Pràctica 7: Control semafòric de cruïlla amb comunicació morse: Esclau

Programació a Baix Nivell — iTIC

Sebastià Vila-Marta Paco del Águila

3 de juny de 2021

Índex

1 Introducció

L'objectiu d'aquesta pràctica és afegir a la pràctica anterior la part de recepció morse. D'aquesta forma serà molt senzill implementar controladors de cruïlla esclaus. Els controladors esclaus no estaran supervisats per la workstation sinó que rebran les ordres a través del canal morse.

El sistema final, una vegada acabada aquesta pràctica, es comportarà d'aquesta manera:

- Cada sistema semafòric controlarà els semàfors d'una cruïlla i es podrà configurar com a sistema mestre o sistema esclau.
- Un sistema mestre podrà ser controlat mitjançant la connexió sèrie usant el supervisor i el protocol que ja coneixeu. A més, el mestre reproduirà les ordres correctes que rep pel supervisor usant el canal morse en forma d'infraroig.

- Un sistema esclau no pot controlar-se usant la connexió sèrie. Un sistema esclau només rep comandes a través del canal morse i les executa de manera habitual.

Amb aquest muntatge podem situar a prop un mestre i diversos esclaus, cadascun dels quals controla una cruïlla i, actuant sobre el màster apagar tots els semàfors!

En aquesta pràctica només es tractarà d'implementar el sistema esclau. Essencialment es tracta d'ampliar el mòdul `ether` de manera que, a banda de tenir un canal de sortida, tingui també un canal d'entrada.

La recepció de dades morse es realitza usant un circuit receptor d'infraroig. Se us proporcionarà un kit transceptor d'infrarojos. La part de transmissió consisteix en un LED emissor i una resistència. La part de recepció consisteix en el receptor TSOP 18438. Aquest dispositiu està format per un fotodetector que incorpora un desmodulador a 38 kHz. Aquests components s'han de muntar adequadament en una placa de prototipatge, preferiblement a la shield iTIC.

De forma resumida podem dir que per fer la pràctica cal:

- Reestructurar l'organització del projecte de forma que tinguem per un costat una llibreria i per altra diverses aplicacions que la utilitzen.
- Fer el muntatge del transceptor d'infraroig i comprovar que es capaç de rebre senyals amb certa qualitat.
- Experimentar amb un filtratge basat en programari del senyal provinent del desmodulador per fer més robusta la detecció de flancs.
- Basant-se en els experiments anteriors, modificar el mòdul `ether` per tal d'afegir la part de recepció de dades.
- Comprovar el correcte funcionament del mòdul ether.
- Finalment implementar l'esclau del control semafòric.

1.1 Organització

1. Cal fer la pràctica en equip de dues persones. Això no obstant, per poder-la provar són necessàries dues plaques Arduino.
2. Cal desenvolupar el programari i la documentació usant els serveis del sistema de gestió de versions de l'EPSEM.

Pel que fa al material necessari, cada equip:

1. Cal que disposi de 12 LED (4 vermells, 4 grocs i 4 verds) o bé els *shield iTIC* per la realització d'aquesta pràctica.
2. Cal que disposi de dos Arduinos i la corresponent placa de prototips amb els seus accessoris habituals.
3. Cal disposar del kit transceptor muntat adequadament. Cal que el demaneu al professor, que us el cedirà mentre duri la pràctica.

TASCA PRÈVIA 1 Assegureu-vos que teniu el material necessari a l'abast.

1.2 Lliuraments

Caldrà lliurar el resultat de la pràctica a través del sistema Atenea de la forma que ja s'anunciarà. Els lliuraments inclouen el següent:

1. Un tar file que conté els fonts (nets i sense objectes, còpies de seguretat, etc.) del projecte incloent el `Makefile` corresponent.

2. El sistema muntat íntegrament i funcionant correctament.

1.3 Mètode de treball

Per fer aquesta pràctica és convenient, una vegada més, estudiar amb cura la forma de repartir-se la feina de manera que els membres de l'equip puguin treballar en paral·lel. Per aconseguir això cal estudiar prèviament el projecte i decidir quin pla de treball se seguirà per executar-lo.

El desenvolupament cal fer-lo sobre el servei de control de versions de l'assignatura (`http://escriny.epsem.upc.edu`). Per tant cal preparar prèviament la disposició de directoris necessària.

2 Reestructuració del codi

En aquesta pràctica caldrà treballar amb una llibreria. La llibreria la constituiran els mòduls més transversals de la pràctica, aquells que podrien ser usats en altres projectes diferents. Aquests mòduls els copiarem literalment de la pràctica anterior.

