UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Trabajo: [illegible]CACIÓN SERIAL

EQUIPO

[illegible]ANTIAGO [illegible]OS

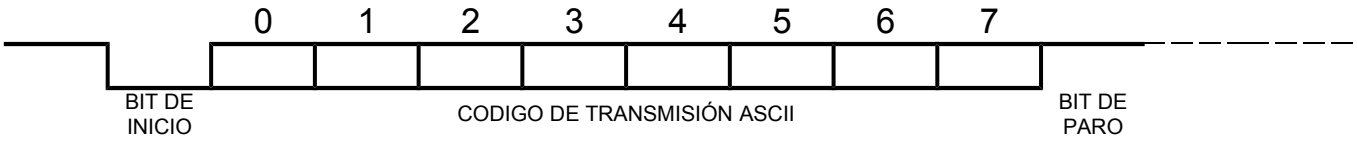
ASIGNATURA: [illegible]

GRUPO: 3

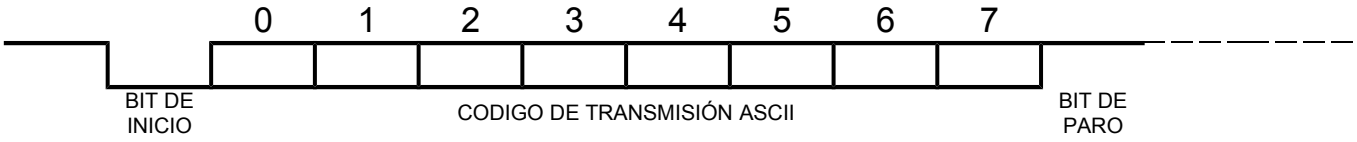
sábado, 28 de octubre de 2017, Ciudad Universitaria, México, DF

COMUNICACIÓN ASINCRONA

EL SCI (Serial Comunicación Interfase) en el µCU es un sistema de comunicación asíncrona en formato NRZ (un bit para empezar, 8 o 9 bits de datos, y un bit de paro) con circuito interno independiente de generación de tasa de baud's y un transmisor y receptor SCI. Este puede ser configurado por 8 o 9 bits de datos (uno el cual podría ser diseñado como un bit de paridad, odd or even). Si esta habilitado, la paridad es generada por hardware para transmisión y recepción de datos. Los errores de paridad recibidos son banderados por hardware. El generador de tasa de baud's esta basado sobre un modulo contador, permitiendo flexibilidad en elegir tasas de baud's diferentes. Hay una característica de receptor de weakup, una característica de línea perezosa (idle), un modo loop back, y varias características de error de detección. Dos pines del puerto proveen la interfase externa para la transmisión de datos (TXD), y el receptor de datos (RXD).



Código de transmisión ASCII



$t_b = \frac{1}{baud}$ t_b= tiempo del bit

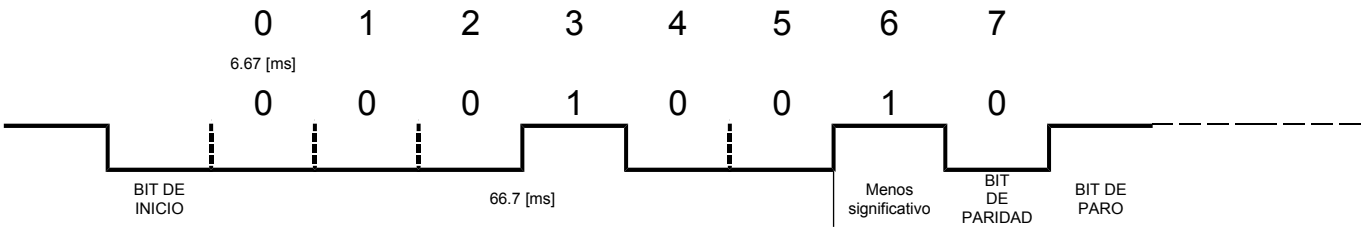
Baud = numero de bits transmitidos por segundo (baud rate)

Ejemplo:

Mensaje = 10 bits
Baud rate = 150
Numero de mensajes por segundo = 10
Tiempo de bit $t_b = \frac{1}{baud} = \frac{1}{150} = 6.67[ms]$
Tiempo de palabra = $6.67[ms](10bits) = 66.7[ms]$

Ejemplo:
Codificar el siguiente mensaje y determinar el bit necesario para una paridad par.

