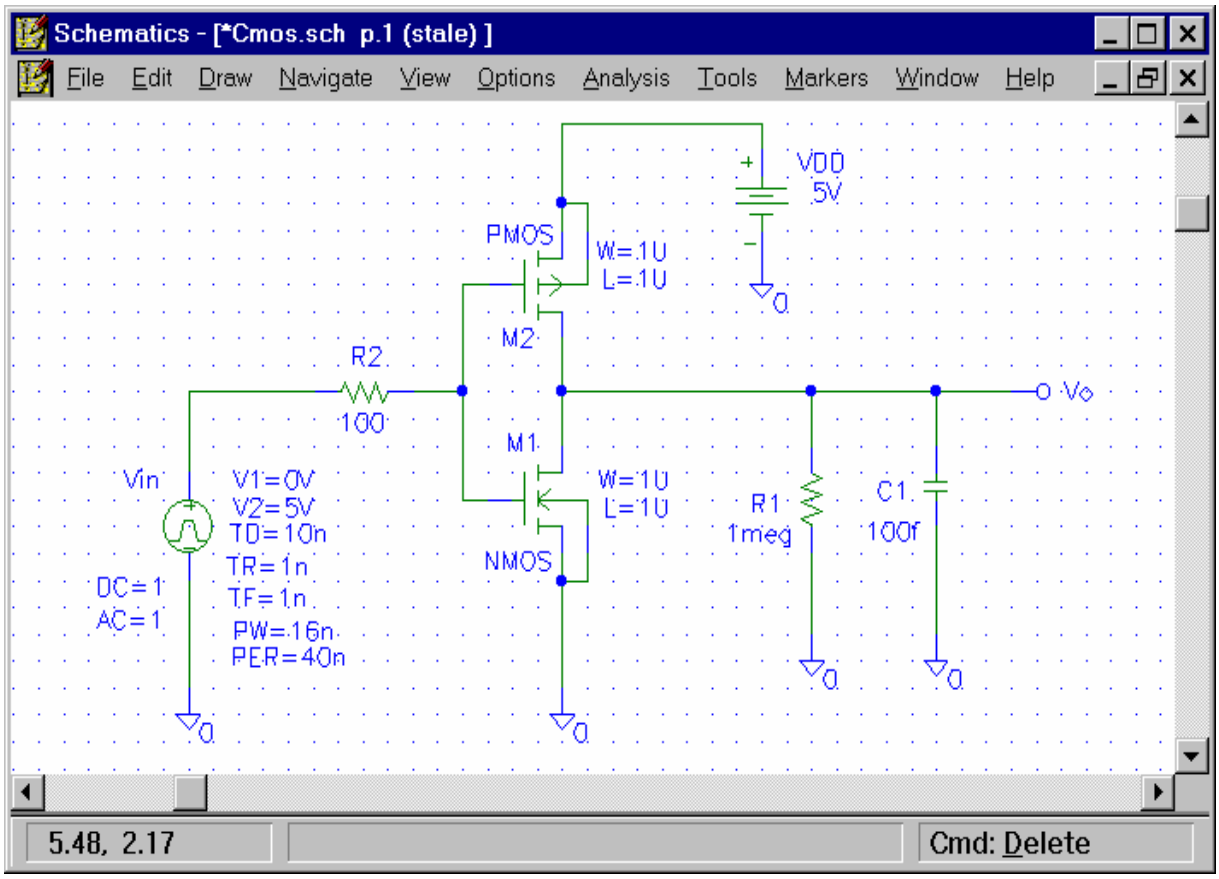
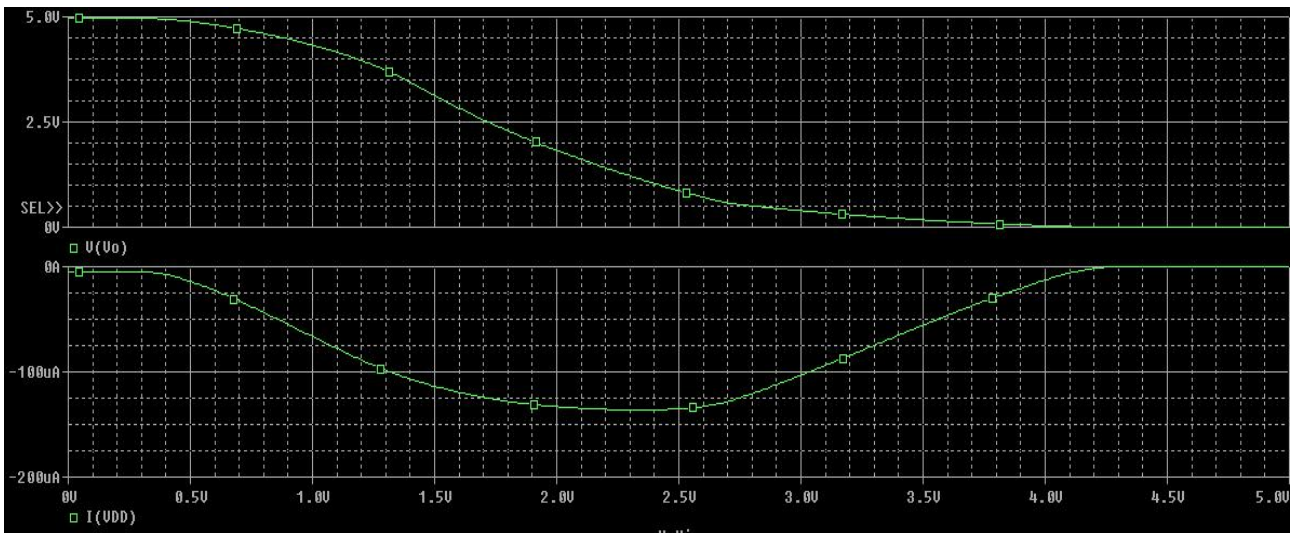
PRÀCTICA 1

Ens disposem a treballar sobre el primer dels circuits proposats:



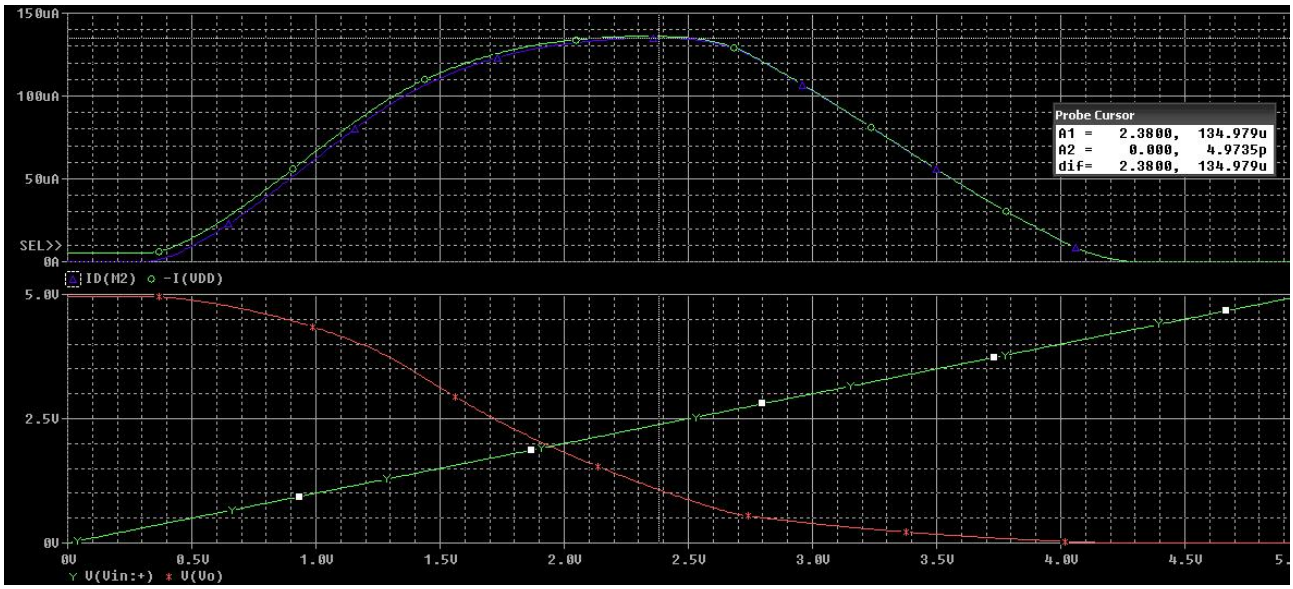
Q1. Per què és negatiu el corrent de VDD?

Verifiquem que efectivament la intensitat de Vdd és negativa. Això s'explica pel fet que la direcció de la intensitat correspon al node allà on el voltatge és més alt cap al node on és menor (en aquest cas de 5V cap a terra) però la intensitat positiva es considera la que "dóna" la font a la resta del circuit. Així, en ésser contraria a aquesta direcció que considerem positiva, la intensitat es mostra com a negativa.



Q2 i Q3: Representeu el corrent de drenador del transistor M1. On se situa el màxim del corrent respecte a les tensions? A què és degut? Quin és el seu valor? (es poden fer mesures numèriques amb la comanda *Cursor1*).
Quines diferències hi ha entre el corrent de drenador del transistor M1 i el de la font d'alimentació ? Per què ?

El corrent màxim es dóna a la zona on tots dos transistors estan conduint. La petita diferència entre la corrent que dóna la font i el corrent que passa pel drenador del transistor és el corrent que passarà per la resistència (molt poc, al ésser molt gran aquesta resistència).



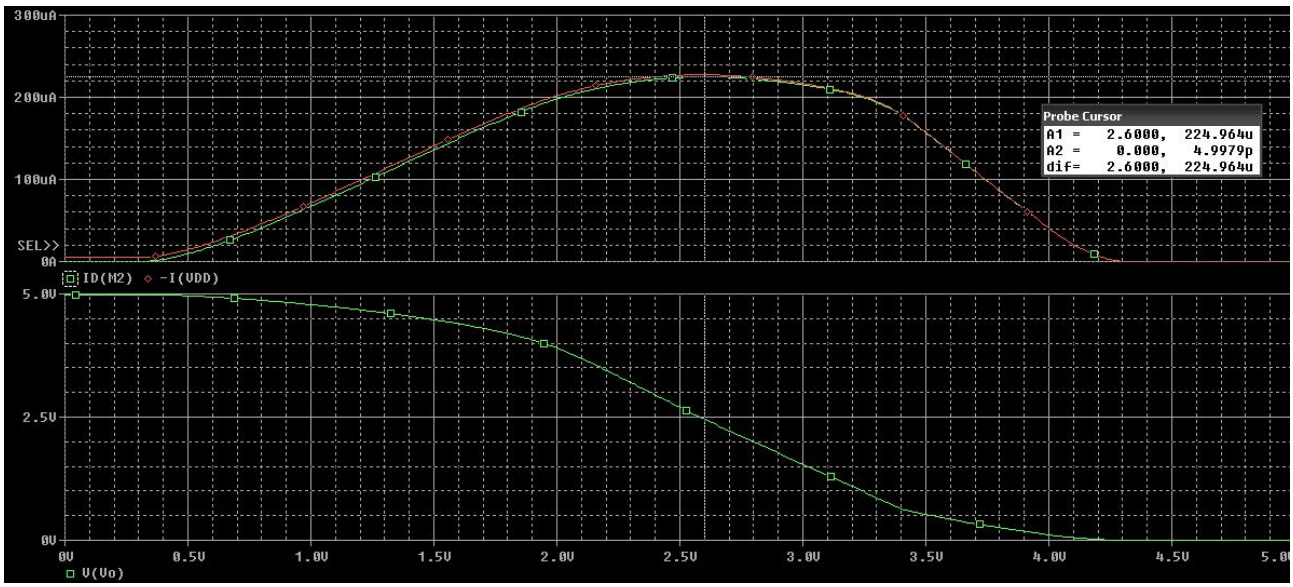
Q4: Quina és la potència dissipada per l'inversor CMOS quan aquest condueix el corrent màxim?

La potència dissipada podem calcular-la com el producte de la intensitat màxima que passa pel transistor pel voltatge entre les seves potes en l'esmentada situació.
En aquest cas la potència seria de aproximadament:

Pdissipada = 134 uA * 2.38V = **0.3 mW**

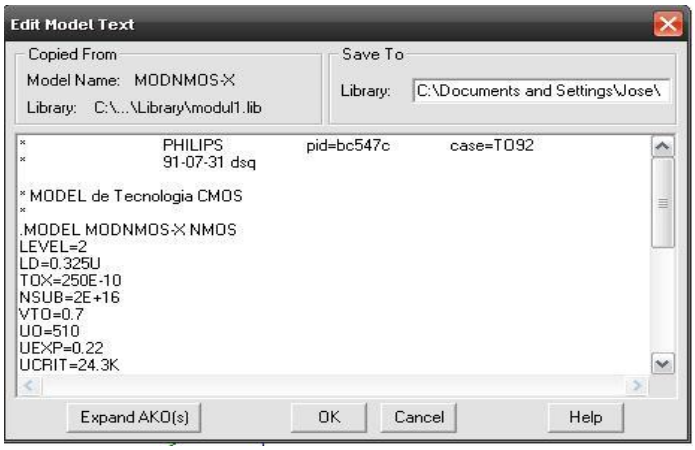
Q5: Tanqueu la finestra *Probe* i modifiqueu els valors d'W dels transistors des de la finestra *Schematics* (per exemple, W=3U al PMOS i W=1U a l'NMOS). Torneu a simular. Indiqueu quines modificacions apareixen a la característica de transferència i a què son degudes.

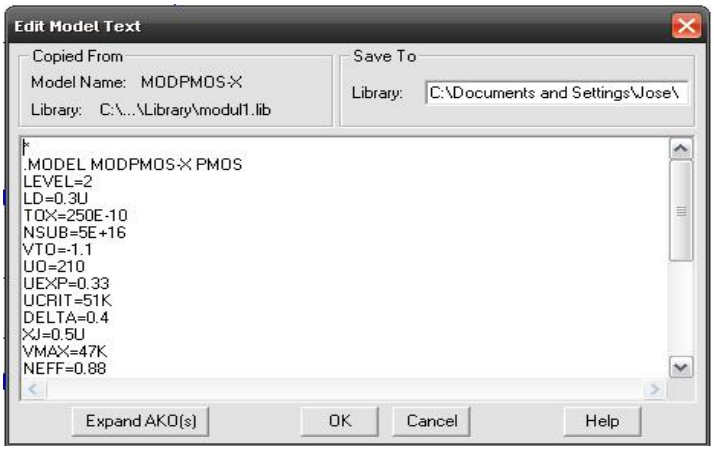
Podem observar una gràfica molt similar, amb alguna petita diferència deguda a que l'estructura física d'un dels transistors és diferent (i per tant condueix de forma diferent). Concretament podem observar que en aquest cas el transistor M2 té una intensitat màxima inferior que en el cas anterior i que aquesta es produeix amb un V d'entrada més alt.



Q6: Quant val la tensió llindar (VTO) del transistor PMOS que hem simulat? I la del transistor NMOS?

Per buscar les tensions llindars de tots dos transistors només cal buscar la dada a les taules següents:



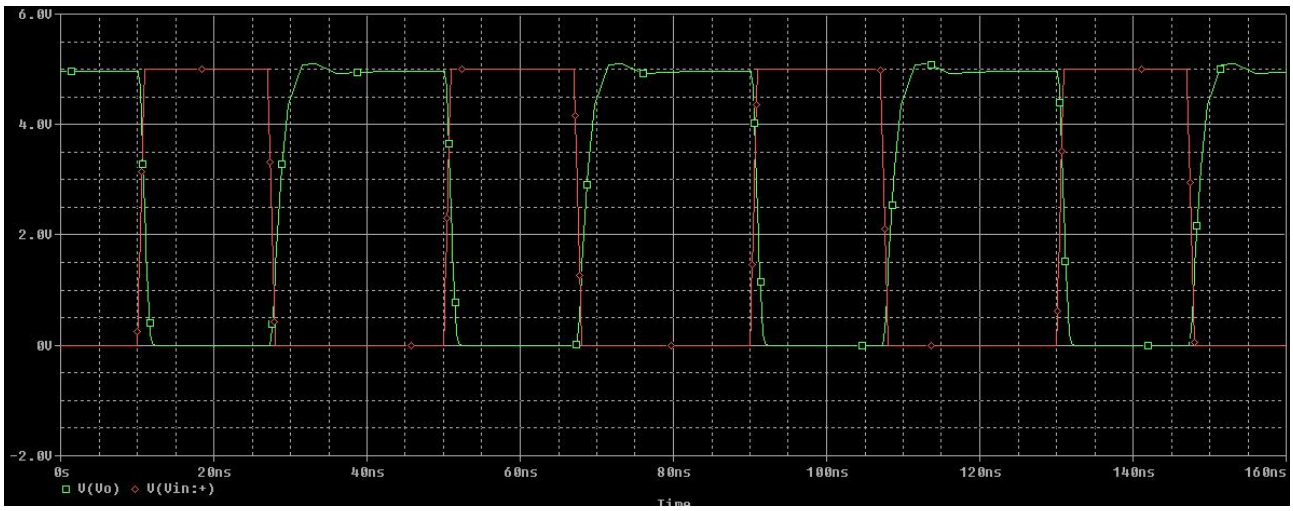


VTOnmos = 0.7 V

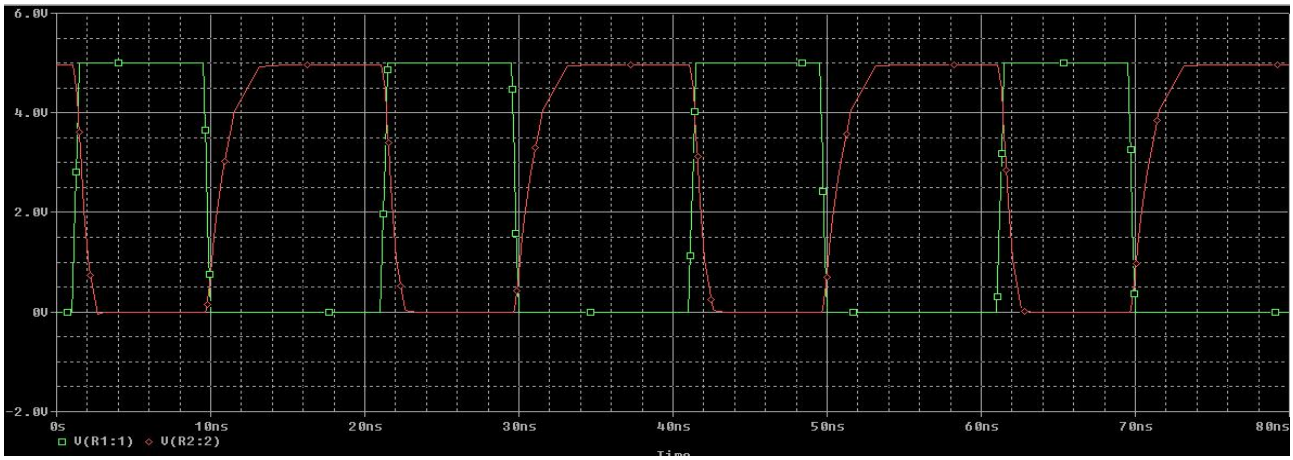
VTOpmos = -1.1 V

Q7 Reduïu els temps de l'anàlisi transitòria en un ordre de magnitud (feu-ho amb tots els valors de temps de la definició de la font Vin, així com amb els paràmetres de l'anàlisi transitòria). Què ha passat amb el senyal de sortida Vo respecte a la simulació anterior? Imprimiu la forma d'ona de la sortida Vo.

