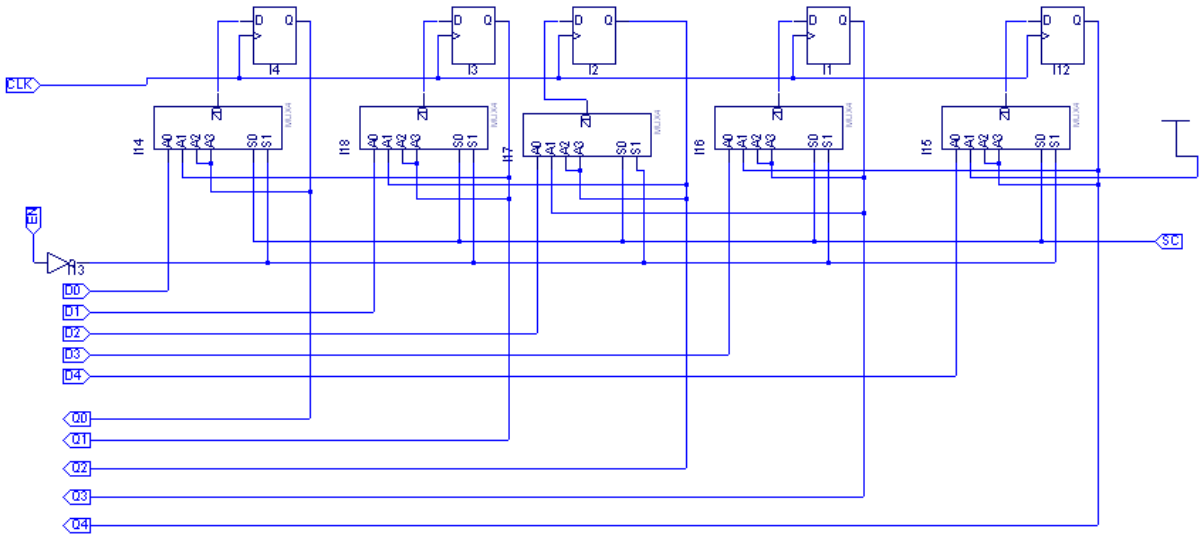
Pràctica 4

Disseny del comandament

Com a primera opció vam optar per a dissenyar el comandament amb els esquemes circuitals que proveeix el ispLever. La llista de components necessaris tal com s'enumera als exercicis previs és: registre desplaçament de 5 bits, divisor de freqüència de 10, divisor de freqüència de 64, escombrat i descodificació de tecla i modulador.

Registre de desplaçament

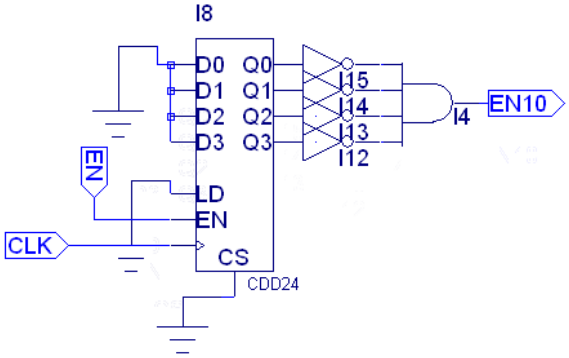
Aquest registre té una senyal d'habilitació (EN) que indica si ha de funcionar (1) o bé si no ha de fer res (0) mantenint el seu valor. En cas d'estar en funcionament tenim dues possibilitats en funció del pin SC. Si SC és 0 carregarà el valor a l'entrada corresponent als bits D0..D4, en cas de ser un 1 desplaçarà el valor del seu registre cap a la dreta (eliminant el de menor pes) i introduirà un 1 al bit de major pes.



La implementació d'aquest bloc és senzilla, utilitzem un registre i un multiplexor per a cada bit del registre. El multiplexor tindrà 3 possibles valors: anterior, desplaçat i càrrega. En el cas de EN=0 triarem l'entrada que manté el valor independentment del valor de SC. En cas de EN=1 triem el valor en funció de SC.