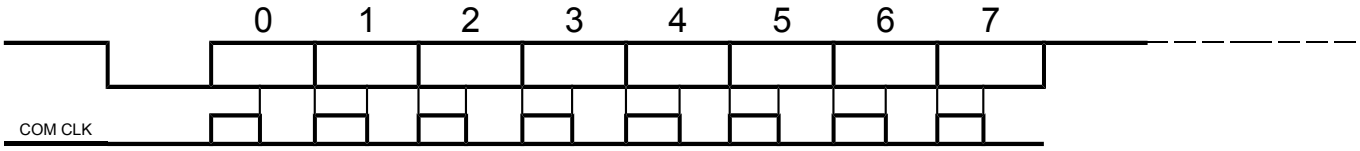
Primero se transmite el bit menos significativo



$48

Para velocidades mayores a 300 bits/s los datos son conducidos junto con una señal de reloj.

Hay un ciclo de reloj por bit de dato con la transición de uno a cero justo a la mitad del tiempo de cada bit.



El receptor puede depender de esta transición para indicar el tiempo cuando la línea de datos puede ser confiablemente muestrada.

REGISTROS:

	7	6	5	4	3	2	1	0	
PORTS	CS_/SS_	SCK	SDO/MOSI	SDI/MISO	I/0	I/0	TXD	RXD	$0248
Al reset	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
SCOBDH	BTST	BSPL	BRLD	SBR12	SB11	SBR10	SBR9	SBR8	$00C0
	Reservado	Para	Funciones de prueba						
Al reset	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
SCOBDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	$00C1
Al reset	0	0	0	0	0	0	0	0	

Dos registros para la velocidad de transmisión SBR0 → SBR12 programar diferentes velocidades de transmisión.

Determinan la velocidad mediante la siguiente formula:

$$SCI_{BAUD\_RATE} = \frac{MCLK}{16xBR}$$

MCLK=8[MHz]
BR= valor escrito en los bits

Para una velocidad de transmisión de 9600 bauds, tenemos:

$$SCI_{BAUD\_RATE} = \frac{8[MHz]}{16x9600} = 52_{10} = 34_{16}$$

REGISTROS DE CONTROL:

	7	6	5	4	3	2	1	0	
SCOCR1	LOOPS	WOMS	RSRC	M	WAKE	ILT	PE	PT	$00C2
Al reset	0	0	0	0	0	0	0	0	

PT- bit de paridad (0-par, 1-impar)
PE – habilita paridad (0 → deshabilitado, 1 → habilitado)
ILT – detecta línea vacía (0→corta, 1→larga)
WAKE – arranque (0→línea vacía, 1→linea marcada)
M – longitud de palabra (0→ 8 bits, 1→9 bits)

	7	6	5	4	3	2	1	0	
SCOCR2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	$00C3
Al reset	0	0	0	0	0	0	0	0	

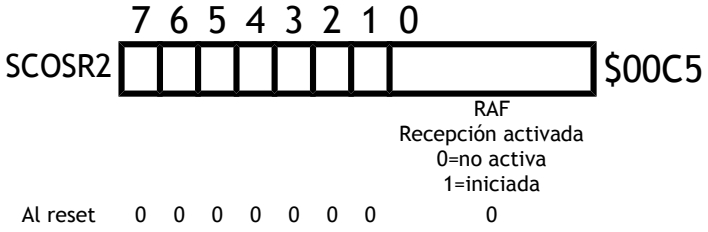
SBK – comando de ruptura (0→deshabilitado, 1→ habilitado)
RWU – recepción de modo inicio (0→arranque normal, 1→arranque con línea marcada)
RE – habilitación para recepción (0→deshabilitado, 1→habilitado)
TE – habilitación para transmisión (0→deshabilitado, 1→habilitado)
ILIE – habilitación de interrupción por línea vacía.
RIE - habilitación de interrupción por recepción
TCIE - habilitación de interrupción por transmisión completa.
TIE - habilitación de interrupción por transmisión.