El transitori del circuit mostra la sortida:



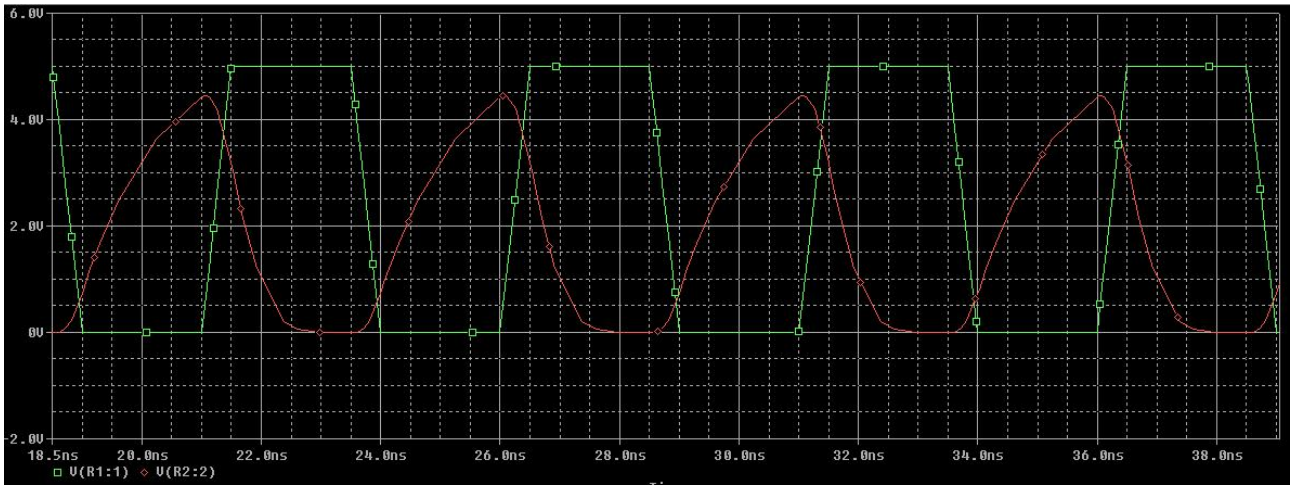
En reduir en un ordre de magnitud tots els temps que intervenen en la simulació (tant del pols d'entrada com de la simulació en sí mateixa) obtenim la sortida:



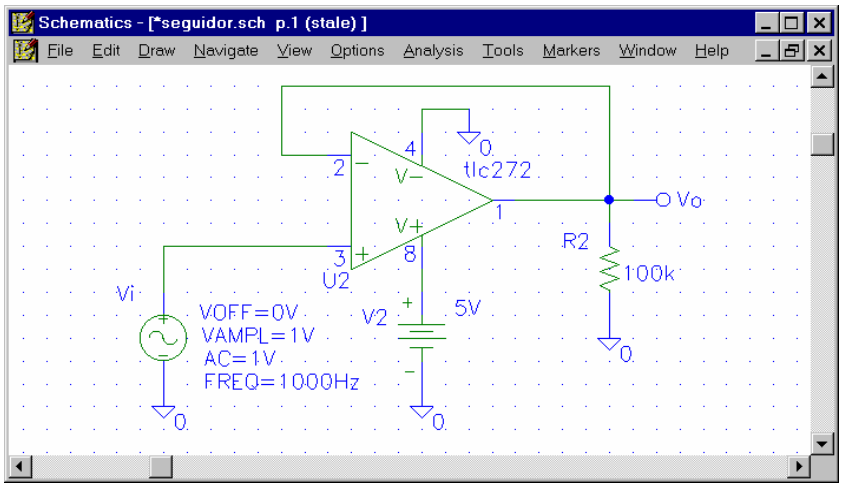
Observem que tot i que la duració del pols s'ha reduït, el temps d'estabilització no ho ha fet en la mateixa proporció. Això dóna pas al següent i últim apartat en què ens proposen reduir la duració del pols (augmentar la freqüència) fins que no tingui prou temps a estabilitzar la sortida.

Q8: Obteniu la freqüència màxima de funcionament correcte de l'inversor. Preneu com a un possible criteri de "funcionament correcte" que la tensió de presenti un nivell alt superior a 4 V i un nivell baix inferior a 1 V (Considerant que l'alimentació està compresa entre 5V i 0 V).

Augmentem la freqüència fins que el sistema no proporcioni una sortida correcte segons el criteri proposat a l'enunciat. Com podem veure a la imatge, això passa quan el període del pols d'entrada és de 5ns (200MHz).

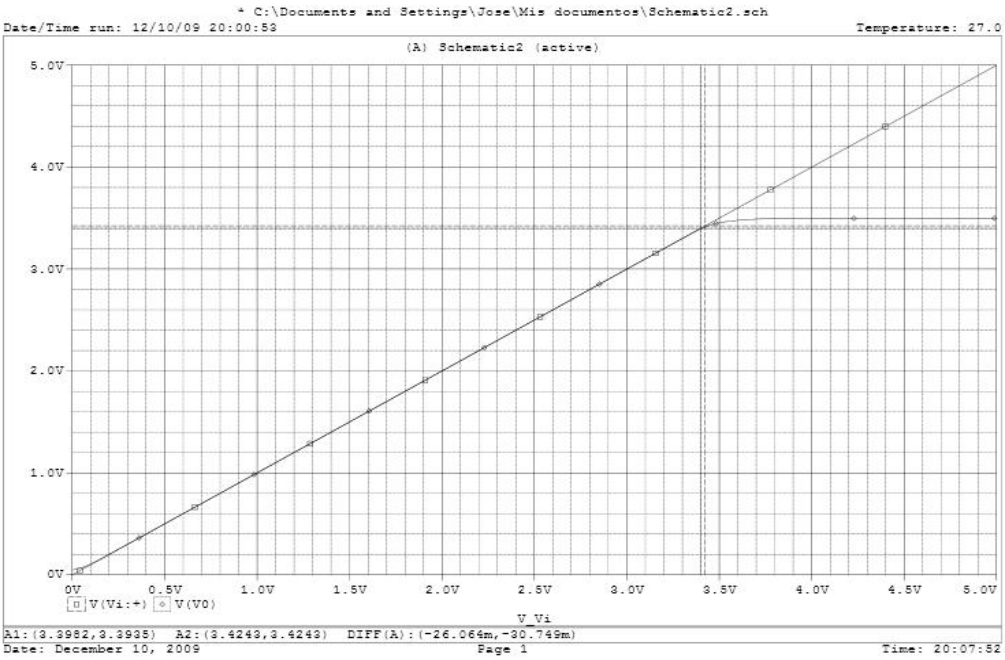


Circuit seguidor de tensió

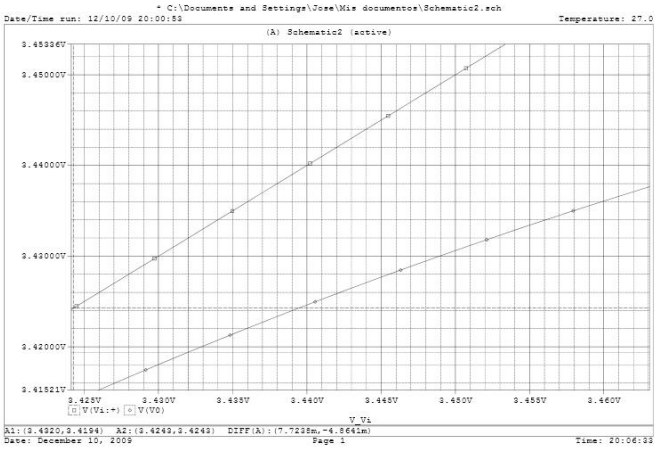


Q9: Per a quins marges de tensió d'entrada Vin l'A.O. es comporta com un seguidor amb un error menor que 10 mV? Quin marge de tensions de sortida té? Expliqueu com heu obtingut els resultats

A partir de la simulació en DC, observem que, tal com s'espera hi ha una certa tensió a la qual l'amplificador es satura i deixa de funcionar com un seguidor de tensió

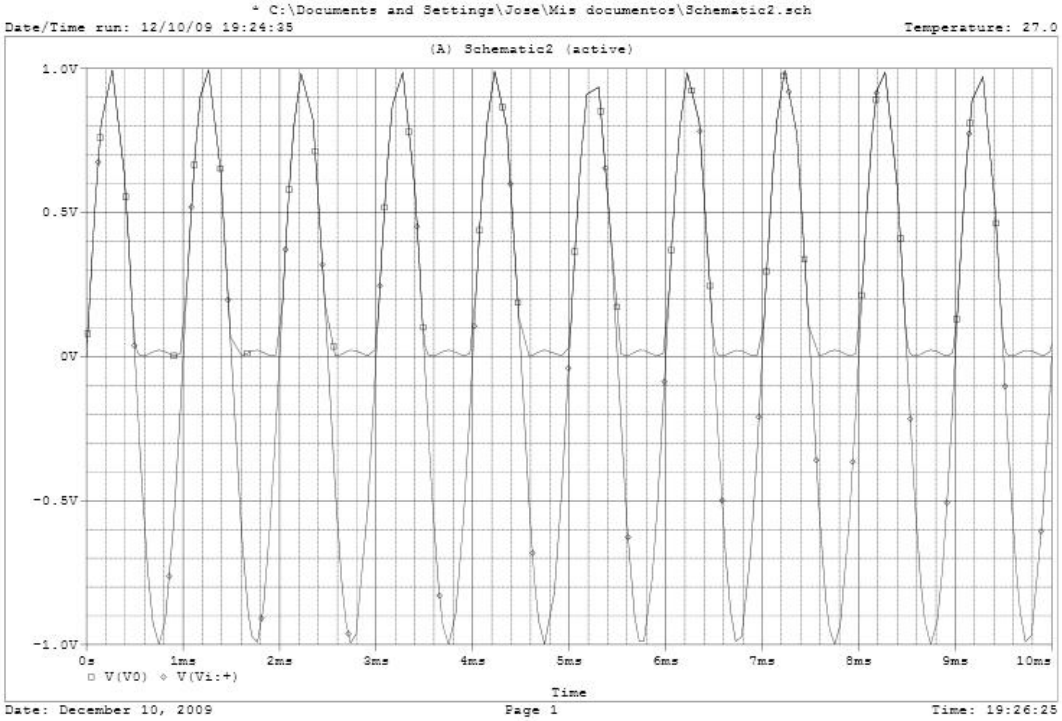


En aquesta imatge veiem aproximadament el valor de V(in) a partir de la qual l'error és més gran que 10mV (aprox. 3.44 V).

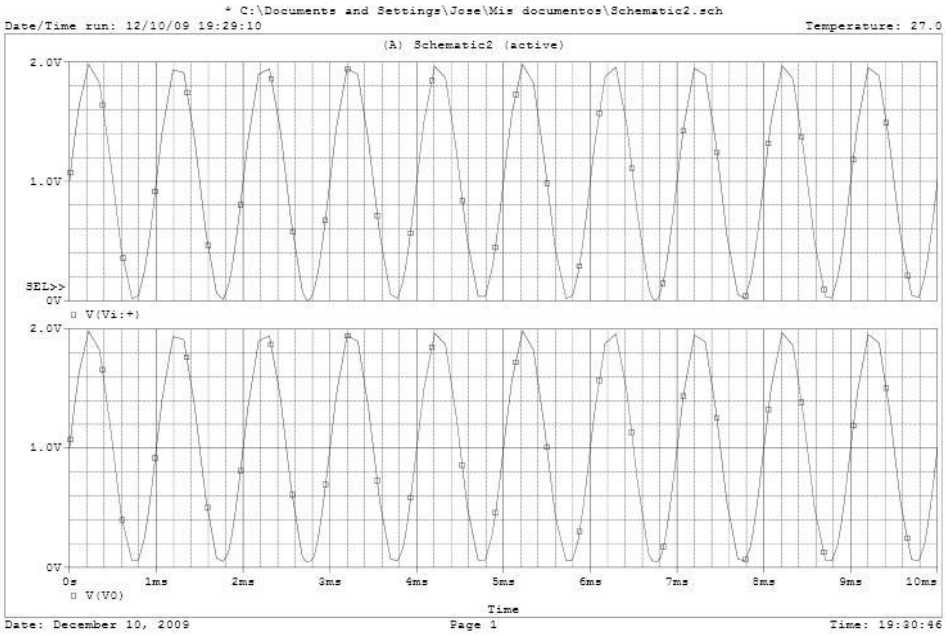


Q10: Si observem la forma d'ona de sortida a l'anàlisi transitòria, podem veure que està distorsionada. Per què?. Afegiu un nivell de continua d'1 V a la tensió d'entrada i observeu la sortida. Quina millora s'ha produït?

Observem com la sortida del seguidor, tal com esperem, força una sortida igual a la tensió d'entrada per tensions positives. Respecte a les tensions negatives podem observar que es produeix l'esperada distorsió deguda a la alimentació asimètrica de l'amplificador operacional (es retalla el senyal).

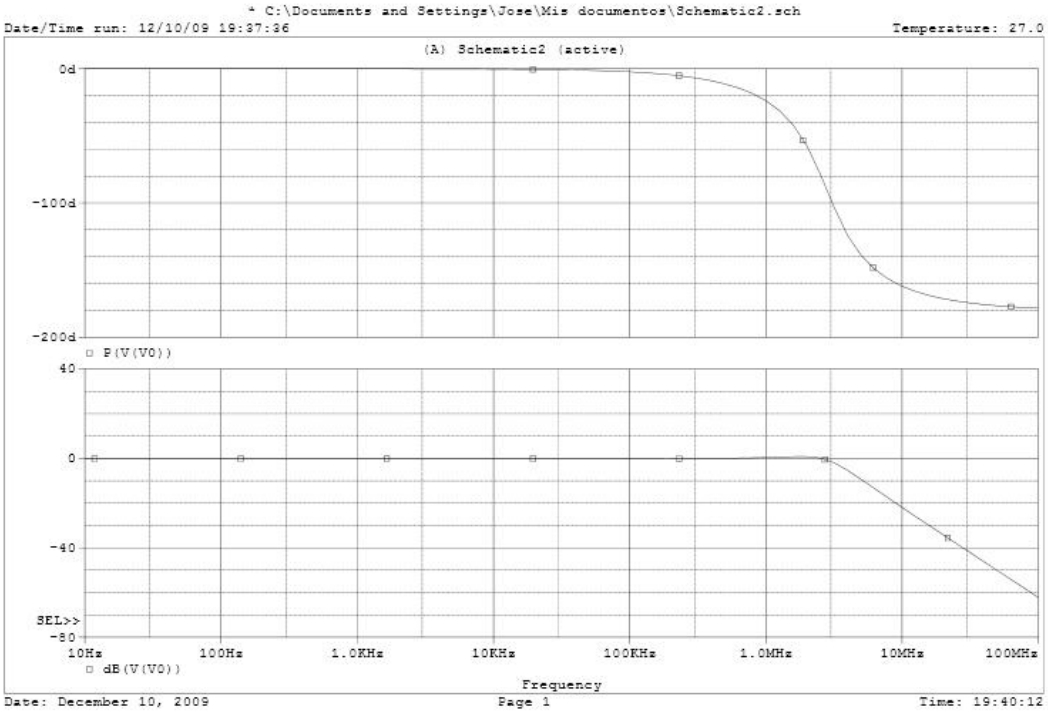


Si afegim 1V de continua a la entrada veiem que solucionem aquest problema ja que l'amplitud de l'entrada es troba en tot moment fitada per 2V i 0V, que són valors de tensió que l'amplificador sí és capaç de proporcionar amb l'alimentació asimètrica amb la que l'alimentem.



Q11: Quant val la freqüència de tall del seguidor?