Amb aquest canvi passarem a tenir dos projectes independents. Un és el que es dedica a mantenir i millorar la llibreria. Un altre és el que construeix aplicacions basant-se en la llibreria.

La llibreria l'anomenarem `libpbn.a` i la constituiran els mòduls: `ether`, `gpio_device`, `lamp`, `mchar`, `modulator`, `mtbl`, `semaph`, `queue`, `serial_device`, `blck_serial` i `timer`. Per facilitar l'ús de la llibreria crearem un fitxer de headers que inclourà tots els headers dels mòduls. El seu nom serà `pbn.h` i el seu contingut el següent:

```
#include <queue.h>
#include <timer.h>
#include <serial_device.h>
#include <bclk_serial.h>
#include <mchar.h>
#include <mtbl.h>
#include <modulator.h>
#include <ether.h>
#include <lamp.h>
#include <gpio_device>
#include <semaph.h>
```

Noteu que no és necessari parentitzar el contingut d'aquest header usant `#ifndef` ja que seria redundant.

A nivell organitzatiu la llibreria (i tots els mòduls que la composen) estarà en un subdirectori específic del directori amb els fonts del projecte (vegeu la figura 1). Aquest subdirectori l'anomenarem `libpbn`, com no podia ser d'altra forma. La llibreria ha de tenir el seu propi `Makefile`, que ha de residir en el mateix directori. El target principal d'aquest makefile ha de ser la pròpia llibreria de forma que invocant `make` sense altres paràmetres es reconstrueixi `libpbn.a`. Make, de fet la implementació de make del projecte GNU, que és la que estem usant, sap mantenir

les llibreries usant regles específiques. És molt recomanable fer-ho amb aquestes regles ja que simplifiquen molt els procés. A tal efecte llegiu [Fre10, cap. 11]. Com és habitual, el makefile ha de contenir també els targets `clean` i `veryclean`.

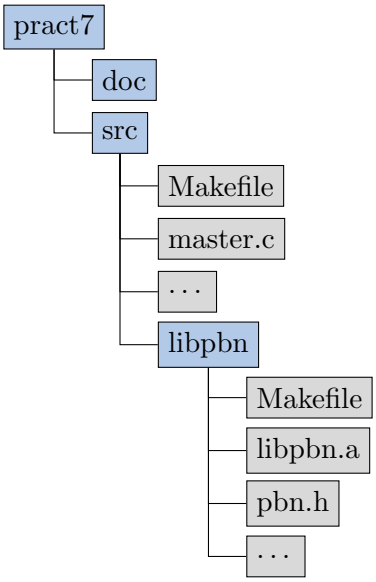

Figura 1: Estructura organitzativa de la pràctica.

El directori `src` contindrà els fonts dels diversos executables del projecte i també el seu propi makefile. Aquest makefile serà especialment simple, ja que la majoria de la feina ara ha quedat circumscrita al manteniment de la llibreria `libpbn`. Com encara no hem desenvolupat nou material, de moment aquest directori i el corresponent makefile només es preocuparan del `master`, que és exactament l'executable de la pràctica anterior.

En aquestes condicions, el `Makefile` hauria de tenir aquest aspecte (deixant a banda les regles genèriques per implantar el codi en l'Arduino):

```
CC=avr-gcc
CPPFLAGS=-DF_CPU=16000000UL
CFLAGS=-Wall -std=c99 -Os -mmcu=atmega328p -fshort-enums -Ilibpbn

vpath lib% libpbn

master: control.o -lpbn
master.o: control.h

.PHONY: libpbn
libpbn:
	$(MAKE) -C libpbn

.PHONY: clean veryclean
clean:
	\rm -f *~ *.o *.s *.hex
	$(MAKE) -C libpbn clean
```

```
veryclean: clean
	\rm -f master
	$(MAKE) -C libpbn veryclean
```

Aquest makefile incorpora nombroses novetats:

- En les flags de compilació s'usa `-Ilibpbn` per indicar al compilador que també ha de cercar headers estàndard en el (sub)directori `libpbn`.
- La sentència `vpath lib% libpbn` indica a make que, si cal, pot cercar fitxers que comencin per `lib` dins del directori `libpbn`. Això permetrà que make trobi de forma automàtica la llibreria.
- En les dependències de `master` hi surt el text `-lpbn`. Així indica a make que aquest executable depen d'aquesta llibreria i que, per muntar-lo, cal fer-ho amb aquesta llibreria. Cal fer notar que la notació `-lpbn` implica la cerca de la llibreria amb nom de fitxer `libpbn.a` (veure manual de gcc).
- Finalment, en diversos llocs s'invoca a make recursivament. Per exemple, la regla:

  ```
  libpbn:
      $(MAKE) -C libpbn
  ```

