Pràctica 6: Control semafòric de cruïlla amb comunicació morse: Mestre

Programació a Baix Nivell — iTIC

Sebastià Vila-Marta Paco del Águila

20 d'abril de 2015

Índex

1 Introducció

Aquesta pràctica és una ampliació de la pràctica anterior en que s'afegeix la possibilitat que el sistema semafòric comuniqui (a sistemes semafòrics veïns) les ordres que rep usant codi morse sobre una portadora audible. En una segona part d'aquesta pràctica, dissenyarem un sistema que descodifiqui el senyal morse rebut i l'interpreti.

El sistema final, una vegada acabada aquesta pràctica i la següent, 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 de senyal audible.
- 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 mestre, cosa que es farà ampliant convenientment la pràctica 5. Per tal de fer aquesta ampliació molts dels mòduls de la pràctica anterior s'utilitzen integralment, alguns cal retocar-los lleugerament i, a més, cal escriure nous mòduls. De forma resumida podem dir que:

- Cal afegir la infraestructura necessària per convertir caràcters a codis morse i viceversa.
- Cal afegir un mòdul per generar el senyal de la portadora a 3.1 kHz.
- Cal afegir un servei, per a l'usuari mol similar al del mòdul `serial_device`, que permeti enviar cadenes de caràcters en morse.
- Com que el sistema ha de fer un ús molt més sofisticat de les accions relacionades amb el temps, caldrà implementar un servei de temporització.

1.1 Organització

1. Cal fer la pràctica en equip de dues o tres persones.
2. Caldrà lliurar un petit informe del temps de dedicació.
3. Cal desenvolupar el programari i la documentació usant els serveis del sistema de gestió de versions de l'EPSEM.
4. La pràctica té una durada de tres sessions.

Pel que fa al material necessari, cada equip:

1. Cal que disposi de 6 LED (2 vermells, 2 grocs i 2 verds) per la realització d'aquesta pràctica o bé del *shield iTIC*.
2. Cal disposar d'un petit altaveu piezoelèctric amb o sense oscil·lador integrat capaç de treballar a 5 V amb una intensitat típica de 20 mA to 30 mA.

3. Cal que disposi d'un Arduino i la corresponent placa de prototips amb els seus accessoris habituals.

   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. La taula de temps de dedicació.
3. 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://escriny3.epsem.upc.edu`). Per tant cal preparar prèviament la disposició de directoris necessària.

Tasca 2 Prepareu la disposició de directoris i fitxers que us convinguin per començar a desenvolupar el projecte sobre el sistema de control de versions.

2 Descripció del sistema

2.1 Arquitectura

En el giny que cal construir intervenen dos sistemes diferenciats que són anàlegs als de la pràctica anterior:

1. El sistema semafòric. S'encarrega d'operar el semàfor d'acord amb una rutina preestablerta i les indicacions del sistema de supervisió. El nucli del sistema semafòric és el control semafòric, que interacciona amb driver del semàfor i amb els sistemes de comunicacions. El sistema semafòric s'implementarà usant el kit d'Arduino i un petit muntatge maquinari auxiliar.
2. El sistema de supervisió. S'encarrega de controlar el sistema semafòric. L'implementarem com un programa escrit en Python que s'executa en l'estació de treball. Es comunica amb el sistema semafòric a través del port sèrie usant un protocol textual molt senzill.

La figura 1 mostra un esquema d'aquesta arquitectura.

El sistema té un funcionament similar al seu predecessor en la pràctica anterior. El sistema de control semafòric governa els semàfors de la cruïlla i els va fent canviar d'estat d'acord amb un cicle de treball preestablert. El sistema de comandament rep ordres del supervisor a través del sistema de comunicacions i les executa afectant el funcionament del control semafòric. El sistema de supervisió es comunica amb el sistema de comandament a través d'un petit protocol.