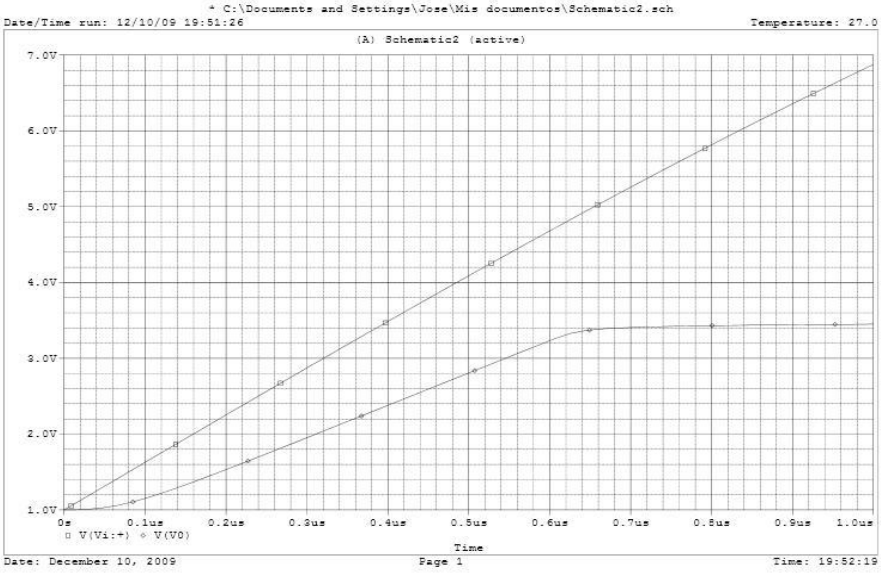
La freqüència de tall podem observar-la a la següent imatge. El pol de l'amplificador operacional es troba allà on apareix la recta de 20 DB/dècada. És a dir, la freqüència de tall és de aproximadament 2MHz (un pèl menys).



Q12: Mitjançant una simulació transitòria obteniu el *slew-rate* del TLC272.

Per buscar el valor del Slew Rate el que fem es posar una tensió gran a l'entrada i fem que l'entrada oscil·li a una freqüència alta. Així, la variació d'amplitud respecte el temps serà més gran que el màxim assolible per l'amplificador operacional (SR). El AO seguirà aquest augment de tensió tan ràpid com pot. Aquesta velocitat constant és definida com SR, així que observant el pendent de la recta podem determinar-lo:

SR = max dv/dt = (2.8V – 1.6V) / (0.5 us – 0.2 us) = 2.8e6



Q13: Contrasteu els valors obtinguts en les simulacions fetes en aquest apartat amb els paràmetres corresponents del full de característiques de l'A.O. donat pel fabricant. En particular compareu els valors de Ample de Banda, *Slew-Rate* i Marge de tensió de sortida.

Cerquem al datasheet del amplificador operacional els valors de Slew Rate, i de la freqüència de tall de l'amplificador i comprovem que efectivament els nostres càlculs s'ajusten prou bé a les dades ofertes pel fabricant:
Un ample de banda, tal com dèiem, inferior a 2 Mhz i un SR entre 2.5 i 2.9 (en el nostre càlcul 2.8).

B_1	Unity-gain bandwidth	V_I = 10 mV, C_L = 20 pF, See Figure 3	25°C	1.7		MHz
			0°C	2		
			70°C	1.3		
ϕ_m	Phase margin	V_I = 10 mV, C_L = 20 pF, f = B_1, See Figure 3	25°C	46°		
			0°C	47°		
			70°C	43°		

PARAMETER		TEST CONDITIONS		T_A	TLC272C, TLC272AC, TLC272BC, TLC277C			UNIT
					MIN	TYP	MAX	
SR	Slew rate at unity gain	R_L = 10 kΩ, C_L = 20 pF, See Figure 1	V_{IPP} = 1 V	25°C		3.6		V/μs
				0°C		4		
				70°C		3		
			V_{IPP} = 2.5 V	25°C		2.9		
				0°C		3.1		
				70°C		2.5		

Pràctica 2

Exercicis previs

L'objectiu de l'estudi previ és simular el circuit a realitzar i adequar-lo per a que compleixi les especificacions i els requisits.

1. Determini el valor mínim del guany del filtre per a garantir una amplitud de 1Vpp al senyal de sortida quan l'emissor es troba a mig metre del capçal receptor.

El receptor capta 100mVpp a uns 50cm del emissor. Si volem 1Vpp necessitem un guany de 10.

2. Suposi fixat C_f, determini el valor que ha de tenir cada una de les resistències en funció de k, Q, w_0 y C_f.

$$H(s)=\frac{-1}{R_{f1}\cdot C_f}\cdot\frac{s}{s^2+\frac{2\cdot s}{C_f\cdot R_{f5}}+\frac{R_{f1}+R_{f2}}{R_{f1}\cdot R_{f2}}\cdot\frac{1}{C_f^2\cdot R_{f5}}} \qquad H(s)=\frac{k\cdot(\omega_0/Q)\cdot s}{s^2+(\omega_0/Q)\cdot s+\omega_0^2}$$

D'aquestes fórmules aïllem terme a terme cada una de les resistències i obtenim:

$$R_{f1}=\frac{-Q}{C_f\cdot\omega_0\cdot k} \qquad R_{f5}=\frac{2\cdot Q}{C_f\cdot\omega_0} \qquad R_{f2}=\frac{Q}{C_f\cdot\omega_0\cdot(2\cdot Q^2-k)}$$

3. Faci l'elecció de C_f i determini el valor numèric de R_{f1}, R_{f2} i R_{f5}. Raoni la idoneïtat de la tria.

Triem C_f = 3,3nF i obtenim les valors per a les resistències de:

Rf1 = 737Ω Rf2 = 122Ω Rf5 = 14,7Ω

Considerem que aquests valors són adequats ja que estan dins d'un valor raonable. Són majors de 100Ω i menors de 100KΩ.

4. Dissenyi R_{f20} i P_{f2} per a que es pugui ajustar R_{f2} amb un valor nominal entre +/- 20%.

$$P_{f2}=2\cdot\frac{20\cdot R_{f2}}{100}\simeq 50\,\Omega \qquad R_{f20}=R_{f2}-\frac{P_{f2}}{2}\simeq 100\,\Omega$$

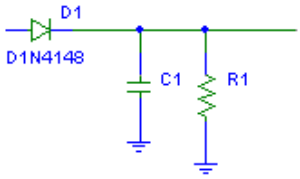
5. Dissenyi R_{f6} i R_{f7} per a que la tensió a l'entrada no inversora de l'operacional es trobi a la meitat del rang de sortida de l'operacional. S'ha de garantir que per aquestes resistències no circula un corrent superior a 5mA.

Suposarem que l'operacional és ideal i, en la pràctica, rectificarem aquesta

imperfecció modificant aquest divisor de tensió a un valor més baix. Utilitzarem dues resistències iguals (V=2,5V) de més de 1kΩ. D'aquesta manera segur que complim la restricció.

6. Determini el valor mínim de la sortida del detector Vd_{min} que es donarà quan no hi hagi senyal portadora (nivell "0").

7. Determini el valor màxim de la sortida del detector Vd_{max} que es donarà quan la sortida de l'operacional sigui màxima (nivell "1").



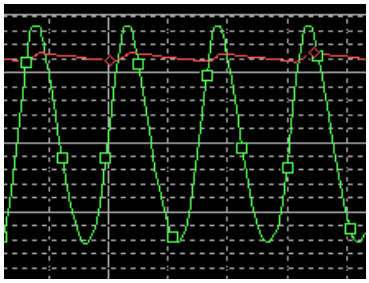
La sortida de l'operacional va connectada a un díode 1N4148 que té una V_γ d'aproximadament 0,6 V. Això provoca que davant una sortida de nivell 0 la sortida de l'operacional sigui la meitat del seu recorregut (2,5V si el considerem ideal), per tant la sortida serà

Vd_{min} = 2,5V – 0,6V = 1,9V

En el cas del nivell alt serà quan l'amplificador estigui saturat (5V si el considerem ideal):

Vd_{max} = 5V – 0,6V = 4,6V

8. Tenint en compte la freqüència portadora de la transmissió, determini la constant de temps mínima que garanteixi que l'arrissat sigui menor al 10% del màxim de la senyal.



L'arrissat es produeix quan la sinusoide portadora passa a través del díode. En el moment en que aquesta puja la intensitat és positiva i el díode condueix i carrega el condensador. En el moment que la senyal disminueix el díode es talla i el condensador es descarrega. La repetició d'aquest procés produeix un arrissat.

$$V_c(t)=V_{cf}+(V_{c0}-V_{cf})\cdot e^{\frac{-t}{\tau}} \qquad 0\text{V}+(4.6\text{V}-0\text{V})\cdot e^{\frac{-T}{\tau}}=\frac{90}{100}\cdot 4.6\text{V}$$

L'equació de descàrrega del condensador considerant que està carregat i es vol descarregar fins arribar a 0V. T és el període de la senyal portadora.

$$4.6\text{V}\cdot e^{\frac{-1/32678}{\tau}}=4.14\text{V} \quad \rightarrow \quad \tau_{min}=\frac{-1/32678}{\ln(4.14\text{V}/4.6\text{V})}=0.29\,ms$$

9. Tenint en compte la velocitat de transmissió calculi la constant de temps màxima per a que el flanc de baixada de la senyal d'entrada tingui un retard inferior al 25% del temps de bit. Consideri el valor mig de 0 i 1 com a llindar de comparació.

$$V_{llindar}=\frac{1.9\text{V}+4.6\text{V}}{2}=3.25\text{V} \qquad 512\,bits/s\rightarrow T_{bit}=1.9\text{ms}$$

$$t=\frac{25}{100}\cdot T_{bit}=0.47\text{ms}$$

Imposem que el voltatge al cap del 25% del temps de bit sigui el del llindar de comparació ("nivell 0").

$$4.6\text{V}\cdot e^{\frac{-0.47\cdot 10^{-3}s}{\tau}}=3.25\text{V} \qquad \tau_{max}=\frac{-0.47\cdot 10^{-3}s}{\ln(3.25\text{V}/4.6\text{V})}=1.35ms$$

10. Tenint en compte els resultats anteriors trii un valor per a la constant de temps que no comprometi el disseny posterior ni forci a l'operacional a donar un corrent excessiu.

Triem un condensador petit per a que l'operacional hagi de subministrar poc corrent C_f = 3.3nF.

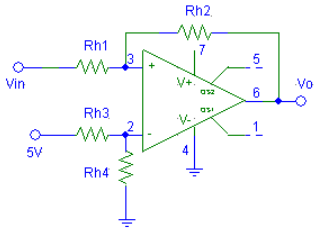
Utilitzem una resistència de 200kΩ el que implica $\tau=R\cdot C=0.66\,ms$ que està a dins del marge calculat anteriorment.

11. Determini l'amplada del cicle d'histèresi que ha de ser el doble del màxim arrissat de Vd.

$$V_{ih}-V_{il}=2\cdot 0.46\text{V}=0.92\text{V}\approx 1\text{V}$$

12. Determini la relació entre V_{ih} i V_{il} i els valors dels components del circuit.

$$V_{\text{im}}=\frac{V_{ih}+V_{il}}{2}=\frac{1.9V+4.6\text{V}}{2}=3.25\text{V} \qquad V_{ih}=3.75\text{V}$$
$$V_{ih}-V_{il}=1\text{V} \qquad V_{il}=2.75\text{V}$$



Ara que coneixem el cicle d'histèresi calculem els valors necessaris de les resistències.

$$V_{il}=V_{ref}\cdot\left(\frac{R_{h1}}{R_{h2}}+1\right)-\frac{R_{h1}}{R_{h2}}\cdot 5\text{V} \qquad \frac{R_{h1}}{R_{h2}}=0.2$$
$$V_{ih}=V_{ref}\cdot\left(\frac{R_{h1}}{R_{h2}}+1\right) \qquad V_{ref}=3.125\text{V}$$

On V_{ref} és la tensió a l'entrada inversora.

13. 14. Determinar el valor de les resistències del comparador.

Donat que aquest circuit no ha de produir efectes de càrrega sobre l'anterior posarem una R_{h1} molt gran. Així per exemple R_{h1} = 2MΩ i R_{h2} = 10MΩ.

Per a V_{ref} utilitzarem un divisor de tensió amb unes resistències prou grans per a que hi circuli poc corrent.

$$V_{ref}=\frac{R_{h4}}{R_{h3}+R_{h4}}\cdot 5\text{V}=3.125\text{V}\rightarrow\frac{R_{h4}}{R_{h3}}=1.66$$

Triem R_{h3} = 10kΩ i R_{h4} = 16.6kΩ

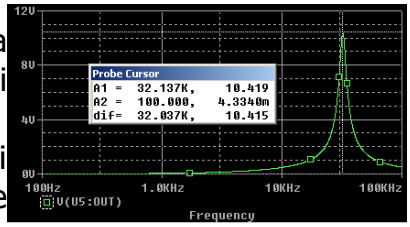
Simulació del receptor d'infrarojos

Ara realitzarem una simulació del circuit amb els valors calculats a l'estudi previ.

1. Realitzeu una simulació del capçal i comproveu que la corrent i la tensió són les esperades.

La sortida del capçal és una senyal quadrada de freqüència 32678Hz que és modulada amb 3 bits: 0,1,0.

2. Realitzeu la gràfica AC del filtre i comproveu que compleix les especificacions.



Realitzem un AC sweep per a trobat la resposta impulsional. Comprovem que k = 10, w_0 = 32kHz i Q = 5.

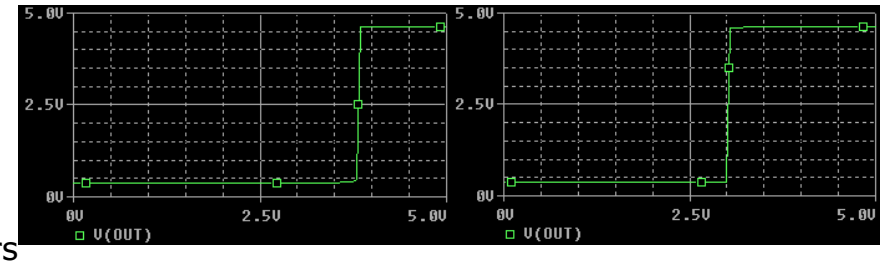
Per a calcular Q busquem a quines freqüències hi ha un guany de -3dB respecte al màxim i fem ús de la fórmula.

$$Q=\frac{f_0}{f_2-f_1}$$

3. Si heu realitzat alguna modificació en els valors del circuit calculeu de nou els paràmetres w_0, Q i k.

En principi no cal canviar els valors a la simulació, ja que es la desitjada. En tot cas serà després durant el muntatge que haurem de variar els valors per adequar-los a valors comercials.

4. Obteniu la gràfica de Vo(Vd) i evelueu V_{ih} i V_{il}.



Realitzem un DC sweep entre 0 i 5V de pujada i un altre entre 5V i 0V de baixada per obtenir el cicle d'histèresi.

Obtenim els valors calculats, la única diferencia és la sortida de l'operacional que en valors alts dóna aproximadament 4,1V enlloc del 5V que donaria si fos ideal.

5. Verifiqueu el circuit complet.

La sortida és l'esperada: 0,1,0. A més els valors de voltatge són 0V per a nivell baix i aproximadament 4,1V per a nivell alt, que compleix amb l'especificació TTL. Comprovem les amplituds de les senyals intermèdies.

La sortida del detector és un pols amb un arrissat que és molt petit i que no s'acosta al marge del comparador. El retard de 0 a 1 és d'aproximadament 0.4ms, que és inferior als 0.47ms que ens exigien les especificacions.

La sortida del filtre és una sinusoïdal centrada en 2,5V sense soroll afegit, que significa que el filtre funciona correctament.

Mesura experimental del receptor d'infrarojos

1. Varieu la freqüència per a trobar el guany màxim. Aquesta serà la freqüència central.

2. Ajusteu el potenciòmetre per aconseguir variar la freqüència central del filtre.

El filtre muntat ens dona una freqüència de pas lleugerament menor a la desitjada. Donat que hem instal·lat un potenciòmetre en en lloc de R_{f2}, podem variar lleugerament la freqüència de pas a costa de variar també K i Q. En el nostre cas utilitzem R_{f5} = 15kΩ i R_{f1} = 820Ω. És per això que tenim que ajustar el potenciòmetre a uns 110Ω aproximadament.

3. Obteniu el guany del filtre a la freqüència central.

El guany és d'aproximadament 9.

4. Obteniu les freqüències de tall a 3dB i obteniu la Q del filtre.

Les freqüències de tall a 3dB són aproximadament 36.5KHz i 28.5KHz, que fan un factor de qualitat de 4.

5. Obteniu la característica entrada-sortida del comparador.

6. Obteniu els valors de V_{ih} i V_{il}.

Vil = 2.8V, Vih = 3.4V. Com es pot veure hem desplaçat el cicle d'histèresi una mica avall per tal d'aconseguir una millor detecció. Això és degut a que la sortida de l'operacional no és ideal i a que la V_γ del díode és bastant elevada, cosa que provoca que la detecció no sigui tan bona.

A més a més el nivell alt de sortida és aproximadament 4,27V, mentre que el baix és ideal (0V).

7. Determineu el soroll a 100Hz de la il·luminació fluorescent.

9. Comproveu l'amplitud del soroll a 100Hz després de passar pel filtre.

Si mesurem a la sortida del capçal receptor observem un soroll prou gran, aproximadament 400mVpp. En canvi després de passar pel filtre el soroll s'ha atenuat bastant i només podem mesurar uns 50mVpp. Tot i que sembli que el soroll és massa gran (ja que el filtre té una atenuació gran a 100Hz) cal dir que la mesura és pos precisa degut als errors de mesura així com el soroll de les altres freqüències que no distingim.