Divisor de freqüència (10)

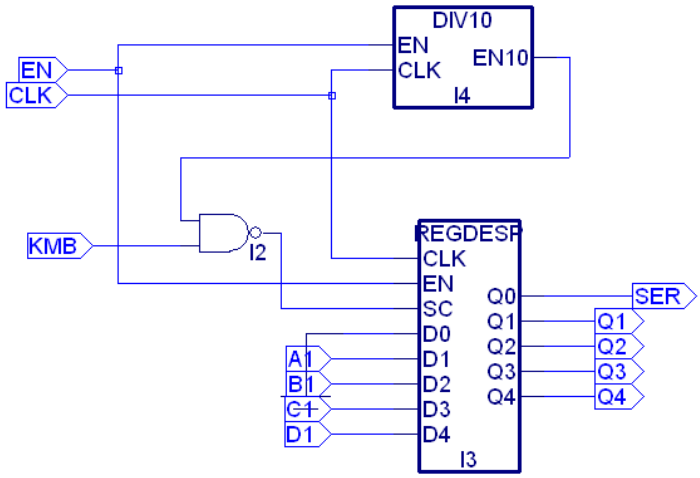
Aquest bloc és molt senzill. Simplement ha de treure un 1 a la seva sortida cada 10 cicles de rellotge. Per a implementar-lo necessitem un comptador.



En el cas del ispLever tenim a les llibreries un comptador de mòdul 10. Farem una AND amb les sortides del comptador invertides, d'aquesta manera quan el comptador valgui tot 0 traurem un 1 a la sortida.

Conversor paral·lel sèrie

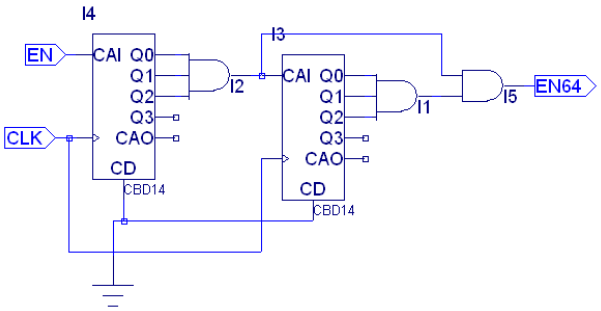
Fent ús dels dos blocs anteriors es fa el conversor paral·lel sèrie. Aquest bloc llegeix 4 bits de dades de forma paral·lela i els treu de forma seqüencial, començant per un 0, el 4 bits de dades i finalment 5 uns, i torna a repetir el procés. Aquest és el bloc que s'encarrega de crear la seqüència que envia el comandament. A més a més traiem uns bits de més per a poder comprovar-ne el funcionament posteriorment.



El funcionament es basa en utilitzar la senyal del divisor entre 10 com a senyal de càrrega, d'aquesta manera carreguem un de cada 10 cicles de rellotge. La resta de cicles el registre de desplaçament va desplaçant la seva sortida obtenint al bit de menor pes (Q0 o SER) una sortida sèrie formada per 0,A,B,C,D,1,1,1,1,1.

Divisor de freqüència (64)

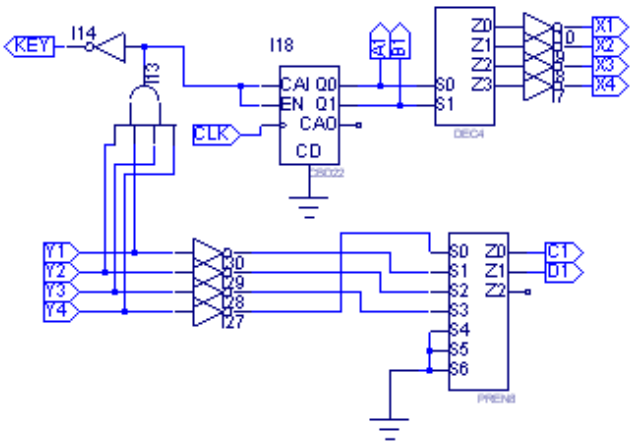
Aquest bloc es fa exactament igual que el divisor per 10. La única diferència és que encadenem dos divisors de freqüència de 8 per a evitar utilitzar un comptador molt gran.



Conversor de teclat

Aquest bloc és l'encarregat d'escombrar les tecles del teclat del comandament i de generar la senyal KEY (que indica tecla pitjada) així com el codi de fila i columna que s'ha pitjat.