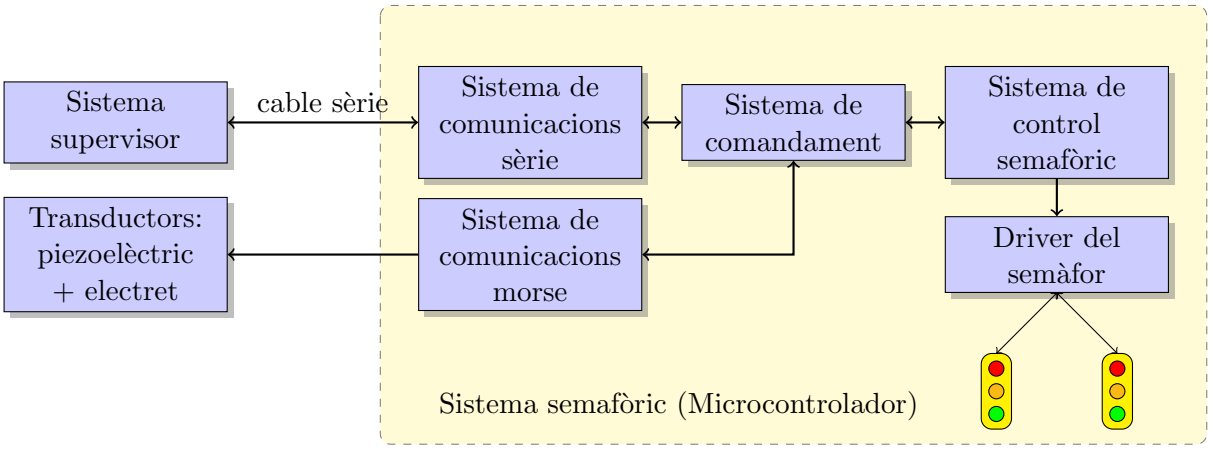

Figura 1: Arquitectura del sistema

2.2 Protocol de supervisió

És el protocol que usen el sistema de supervisió i el control semafòric per a col·laborar. És un protocol en el que els missatges sempre són línies de text acabades en *carriage return* i *linefeed* (`"\r\n"`).

El sistema semafòric envia un primer missatge al sistema de supervisió quan ja s'ha inicialitzat i està a punt per funcionar. El supervisor no pot actuar fins que ha rebut aquest missatge. Aquest missatge és `START`. A partir d'aquest moment la iniciativa sempre la té el supervisor.

El sistema supervisor identifica els dos carrers que controla amb els noms `A` i `B`.

Els missatges que el supervisor pot enviar són els següents:

FA *Force A.* Força el semàfor del carrer `A` a verd. El pas a verd pot no ser immediat atès que sempre cal respectar el tems de carabassa corresponent. El sistema torna el missatge `OK` si tot va bé.

FB *Force B.* Idènticament a `FA` però pel carrer `B`.

SA *A state?.* Consulta quin és l'estat del semàfor del carrer `A`. El sistema torna els missatges `CLEAR`, `APPROACH` o `STOP` segons sigui el cas.

SB *B state?.* Idènticament que `SA` pel semàfor del carrer `B`.

R *Run.* Si el sistema està parat l'engega i aquest contesta amb el missatge `OK`. En cas contrari no fa res i contesta també amb `OK`.

H *Halt.* Atura els semàfors i els apaga fins a nova ordre. Si tot va bé el sistema contesta `OK`. Si el sistema està aturat els missatges `FA` i `FB` no tenen cap efecte.

Qualsevol altre missatge que envii el supervisor es considera un error i el sistema de semàfors respon amb el missatge `COMERR` per indicar aquesta condició. Noteu que el protocol en cap cas fa echo del que s'escriu.

2.3 Estructura de mòduls

A banda del programa `Python` que actua de sistema supervisor, el programari que governa el microcontrolador s'escriurà usant `C` i s'organitzarà en diversos mòduls que responen al diagrama de mòduls de la figura 2.