8. Determineu l'amplitud de Vc quan l'emissor es troba a mig metre

del receptor.

L'amplitud de Vc és molt elevada, en la majoria d'ocasions satura i produeix una entrada distorsionada. Malgrat això el circuit segueix funcionant, ja que les components freqüencials del senyal són filtrats pel filtre. És per aquest motiu que el filtre és una part crítica del circuit, més que el comparador o el demodulador.

10. Estimeu a quina distància la sortida del filtre té una amplitud de 1V.

En el nostre cas la distància es ben petita, d'aproximadament 25cm. És per això que una de les possibles solucions que s'en acut per arreglar-ho és doblar el guany del filtre. Això ho aconseguim modificant Rf1 i Rf2. Si realitzem aquesta modificació la distància és de 0,5m aproximadament.

En cas que volguéssim augmentar encara més la distància seria recomanable afegir un segon filtre idèntic al ja existent. Així, a més de tenir el doble de guany atenuaríem més els sorolls i la sortida seria més fiable.

11. Obteniu l'arrissat del detector quan es detecta senyal.

L'arrissat és d'aproximadament 100mVpp, que és l'esperat a la simulació (uns 130mVpp).

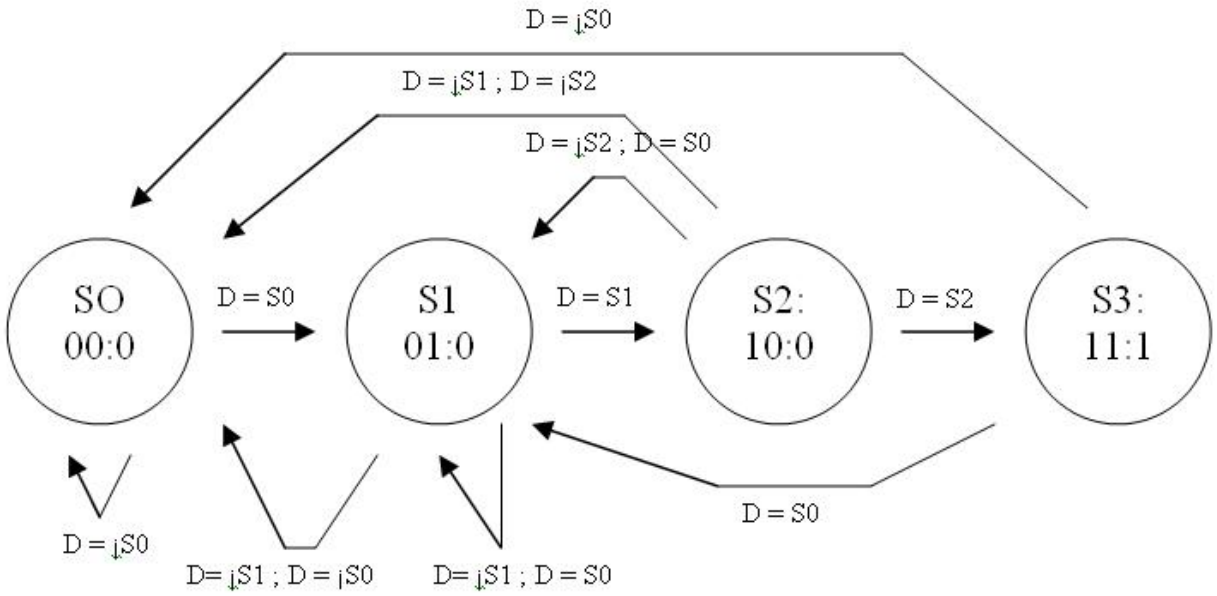
12. Verifiqueu que la sortida del comparador és l'esperada. Digueu a quina distància màxima opera el receptor.

La sortida del comparador és correcta. Si volem una millor detecció de la senyal podríem baixar una mica més el cicle d'histèresi, ja que en absència de senyal tenim una entrada de 1,5V aproximadament. El llindar de comparació és més alt, pel que seria segur abaixar-lo una mica més.

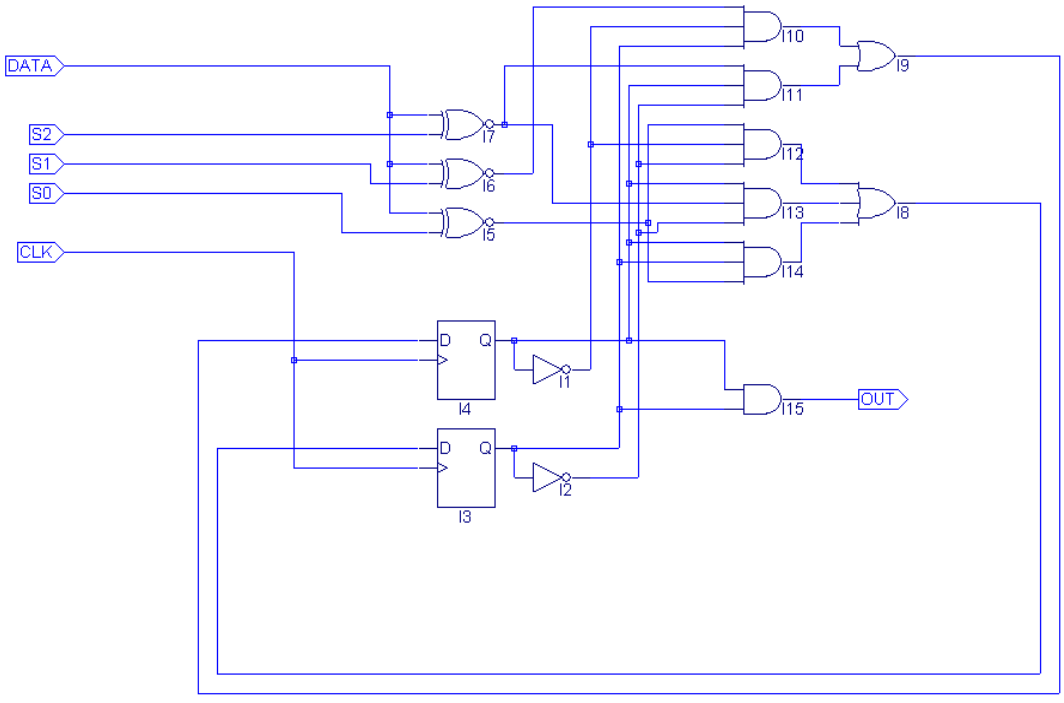
De tota manera una millora del filtre sempre és prioritària, ja que un millor filtratge garanteix que la senyal serà lliure de soroll, que afecta molt al comparador.

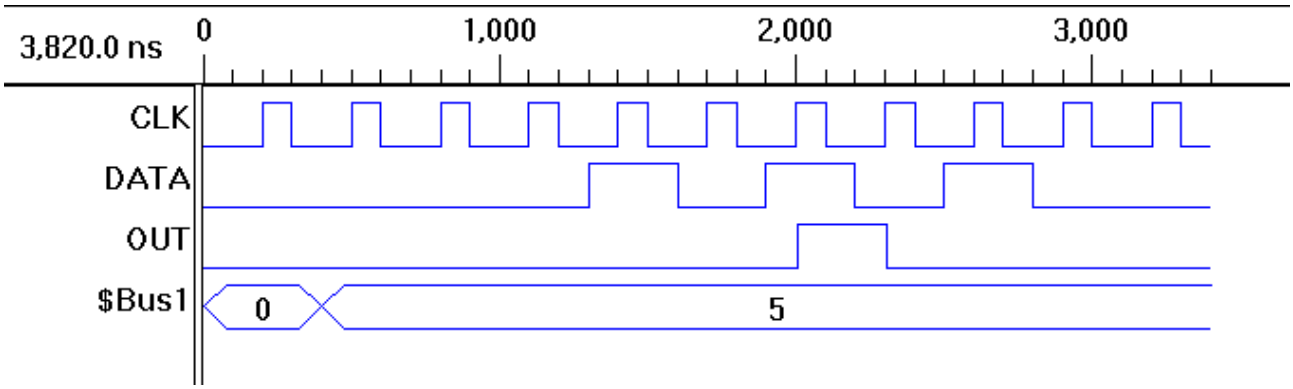
PRÀCTICA 3

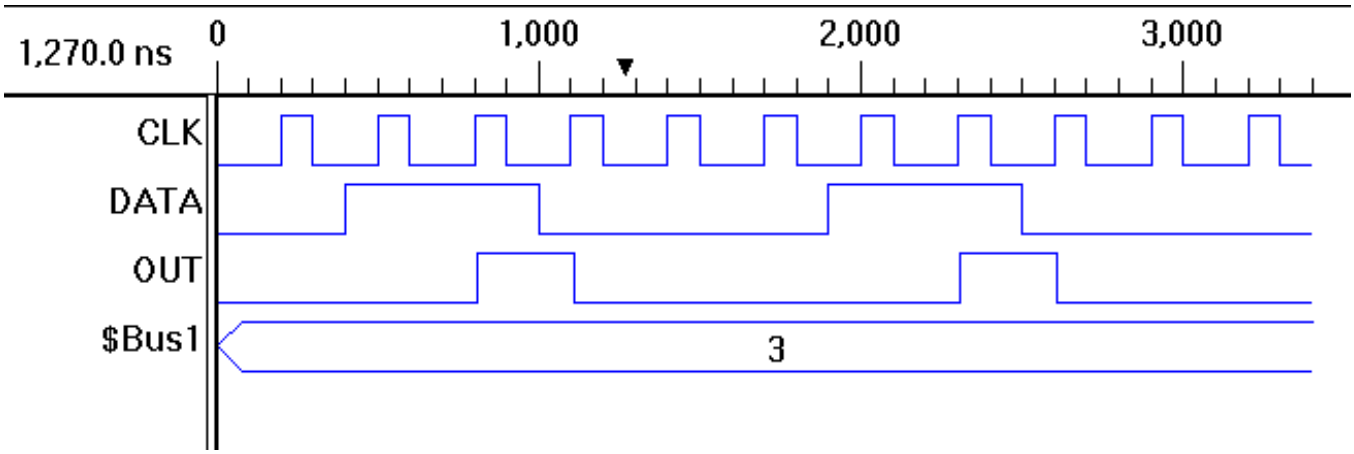
El diagrama d'estats de Moore que usarem per dissenyar el detector de seqüència és el següent:



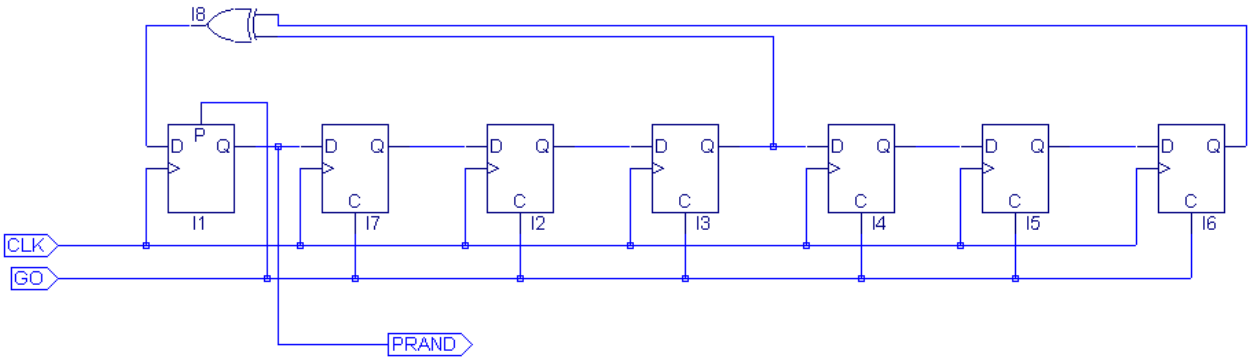
El disseny que es comporta com el diagrama de Moore exposat i la seva simulació s'exposen a continuació on comprovem que sigui capaç de reconèixer una determinada seqüència (en el primer cas 101 i en el segon 011):





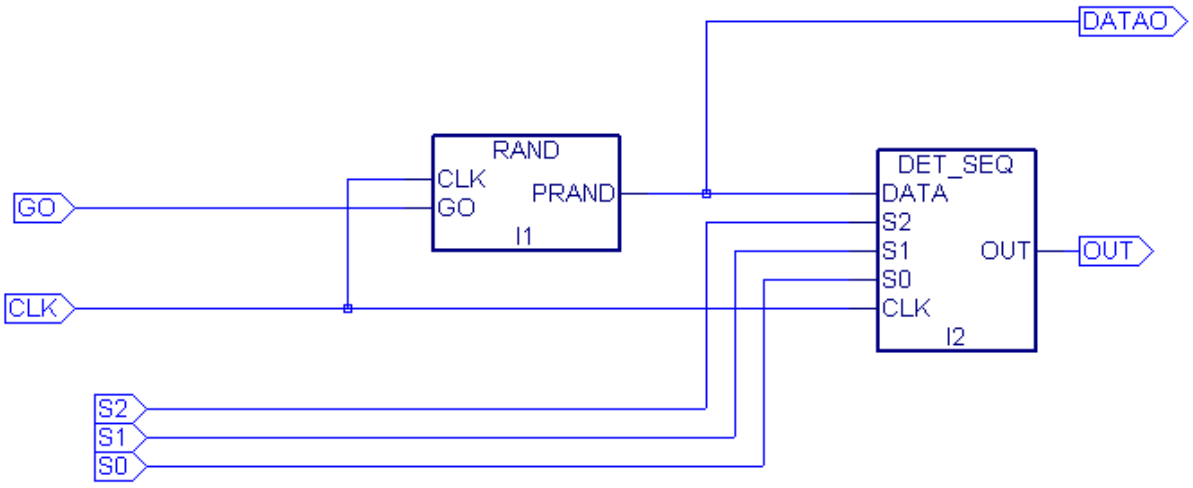


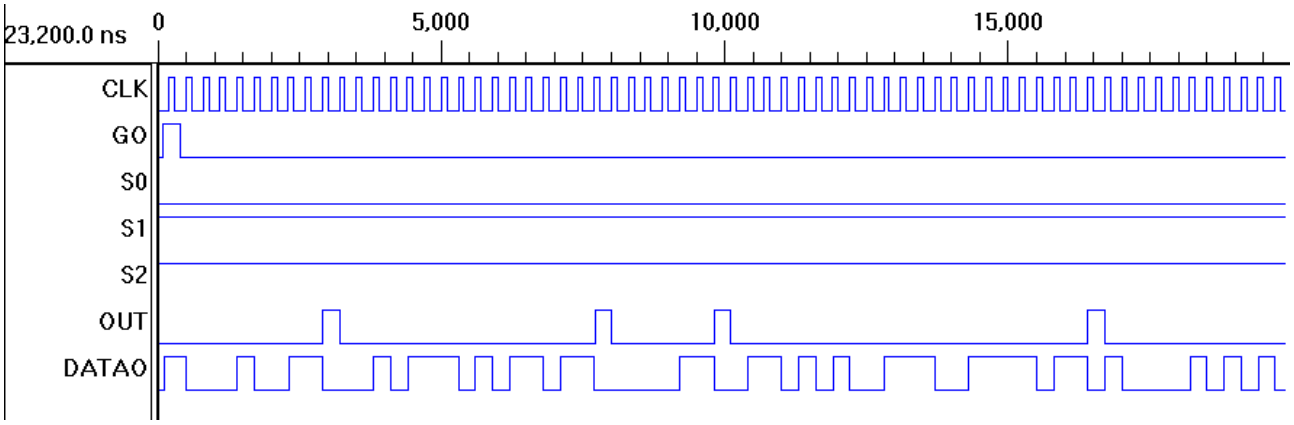
Un cop fet això muntem el generador de seqüència aleatòria proposat a la pràctica:



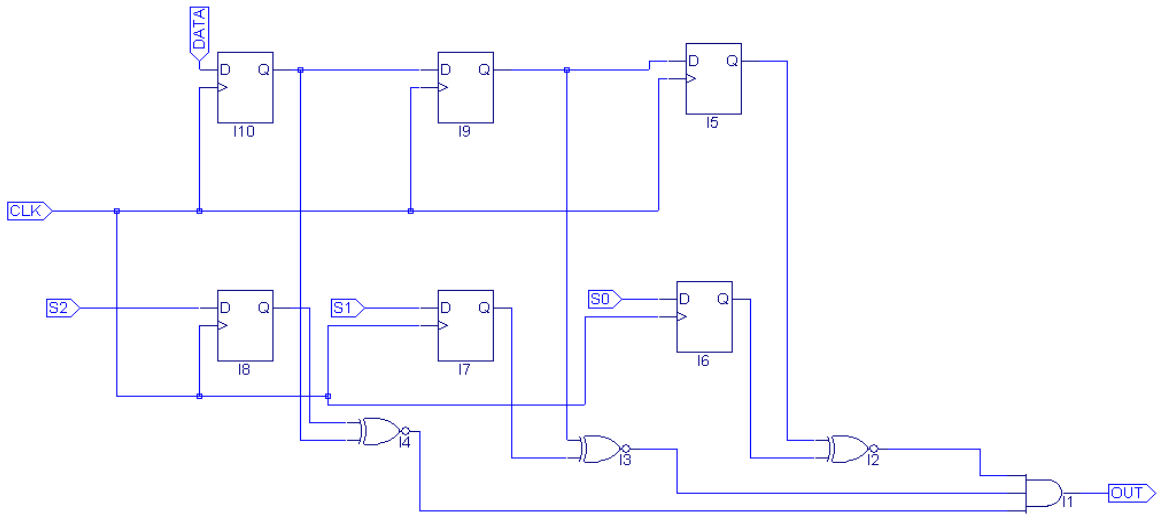
El muntatge del circuit complert (detector de seqüència + generador seqüència aleatòria) té el comportament esperat tal com podem veure a la simulació. Tot i així observem que, tal com hem dissenyat els estats del circuit, el sistema perd moltes seqüències vàlides. Això és pel fet que si rebem un bit que no correspon a la seqüència que esperem detectar un cop que hem començat la detecció de la seqüència en qüestió no és capaç de retornar més que a l'estat inicial.