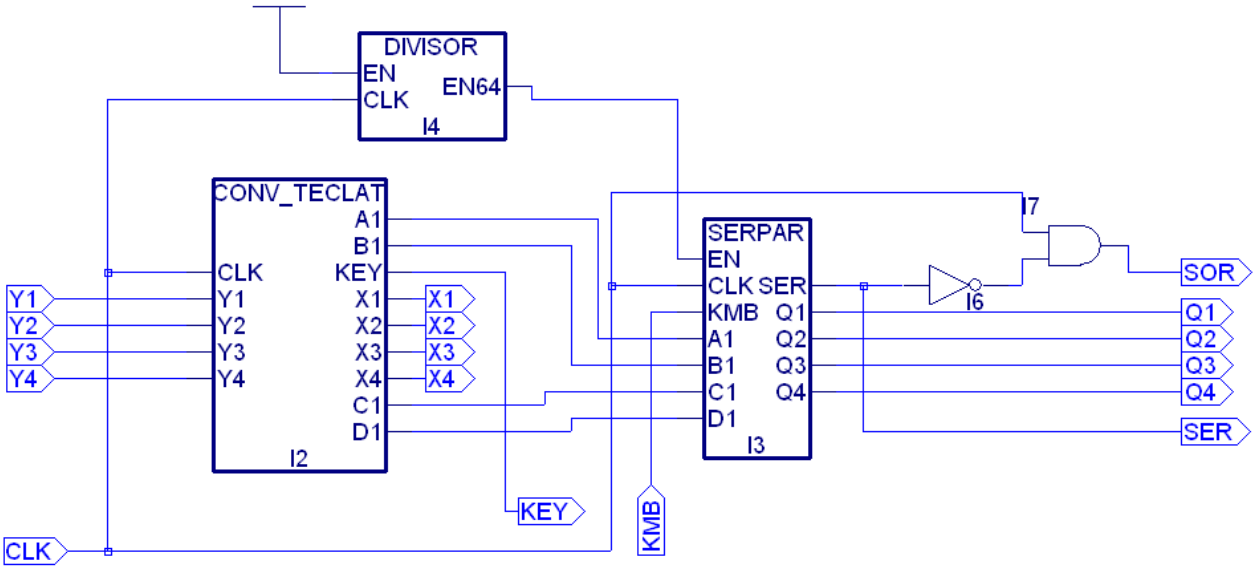
Per a realitzar-lo hem utilitzat un comptador que va canviant



contínuament de fila a escombrar. Per cada fila els senyals Y_i reben l'estat de les tecles. Si una d'elles és 0 vol dir que s'ha pitjat una tecla. En aquest cas s'ha d'aturar l'escombrat mitjançant la senyal EN del comptador. Això es fa per dues coses: poder detectar quan s'aixeca el dit i poder enviar la descodificació de fila columna. Les descodificacions les hem fet amb uns mòduls que proporciona el ispLever.

Circuit complet

El circuit complet es basa en el divisor de freqüència de 64, el conversor de teclat i el conversor sèrie paral·lel.



Les senyals KEY i KMB són les mateixes llevat que es treuen a fora de la PLD per a poder afegir un petit circuit que eviti el rebot produït pel teclat. SOR és la senyal modulada a la freqüència del circuit (32KHz), aquesta senyal és inversa de SER, ja que així ho requereix el receptor.

El funcionament és el següent: el conversor de teclat detecta una tecla i activa KEY així com els codis de fila i columna A, B, C i D, el conversor paral·lel sèrie carrega els valors i comença una conversió de forma periòdica de 10 bits. Aquest últim bloc funcionarà a una freqüència de 512Hz ja que la seva senyal d'habilitació prové del divisor de freqüència. Això fa que la velocitat de transmissió sigui de 512 bauds.

Finalment el modulador és molt senzill, simplement és una porta AND entre el rellotge i la sortida. Això provoca que la sortida sigui una senyal de 32KHz com a valor lògic 1 o absència de senyal com a valor lògic 0.

Conclusions de l'experiència

Durant aquesta pràctica vam trobar molt problemes amb el ispLever principalment. El primer i més elemental és l'absència d'etiquetes o comentaris que identifiquessin els elements circuitals que porta l'editor incorporats. Això

ens va obligar a simular moltes vegades components que semblaven una cosa i eren una altra. Un exemple clar és el comptador de mòdul 10. Aparentment semblava un comptador de 4 bits (és a dir, fins a 15), però no va ser així i això ens va provocar contratemps en el disseny i la simulació.

Una altra cosa molt més generalitzada va ser el mal comportament del programa en general: problemes amb l'editor així com una poca claredat en els errors que reportava. El major problema en aquest sentit va ser bastant gros: la generació del fitxer JEDEC per la PLD. En la majoria de casos ens deia que era impossible encabir el circuit en la nostra PLD ja que era massa petita. Això és estrany per molts motius: utilitzant una lògica infinitament més simple ens deia que no hi cabia, mentre que si posàvem unes portes de més per a complicar-ho no hi havia cap problema. Aquest problema ens va portar a implementar alguns mòduls utilitzant ABEL, però donat que el problema no va desaparèixer vam decidir tornar a fer la pràctica en ABEL. Cal dir però que les dues versions de la pràctica funcionen perfectament.

Pràctica alternativa ABEL

Després de fer la primera versió de la pràctica usant components circuitals, vam arribar a la conclusió que dissenyar els blocs utilitzant circuits no és el més eficient i que el programa de Lattice està optimitzat per a fer servir el llenguatge ABEL. És per aquest motiu que vam decidir realitzar la pràctica de nou utilitzant exclusivament el llenguatge ABEL.