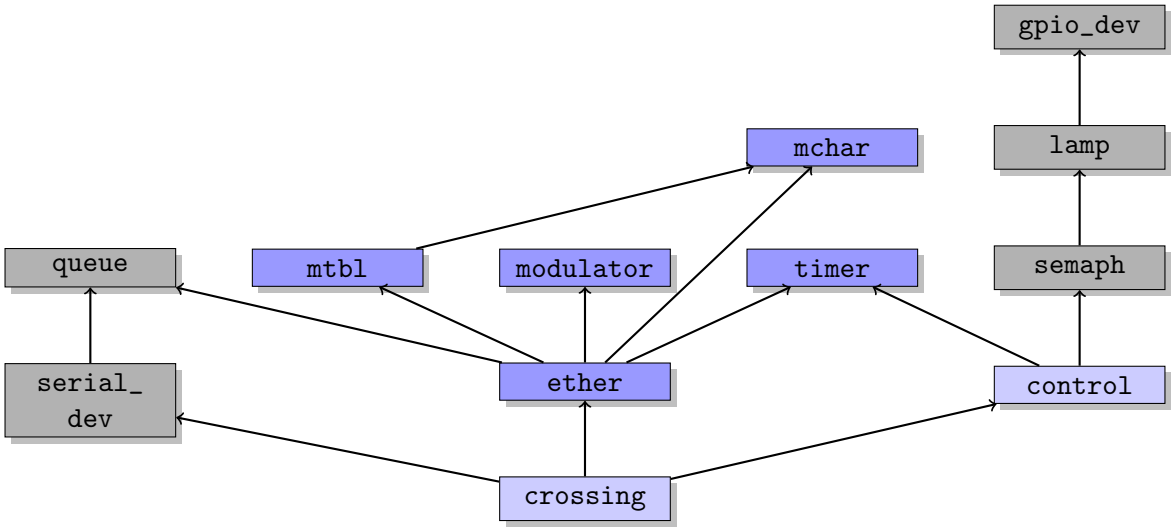

Figura 2: Diagrama de mòduls de la pràctica. Els de color blau són nous, els blau cel modificats i els grisos totalment reutilitzats.

Els mòduls compartits amb el projecte anterior tenen les mateixes responsabilitats (tot i que en alguns es modifiquen les implementacions). Els nous mòduls tenen la següents responsabilitats:

mchar Defineix el tipus mchar_t, que representa un caràcter codificat en morse, i les operacions associades.

mtbl Defineix les operacions per convertir d'ASCII a Morse i viceversa segons una taula de codificació constant.

modulator El mòdul ofereix un modulador amb una portadora a 3.1 kHz. S'usa per conduir el transductor piezoelèctric.

timer El mòdul és una abstracció basada en programari d'un temporitzador universal. Permet gestionar accions que cal realitzar en un temps o amb una periodicitat concreta.

ether El mòdul ofereix un servei de comunicacions a través de l'èter usant codi morse. En aquesta pràctica només s'implementarà la part de sortida. Té una API molt similar a la del mòdul `serial`.

Noteu que el supervisor tampoc cal modificar-lo: el protocol de supervisió és el mateix i per tant el programa supervisor pot ser el mateix.

3 Implementació del semàfor

La implementació física és com en la pràctica anterior. Cal tenir en compte, però, que en aquest cas el transductor piezoelèctric es condueix usant el generador de senyal associat al temporitzador 2 del microcontrolador i, per tant, està fixat al pin `PORTB3` (`OC2A` en el sub-sistema del temporitzador). Això significa que els semàfors no poden usar aquest pin.

Tasca 3 Munteu els LED i les corresponents resistències, si escau, en la placa de prototips. Munteu també el transductor piezoelèctric connectat al pin `PORTB3`. Cablegeu les connexions a l'Arduino. Procureu que cada semàfor tingui els pins assignats correlativament i que el LED verd correspongui al pin de menor pes.