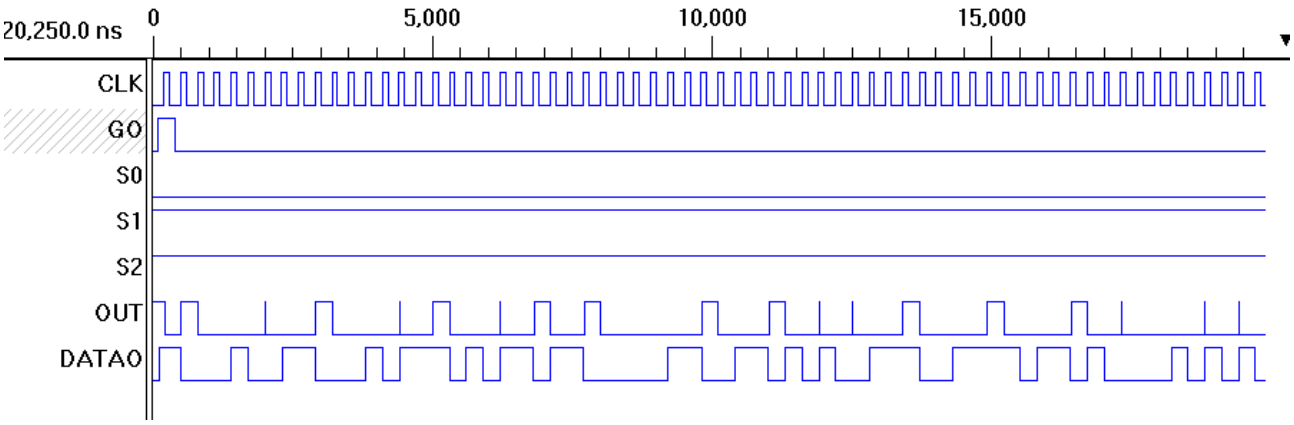
* Les simulacions següents es fan detectant la seqüència 011.





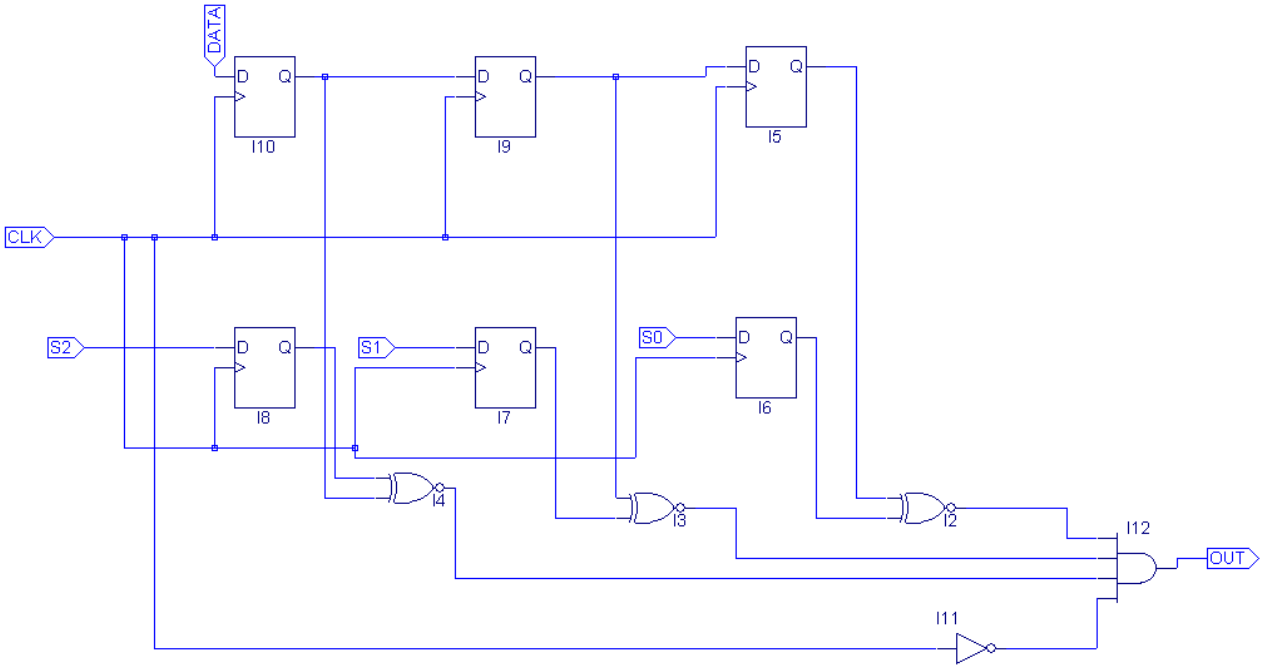
La complexitat d'implementar aquest circuit per tal de corregir aquest problema pot ser costosa, però podem optar per un sistema ben senzill i prou efectiu en el qual utilitzarem alguns biestables 'extres'.

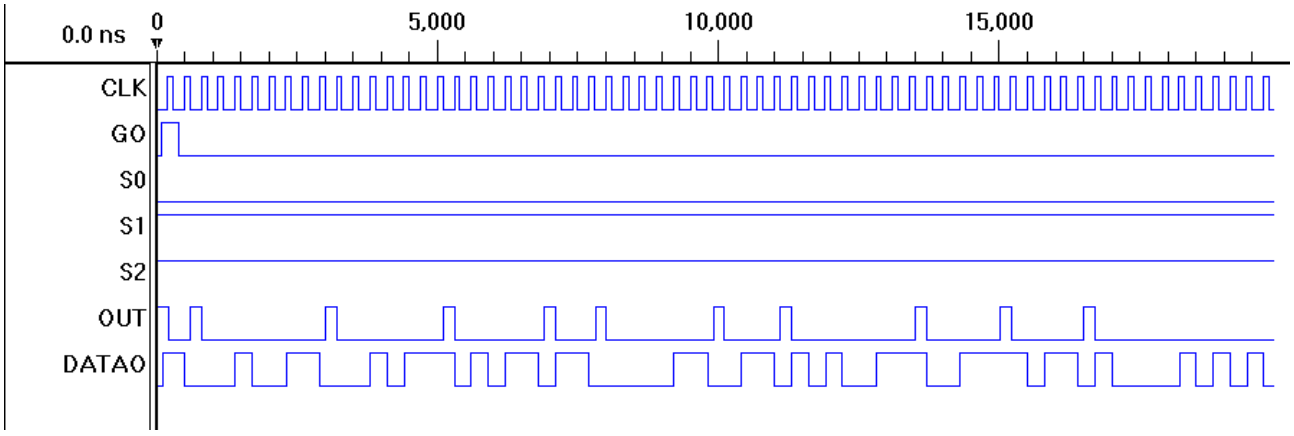




A la simulació veiem que apareixen glitches que no volem. Tenim 2 opcions per solucionar-ho.

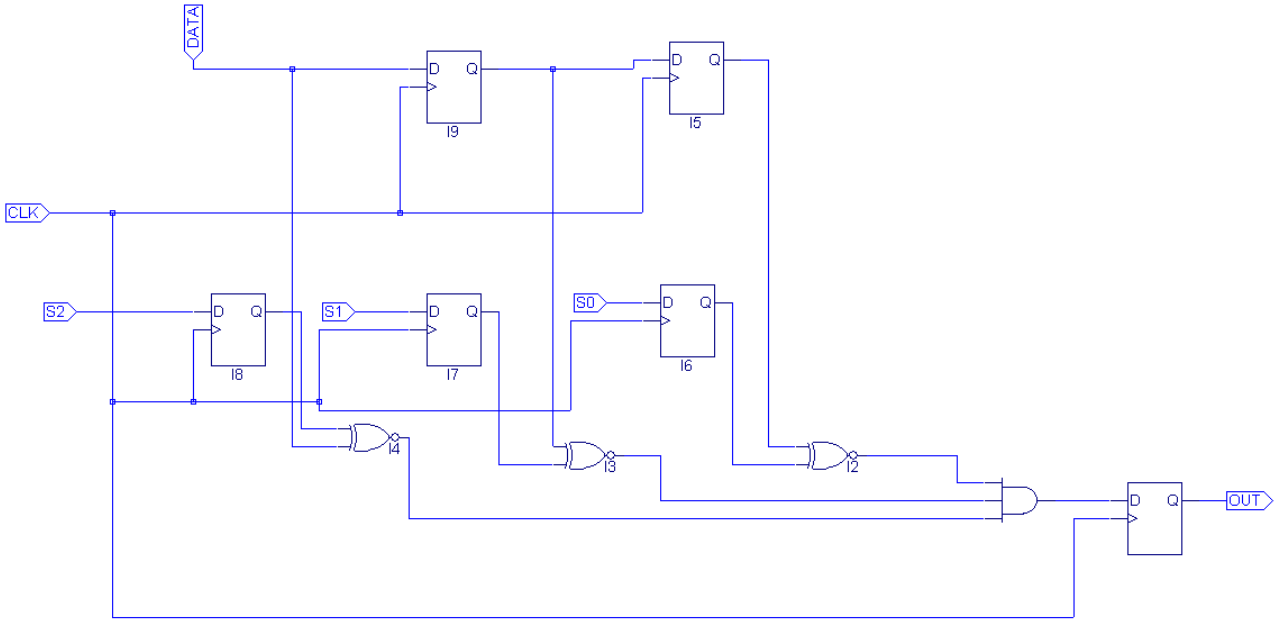
– Una un pèl rústica, que consistirà en regular la sortida validant-la únicament quan el rellotge està a nivell baix. Aquesta té el problema de que no garantim la sortida esperada, sinó que la sortida indica que s'ha detectat la seqüència únicament durant mig cicle

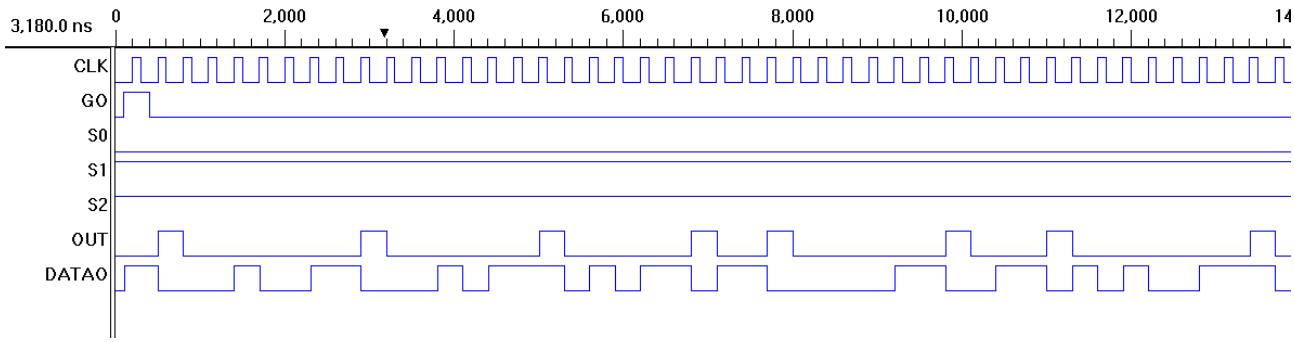




– Una segona solució millor és col·locar un biestable a la sortida del circuit,

de manera que la mantindrà estable durant tot el cicle. Amb aquesta solució sí que aconseguim el nostre objectiu: sortida estable durant tot un cicle i no perdem cap seqüència vàlida.





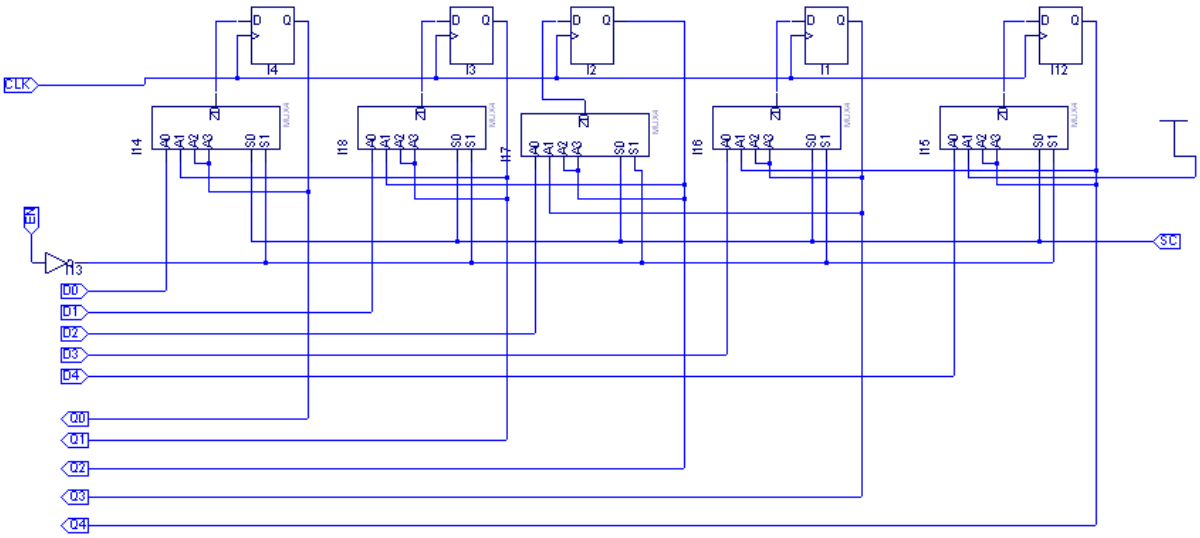
Pràctica 4

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.

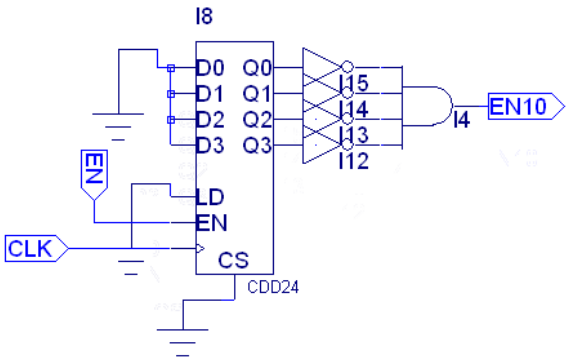
Registre de desplaçament

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)

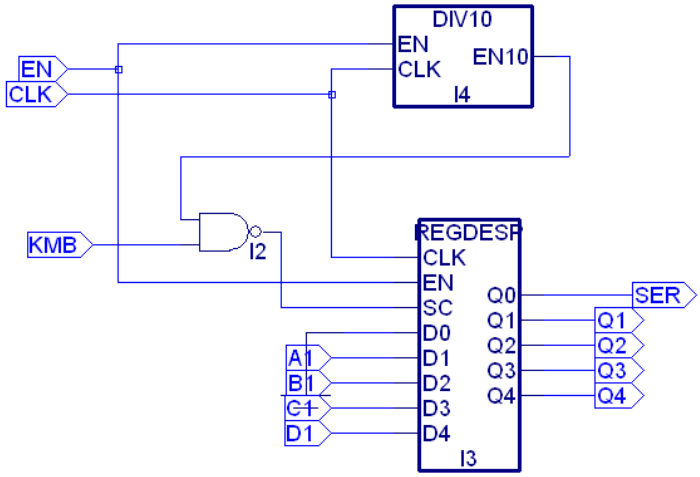


Aquest bloc és molt senzill. Simplement ha de treure un 1 a la seva sortida cada 10 cicles de rellotge. Per a implementar-lo necessitem un comptador. En el cas del ispLever tenim a les llibreries un comptador de mòdul 10. Farem una AND amb les sortides del

comptador invertides, d'aquesta manera quan el comptador valgui tot 0 traurem un 1 a la sortida.

Conversor paral·lel sèrie

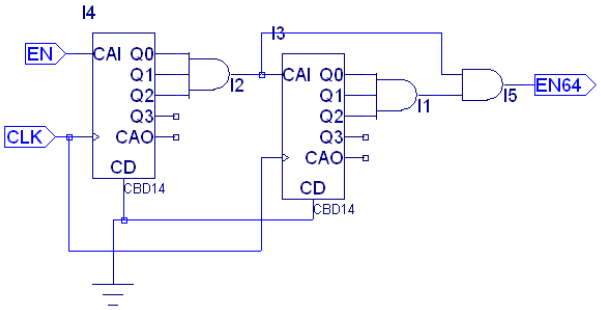
Fent ús dels dos blocs anteriors es fa el conversor paral·lel sèrie. Aquest bloc llegeix 4 bits de dades de forma paral·lela i els treu de forma seqüencial, començant per un 0, el 4 bits de dades i finalment 5 uns, i torna a repetir el procés. Aquest és el bloc que s'encarrega de crear la seqüència que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

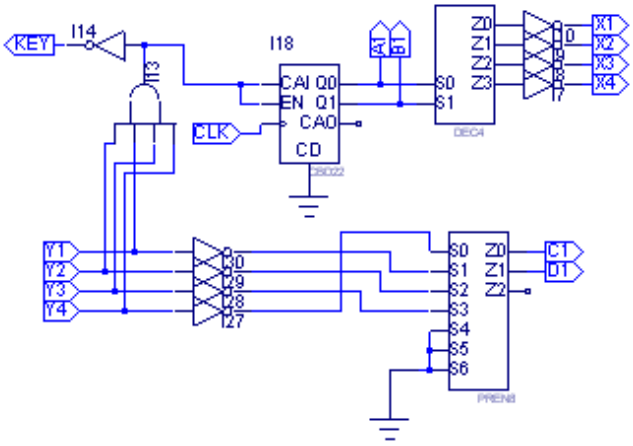
Divisor de freqüència (64)

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.