L'orientació que li vam donar a aquesta segona versió va ser molt més modular i divisible.

Vam descompondre el divisor de freqüència de 64 en dos divisors de 8 encadenats.

Vam implementar un comptador de 2 bits, un codificador de 4 a 2 i un descodificador de 2 a 4.

Vam implementar el registre de desplaçament utilitzant ABEL.

Divisor de freqüència de 8

Aquest bloc està implementat com una màquina de 8 estats en la qual s'avança si EN=1 i s'emet un 1 al cap de 8 cicle de EN=1.

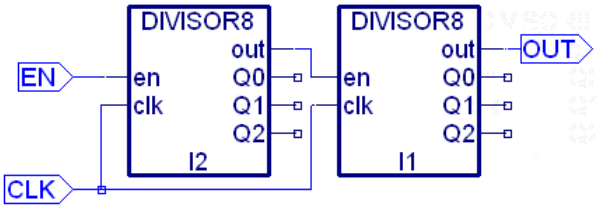
```
MODULE divisor8

title 'div freq de 8';
C,X = .C.,.X.;

"Entrades
        en      pin;
        clk     pin;
"Sortides
        out     pin istype 'reg';
        Q2..Q0  pin istype 'reg';
        sreg = [Q2,Q1,Q0];
```

```
equations
        sreg.clk = clk;
        out.clk = clk;

state_diagram sreg

state 0:
        if (en) then 1 WITH out = 0;
        else 0 WITH out = 0;
state 1:
        if (en) then 2 WITH out = 0;
        else 1 WITH out = 0;
state 2:
```

```
        if (en) then 3 WITH out = 0;
        else 2 WITH out = 0;
state 3:
        if (en) then 4 WITH out = 0;
        else 3 WITH out = 0;
state 4:
        if (en) then 5 WITH out = 0;
        else 4 WITH out = 0;
state 5:
        if (en) then 6 WITH out = 0;
        else 5 WITH out = 0;
state 6:
        if (en) then 7 WITH out = 0;
        else 6 WITH out = 0;
state 7:
        if (en) then 0 WITH out = 1;
        else 7 WITH out = 0;

test_vectors ([clk,en] -> [out])
@repeat 50 { [C,1] -> [X]; }
@repeat 30 { [C,0] -> [X]; }
@repeat 20 { [C,1] -> [X]; }

END
```

Divisor de freqüència de 64

Unió de dos divisor de 8 en cascada.



Comptador de 2 bits.

```
MODULE comptador4
title 'comptador de 0 a 3';

C,X = .C.,.X.;

"Entrades
        en      pin;
        clk     pin;
"Sortides
        q1 .. q0   pin istype 'reg';
"Conjunts
        data = [q1..q0];

Equations
        when en then data:= data+1
        else data := data;
        data.clk = clk;

test_vectors ([clk,en] -> [q1,q0])
        @repeat 20 { [C,1] -> [X,X]; }

END
```

Descodificador de 2 a 4

```
MODULE descodificador24

title 'descodificador 2 a 4';
C,X = .C.,.X.;

"Entrades
        a0      pin;
        a1      pin;
"Sortides
        q3 .. q0   pin istype 'com';

Equations
        q0 = (!a0)&(!a1);
        q1 = (a0)&(!a1);
        q2 = (!a0)&(a1);
        q3 = (a0)&(a1);

test_vectors ([a1,a0] -> [q3,q2,q1,q0])
        [0,0] -> [X,X,X,X];
        [0,1] -> [X,X,X,X];
        [1,0] -> [X,X,X,X];
        [1,1] -> [X,X,X,X];

END
```

Codificador de 4 a 2

```
MODULE codificador42

title 'codificador 4 a 2';
C,X = .C.,.X.;

"Entrades
        a3,a2,a1,a0       pin;
"Sortides
        q1 .. q0   pin istype 'com';

Equations
        q0 = a1#a3;
        q1 = a2#a3;

test_vectors ([a3,a2,a1,a0] -> [q1,q0])
        [0,0,0,1] -> [X,X];
        [0,0,1,0] -> [X,X];
        [0,1,0,0] -> [X,X];
        [1,0,0,0] -> [X,X];

END
```

