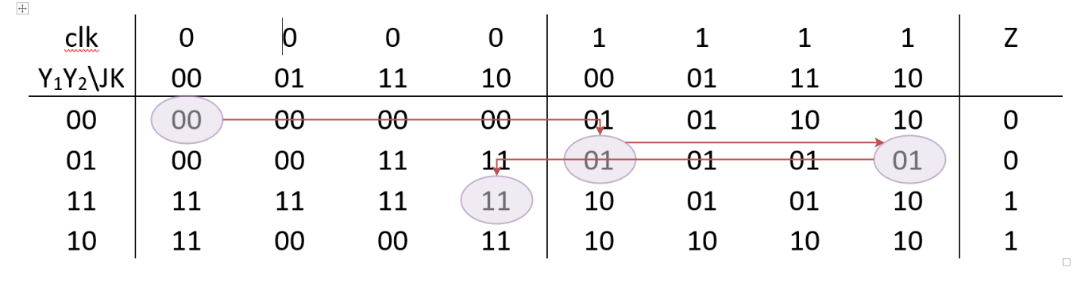
JKc = 000 con Y1Y2=00 il sistema correttamente mantiene lo stato Y1Y2=00

JKc = 001 (il clock si alza) il sistema cambia stato e si pone correttamente in Y1Y2=01

JKc = 101 (si alza J) il sistema mantiene lo stato Y1Y2=01

JKc = 100 (si abbassa il clock - fronte di discesa e di sincronismo) come da tabella il sistema dovrebbe mettersi nello stato Y1Y2=11, ma prima che il segnale arrivi su Y1 la variabile Y2 si è venuta subito ad abbassare (vedasi lo schematico) In pratica con i ritardi assegnati vengono a commutare contemporaneamente entrambe le porte XOR

portando per un certo tempo il sistema nello stato 00, a questo punto il segnale è finalmente arrivato su Y1 che assume il livello alto, ma nel frattempo il segnale basso di Y2 si è già propagato all’ingresso del loop innescando a sua volta una nuova commutazione di Y1, che a sua volta porta una commutazione di Y2 e così via ….



Il problema nasce dal fatto che Y2 , per come è stata pensata la logica “dovrebbe” dipendere (e pertanto “seguire temporalmente”) il valore di Y1, mentre per come è stato realizzato il circuito essa risponde con tempi estremamente brevi già alle variazioni del clock, molto prima di aspettare che queste abbiano qualche effetto su Y1. Se poi, come accade, tali variabili di stato si propagano verso gli ingressi prima che le altre variabili abbiano “percepito” la modifica dell’ingresso che ha creato tale variazione, l’evoluzione può risultare assolutamente imprevedibile.

L’unica soluzione da adottare è quella di garantire che le variabili di stato vengano percepite all’ingresso del loop solo dopo che le variazioni degli ingressi abbiano propagato la loro influenza all’intero circuito.

Ciò può essere “simulato” allungando i tempi di ritardo delle porte che riportano in ingresso le variabili di stato (ed in particolare Y2), mentre dal punto di vista realizzativo si può risolvere introducendo logica ridondante (ad esempio serie pari di invertitori) atte a ritardare sufficientemente il segnale.

assign #1 l1 = notK & y1r;

assign #1 l2 = J & f;

assign #1 l3 = y1r & f;

assign #1 notK = ~K;

assign #1 y1 = l2 | l3 | l1;

assign #1 y2 = y1 ^ clk;

assign #6 y2r = y2 & ~Res;

assign #6 y1r = y1 & ~Res;

assign #1 f = y2r ^ clk;

assign Z = y1;