  Indica que, per reconstruir la llibreria, cal invocar a `make` usant com a directori `libpbn`. Això significa que s'invocarà a make usant el makefile corresponent a aquest directori. En certa manera és molt similar a haver escrit una regla així:

  ```
  libpbn:
      cd libpbn; make; cd ..
  ```

El programa `master`, que en la pràctica anterior havíem anomenat `crossing`, i el mòdul `control` també necessiten uns petits canvis al copiar-los de la pràctica anterior. Essencialment cal substituir els diversos `#include` que feien referència als mòduls per un únic `#include <pbn.h>`. Efectivament, la nostra llibreria ha pujat de categoria i ara té el mateix tractament que els headers estàndards. Això és la conseqüència directa d'haver usat el flag `-Ilibpbn` al compilar.

Tasca 2 Prepareu l'estructura de directoris per a aquesta pràctica. Copieu els fitxers escaients de la pràctica anterior i feu els canvis de nom que siguin adequats. No cal que transferiu els programes de test.

A continuació creeu el makefile corresponent a la llibreria i el header `pbn.h`. Construïu la llibreria.

Modifiqueu el fitxer `master.c` tot i canviant convenientment els includes corresponents. Creeu el makefile corresponent als programes i comproveu que podeu implantar el màster a l'Arduino correctament.

Tasca 3 Modifiqueu el temps de tic del mòdul`timer` a 5 ms. Assegureu-vos que el temps de DOT és de 90 ms. Si és necessari modifiqueu-lo. Amb temps més curts tindreu més problemes de recepció.

3 Experimentació amb el desmodulador

El circuit desmodulador que usarem correspon a l'integrat TSOP18438 [www21]. Aquest integrat disposa únicament de 3 potes. La pota 1 correspon a la sortida desmodulada, la pota 2 va connectada a `GND` i la pota 3 va connectada a alimentació de l'Arduino de 5 V. Aquest circuit ens ofereix una sortida digital que indica si el circuit detecta senyal (0) o silenci (1). Observeu que quan hi ha senyal la pota de sortida val 0!

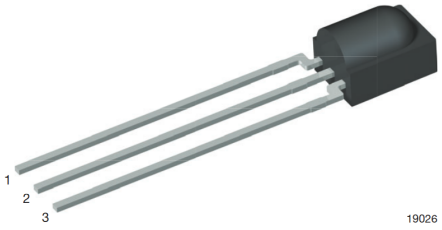

Figura 2: Integrat desmodulador d'infraroig del senyal morse.

El treball amb el desmodulador és convenient experimentar-lo prèviament. La recepció i descodificació del senyal és farcida de dificultats, entre les que cal comptar-hi: la sensibilitat del receptor, l'atenuació produïda per la distància emissor-receptor, el soroll en la transmissió, la interferència produïda per altres fonts de llum (solar, fluorescents, etc.). Això fa interessant experimentar prèviament una mica amb el desmodulador per tal d'assumir les seves característiques.

En aquest apartat construirem dos receptors experimentals que després ens serviran per dissenyar el definitiu.

3.1 Primer experiment

El primer experiment que cal fer és dissenyar i implementar un programa que permeti captar el senyal del desmodulador i encendre un LED quan arriba un senyal baix. Per governar el LED cal usar el mòdul `gpio_device` de la llibreria. Per captar el senyal del circuit desmodulador caldrà usar la detecció de flancs en un port d'entrada, tant de pujada com de baixada, i la generació de la conseqüent interrupció. És convenient mirar-se la documentació corresponent de l'AVR, [Atm18, cap. 13, *External interrupts*]. En el nostre cas usarem la interrupció INT0, que està associada al pin PD2 del PORTD tal com es diu a [Atm18, ap. 14.3.3, *Alternate Functions of Port D*].

Tasca 4 Feu el muntatge corresponent a aquest primer experiment. Connecteu l'alimentació del desmodulador als pins corresponents d'un dels Arduinos: `GND` i `5V`. Connecteu la sortida de senyal del desmodulador al pin PD2 de l'AVR. El LED podeu connectar-lo en el pin que voleu sempre que després inicialitzeu un pin convenientment.

En aquest experiment, la rutina d'interrupció corresponent ha de commutar l'estat del LED. Així el flanc de baixada l'encén i el de pujada l'apaga.

Per generar el senyal infraroig, el programa de test del mòdul `modulator` que vàreu usar en la pràctica anterior us pot fer servei. Si és necessari podeu copiar-lo cap aquesta pràctica. Podeu provar l'efecte que tenen altres fonts lluminoses com fluorescents, comandaments a distància, la llum del sol, etc. Si comproveu que aquestes fonts no provoquen canvis és que el desmodulador és prou robust a interferències.