Divisor de freqüència de 10

Aquest bloc està realitzat de la mateixa manera que l'altre divisor però afegint dos estats més.

```
MODULE divisor10

title 'div freq de 10';
C,X = .C.,.X.;

"Entrades
        en        pin;
        clk       pin;
"Sortides
        out       pin istype 'reg';
        Q3..Q0   pin istype 'reg';
        sreg = [Q3,Q2,Q1,Q0];

Equations
        sreg.clk = clk;
        out.clk = clk;

state_diagram sreg

state 0:
        if (en) then 1 WITH out = 0;
        else 0 WITH out = 0;
state 1:
        if (en) then 2 WITH out = 0;
        else 1 WITH out = 0;
state 2:
        if (en) then 3 WITH out = 0;
        else 2 WITH out = 0;
state 3:
        if (en) then 4 WITH out = 0;
        else 3 WITH out = 0;
state 4:
        if (en) then 5 WITH out = 0;
        else 4 WITH out = 0;
state 5:
        if (en) then 6 WITH out = 0;
        else 5 WITH out = 0;
state 6:
        if (en) then 7 WITH out = 0;
        else 6 WITH out = 0;
state 7:
        if (en) then 8 WITH out = 0;
        else 7 WITH out = 0;
state 8:
        if (en) then 9 WITH out = 0;
        else 8 WITH out = 0;
state 9:
        if (en) then 0 WITH out = 1;
        else 9 WITH out = 0;


test_vectors ([clk,en] -> [out])
@repeat 50 { [C,1] -> [X]; }
@repeat 30 { [C,0] -> [X]; }
@repeat 20 { [C,1] -> [X]; }

END
```

Registre de desplaçament de 5 bits

Aquest bloc el vam realitzar utilitzant equacions de ABEL. Per a desplaçar vam utilitzar dos conjunts que representaven els 4 primers bits i els últims 4 bits. Per a carregar o desplaçar simplement assignem conjunts amb l'operador =.

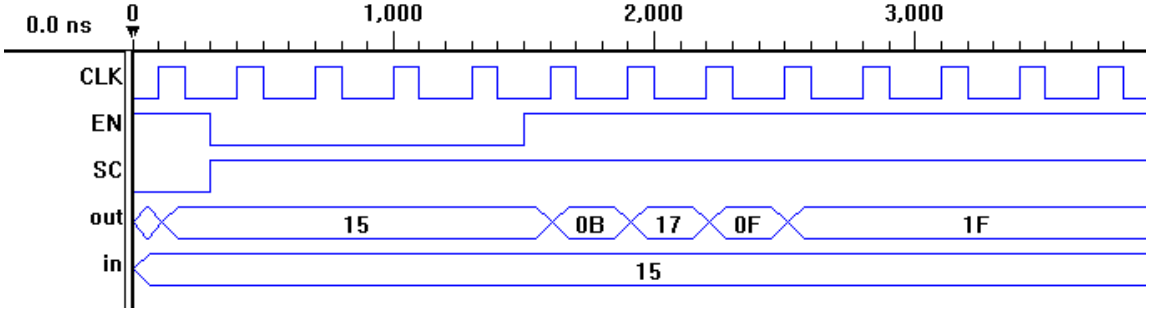
```
MODULE regdesp5
C,X = .C.,.X.;

"Entrades
        en      pin;
        sc      pin;
        clk     pin;
        d4..d0  pin;
"Sortides
        q4 .. q0   pin istype 'reg';
"Conjunts
        data = [q4..q0];
        inputs = [d4..d0];
        datah = [q4..q1];
        datal = [q3..q0];

Equations
        when en&(!sc) then data :=
inputs;
        else when en&sc then { q4 := 1;
datal := datah }

        when !en then data := data;

        data.clk = clk;

test_vectors ([clk,en,sc,d4,d3,d2,d1,d0]
 -> [q4,q3,q2,q1,q0])
        [C,1,0,1,0,1,0,1] -> [X,X,X,X,X];
        @repeat 4 { [C,0,1,1,0,1,0,1]
-> [X,X,X,X,X]; }
        @repeat 8 { [C,1,1,1,0,1,0,1]
-> [X,X,X,X,X]; }
END
```

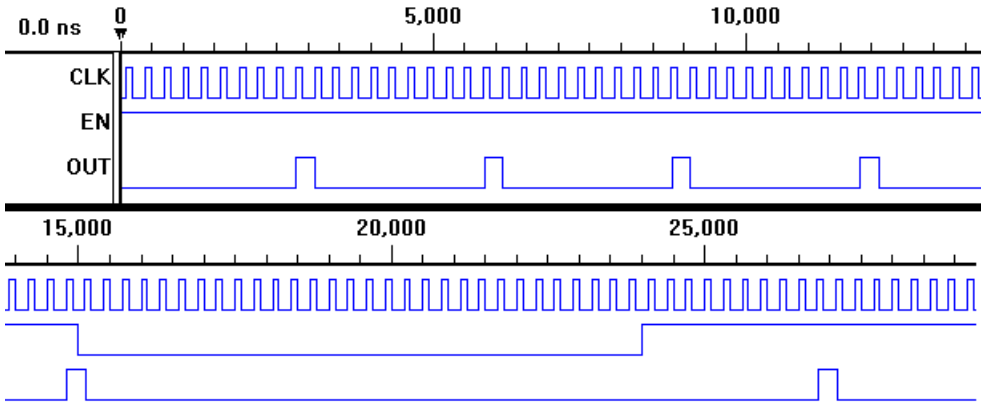
La resta de circuits queden iguals en principi, llevat d'una petita modificació en el registre paral·lel-sèrie. Cal retardar l'entrada de la senyal EN que va connectada al registre de desplaçament, ja que necessitem que EN, SC i KEY vagin sincronitzades per a carregar el valor al registre de desplaçament. Això es pot fer afegint un registre que retardi l'entrada o bé optant per afegir una operació lògica que activi la senyal EN si la senyal SC val 0. També es podria fer canviat les especificacions del registre de desplaçament.