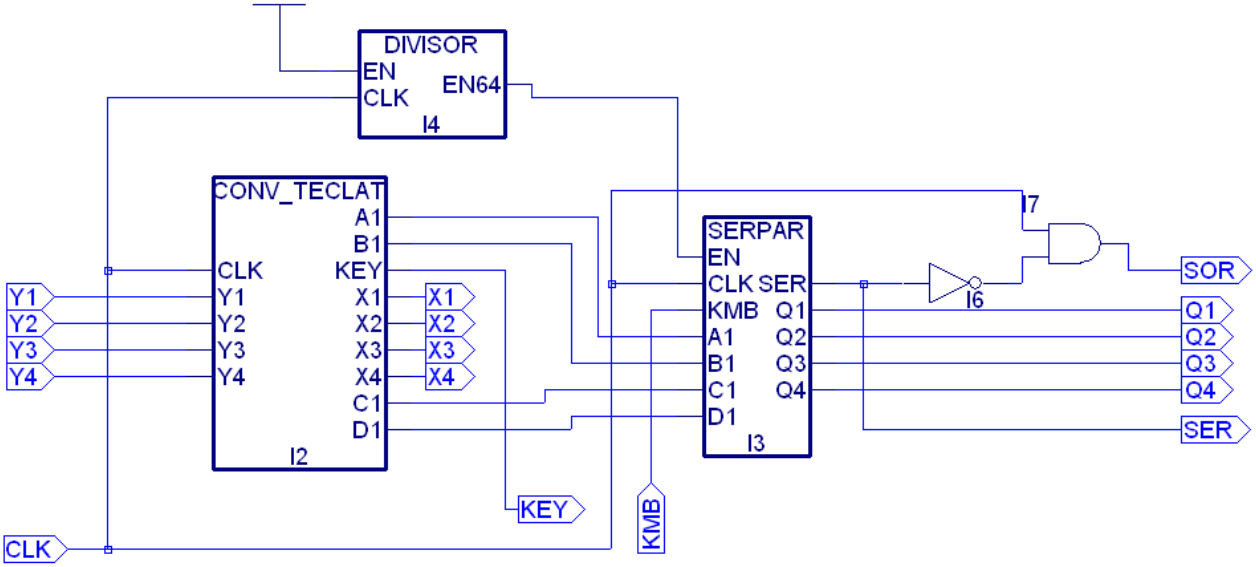


Per a realitzar-lo hem utilitzat un comptador que va canviant contínuament de fila a escombrar. Per cada fila els senyals Y_i reben l'estat de les tecles. Si una d'elles és 0 vol dir

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.

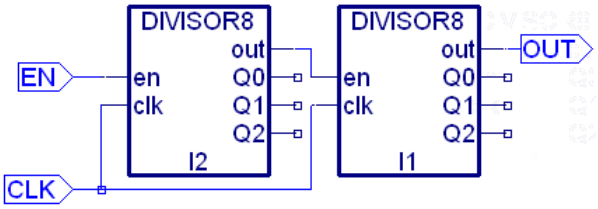
```
MODULE divisor8

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 =.

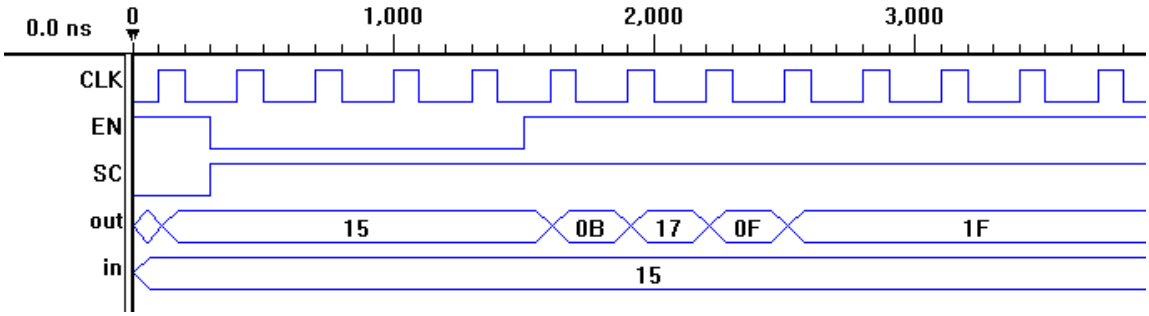
```
MODULE regdesp5
C,X = .C.,.X.;

"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.

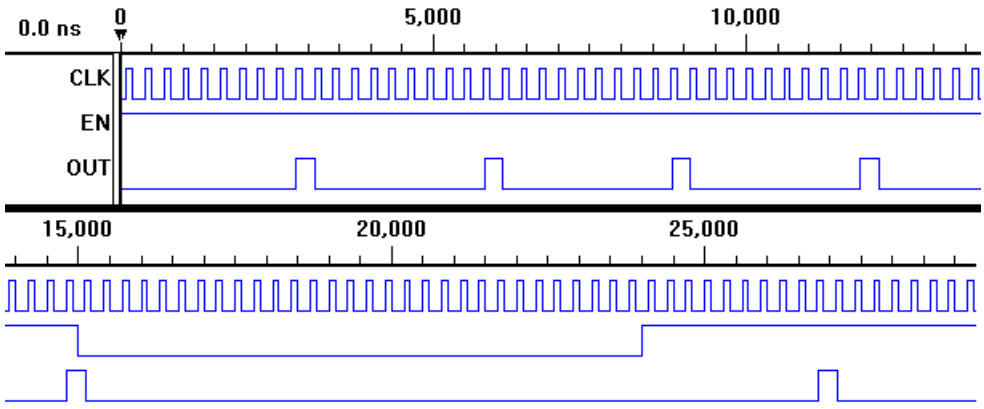
Simulació

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.

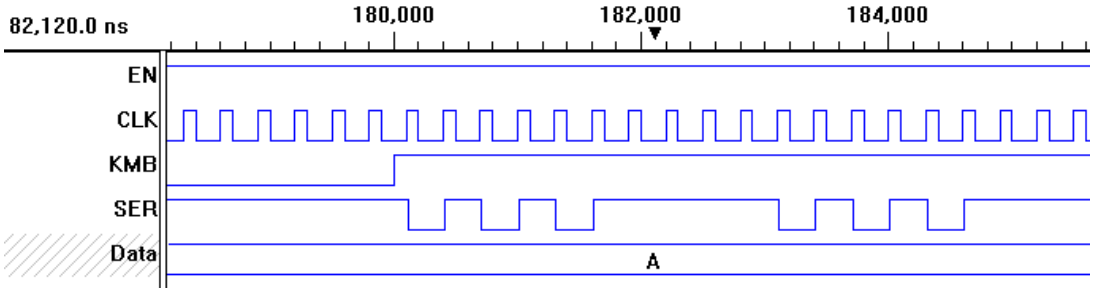
Registre de desplaçament. Es comprova el funcionament de SC i EN.



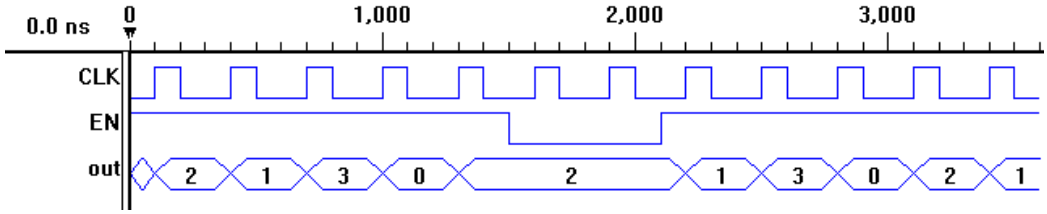
Divisor de freqüència entre 10. Només cal comprovar la senyal EN



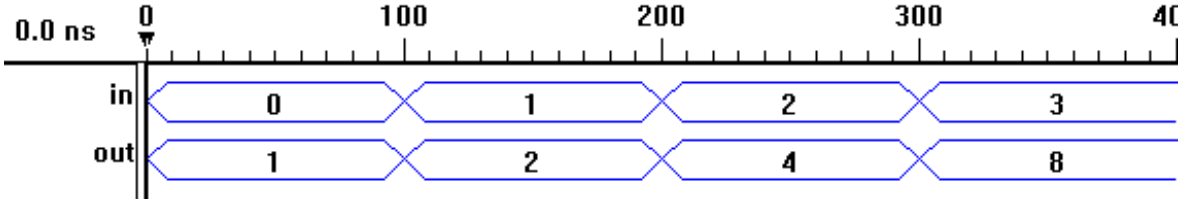
Conversor paral·lel-sèrie



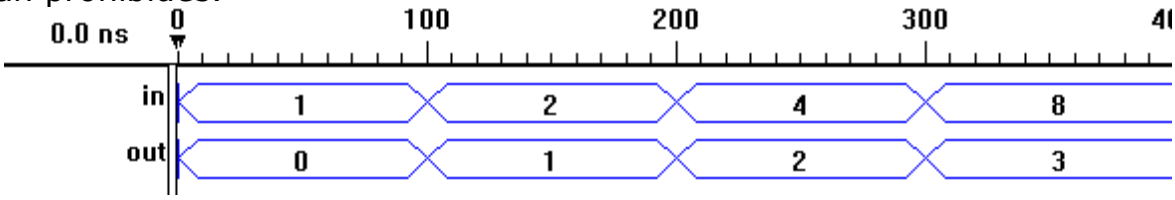
Comptador de 2 bits. Noteu que no compta si EN=0.



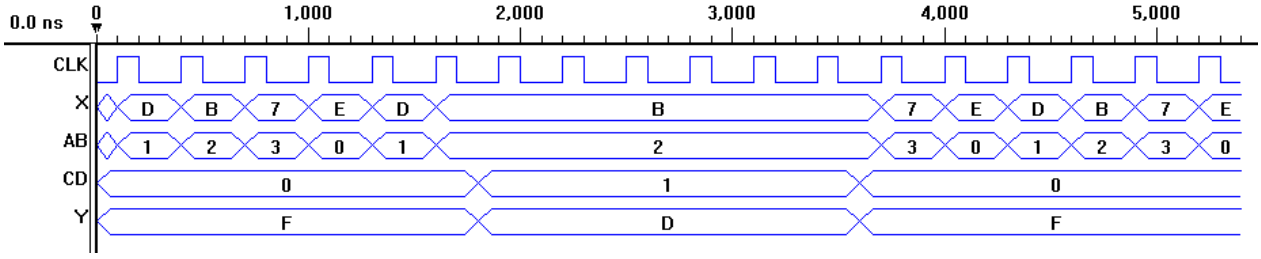
Descodificador 2 a 4. Les sortides i entrades estan en hexadecimal.



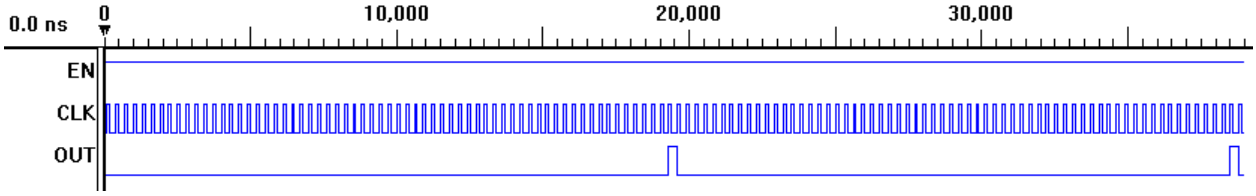
Codificador 4 a 2. No hem comprovat les codificacions no vàlides ja que estan prohibides.



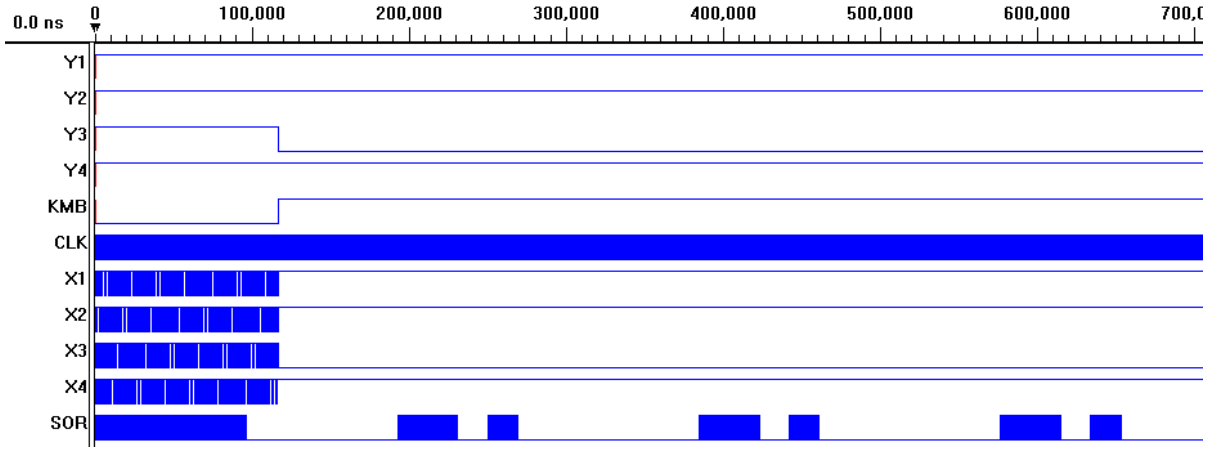
Conversor de teclat. En el moment que pitgem una tecla (alguna Y <> 0) s'atura.



Divisor de freqüència entre 64.

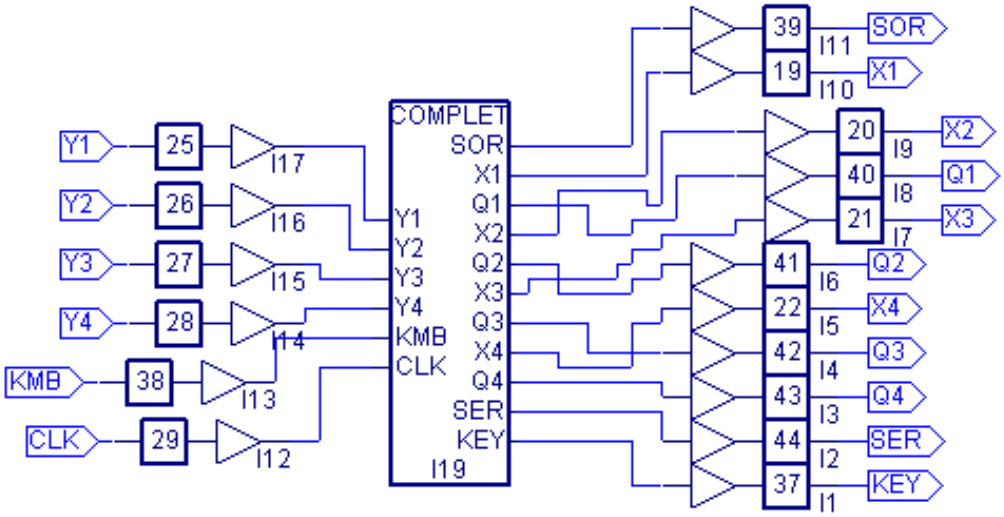


Circuit complet. Es pot veure la sortida dels leds a la senyal SOR.



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 $\overline{\text{KMB}}$). 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.

Pràctica 5:

Com a primera part d'aquesta pràctica hem preparat tal com es demanava un programa que encengui i apagui un LED de forma intermitent i vagi canviant el LED encès cada determinat temps. S'ha fet:

- amb un sol timer (base de temps)
- amb dos timers

1- Amb un sol timer

Amb un sol timer utilitzem les variables globals 'contador', 'encendido', posició i 'pbyReg' que utilitzarem com a punter per escriure sobre els registres que ens calgui.
Posició indicarà el LED (del primer al vuitè) que estem encenent en aquell moment i contador ens servirà per regular el temps que romandrà encès cada LED ja que base de temps provoca una interrupció cada 2.05 ms (de fet el que estem fent es utilitzar la base de temps com un pre-scaler).

```
int contador = 0;
int encendido = 0;
BYTE posicion = 1;
BYTE far *pbyReg;
```

Necessitarem inicialitzar el timer de la base de temps. Ho fem amb la següent rutina:

```
void IniTimers(void)
{
   BYTE far *pbyReg;                          //punter a un registre

                                              //substituir el vector d'interrupció de la base de temps :
  disable();                                        //no permetre cap INT
  setvect(INTBASETEMPS, RSIBaseTemps);              //indiquem la ISR que atendrà la interrupció
  enable();

  pbyReg = (BYTE far *)MK_FP(RSEG, PRC);            //crear el punter al registre de control del
                                                    //processador PRC

  *pbyReg = 0x04;                             //modificar el registre PRC per a una int cada 2.05 ms
                                              //RAM interna deshabilitada
                                              //Fclk de 4 MHz

  pbyReg = (BYTE far *)MK_FP(RSEG, TBIC);           //apuntar al registre TBIC
  *pbyReg &= 0xBF;                                  //activar interrupcions BT
}
```

La RSI que utilitzarem per atendre cadascuna de les interrupcions que generi aquesta base de temps es la següent:

```
void interrupt RSIBaseTemps(void)
{
    disable();

    if (contador%50 == 0) {
          if (encendido == 0) encendido = 1;
           else encendido = 0;

          if (encendido == 1) SetP2(posicion);
          else SetP2(0);
    }

    if (contador >= 500) {
          posicion = posicion<<1;
          if (posicion == 0) posicion = 1;
           contador = 0;
    }

     contador++;

     enable();
     FINT;
}
```

Cada vegada que es produeix una interrupció l'atenem amb aquesta RSI, on incrementem la variable contador, de manera que sabem quantes vegades hem atès la interrupció i podem controlar els temps com ens convingui.