TASCA 5 Copieu el programa de test del mòdul `modulator` i modifiqueu-lo per tal que generi un senyal periòdic de so-silenci amb un període de $T = 300\,\text{ms}$. Adapteu el makefile corresponent i proveu el programa sobre un dels Arduinos.

Dissenyeu un programa que rebi el senyal del demodulador i actuï com s'ha explicat anteriorment. Adapteu el makefile. Implanteu-lo i comproveu que esteu reben correctament el senyal original. Experimenteu variant la posició del receptor i introduint soroll lluminós.

3.2 Segon experiment

Per tal de fer més robusta la detecció de flancs, modifiqueu la rutina d'interrupció per tal que decideixi quan es troba davant d'un flanc "veritable" o davant d'un espuri que cal ignorar.

La modificació consisteix en implementar una detecció per votació. A tal efecte cal tenir una variable que emmagatzemi l'estat del senyal que prové del desmodulador, que pot ser Baix o Alt. Cada vegada que arriba una interrupció, comprovem el valor del port al cap de 2 µs, 4 µs i 8 µs. A cada comprovació el senyal pot ser Baix o Alt. Guanya l'estat que més vots obté d'entre les tres comprovacions. Naturalment, només farem cas de la votació si el resultat és coherent amb l'estat en que ens trobem. Abans de retornar de la rutina d'interrupció cal esborrar les interrupcions pendents a fi i efecte de que no provoquin interrupcions encadenades. L'efecte d'aquesta tècnica de decisió el podeu veure en el cronograma de la figura 3.

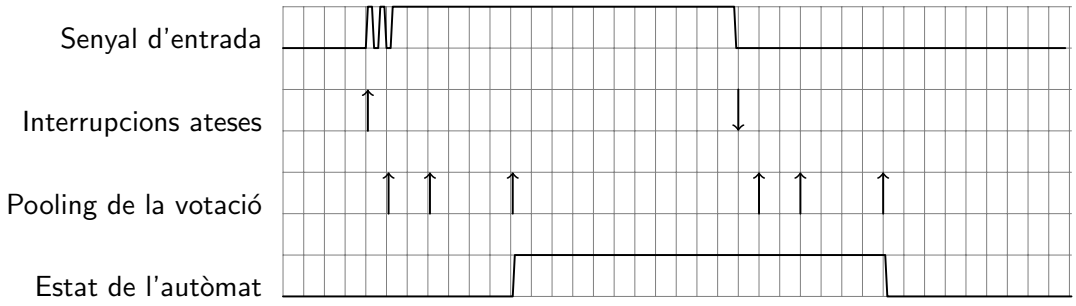

Figura 3: Efecte de l'algoritme de decisió per votació. Observeu com filtra el transitori inicial.

A continuació teniu un esquema en pseudocodi del procediment:

```
votacio = 0
for tau in [2,4,8]:
    wait_us(tau)
    if bit_set(INPUT_PIN):
        votacio += 1
    else:
        votacio -= 1
if votacio > 0 and estat == Baix:
    toggle_led()
    estat = Alt
elif votacio < 0 and estat == Alt:
    toggle_led()
    estat = Baix
cancela_interrupcions_pendents()
```

Fixeu-vos que en realitat implementa un petit autòmat de dos estats. El resultat de les votacions decideix si cal canviar d'estat o no. Cada transició de l'autòmat provoca una acció, en aquest cas commuta l'estat del LED. Tingueu en compte quina relació ha de tenir l'estat d'aquest autòmat i l'estat del LED.

TASCA 6 Implementeu la detecció per votació i observeu si és menys sensible al soroll. Proveu de variar els temps (sempre petits ja que afecten al procés de les interrupcions) o de votar més vegades i estudieu-ne les conseqüències.

4 Implementació de cronòmetres

L'objectiu d'aquesta tasca és ampliar els serveis relacionats amb el temps que ofereix el mòdul `timer` de la llibreria. En la versió que tenim ofereix un servei d'accions temporitzades, que permet llençar l'execució d'una acció en un temps futur específic.

Ara caldrà afegir un nou tipus de servei al mateix mòdul: cronòmetres. Un cronòmetre és un objecte que permet mesurar el temps transcorregut des d'un cert instant. Els cronòmetres seran importants per a poder mesurar el temps durant la descodificació del senyal morse.