Simulació

Abans de gravar el circuit cal simular-ne els components per a verificar que realitzen el que nosaltres volem. Aquí incloem algunes imatges de la simulació dels diferents blocs del circuit.

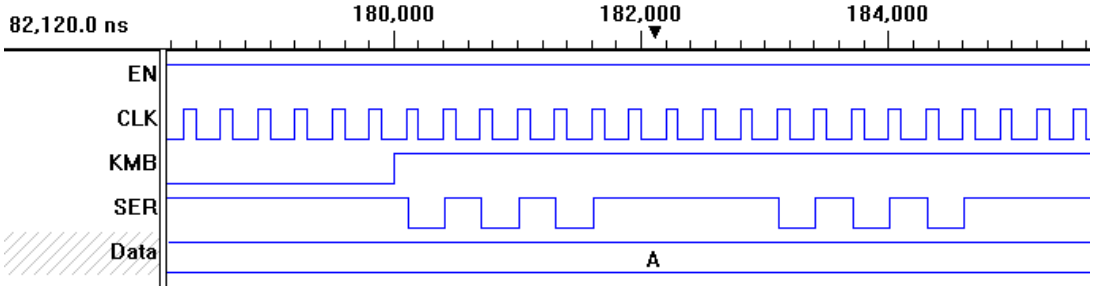
Registre de desplaçament. Es comprova el funcionament de SC i EN.



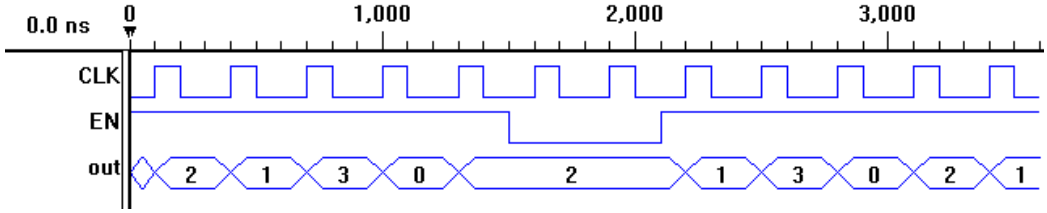


Divisor de freqüència entre 10. Només cal comprovar la senyal EN

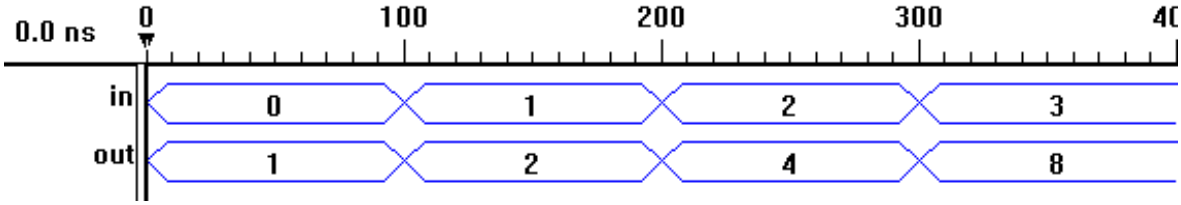
Conversor paral·lel-sèrie



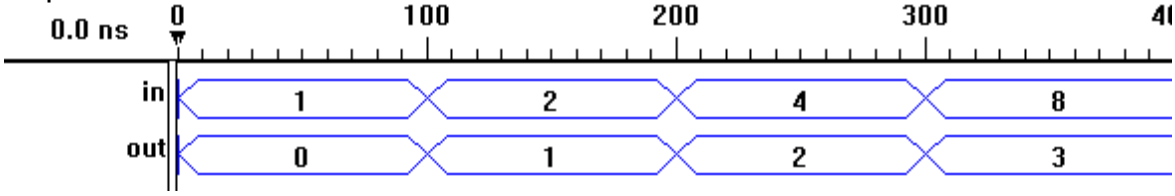
Comptador de 2 bits. Noteu que no compta si EN=0.