Així, cada 50 vegades que es generi la interrupció encendrem/apagarem el LED posicio-èsim LED. Tenint en conte que es genera una interrupció cada 2.05 ms, cada 0.2 segons s'encendrà un LED (0.1 segon encès, 0.1 segon apagat).

Quan el comptador arriba a 500 canviem el LED que estàvem encenent i reiniciem el contador. Així aconseguim que cada LED s'encengui i s'apagui durant 1 segon (és a dir: abans de que canvii el LED que encenem i apaguem, el LED en qüestió s'encendrà i apagarà 5 vegades).

2**- Amb dos timers**

Bàsicament dividirem la feina que feia la RSI de la base de temps en 2 rutines d'interrupció. Una rutina per cada timer: una que encendrà i apagarà el posició-èsim LED i l'altra que modificarà aquesta posició.

```
void interrupt RSITimer1(void) {
   disable();

   posicion = posicion<<1;
    if (posicion == 0) posicion = 1;
    enable();      //permetre altres INT.
    FINT;
}
```

```
void interrupt RSITimer0(void) {
  disable();

  if (encendido == 0) encendido = 1;
  else encendido = 0;

  if (encendido == 1) SetP2(posicion);
  else  SetP2(0);

  enable();        //permetre altres INT.
  FINT;
}
```

El registre MD0 d'un dels timers serà aproximadament 10 vegades menor que el de l'altra timer per tal d'aconseguir que es cridi 10 vegades a una de les rutines per cada vegada que cridem a l'altra. La rutina cridada amb més freqüència serà l'encarregada de encendre i apagar el LED, deixant a l'altra la tasca de modificar la posició (aconseguim que cada LED s'encengui i s'apagui 5 vegades abans de que es produeixi la interrupció que canviarà de LED).

Així, els timers s'inicialitzaran tal com s'indica a continuació:

```
void IniTimers(void) {

BYTE far *pbyReg;                                   //punter a un registre

   // Iniciar el timer 0

  pbyReg = (BYTE far *)MK_FP(RSEG, MD0);            //indiquem a MD0 quin valor ha de decrementar
  pbyReg[0] = 0;
  pbyReg[1] = 13;

  pbyReg = (BYTE far *)MK_FP(RSEG, TMC0);           //permetem interrupció i escollim que decrementi
  *pbyReg = 0x80 | 0x40;                            //MD0 amb una freq de Fclk/128

  disable();
  setvect(INTCOUNTER0, RSITimer0);                  //indiquem la rutina que atendrà la interrupció
  enable();

  pbyReg = (BYTE far *)MK_FP(RSEG, TMIC0);
  *pbyReg &= 0xBF;                                  //activar interrupcions timer0


  // Iniciarel timer 1

  pbyReg = (BYTE far *)MK_FP(RSEG, MD1);
  pbyReg[0] = 0;
  pbyReg[1] = 128;

  pbyReg = (BYTE far *)MK_FP(RSEG, TMC1);
  *pbyReg = 0x80 | 0x40;

  disable();
  setvect(INTCOUNTER2, RSITimer1);
  enable();
```

```
  pbyReg = (BYTE far *)MK_FP(RSEG, TMIC2);
  *pbyReg &= 0xBF;                                    //activar interrupcions timer1
}
```

NOTA: MD0 és un registre de 16 bits! Hem estat hores barallant amb els timers ja que si modifiquem el byte de menor pes, al estar inicialitzat a 1's el registre en qüestió els bits de major pes fan que aquesta modificació que nosaltres fèiem passés desapercebuda, donant la impressió de que els timers no estan ben inicialitzats. Cal fer dos accessos i guardar dos bytes o bé fer un accés a WORD (nosaltres com es pot veure al codi hem optat per la primera opció).

Pràctica 6

Programa de control del robot

La nostra proposta de pràctica de moviment del robot cobreix tots els casos enunciats a la pràctica així com l'apartat opcional del LCD. Tot seguit llistarem els diferents blocs en que es pot dividir el codi. Al final de la memòria hi ha el llistat complet. Cal dir que tot el codi adjuntat funciona perfectament i ha estat comprovat al robot del laboratori.

Hem afegit un arxiu anomenat *lcd.c* per al control del LCD així com un *mis_includes.h* on es trobem macros auxiliars així com totes les constants que el programa fa servir. D'aquesta manera només caldrà canviar les definicions i no la resta de codi.

Recepció de les dades

La recepció de les dades es fa amb el port sèrie RxD0 i la seva configuració inicial es troba a la rutina **InicialitzaSerie**. Bàsicament el que fem és definir el vector d'interrupcions, inhabilitar els errors diversos que ens puguin generar interrupció i definir el mode de funcionament.

El mode triat és 8 bits de dades, sense bit de paritat i amb un bit de stop. Si nosaltres enviem les dades de la forma [A,B,C,D,1,1,1,1] el registre de dades rebudes del canal 0 (RxB0) prendrà el valor de [1,1,1,1,D,C,B,A]. És per aquest motiu que els codis del comandament estan definits girats al fitxer *mis_includes.h*.

Respecte al tractament d'errors nosaltres hem decidit no utilitzar-lo ja que trobem que no aporta res el fet de saber que s'ha rebut un codi invàlid. Podria succeir (com vam experimentar) que donat que hi ha un obstacle o bé movem el comandament es rep un codi que és vàlid (que té una ordre associada) però que no és el de la tecla que estem pitjant. Aquests errors no es poden arreglar de forma trivial. Una possible solució seria modificar el comandament per a que enviés una seqüència com per exemple [A,B,C,D,A,B,C,D] i la probabilitat que variïn els bits formant una seqüència vàlida és menor. Per altra banda sí que comprovem i exigim a les dades rebudes que tinguin els últims 4 bits de stop a 1. Si això no es compleix rebutgem la dada per invàlida.

Tractament de les comandes

El tractament de comandes es fa de mode immediat al rebre-la i s'encarrega al timer 0 la feina d'aturar aquest moviment en cas de no rebre cas més dada. En el cas de guardar o esborrar el tractament és guardar la posició en un vector, en cas d'esborrar només cal invalidar les dades del vector. El cas de la reproducció és bastant diferent. Caldrà comptar les passos, però haurem d'ignorar les dades rebudes pel comandament així com evitar que el timer0 aturi el robot passats 20ms a la vegada que modulem la velocitat dels motors amb el timer.

Mode de reproducció

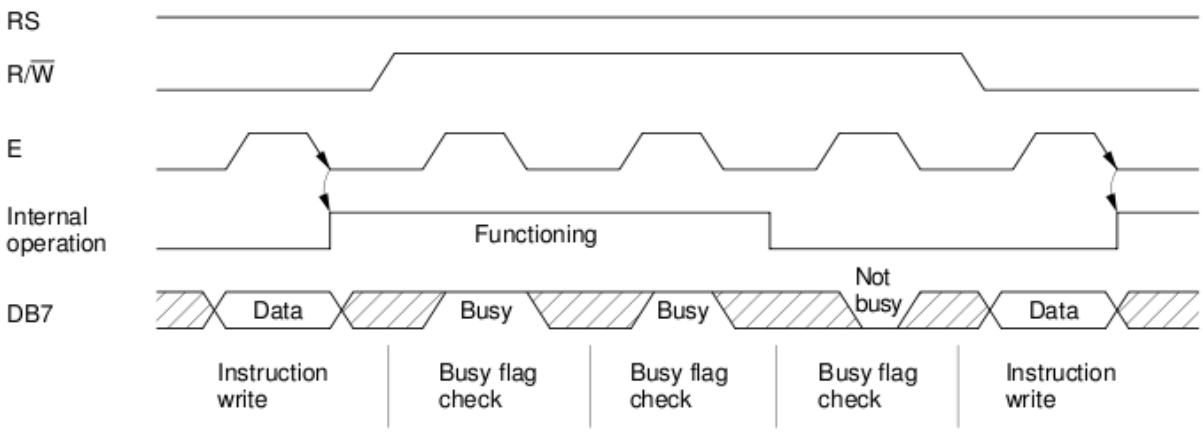
La solució més senzilla i ràpida és tenir una variable que ens ho indiqui i

utilitzar uns ifs per a canviar el comportament dels timers en el cas de la reproducció. A més farem que el *main* sigui l'encarregat de calcular quin moviment cal realitzar per a portar el robot a la posició desitjada. Per a fer això només cal comprovar quina és la posició actual i quina és la següent, en funció de si és major o menor haurem de moure'ns cap endavant o enrere. Anirem comprovant això contínuament fins que arribem a la posició. Unes variables globals indicaran com ens hem de moure i el timer serà l'encarregat de moure els motors. En cas que ens apropem a la posició desitjada caldrà frenar el motor i per això definirem unes variables de cicle de treball que ens permetran fer funcionar els motors més lentament. El timer doncs, activarà cada un dels motors segons els seu cicle de treball que serà 100% normalment i el 25% en apropar-nos a menys de 5 passos del final. La modulació es realitza aturant el motor el 75% del temps i activant-lo el 25% restant. És important notar que a l'enunciat diu que el període sigui de 80ms, però la nostra experiència ens diu que un període molt menor com 40 o 20ms.

LCD

Com a part opcional de la pràctica hem fet servir la pantalla LCD disponible al port 0 del microcontrolador. El codi el podem trobar a lcd.c comentat. Bàsicament el que fem és inicialitzar el LCD amb una seqüència que trobem explicada al manual d'HITACHI i llavors definim unes funcions que escriuen frases a la pantalla. Aquestes funcions situen el cursor a l'origen i escriuen caràcter a caràcter la frase.

Per a que tot això funcioni després d'enviar una dada cal garantir un temps a la senyal enable així com un temps d'execució d'instrucció. Això es veu al cronograma corresponent al manual. Notem que fem servir el mode 8 bits, amb el que les dades s'envien en un sol cicle.



Com podem veure cal enviar la dada i al cap de 40μs aproximadament hem de desactivar la senyal enable i la senyal de write i esperar el temps d'execució d'una instrucció (uns 40μs però depèn de la instrucció).

A continuació el llistat de codi complet. Els fitxers que no es defineixen són els mateixos que ens donàveu com a enunciat.

Codi pràctica 6

mis_includes.h

```
#ifndef  MIS_INCLUDES
#define  MIS_INCLUDES

// Definició molt útil per a assignar bits
#define BIT0  0x01
#define BIT1  0x02
#define BIT2  0x04
#define BIT3  0x08
#define BIT4  0x10
#define BIT5  0x20
#define BIT6  0x40
#define BIT7  0x80

// Funció que posa un motor en un estat
void motor_estat (BYTE num_motor, BYTE codi_estat);

// Macros per a la utilització senzilla de la funció anterior
#define pinza_abrir motor_estat(4,0x2)
#define pinza_cerrar motor_estat(4,0x1)
#define pinza_parar motor_estat(4,0x3)
#define rot_antihorario motor_estat(1,0x2)
#define rot_horario motor_estat(1,0x1)
#define rot_parar motor_estat(1,0x3)
#define movh_atras motor_estat(2,0x2)
#define movh_delante motor_estat(2,0x1)
#define movh_parar motor_estat(2,0x3)
#define movv_arriba motor_estat(3,0x2)
#define movv_abajo motor_estat(3,0x1)
#define movv_parar motor_estat(3,0x3)
// Ho para tot
#define parar_todo pinza_parar;rot_parar;movh_parar;movv_parar

Codis de teclat rebuts girats (primer bit rebut és el de menor pes)
#define ABRIR_PINZA       0x00
#define CERRAR_PINZA      0x04
#define DELANTE           0x06
#define ATRAS             0x02
#define ARRIBA            0x01
#define ABAJO             0x05
#define ANTIHORARIO       0x03
#define HORARIO           0x07
#define INICIO            0x0E
#define ALMACENAR         0x09
#define REPRODUCIR        0x0D
#define BORRAR            0x0A

// Estructura que defineix una posició del robot
struct posicio {
   BYTE pos_pinca;
   BYTE pos_horit;
   BYTE pos_vert;
   BYTE rotacio;
};

#define MAX_ALMACENAR 32   // Nombre màxims de posicions que es poden guardar
#define MARGE_LENT  5      // Nombre de passos per a entrar en mode lent
#define MARGE_ERROR 3      // Marge que deixem del final per errors
#define TEMPS_MISSATGE  800  // Temps de mostra dels missatges

void escriu_frase(BYTE * frase);            // Escriu una frase per l'LCD
void init_lcd();                       // Inicialitza l'LCD
#endif
```

lcd.c

```
#include <embedded.h>
#include "register.h"
#include "typedefs.h"
#include "ports.h"
#include "mis_includes.h"

BYTE valor_P1 = 0x00;      // Valor del port P1
BYTE valor_P0 = 0x00;      // Valor del port P0
DWORD wait_ = 0;
BYTE cont;

// Definim algunes macros d'espera (molt aprox.)
#define ESPERA_40US for(wait_ = 0; wait_ < 20; wait_++){}
#define ESPERA_100US for(wait_ = 0; wait_ < 60; wait_++){}
#define ESPERA_15MS for(wait_ = 0; wait_ < 200; wait_++){}

// Marquem el fi de dada abaixant la senyal Enable i Read/Write
void fi_inst_dada(void) {
    valor_P1 = BIT5;
    SetP0(valor_P0);
}

// Escriu una instrucció al LCD
void escriu_inst_lcd(BYTE inst) {
    // RS = 0, R/W* = 0, E = 1
    valor_P1 = BIT6;
    valor_P0 = inst;

    SetP0(valor_P0);
    SetP1(valor_P1);

    ESPERA_40US    // Fem un pols amb E de uns 40US

    valor_P1 = 0;  // Netegem E però mantenim WRITE
    SetP1(valor_P1);
}

// Escriu una dada al LCD
void escriu_dada_lcd(BYTE dada) {
    // RS = 1, R/W* = 0, E = 1
    valor_P1 = BIT6|BIT4;
    valor_P0 = dada;

    SetP0(valor_P0);
    SetP1(valor_P1);

    ESPERA_40US    // Fem un pols amb E de uns 40US

    valor_P1 = 0;  // Netegem E però mantenim WRITE
    SetP1(valor_P1);
}

// Inicialitzar el LCD, extret del manual
void init_lcd() {
    ESPERA_15MS   // Esperem per garantir l'alimentació Vcc
    escriu_inst_lcd(BIT6|BIT5);   // Enviem la seqüència d'inici 3 cops
    ESPERA_15MS
    fi_inst_dada();

    escriu_inst_lcd(BIT6|BIT5);
    ESPERA_100US
    fi_inst_dada();
```

```
    escriu_inst_lcd(BIT6|BIT5);
    ESPERA_100US
    fi_inst_dada();

    // Activem el mode de dues línies i caràcters grans
    escriu_inst_lcd(BIT6|BIT5|BIT4|BIT3);
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT3);  // Apagar!
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT3|BIT2);  // Encendre!
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT4|BIT2);  // El cursor es desplaça a la dreta
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT1);  // Situem el cursor a l'inici
    ESPERA_40US
    fi_inst_dada();
}

// Escriu una frase sencera al LCD substituint l'anterior
// Escriurem espais en blanc si la frase no ocupa tot l'LCD
void escriu_frase(BYTE * frase) {
    BYTE maux;
    escriu_inst_lcd(BIT1);      // Cursor a l'inici
    ESPERA_40US
    fi_inst_dada();
    maux = 0;
    for (cont = 0; cont < 16; cont++) {
        if (frase[cont] == 0x00 || maux==1) {  // Fi de la frase
              escriu_dada_lcd(' ');
              maux = 1;
        }else{
              escriu_dada_lcd(frase[cont]);
        }
        ESPERA_40US
        fi_inst_dada();
    }
}

// Conversió de integer a character
void itoa(BYTE * cad, BYTE num, BYTE digit) {
   BYTE cont=digit-1;
   do {
         cad[cont] = (num%10)+'0';
         num = num /10;
         cont--;
   } while (cont != 0xFF) ;
}

// Funció que escriu a la pantalla les diferents posicions del robot de la
// forma R000H000V000P000
BYTE buffer[17];
void escriu_valors(struct posicio * pos) {
   buffer[0] = 'R';
   buffer[0+4] = 'H';
   buffer[0+8] = 'V';
   buffer[0+12] = 'P';
   itoa(&buffer[1],pos->rotacio,3);
   itoa(&buffer[1+4],pos->pos_horit,3);
   itoa(&buffer[1+8],pos->pos_vert,3);
```

```
   itoa(&buffer[1+12],pos->pos_pinca,3);
   escriu_frase(buffer);
}

// Escriu una frase seguida d'un enter, similar a fer
// printf ("Frase %d",enter);
void escriu_frase_integer (BYTE * frase, BYTE enter) {
   cont=0;
   while (frase[cont] != 0) {
         buffer[cont] = frase[cont];
         cont++;
   }
   itoa(&buffer[cont],enter,16-cont);
   escriu_frase(buffer);
}
```