L'API dels cronòmetres que cal afegir és la següent:

typedef int8_t timer_chrono_t	Defineix el tipus cronòmetre. En realitat és un handler com el de les accions temporitzades però que fa referència a un cronòmetre. Els cronòmetres es manipulen sempre a través del seu handler.
timer_chrono_t chrono(**void**)	Crea un nou cronòmetre i en retorna el handler o bé TIMER_ERR si no s'ha pogut crear
void chrono_start(timer_chrono_t c)	Posa a zero el cronòmetre i fa que comenci a comptar el temps transcorregut.
uint8_t chrono_get(timer_chrono_t c)	Retorna el temps en tics transcorregut des del darrer start del cronòmetre. No para el cronòmetre.
void chrono_stop(timer_chrono_t c)	Atura el cronòmetre però no el destrueix. Un cronometre aturat no s'actualitza.
void chrono_cancel(timer_chrono_t c)	Destrueix el cronòmetre. A partir d'aquest moment el handler és invàlid i no representa a cap cronòmetre.

La implementació dels cronòmetres es fa, principalment, modificant la taula del mòdul. Si fins ara tenia només accions planificades, ara passa a tenir també un altre tipus de cel·les possibles: cronòmetres. Dels cronòmetres cal conservar únicament l'estat en que es troben (parats o engegats) i el temps acumulat que ha transcorregut des del darrer reset. Hi ha diverses formes d'implementar aquesta modificació. Una possibilitat rau en l'ús de la construcció **union** de `C`. Naturalment, també caldrà modificar de manera conseqüent altres funcions del mòdul i, molt especialment, la rutina d'interrupció que actualitza la taula. Noteu que el rang de mesura dels cronòmetres està fitat. Si se supera el rang màxim el cronòmetre recomença de nou a comptar per 0.

TASCA 7 Amplieu el mòdul `timer` per a que ofereixi també cronòmetres.

5 Implementació del receptor

El receptor cal implementar-lo estenent el mòdul `ether`. Fixeu-vos que el mòdul `ether` recorda al mòdul `serial`. Aquesta similitud és deliberada. Per completar `ether` caldrà afegir la part de recepció de codi morse. L'API que caldrà afegir al mòdul és totalment paral·lela a la de `serial`:

bool ether_can_get(**void**)	Retorna true sii hi ha un o més caràcters rebuts pel canal morse disponibles per a ser llegits. Si aquesta funció retorna true es garanteix que una posterior crida a ether_get() no es bloquejarà.
uint8_t ether_get(**void**)	Retorna el següent caràcter rebut pel canal morse. En cas que no n'hagi cap de disponible es bloqueja fins que n'hi ha un.

La recepció morse cal organitzar-la de la següent forma. En primera instància hi haurà una rutina d'interrupció que, usant la tècnica de votació vista abans, detectarà flancs ascendents i descendents provinents del desmodulador. D'aquest autòmat en direm l'autòmat de nivell 1 i serà capaç de produir els esdeveniments FEdge i REdge, *Falling* i *Raising edge* respectivament. En aquest primer autòmat s'ha filtrat possibles flancs espuris d'una amplada de fins a 8 µs. Degut a la natura del desmodulador, és possible que es produeixin flancs espuris d'amplada similar alguns períodes del senyal modulador de 38 kHz. Això implica que es poden donar flancs espuris de l'ordre de pocs mil·lisegons. El filtrat d'aquests flancs no es pot delegar a la mateixa rutina d'interrupció ja que bloquejaria el sistema un temps excessiu. Per aquest motiu cal la construcció d'un autòmat de nivell 2 encarregat d'aquesta tasca.

Els esdeveniments generats per l'autòmat de nivell 2 conjuntament amb altres esdeveniments que veurem més endavant són els que mouran un tercer autòmat que tindrà com a responsabilitat decidir quina seqüència de punts i ratlles s'està rebent. Aquest autòmat l'anomenarem l'autòmat de nivell 3. Cada vegada que aquest autòmat decideixi que s'ha rebut un caràcter, el caràcter s'encuarà en una cua de recepció. Les operacions de l'API treballaran desencuant caràcters d'aquesta cua.

5.1 L'autòmat de nivell 1

L'autòmat de nivell 1 s'implementa seguint les idees que s'han treballat a l'apartat 3.2. Cada vegada que detecta un flanc emet l'esdeveniment adequat i informa l'autòmat de nivell 2.

A tal efecte, suposarem que l'autòmat de nivell 2 té una funció associada que s'usa per comunicar-li un esdeveniment: l2_event(l2_event_t e). Aquesta funció rep com argument l'esdeveniment e i, segons l'estat en que es troba, dispara la transició adequada en l'autòmat de nivell 2. El tipus l2_event_t es defineix per enumeració i conté tots els esdeveniments possibles que pot rebre l'autòmat de nivell 2:

typedef enum {REdge, FEdge, ...} l2_event_t;

Així doncs, l'autòmat de nivell 1, cada vegada que detecta un nou flanc, n'informa a l'autòmat de nivell 2 emetent el corresponent esdeveniment. Aquesta arquitectura es coneix sovint amb el nom de *jerarquia d'autòmats*.