4 El mòdul «modulator»

El mòdul `modulator` permet generar un senyal digital modulat sobre una portadora de 3.1 kHz. Aquest tipus de modulació és un cas particular de la modulació d'amplitud. Quan s'usa el codi Morse per conmutar on-off l'ona portadora es diu que s'està fent una modulació en ona contínua (CW). Per fer això usa el temporitzador 2 del microcontrolador. Aquest temporitzador es pot configurar de manera que generi autònomament un senyal quadrat d'una freqüència preestablerta sobre el pin `PORTB3`.

> Tasca prèvia 4 Estudieu el funcionament del temporitzador 2 de l'AVR, [Atm11, capítol 15], i com pot usar-se per generar un senyal de manera autònoma quan es configura en mode CTC. Determineu amb quins paràmetres cal configurar-lo per tal que el senyal generat sigui de 3.1 kHz. Esbrineu també com cal fer-ho per aturar i engegar la generació de senyal. Tingueu en compte que si atureu la generació mentre el senyal de sortida del pin `PORTB3` és alt, restarà alt indefinidament. Per tant en aturar la generació cal forçar a baix el senyal de sortida.

El mòdul té la següent especificació:

void modulator_init(**void**)	Inicialitza el mòdul. Després de la inicialització el senyal de sortida és baix.
void modulator_set(bool l)	Canvia l'estat del senyal de sortida. Si l és true activa la generació de portadora, si és false desactiva la generació de portadora.

En el cas que generador piezoelèctric ja tingui incorporat l'oscil·lador, no s'hauria de fer servir el pin `PORTB3`. En aquest cas s'hauria de fer servir un pin que s'activés a 5 V de manera constant el temps que es vulgui que el generador reprodueixi el to. Per tal de fer una implementació el més genèrica possible i compatible amb els dos casos, es farà que el pin del `PORTB5` corresponent al LED serveixi per governar un generador amb oscil·lador incorporat. Així, es podrà fer servir un generador o un altre en funció del pin on es connecti sense necessitat de canviar cap paràmetre del programa.

Tasca 5 Implementeu el mòdul `modulator` i també un programa de test `test_modulator` per provar el seu funcionament correcte.

5 El mòdul «mchar»

El mòdul `mchar` implementa el tipus mchat_t, que representa la codificació morse d'un caràcter. La implementació del tipus es fa sobre un byte, de manera que els 5 bits de més pes representen els signes morse (0 dot i 1 dash) i els 3 bits de menys pes són la longitud (en nombre de signes morse que intervenen en el caràcter). Així, per exemple, la lletra `B`, que es codifica en morse com "`-...`", es representaria com el byte `0b10000100`.

El mòdul ofereix un seguit d'operacions per manipular valors de tipus mchar_t. Algunes d'aquestes operacions s'implementen directament com a macros del pre-processador atesa la seva simplicitat o bé per que són més convenients a l'hora de representar constants.

Per tal de representar els signes morse es defineix també un nou tipus enumerat msign_t.

typedef enum {MDot, MDash} msign_t	Defineix els dos tipus de signes que usa la codificació morse.
typedef uint8_t mchar_t	Defineix la codificació d'un caràcter en morse.
#define mchar(c) (mchar_t)((0b##c << (8−**sizeof**(#c)+1)) | (**sizeof**(#c)−1))	Funció que retorna un mchar_t constant (vegeu explicació més endavant).
#define mchar_len(m) (uint8_t)(m & 0x7)	Funció que retorna la longitud d'un mchar_t. És a dir el nombre de signes que conté. L'expressió mchar_len(mchar(010)), per exemple, val 3.
#define mchar_empty (mchar_t)0	Constant que defineix un objecte de tipus mchar_t buit, sense cap signe.
mchar_t mchar_add(mchar_t m, msign_t s)	Retorna el resultat d'afegir a m un nou signe s. Si m ja contenia 5 signes, el resultat és erroni. Noteu que l'expressió mchar_add(mchar(0),MDash), per exemple, equival al codi morse de la lletra `A`.