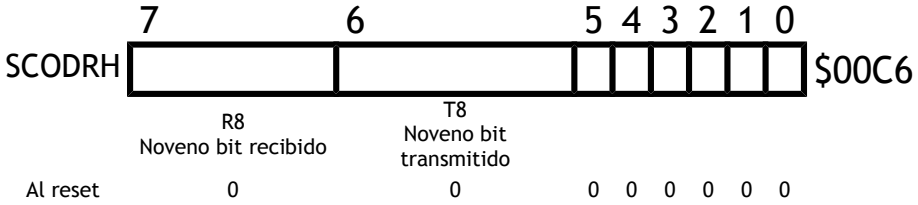
	7	6	5	4	3	2	1	0	
SCOSR1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	$00C4
Al reset	0	0	0	0	0	0	0	0	

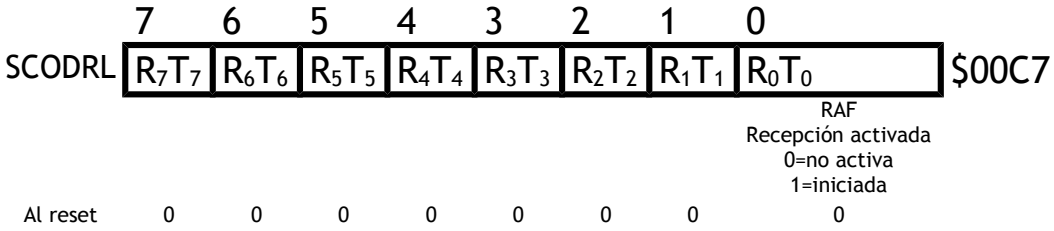
PF – error de paridad (0→correcta, 1→incorrecta)
FE – error de configuración en el bit de paro 0=OK!, 1=error.
NF – bandera de ruido en recepción 0=ok!, 1=ruido.
OR – sobre velocidad en recepción, 0=ok!, 1=sobrevelocidad
IDLE – detección de línea vacía, 0=línea activa, 1=línea vacía.
RDRF – registro de recepción de datos lleno (SCODR)
TC – transmisión completa, 0=ocupada, 1=completa.
TDRE – registro de transmisión de datos (SCODR)

PF –
FE – se usan dependiendo del tipo de interfase serial de comunicación utilizada.
NF –
OR –

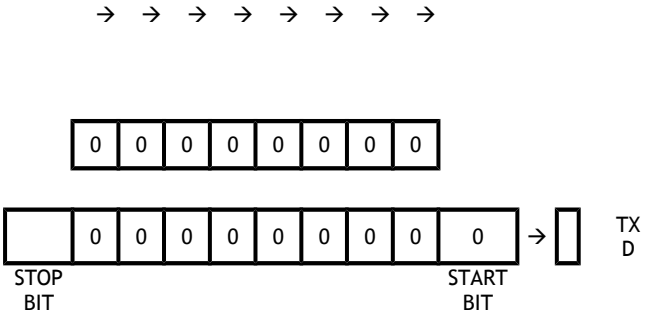
IDLE -
TC - Normalmente no se usan.