5.2 L'autòmat de nivell 2

La funcionalitat d'aquest autòmat serà eliminar espuris de l'ordre dels mil·lisegons. En aquest sentit, es pot pactar que interessa eliminar espuris que tinguin una durada del 10% de la unitat de temps bàsica del senyal morse (DOT). Anomenarem aquest temps per T_FILTER.

Aquest autòmat validarà tant un *Falling edge* com un *Raising edge* sempre i quan durant el temps de T_FILTER posterior no es produeixi un *edge* de sentit contrari. Just en aquest moment l'*edge* quedarà validat i es procedirà a la crida del tercer autòmat condicionat a l'esdeveniment de l'*edge* que s'hagi validat.

Per facilitar-vos la feina, a la figura 4 teniu el graf de com es pot muntar aquest autòmat. L'autòmat és un autòmat de Mealy i, per tant, les accions es produeixen quan s'activen les transicions, [Wik12].

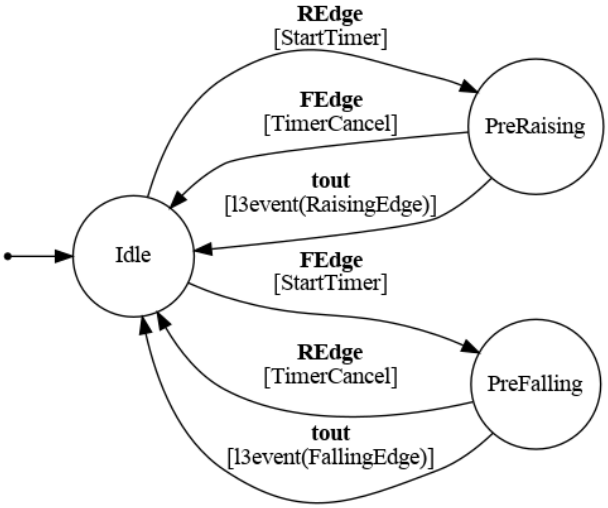

Figura 4: Filtre d'espuris de l'ordre del 10% de la mida del símbol bàsic.

En aquest autòmat els estats són:

Idle Indica que no s'ha rebut cap flanc candidat.

PreRaising Indica que hi ha un candidat a *Raising edge*.

PreFalling Indica que hi ha un candidat a *Falling edge*.

Les transicions corresponen als esdeveniments produïts. En aquestes transicions també estan indicades amb claudàtors les accions que s'han de fer.

Observeu que l'autòmat de nivell 2 és una peça que es pot ficar i treure del sistema molt fàcilment. És a dir, l'autòmat de nivell 1 (implementat a dins de la interrupció) pot cridar a l'autòmat de nivell 2 o directament a l'autòmat de nivell 3. En els dos casos els esdeveniments són els mateixos (REdge o FEdge). En un cas estaran filtrats i en l'altre no.

Tasca 8 Podeu analitzar quin efecte té sobre la correcció dels codis morse rebuts fer un filtrat amb un percentatge diferent al proposat.

5.3 L'autòmat de nivell 3

Per dissenyar l'autòmat de nivell 3 cal fer primer una anàlisi del senyal morse. A la figura 5 s'observa les possibles evolucions que pot tenir el senyal morse. Cal considerar que en aquesta figura s'indica la presència de senyal amb senyal de nivell alt i l'absència de senyal amb nivell baix, contrari a com ho fa el descodificador escollit.

S'observa que:

- Quan arriba un flanc de pujada, l'estat del decodificador està a l'espera de decodificar un **Dot** o un **Dash** o bé, si el senyal continua actiu, un **Error**.
- Quan arriba un flanc de baixada, l'estat del decodificador està a l'espera de decodificar un **Gap de Símbol**, o un **Gap de Caràcter** o bé un **Gap de Paraula**.

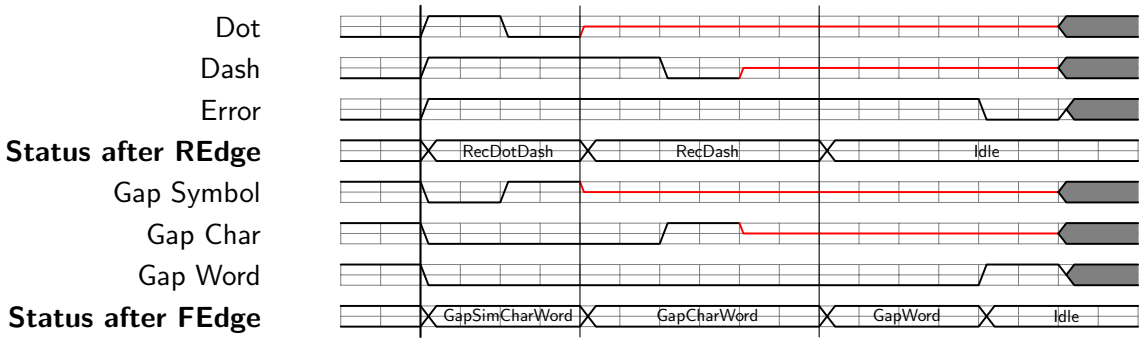

Figura 5: Relació Inici de To o Silenci vs estat del decodificador Morse.