La funció (macro) mchar permet definir constants de tipus mchar_t de manera molt senzilla. Per exemple, el codi morse del caràcter `'A'` correspon a la seqüència "`.-`". Si volem assignar a una variable de tipus mchar_t el codi de la `'A'` faríem:

```
mchar_t v = mchar(01);
```

L'estudi de com està implementat mchar() és interessant per perfeccionar el coneixement del pre-processador de `C`. Si us decidiu a fer-ho, consulteu el manual [Pro12].

A més de les operacions anteriors, el mòdul ofereix també un iterador per recórrer fàcilment els signes que conformen el codi morse d'un caràcter. Aquest iterador està format per un tipus de dades i tres operacions específiques que responen a la següent especificació:

typedef struct{ mchar_t m; uint8_t mask,sentinel; } mchar_iter_t;	El tipus és un iterador que serveix per recórrer els signes del codi morse d'un caràcter. m és el codi que s'està recorrent. mask és una màscara amb un únic 1 que just indica el següent símbol d'm que cal recórrer. Quan mask == sentinel s'ha acabat vol dir que no hi ha més símbols per recórrer.
mchar_iter_t mchar_iter(mchar_t m)	Donat un codi m retorna un iterador inicialitzat sobre el primer signe.
msign_t mchar_next(mchar_iter_t ∗**const** i)	Torna el signe que indica l'iterador i posteriorment l'avança.
bool mchar_valid(mchar_iter_t i)	Retorna true si l'iterador i està sobre un signe i se li pot aplicar l'operació mchar_next().

Els iteradors són una eina fantàstica per recórrer estructures. Si, per exemple, volguéssim escriure amb punts i ratlles el codi contingut en una variable de tipus mchar_t podríem escriure una funció com aquesta:

```
void print_morse(mchar_t m) {
  mchar_iter_t i = mchar_iter(m);
  while (mchar_valid(i)) {
    if (mchar_next(i) == MDot)
      print(".");
```

```
    else
      print("-");
  }
}
```

TASCA 6 Implementeu el mòdul `mchar` d'acord amb l'API indicada. Implementeu també un mòdul `test_mchar` que us permeti comprovar-ne el correcte funcionament.

6 El mòdul «mtbl»

Aquest mòdul implementa les funcions per codificar i descodificar de ASCII a codi morse. La taula de codificació usada és la estàndard de ITU, [Wik12], i és fixa. El domini inclou els caràcters `A`–`Z`, els dígits `0`–`9` i el caràcter espai. Com l'espai no pertany a la taula ITU, el representarem amb la seqüència "`.-.-.`".

L'especificació del mòdul és la següent:

char mtbl_m2a(mchar_t c)	Donat un codi morse `m` retorna el caràcter corresponent. En cas que `m` no tingui un caràcter associat, retorna el caràcter `NUL` (`'\0'`).
mchar_t mtbl_a2m(**char** c)	Donat un caràcter `c` retorna el codi morse corresponent. En cas que no hi hagi cap codi morse associat retorna un `mchar_t` buit.

TASCA 7 Implementeu el mòdul `mtbl`. Decidiu com implementareu la taula de traducció i les dues funcions de traducció. Trieu aquella implementació que consumeixi menys memòria.

7 El mòdul «timer»

7.1 Especificació

Aquest mòdul ofereix un servei de temporització que generalitza i simplifica la utilització de les interrupcions del rellotge. Amb aquest mòdul se simplifica notablement l'execució de codi en instants de temps determinats.