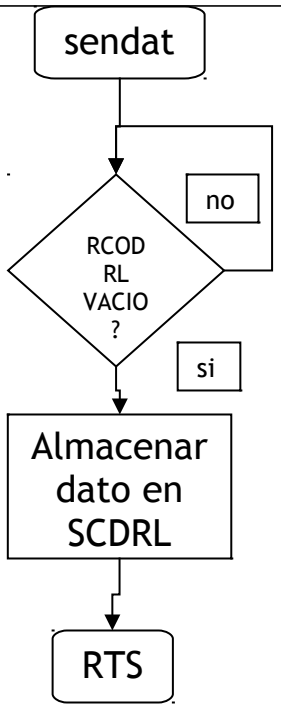
MODOS TRANSMISIÓN:



Bandera TDRE del SCOSR1 (bit7)

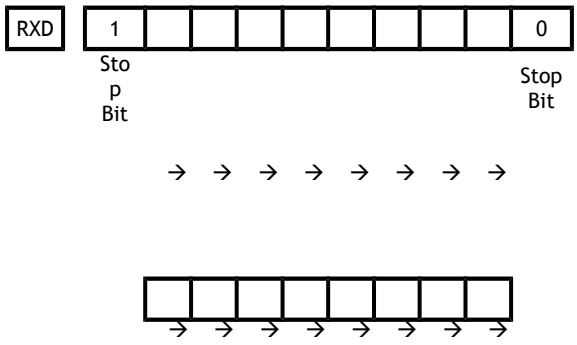
TDRE=1, cada vez que un nuevo dato se transfiere del SCODRL al registro de corrimiento TSR

SUBRUTINA

DIAGRAMA DE FLUJO	CÓDIGO ENSAMBLADOR		
	SENDAT	BRCLR STAA RTS	SC0SR1,X,$80,SENDAT SC0DRL,X

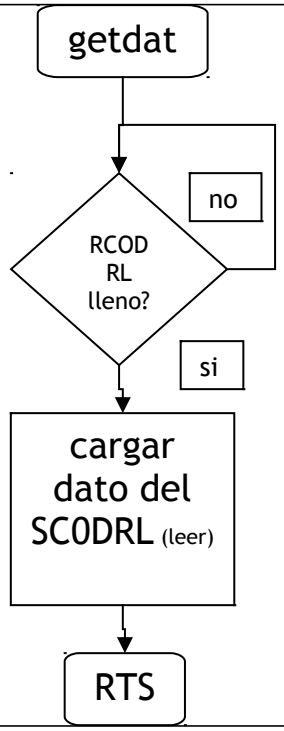
RECEPCION:

REGISTRO DE CORRIMIENTO (RSR)



Bandera: RDRF del SCOSR1 (bit 5)

RDRF=1, cada vez que un nuevo dato se transfiere del registro de corrimiento (RSR) al SCODRL.

DIAGRAMA DE FLUJO	CÓDIGO ENSAMBLADOR		
	GETDAT	BRCLR LDAA RTS	SC0SR1,X,$80,GETDAT SC0DRL,X

EJEMPLO:

Desarrollar un programa para establecer la comunicación entre una terminal tonta y un microprocesador.

El microprocesador recibe el carácter ASCII mandado por la terminal tonta, cuando termina el microprocesador debe de mandar de regreso a la terminal tonta:

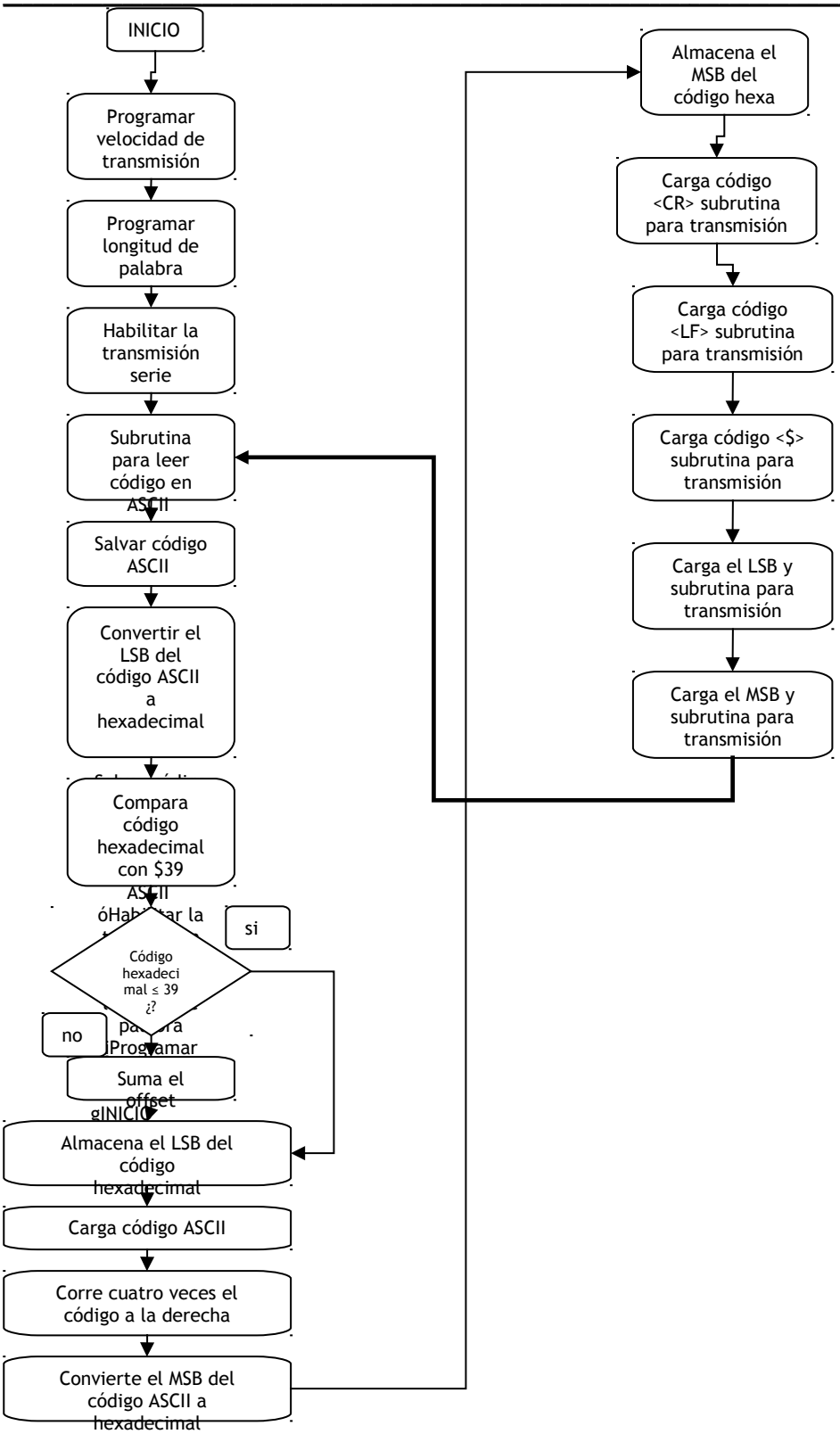
1. <CR> en ASCII
2. <LF> en ASCII
3. <$> en ASCII
4. Los dos números en hexadecimal que representa el carácter ASCII recibido

Ejemplo:

ASCII	“A”	→	$	4	1		
HEXADECIMAL				$34	$31		
ASCII	“Z”	→	$	5	A		
HEXADECIMAL				$35	$41		

En ASCII los números del cero al nueve.