- Per diferenciar entre un **Dot** (1 unitat de temps bàsic) i un **Dash** (3 unitats de temps bàsic) es defineix un llindar temporal T_SHORT al punt mig entre els dos valors. Aquest punt mig es troba a 2 unitats de temps des de l'instant que es produeix el flanc de pujada. Conseqüentment es pot afirmar que:
  - Es detecta un **Dot** si es produeix un flanc de baixada *abans* de passar un temps T_SHORT.
  - Es detecta un **Dash** si es produeix un flanc de baixada *després* de passar un temps T_SHORT.
- Per diferenciar entre un **Gap de Símbol** i un **Gap de Caràcter** s'aplica el mateix criteri que en el cas anterior.
- Per diferenciar entre un **Gap de Caràcter** (3 unitats de temps bàsic) i un **Gap de Paraula** (7 unitats de temps bàsic) es defineix un nou llindar temporal T_LONG al punt mig entre aquests valors, sent de 5 unitats de temps des de l'inici del flanc. En aquest cas:
  - Es considera un **Gap de Caràcter** si es produeix un flanc de pujada abans de passar un temps T_LONG.
  - Es considera un **Gap de Paraula** si es produeix un flanc de pujada després de passar un temps T_LONG.
- Podem considerar un senyal erroni el cas que es superi aquest llindar T_LONG quan estem rebent un **Dash**. En aquest cas descartarem els símbols recuperats del caràcter morse i començarem de nou la descodificació.

La figura 5 mostra també els possibles estats en que es pot trobar el descodificador morse:

Idle Correspon a l'estat inicial i quan no hi ha senyal a descodificar.

RecDotDash En aquest estat s'ha començat a rebre senyal però encara no està definit si serà un **Dot** o un **Dash**.

RecDash En aquest estat s'esta rebent un **Dash** però cal confirmar que no es produirà un error per excés de durada.

GapSimCharWord A aquest estat s'arriba a l'inici de manca de senyal o inici de silenci. Però s'ha d'esperar al següent flanc o un temps suficient per confirmar quina mida tindrà aquest silenci i així distingir entre els 3 casos possibles.

GapCharWord Es passa a aquest estat quan s'ha superat el temps T_SHORT. En aquest estat s'ha acabat la rebuda de símbols corresponents a un caràcter morse. Encara cal diferenciar entre un **Gap de Caràcter** o un **Gap de Paraula**.

GapWord Aquest últim estat confirma que s'ha superat el T_LONG i per tant correspon a **Gap de Paraula**. Cal mantenir el descodificador en aquest estat el temps suficient abans de passar a l'estat Idle per evitar possibles solapaments temporals dels senyals descodificats.

A la figura 6 es mostra el graf corresponent a l'autòmat de nivell 3. En aquest graf no es mostren les accions que s'han de fer. Aquest autòmat correspon a la versió completa del desmodulador morse. Per a aquesta pràctica es podrien fer simplificacions ja que el caràcter espai també es codifica com a combinacions de símbols **Dot** i **Dash**.

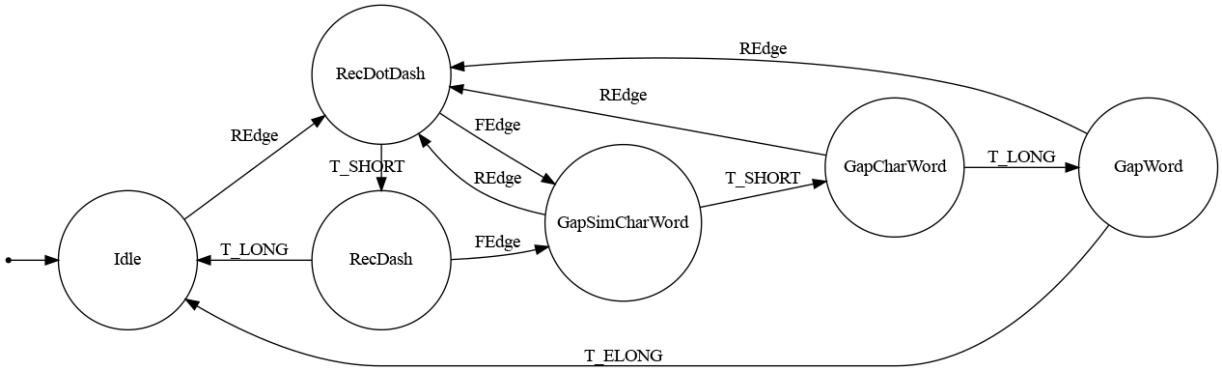

Figura 6: Graf del desmodulador morse.