La forma més bàsica d'usar aquest servei consisteix a escriure una funció `f()` que fa un cert càlcul i demanar al `timer` que planifiqui la seva execució per d'aquí a k ms. Al cap d'aquest temps, el servei cridarà automàticament la funció `f()`. El servei ofereix algunes funcionalitats més, com ara invocar la funció `f()` reiteradament cada k ms i algunes altres que veureu més endavant. Sovint, aquestes funcions que són cridades automàticament al cap d'un cert temps se les anomena *callbacks*.

> TASCA PRÈVIA 8 Per poder implementar aquesta estratègia és necessari entendre com es treballa amb `C` amb apuntadors a funcions. Els apuntadors a funcions permeten, per exemple, passar funcions com a paràmetres. En cas que no conegueu aquest mecanisme consulteu [KR88] i familiriaritzeu-vos-hi.

L'API d'aquest mòdul és el següent:

#define TIMER_MS(ms) (ms/10)	TIMER_MS() és una funció que converteix unitats de temps: de milisegons a ticks. Cal recordar que un tick en aquest context és la unitat mínima de temps amb que treballa aquest mòdul.
#define TIMER_ERR (timer_handler_t)−1	Constant de tipus timer_handler_t que indica un error en el moment de planificar una nova funció.
typedef void (∗timer_callback_t)(**void**)	timer_callback_t és un tipus de dades que denota un apuntador a una funció sense paràmetres i que retorna **void**.
typedef int8_t timer_handler_t	Un timer_handler_t identifica una funció que s'ha planificat per a ser executada en un futur sota el control d'aquest mòdul.
void timer_init(**void**)	Inicialitza el mòdul. Cal cridar-la com a mínim una vegada abans d'usar el mòdul. Només pot cridar-se amb les interrupcions inhabilitades.
timer_handler_t timer_after(uint8_t ticks, timer_callback_t f)	Planifica la funció f() per a ser executada al cap de ticks ticks. Retorna un handler que identifica aquesta acció planificada o bé val TIMER_ERR en cas que l'acció no es pugui planificar per alguna raó.
timer_handler_t timer_every(uint8_t ticks, timer_callback_t f)	Planifica la funció f() per a ser executada cada ticks ticks de manera indefinida. Retorna un handler que identifica aquesta acció planificada o bé val TIMER_ERR en cas que l'acció no es pugui planificar per alguna raó.
timer_handler_t timer_ntimes(uint8_t n, uint8_t ticks, timer_callback_t f)	Planifica la funció f() per a ser executada cada ticks ticks n vegades. Retorna un handler que identifica aquesta acció planificada o bé val TIMER_ERR en cas que l'acció no es pugui planificar per alguna raó. En cas que n sigui zero s'interpreta que la funció ha de ser cridada indefinidament. Notesu que aquesta funció subsumeix les dues anteriors.
void timer_cancel(timer_handler_t h)	Cancel·la l'acció planificada identificada per h. Si h no és un handler vàlid, no fa res.
void timer_cancel_all(**void**)	Cancel·la totes les accions planificades del servei.

7.2 Exemples d'ús

Les formes d'usar un servei com aquest són moltes. Imaginem, per exemple, que volem encendre i apagar un LED cada 500 ms de manera indefinida. Per fer-ho podem escriure una acció que commuti el LED d'estat i planificar-la usant el servei:

```
#include "timer.h"
void commuta_led(void) {...};

int main(void) {
  init_timer();

  timer_handler_t h = timer_every(TIMER_MS(500), commuta_led);
  if (h == TIMER_ERR))
    abort();

  for(;;);
}
```

Modifiquem ara el programa per tal que commuti el LED cada 200 ms i que aquest estat d'intermitència duri 2.5 s després dels quals el LED s'apaga indefinidament.

```
#include "timer.h"
static timer_handler_t h;

static void commuta_led(void) {...};
static void apaga_led(void) {...};

static void atura_intermitent(void) {
  timer_cancel(h);
  apaga_led();
}

int main(void) {
  init_timer();

  h = timer_every(TIMER_MS(200), commuta_led);
  timer_after(TIMER_MS(2500), atura_intermitent);

  for(;;);
}
```