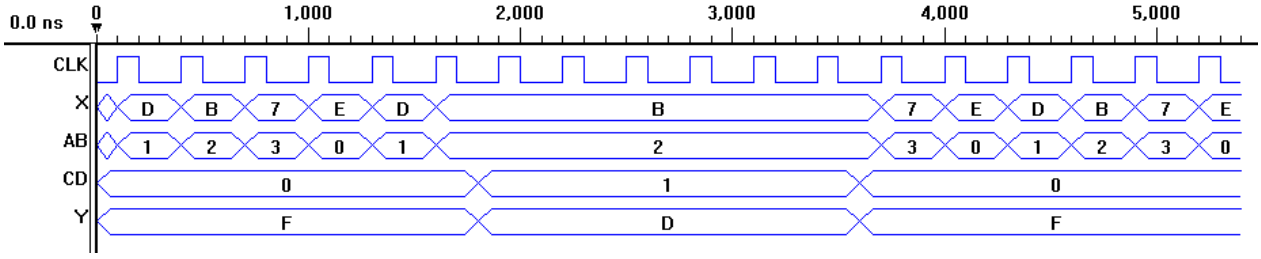
Descodificador 2 a 4. Les sortides i entrades estan en hexadecimal.



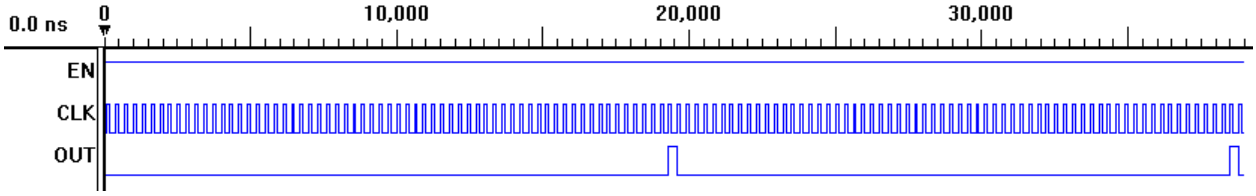
Codificador 4 a 2. No hem comprovat les codificacions no vàlides ja que estan prohibides.



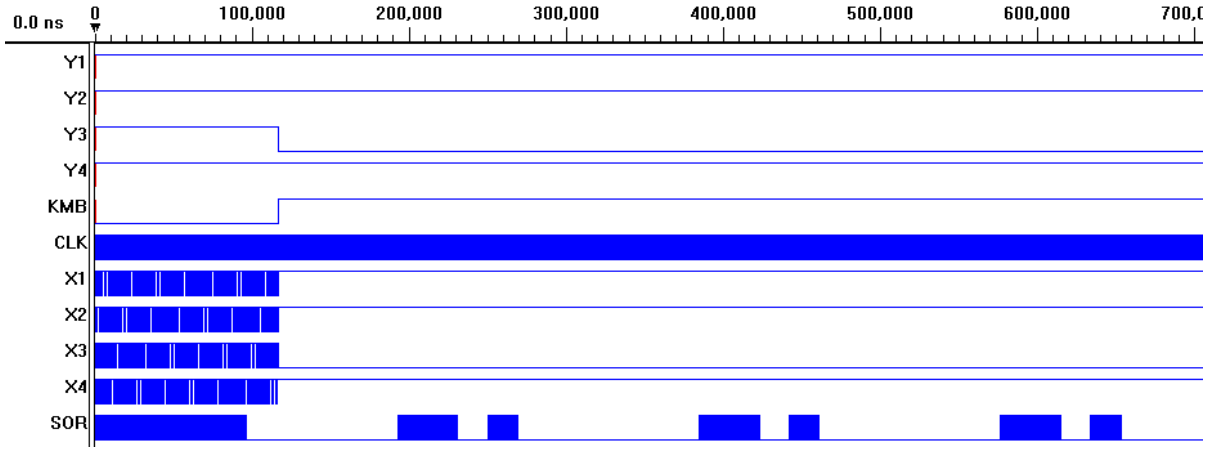
Conversor de teclat. En el moment que pitgem una tecla (alguna Y <> 0) s'atura.



Divisor de freqüència entre 64.