Quan l'autòmat detecta un silenci llarg recopila els punts i ratlles detectats fins el moment sobre un mchar_t, el tradueix a caràcter i l'encua. Cal destacar que aquest autòmat és capaç de descodificar de la millor manera possible qualsevol senyal morse encara que presenti anomalies a causa del soroll o de deficiències de la recepció. Per garantir un comportament més robust, cal que l'autòmat tingui una bona resposta en cas que es rebin dos esdeveniments FEdge o REdge seguits.

Tasca 9 Definiu les accions que cal fer a les transicions de l'autòmat de nivell 3.

5.4 Implementació

Com ja s'ha dit l'autòmat de nivell 1 s'implementa de manera homòloga a com s'ha fet en l'experiment de la secció 3.2.

Pel que fa a l'autòmat de nivell 2 i el de nivell 3, el que cal és implementar la funció l2_event(l2_event_t e) i l3_event(l3_event_t e). Aquestes funcions són les encarregades de processar els esdeveniments que mouen aquests autòmats. La majoria d'aquests esdeveniments són moviments de flanc del tipus REdge i FEdge. Però també existeixen transicions que es disparen quan ha passat cert temps. La forma d'implementar aquestes transicions es basa en el següent mecanisme.

Per una banda es defineix un esdeveniment específic lligat a aquest fet, diguem-ne TimeOut. D'altra banda, quan és el moment s'activa una funció planificada per a ser executada al cap de T_OUT que la única cosa que fa és informar a l'autòmat d'aquest esdeveniment. Un esquema d'aquest mecanisme podria ser el següent:

```
typedef enum {FEdge, REdge, TimeOut, ...} l2_event_t;

void l2_event(l2_event_t e);

void raise_stimeout(void) {
  l2_event(TimeOut);
}

void l2_event(l2_event_t e) {
  switch (current_state_l2) {
    ...
    timer_after(raise_timeout, T_OUT);
    ...
  }
}
```

TASCA 10 Implementeu la part de recepció del mòdul `ether`. Per tal de provar el seu funcionament cal que implementeu dos programes (a implantar sobre dos Arduinos). Un llegeix del port sèrie i transmet els caràcters en morse. L'altre escolta el canal morse i envia pel port sèrie els caràcters que ha llegit. Si tot funciona com cal, haureu de poder establir un enllaç morse entre dos computadors o entre dues terminals d'un mateix computador. Escrivint en una terminal haureu de veure tant la transmissió com la recepció en l'altra terminal. Observeu el retard a causa de la baixa velocitat de transmissió i observeu l'efecte dels buffers o cues.

6 Implementació final de l'esclau

Finalment només queda enganxar totes les peces. Per un costat ja teníeu de la pràctica anterior el màster, que s'encarrega de controlar una cruïlla i, simultàniament, transmetre les ordres correctes per morse.

Ara us queda escriure el programa principal de l'esclau. El programa és molt similar al mestre però en aquest cas només rep les ordres pel canal morse i no usa el canal sèrie per res.

TASCA 11 Implementeu l'esclau. Noteu que en cas de rebre ordres desconegudes o incoherents l'esclau simplement les ignora.

Referències

[Atm18] Atmel. *ATmega48A/PA/88A/PA/168A/PA/328/P Complete datasheet*. Anglès. Versió DS40002061A. 2018. URL: `http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf` (consultat 30 d'abr. de 2020).

[Fre10] Free Software Foundation – GNU Project. *GNU Make User Manual*. Anglès. Versió 3.82. 2010. URL: `http://www.gnu.org/software/make/manual` (consultat 20 de febr. de 2017).

[Wik12] Wikipedia contributors. *Mealy Machine*. Anglès. Wikipedia, The Free Encyclopedia. 2012. URL: `http://en.wikipedia.org/wiki/Mealy_machine` (consultat 20 de febr. de 2017).

[www21] www.mouser.es. *tsop184-1767178*. Anglès. www.vishay.com. 2021. URL: `https://www.mouser.es/datasheet/2/427/tsop184-1767178.pdf` (consultat 1 de juny de 2021).