Finalment fem un darrer exemple d'aplicació. En aquest cas volem encendre i apagar intermitentment un LED de forma que estigui 30 ms encès i 60 ms apagat. En aquest cas podem seguir una estratègia com aquesta:

```
#include "timer.h"

static void commuta_led(void) {...};
static void apaga_led(void) {...};

static void final_engegat(void) {
  commuta_led();
  time_after(TIMER_MS(60), final_apagat);
}

static void final_apagat(void) {
  commuta_led()
  time_after(TIMER_MS(30), final_engegat);
}

int main(void) {
  init_timer();

  apaga_led();
  final_apagat();

  for(;;);
}
```

7.3 Implementació

Pel que fa a la implementació, la idea fonamental es basa en tenir una taula de funcions $f_0, \ldots, f_k$ i quants ticks falten per a executar cadascuna d'aquestes funcions $t_0, \ldots, t_k$. Cada vegada que la interrupció de rellotge indica que ha transcorregut un tick, que en aquest cas és de 10 ms, es recorre la taula i es decrementen els t_i. Si un cert t_i esdevé zero, cal cridar la funció f_i.

Podeu definir una estructura com la següent per tal de representar la taula de funcions planificades:

```
// Maxim number of entries
#define N 20

/*
 * A record of the timer table:
 * 'remaining': ticks left to call callback
 * 'every': call callback every 'every' ticks
 * 'ntimes': number of times callback must be called.
 * 0 means ad infinitum
 */
typedef struct {
  uint8_t remaining, every, ntimes;
  timer_callback_t callback;
} entry;

/*
 * The timed tasks table.
 * If the 'every' field is 0, the entry is empty.
 */
static struct {
  entry t[N];
  uint8_t n; // The number of valid entries
} tt;
```

El procés d'actualitzar la taula cada 10 ms està conduït per les interrupcions que proporcionen el temporitzador de 16 bit del microcontrolador. Podeu aprofitar la feina feta en el mòdul `control` de la pràctica anterior de cara a configurar aquest temporitzador.

Recordeu que, quan no hi ha accions planificades el temporitzador no ha de provocar interrupcions. Així, quan executem la darrera acció planificada o bé cancel·lem totes les accions planificades del servei cal deshabilitar el temporitzador. De la mateixa forma, quan afegim la primera acció planificada cal habilitar-lo.

Tasca 9 Implementeu el mòdul `timer`. Implementeu també un programa per a comprovar el funcionament correcte de `timer`. Anomeneu-lo `test_timer`.

8 El mòdul «ether»

8.1 Especificació

Aquest mòdul té com a principal funció aïllar la resta del programari de les especificitats del dispositiu de comunicacions morse. Des del punt de vista de l'usuari es comporta de manera

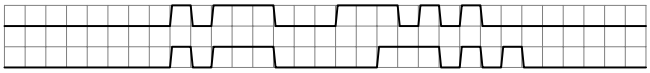

Figura 3: Diferents cronogrames que corresponen a la transmissió del missatge `AD`. Noteu que el silenci entre caràcters només té una longitud mínima.

molt similar al mòdul `serial`. En aquesta pràctica només implementarem la part de transmissió. La part de recepció serà l'objectiu de la següent pràctica.

El mòdul té la següent API:

void ether_init(**void**)	Inicialitza el mòdul. Cal cridar-la abans d'usar el mòdul i amb les interrupcions inhibides. Es pot cridar més d'una vegada i s'obté el mateix efecte.
void ether_put(uint8_t c)	Envia el caràcter `c` usant el canal morse. `c` ha de contenir un dels caràcters que admeten codificació morse: els caràcters `A`–`Z`, els dígits `0`–`9` i el caràcter espai.