ports.c

```
#include <embedded.h>
#include "register.h"
#include "typedefs.h"
#include "ports.h"
#include "mis_includes.h"

// Inicialització dels ports P0,P1,P2,PT
void IniPorts(void) {
   BYTE far *pbyReg; //punter a un registre
   // Port P2
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC2);
   *pbyReg = 0x00;   //PMC2 = '00' -> mode port I/O
   --pbyReg;         //apuntar a PM2
   *pbyReg = 0x00;   //PM2 = '00' -> port 2 de sortida
   --pbyReg;         //apuntar a P2
   *pbyReg = 0x00;   //P2 = '00' -> port 2 valor inicial

   // Port 1
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC1);
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;

   // Port 0
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC0);
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;

   //config PT
   pbyReg = (BYTE far *)MK_FP(RSEG, PMT);
   *pbyReg = 0x08;   //llindar entrades PT a 1.25V
}

// Funcions per llegir/escriure els diferents ports
void SetP2(BYTE byVal) {
   BYTE far *pbyP2;
   pbyP2 = (BYTE far *)MK_FP(RSEG, P2);
   *pbyP2 = byVal;
}

void SetP1(BYTE byVal) {
   BYTE far *pbyP1;
   pbyP1 = (BYTE far *)MK_FP(RSEG, P1);
```

```
   *pbyP1 = byVal;
}

void SetP0(BYTE byVal) {
   BYTE far *pbyP0;
   pbyP0 = (BYTE far *)MK_FP(RSEG, P0);
   *pbyP0 = byVal;
}

BYTE GetPT(void) {
   BYTE far *pbyPT;
   pbyPT = (BYTE far *)MK_FP(RSEG, PT);
   return(*pbyPT);
}
```

robot.c

```
#include <embedded.h>
#include "typedefs.h"
#include "register.h"
#include "timers.h"
#include "ports.h"
#include "mis_includes.h"

BYTE estat_P2 = 0x00;
BYTE reset = 1;
BYTE no_show = 0;
extern BYTE cicle_treball[4];
extern int contador_lcd;

BYTE reproduir = 0;  // Si es 0, mode normal, 1 mode reproduccio!

// S'actualitza sola indicant la posició del robot!
// Cal posarla a 0 al fer un reset!
struct posicio posicio_actual;
// Llista de posicions a guardades
struct posicio vector_posicions[MAX_ALMACENAR];
// Indica la pròxima posició que es guardara al enviar la tecla
BYTE proxima_posicio_gravar = 0;
// Per a esborrar les dades només cal posar-la a 0.
// Si és 0 també sabem que no hi ha res guardat.

BYTE paraula_moviment;
// Indica l'estat dels motors a cada instant al reproduir
BYTE cont_pinca = 0, cont_posh = 0, cont_posv = 0, cont_rot = 0;
// Valors dels pins compta-passos, necessaris per detectar flancs
BYTE moviment_1 = 0, moviment_2 = 0, moviment_3 = 0, moviment_4 = 0;
// Serà 1 o -1 segons el sentit del moviment

// Funció que calcula la paraula d'estat dels motors
BYTE calc_paraula_estat (BYTE num_motor, BYTE codi_estat, BYTE paraula) {
   BYTE mascara = (0x3 << 2*(num_motor-1));
   paraula &= ~(mascara);  // Netejem l'estat! El posem a 00
   // Enmascarem el codi per si de cas for incorrecte!
   paraula |= ( (codi_estat & 0x3) << 2*(num_motor-1) );
   return paraula;
}

// Actualitza l'estat d'un motor
void motor_estat (BYTE num_motor, BYTE codi_estat) {
   estat_P2=calc_paraula_estat(num_motor,codi_estat,estat_P2);
   SetP2(estat_P2);
}

// Funció que acaba quan el robot es troba a l'origen
void robot_inici() {
```

```
      BYTE status;
      pinza_abrir; rot_antihorario; movh_atras; movv_arriba;
      status = GetPT();
      // Esperem mentre no estigui a l'origen
      while ( (status & 0x01) != 0 ) {status = GetPT();}
      parar_todo;

      posicio_actual.pos_pinca = 0;
      posicio_actual.pos_horit = 0;
      posicio_actual.pos_vert = 0;
      posicio_actual.rotacio = 0;
   }

   void main() {
      BYTE i,temp;

      // Important iniciar els ports de I/O correctament
      IniPorts();
      // Inicialitza el LCD
      init_lcd();
      // Inicialitzem el port serie
      InicialitzaSerie();
      // Parem el robot
      parar_todo;
      // Activem els timers
      IniTimers();
      enable(); // Permetem interrupcions!

      while (1) {

      // Escribim els valors actuals si no_show==0
            if (no_show == 0) escriu_valors(&posicio_actual);

      // Es vol fer un reset!
            if (reset == 1) {
                  escriu_frase("Robot: reset");
            disable();  // Parem interrupcions per a fer el reset, important!
            robot_inici();
            reset = 0;
            enable();   // Tornem al mode habitual
      }

      if (reproduir == 1) {  // Mode reproducció!
            escriu_frase("Robot: reproduir");
            // Comprovar que hi hagi alguna cosa a la llista de reproduccions
            if (proxima_posicio_gravar != 0) {
                  parar_todo;  // Parem, resetegem i comencem a reproduir
                  disable(); robot_inici(); enable();
                  posicio_actual.pos_pinca = 0;
                  posicio_actual.pos_horit = 0;
                  posicio_actual.pos_vert = 0;
                  posicio_actual.rotacio = 0;

                  for (i = 0; i < proxima_posicio_gravar; i++) {
                        // Anem escrivint el moviment que estem fent
                        escriu_frase_integer("Moviment: ",i+1);
                        temp = 0x00;  // Variable que indica el moviment
                        while (!(temp == 0xFF)) {
                              // Ens quedem recalculant la paraula d'estat
                              // mentre no arribem  la següent posició
                              temp = 0xFF;
                              // Calcular com han d'estar els motors i revisar
                              // si s'ha assolit la posició!

                              if (vector_posicions[i].pos_pinca >
posicio_actual.pos_pinca) {  // Tancar la pinça
                                    moviment_4 = 1;
                                    temp=calc_paraula_estat(4,0x1,temp);
```

```
                              }else if (vector_posicions[i].pos_pinca <
posicio_actual.pos_pinca) {
                                    moviment_4 = -1;
                                    temp=calc_paraula_estat(4,0x2,temp);
                              }

                              if (vector_posicions[i].rotacio >
posicio_actual.rotacio) {  // Moviment horari
                                    moviment_1 = 1;
                                    temp=calc_paraula_estat(1,0x1,temp);
                              }else if (vector_posicions[i].rotacio <
posicio_actual.rotacio) {
                                    moviment_1 = -1;
                                    temp=calc_paraula_estat(1,0x2,temp);
                              }

                              if (vector_posicions[i].pos_horit >
posicio_actual.pos_horit) {  // Endevant
                                    moviment_2 = 1;
                                    temp=calc_paraula_estat(2,0x1,temp);
                              }else if (vector_posicions[i].pos_horit <
posicio_actual.pos_horit) {
                                    moviment_2 = -1;
                                    temp=calc_paraula_estat(2,0x2,temp);
                              }

                              if (vector_posicions[i].pos_vert >
posicio_actual.pos_vert) {  // Avall
                                    moviment_3 = 1;
                                    temp=calc_paraula_estat(3,0x1,temp);
                              }else if (vector_posicions[i].pos_vert <
posicio_actual.pos_vert) {
                                    moviment_3 = -1;
                                    temp=calc_paraula_estat(3,0x2,temp);
                              }

                              // Mirem a quina velocitat hem d'anar
                              if (abs(vector_posicions[i].rotacio -
posicio_actual.rotacio) < MARGE_LENT) cicle_treball[0] = 4;
                              else cicle_treball[0] = 1;
                              if (abs(vector_posicions[i].pos_horit -
posicio_actual.pos_horit) < MARGE_LENT) cicle_treball[1] = 4;
                              else cicle_treball[1] = 1;
                              if (abs(vector_posicions[i].pos_vert -
posicio_actual.pos_vert) < MARGE_LENT) cicle_treball[2] = 4;
                              else cicle_treball[2] = 1;
                              if (abs(vector_posicions[i].pos_pinca -
posicio_actual.pos_pinca) < MARGE_LENT) cicle_treball[3] = 4;
                              else cicle_treball[3] = 1;

                              paraula_moviment = temp;
                        }
                  }
            }else{
                  // Cas de reproducció però no hem guardat res
                  contador_lcd=0;
                  no_show = 1;
                  escriu_frase("Res a reproduir!");
            }
      reproduir = 0;
      }

      }  // while (1)

   }
```

timers.c

```
#include <embedded.h>
#include "typedefs.h"
#include "int.h"
#include "register.h"
#include "ports.h"
#include "timers.h”
#include "mis_includes.h"

// Variables globals
int n_codivalid = 0;
int contador = 0;
int encendido = 0;
int contador_lcd = 0;
BYTE posicion = 1;
extern BYTE no_show;

void interrupt RSITimer0(void);
void interrupt RSITimer1(void);
void interrupt RSIRecepcio0 (void);

BYTE far *pbyReg;
BYTE cicle_treball[4];
BYTE cont_treball[4];

extern BYTE moviment;
extern BYTE reset;
extern BYTE reproduir;
extern struct posicio posicio_actual;
extern struct posicio vector_posicions[32];
extern BYTE proxima_posicio_gravar;
extern BYTE paraula_moviment;
extern BYTE cont_pinca, cont_posh , cont_posv , cont_rot ;
extern BYTE moviment_1, moviment_2, moviment_3, moviment_4;
BYTE almacenado = 0;


// Inicialització dels temporitzadors
void IniTimers(void) {
   BYTE far *pbyReg; //punter a un registre
   WORD far *pwoReg; //punter a un registre

   disable();        // no permetre cap interrupció

   setvect(INTBASETEMPS, RSIBaseTemps);
   pbyReg = (BYTE far *)MK_FP(RSEG, PRC);
   *pbyReg = 0x04;         //Fclk de 4 MHz
   pbyReg = (BYTE far *)MK_FP(RSEG, TBIC);
   *pbyReg &= 0xBF;  //activar interrupcions BT

   // Iniciarel timer 0
   pwoReg = (WORD far *)MK_FP(RSEG, MD1);
   *pwoReg = 512;

   pbyReg = (BYTE far *)MK_FP(RSEG, TMC0);
   *pbyReg = 0x80 | 0x40;

   setvect(INTCOUNTER0, RSITimer0);

   pbyReg = (BYTE far *)MK_FP(RSEG, TMIC0);
   *pbyReg &= 0xBF;  //activar interrupcions timer0

   // Iniciarel timer 1
   pwoReg = (WORD far *)MK_FP(RSEG, MD1);
   *pwoReg = 128;

   pbyReg = (BYTE far *)MK_FP(RSEG, TMC1);
```

```
      *pbyReg = 0x80 | 0x40;

      setvect(INTCOUNTER2, RSITimer1);

      pbyReg = (BYTE far *)MK_FP(RSEG, TMIC2);
      *pbyReg &= 0xBF;  //activar interrupcions timer1

      enable();
   }

   void InicialitzaSerie() {
      BYTE far *pbyReg;

      disable();

      pbyReg = (BYTE far *)MK_FP(RSEG, SCM0);   // Registre de descripció del
senyal
      *pbyReg = BIT3 | BIT6 | BIT0;
      // 8 bits de dades + recepció enabled  (no paritat , 1 bit stop)
      // MOLT IMPORTANT el BIT0 per activar mode ASINCRON

      pbyReg = (BYTE far *)MK_FP(RSEG, SCE0);   // Registre de control de errors
      *pbyReg = 0;  // Tots els errors capats + NO transmissió

      // Registre de control de d'interrupció d'errors
      pbyReg = (BYTE far *)MK_FP(RSEG, SEIC0);
      *pbyReg = 0;  // No volem petició d'error

      // Registre de control de d'interrupció de transmissió completada
      pbyReg = (BYTE far *)MK_FP(RSEG, SRIC0);
      *pbyReg = 0;  // No volem petició d'error

      // Registre de control de d'interrupció de recepció completada.
      pbyReg = (BYTE far *)MK_FP(RSEG, SRIC0);
      *pbyReg = BIT7;  // Volem petició

      pbyReg = (BYTE far *)MK_FP(RSEG, BRG0);
      *pbyReg = 15;  // 512bauds

      pbyReg = (BYTE far *)MK_FP(RSEG, SCC0);
      *pbyReg = 0x08;  // 512bauds

      setvect(INTSRX0, RSIRecepcio0);
      setvect(INTSERR0, RSIRecepcio0);
      setvect(INTSTX0, RSIRecepcio0);
      enable();
   }

   BYTE decide_signo(BYTE valor, BYTE medio) {
      if(valor > medio) return -1;
      return 1;
   }

   void interrupt RSIRecepcio0 (void) {
      BYTE j,dada;
      disable();

      if (reproduir == 0) {

            // Tractar recepció de dades! La dada esta a RxB0
            pbyReg = (BYTE far*)MK_FP(RSEG,  RxB0);   // Dada!
            dada = *pbyReg;
            if ( (dada & 0xF0) == 0xF0) {  // Els 4 primers bits d'stop
                  dada = dada & 0x0F;
                  moviment_1 = 0;moviment_2 = 0; moviment_3 = 0; moviment_4 = 0;
                  switch (dada) {
                  case ABRIR_PINZA:
                        moviment_4 = -1; almacenado=0;
```

```
                        parar_todo; pinza_abrir; break;
                  case CERRAR_PINZA:
                        moviment_4 = 1; almacenado=0;
                        parar_todo; pinza_cerrar; break;
                  case DELANTE:
                        moviment_2 = 1; almacenado=0;
                        parar_todo; movh_delante; break;
                  case ATRAS:
                        moviment_2 = -1; almacenado=0;
                        parar_todo; movh_atras; break;
                  case ARRIBA:
                        moviment_3 = -1; almacenado=0;
                        parar_todo; movv_arriba; break;
                  case ABAJO:
                        moviment_3 = 1; almacenado=0;
                        parar_todo; movv_abajo; break;
                  case ANTIHORARIO:
                        moviment_1 = -1; almacenado=0;
                        parar_todo; rot_antihorario; break;
                  case HORARIO:
                        moviment_1 = 1; almacenado=0;
                        parar_todo; rot_horario; break;
                  case INICIO:
                        almacenado=0; reset = 1; parar_todo; break;
                  case REPRODUCIR:
                        almacenado=0; paraula_moviment=0xFF; reproduir = 1;
                        parar_todo; break;
                  case ALMACENAR:
                        if (almacenado==1) break;
                        almacenado=1;
                        if (proxima_posicio_gravar >= MAX_ALMACENAR) {
                              contador_lcd=0;
                              no_show = 1;
                              escriu_frase("Cua plena!");
                              break;  // No n'hi caben més!
                        }
                        vector_posicions[proxima_posicio_gravar] =
posicio_actual;   // Guardar la posició
                        vector_posicions[proxima_posicio_gravar].rotacio +=
decide_signo(vector_posicions[proxima_posicio_gravar].rotacio,115)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_pinca +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_pinca,15)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_horit +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_horit,90)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_vert +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_vert,55)*MARGE_ERROR;
                        proxima_posicio_gravar++;
                        contador_lcd=0;
                        no_show = 1;
                        escriu_frase("Moviment guardat");
                        parar_todo; break;
                  case BORRAR:
                        almacenado=0;
                        proxima_posicio_gravar = 0;
                        contador_lcd=0;
                        no_show = 1;
                        escriu_frase("Tot borrat");
                        parar_todo; break;
                  default:
                        goto CODI_NO_VALID;
                        break;
                  }
                  // Codi valid! Seguir el moviment!
                  n_codivalid = 0;
            }
      }
   CODI_NO_VALID:
      enable();
```

```
      FINT;
   }

   // Servei d'interrupció de la base de temps
   void interrupt RSIBaseTemps(void) {
      disable();

      contador_lcd++;
      if (contador_lcd > TEMPS_MISSATGE) no_show=0;
      enable();

      FINT;
   }

   // Cicle de treball variable segons l'array cicle_treball
   void interrupt RSITimer1(void) {
      BYTE temp2,j;
      disable();  //permetre altres INT.

      if (reproduir == 1) {
            // Modular la sortida amb tren de polsos
            temp2 = paraula_moviment;
            for (j = 0; j < 4; j++) {
               if (cont_treball[j] != 0) {
                  temp2 |= (0x3 << (2*j));
               }
            }
            SetP2(temp2);
            for (j = 0; j < 4; j++) cont_treball[j] = (cont_treball[j]+1) %
cicle_treball[j];
      }
      enable();
      FINT;
   }

   void interrupt RSITimer0(void) {
      BYTE dada;
      disable();

      /// Comptapassos mostrejat a la freq del timer
      dada = GetPT();

      if ( ((dada & BIT1) == 0) && cont_rot == 1) { cont_rot=0;
posicio_actual.rotacio+=moviment_1; }
      if ( ((dada & BIT1) != 0) && cont_rot == 0) { cont_rot=1;
posicio_actual.rotacio+=moviment_1; }

      if ( ((dada & BIT2) == 0) && cont_posh == 1) { cont_posh=0;
posicio_actual.pos_horit+=moviment_2; }
      if ( ((dada & BIT2) != 0) && cont_posh == 0) { cont_posh=1;
posicio_actual.pos_horit+=moviment_2; }

      if ( ((dada & BIT3) == 0) && cont_posv == 1) { cont_posv=0;
posicio_actual.pos_vert+=moviment_3; }
      if ( ((dada & BIT3) != 0) && cont_posv == 0) { cont_posv=1;
posicio_actual.pos_vert+=moviment_3; }

      if ( ((dada & BIT4) == 0) && cont_pinca == 1) { cont_pinca=0;
posicio_actual.pos_pinca+=moviment_4; }
      if ( ((dada & BIT4) != 0) && cont_pinca == 0) { cont_pinca=1;
posicio_actual.pos_pinca+=moviment_4; }

      if (reproduir == 0) {  // Només si estem en mode normal
            // Aquesta rutina s'encarrega de parar el
            // motor si no s'ha apretat cap tecla!
            if (n_codivalid >= 2) {
                  parar_todo;  // to's quietos
            }else{
```

```
                n_codivalid++;
            }
        }

        enable();
        FINT;
    }
```