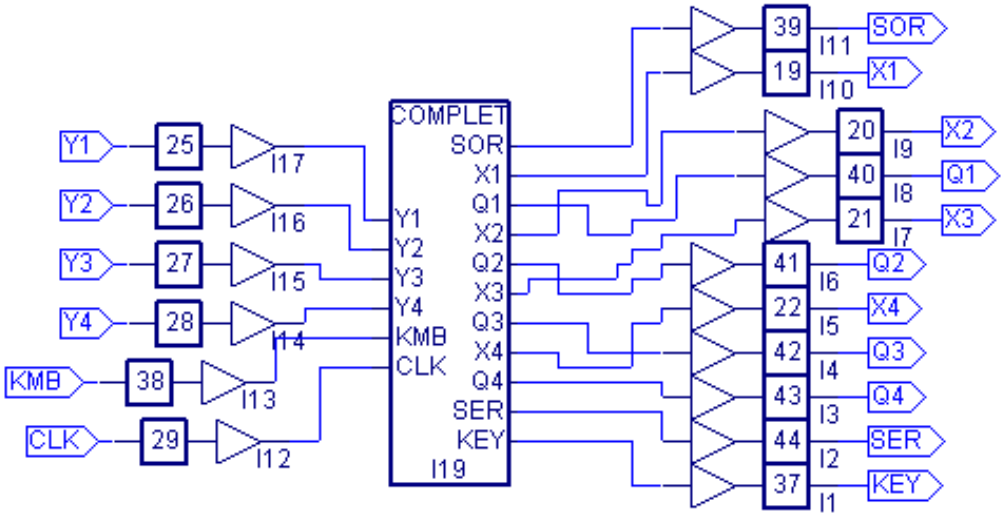


Circuit complet. Es pot veure la sortida dels leds a la senyal SOR.



Encapsulat i gravat a la PLD

Per a gravar la PLD es defineix un circuit a un nivell superior tal com es veu a la imatge:



Anomenant als pins amb el seu nombre corresponent que hi ha muntat al

circuit del comandament obtenim un circuit que podem gravar a la PLD. Un cop generat el fitxer JEDEC podem gravar-lo a la PLD i provar-lo. També es pot connectar a l'oscil·loscopi per a veure com les senyals es comporten al circuit real.

Qüestions

Q1: Taula de codificació de les tecles.

Cal notar que les codificacions aquí donades són en l'ordre en que s'emeten, a l'hora de rebre aquestes dades cal tenir en compte que el primer bit rebut serà el de menor pes.

AB\CD	00	10	01	11
00	Obrir 0h	Tancar 2h	(Buit) 1h	(Buit) 3h
10	Pujar 8h	Baixar Ah	Emmagatzemar 9h	Reproduir Bh
01	Enrere 4h	Endavant 6h	Esborrar 5h	Inici 7h
11	Antihorari Ch	Horari Eh	(Buit) Dh	(Buit) Fh

En principi les combinacions són arbitràries, l'únic requisit és que siguin les mateixes que posarem al receptor.

Q3: Si les sortides són configurades com a *push-pull* significa que no es poden connectar amb altres sortides, ja que *push-pull* és capaç de forçar la sortida al nivell que desitgi 0 o 1. En canvi si utilitzem una sortida en drenador obert necessitarem resistències de *pull-up*, ja que la sortida només serà capaç de forçar un 0.

Q4: Si no aturéssim el comptador seguiria escombrant tecles, i el codi que s'enviaria al conversor paral·lel-sèrie seria erroni, això provocaria que el codi enviat seria erroni i el robot no faria el que nosaltres volem. De fet quan escombrem una fila que no té cap tecla pitjada estem rebent un 1111, que al negar-lo queda 0000 i la sortida del codificador no està definida per a una entrada que no té només un 1.

Q5: Una emissió completa són 10 bits: 1 de start, 4 de informació i 5 de stop. Si sabem que enviem 512 bits/segon (bauds) tenim:

$$t_{emissió} = 10\,bits \cdot \frac{1\text{s}}{512\,bits} \approx 20\text{ms}$$

Q6: Si no utilitzem el divisor de freqüència de 10 a la senyal S/C succeeix que sempre desplacem el contingut del registre i mai el carreguem (si connectem KMB) o bé que només carreguem (si connectem $\overline{KMB}$). En qualsevol cas el codi que enviem es manté constant al llarg del temps, cosa que provoca que enviem informació no vàlida o que en el cas que sigui vàlida no serà la desitjada.

Q7: Si sabem que cada segon s'envien 512bits, en 10 segons s'enviarà 5120 bits, a 10 bits per codi són 512 codis en total.

Q8: Si utilitzem un divisor de 120 enlloc de un de 64 tenim que en 1 segon s'envien 273 bits aproximadament. Això vol dir que hem de refer tots els càlculs amb una velocitat de 273 bauds.

$$t_{emissió} = 10\,bits \cdot \frac{1\text{s}}{273\,bits} \approx 36.6\text{ms}$$

En 10 segons s'enviaran 273 codis al receptor.