Pel que fa al protocol morse, la unitat de dades del protocol correspon a un caràcter. Entre dos caràcters sempre hi ha un silenci de mida LETTERGAP com a mínim. Un caràcter es composa d'una seqüència de punts i ratlles separades per un silenci de mida GAP. Els punts corresponen a un senyal de portadora de longitud DOT i les ratlles a un senyal de portadora de longitud DASH.

Tipicament DOT val 50 ms i a partir d'aquest valor $\texttt{GAP} = \texttt{DOT}$, $\texttt{DASH} = 3 * \texttt{DOT}$ i $\texttt{LETTERGAP} = 3 * \texttt{DOT}$. La figura 3 mostra dos cronogrames diferents i igualment correctes que il·lustren la transmissió del missatge `AD`. Noteu que aquest missatge està format per dues unitats de dades.

8.2 Implementació

La implementació d'aquest mòdul es basa en una cua de transmissió. La funció ether_put() escriu a la cua de transmissió. Quan la cua té algun caràcter, s'engega un autòmat conduït pel temps que obté un caràcter de la cua, el tradueix a codi morse i va enviant els signes temporitzats adequadament. L'autòmat està actiu mentre hi ha caràcters a la cua.

Per implementar aquest autòmat s'usen els serveis del mòdul `timer`.

Tasca 10 Dissenyeu el mòdul `ether` i també el programa de prova corresponent `test_ether`.

9 La modificació del mòdul «control»

El mòdul control de la pràctica anterior cal modificar-lo lleugerament per adaptar-lo als serveis del mòdul `timer`. Essencialment cal fer dues modificacions:

1. Cal eliminar tot allò que fa referència a l'ús del temporitzador hardware del microcontrolador. Recordeu que aquest temporitzador ara l'usa el mòdul `timer`.

2. Cal connectar l'antiga rutina d'interrupcions al servei de `timer`, planificant-la per que sigui cridada indefinidament cada 100 ms tal i com estava planificat.

Tasca 11 Modifiqueu el mòdul `control` i proveu-lo usant exactament el mateix test que vàreu fer servir en l'anterior pràctica.

10 La modificació del mòdul «crossing»

El mòdul `crossing` també cal modificar-lo lleugerament. En aquest cas cal afegir la "replicació d'ordres". Recordeu que el sistema que estem implementant, que es tracta d'un mestre, rep ordres del supervisor i en replica algunes a través de canal morse. Per ser precisos:

- Només replica les ordres que s'han considerat correctes. Així, per exemple, no replicarà l'ordre `FA` si el control té els semàfors apagats.
- Només replica les ordres `R`, `H`, `FA` i `FB`.
- Cada ordre la finalitza amb un caràcter espai.
- No replica cap altre missatge (confirmacions, start, etc.)

Cal doncs modificar el modul per tal que, una vegada detectada una ordre del supervisor, decideixi si cal replicar-la pel canal morse i, en cas afirmatiu ho faci.

Tasca 12 Feu els canvis convenients al mòdul `crossing`. Comproveu que ara el vostre control semafòric ja parla morse!.

Referències

[Atm11] Atmel. *ATmega48A/PA/88A/PA/168A/PA/328/P Complete datasheet*. Anglès. Versió Rev. 8271D–AVR–05/11. 2011. URL: `http://www.atmel.com/Images/doc8271.pdf` (consultat 7 de jun. de 2012).

[KR88] Brian Kernighan i Denis Ritchie. *The C Programming Language*. Anglès. 2a edició. Addison-Wesley, Pearson Education, 1988. 274 pàgines. ISBN: 9780131103627.

[Pro12] GNU Project. *The C preprocessor*. Anglès. 2012. URL: `http://gcc.gnu.org/onlinedocs/cpp` (consultat 8 de mai. de 2012).

[Wik12] Wikipedia contributors. *Morse Code*. Anglès. Wikipedia, The Free Encyclopedia. 2012. URL: `http://en.wikipedia.org/wiki/Morse_code` (consultat 26 de feb. de 2012).