0 → 30
1 → 31
2 → 32
3 → 33
4 → 34
5 → 35
6 → 36
7 → 37
8 → 38
9 → 39


INICIO
Programar velocidad de transmisión
Programar longitud de palabra
Habilitar la transmisión serie
Subrutina para leer código en ASCII
Salvar código ASCII
Convertir el LSB del código ASCII a hexadecimal
Compara código hexadecimal con $39
Código hexadecimal ≤ 39 ¿?
si
no
Suma el offset
Almacena el LSB del código hexadecimal
Carga código ASCII
Corre cuatro veces el código a la derecha
Convierte el MSB del código ASCII a hexadecimal
Almacena el MSB del código hexa
Carga código <CR> subrutina para transmisión
Carga código <LF> subrutina para transmisión
Carga código <$> subrutina para transmisión
Carga el LSB y subrutina para transmisión
Carga el MSB y subrutina para transmisión

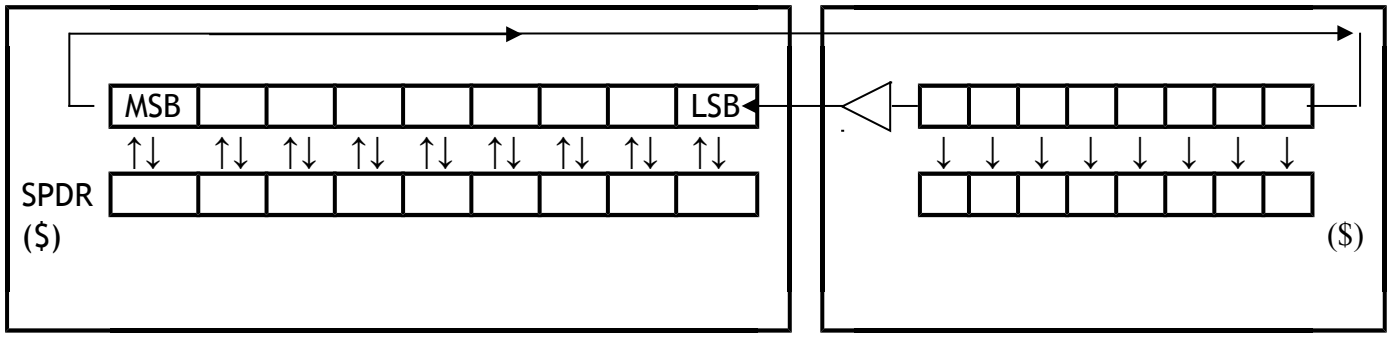
```
SCOBDL    EQU    $00C1
SCOCR1    EQU    $00C2
SCOCR2    EQU    $00C3
SCOSR1    EQU    $00C4
SCODRL    EQU    $00C7
TEMP      EQU    $4000
TEMPL     EQU    $4001
TEMPH     EQU    $4002

          LDX    #$0000

          LDAA   #$34
          STAA   SCOBDL          ;programar velocidad de transmisión

          CLRA
          STAA   SCOCR1          ;programar longitud de palabra

          LDAA   #$C0
          STAA   SCOCR2          ; habilitar trans. Y recep. Y no hab. Interrupción

PROX      JSR    GETDATA         ; subrutina para leer dato en ASCII
          STAA   TEMP            ; salva dato de código ASCII.
          ANDA   #$0F            ; convierte el LSB del dato a hexadecimal.
          OR     #$31
          CMPA   #$39            ; compara el LSB con 9
          BLS    GUARDL                BLS        C+Z=1        Regm ≤ regn
          ADDA   #$07            ; suma el offset
GUARDL    STAA   TEMPL           ; carga dato
          LSRA
          LSRA
          LSRA
          LSRA                   ;coloca el MSB a la derecha
          ANDA   #$07
          ORA    #$30            ;convierte el MSB a hexadecimal.
          STAA   TEMPH
          LDAA   #$0D            ; manda el código <LF>
          JSR    SENDATA
          LDAA   #$24
          JSR    SENDATA         ;
          LDAA   TEMPL
          JSR    SENDATA
          LDAA   TEMPH
          JSR    SENDATA
          BRA    PROX

GETDATA   BRCLR  SCOSR1,X,$20,GETDATA
          LDAA   SCODRL,X
          RTS
SENDATA   BRCLR  SCOSR1,X,$80,SENDATA
          STAA   SCODRL,X
```

RTS

COMUNICACIÓN SÍNCRONA

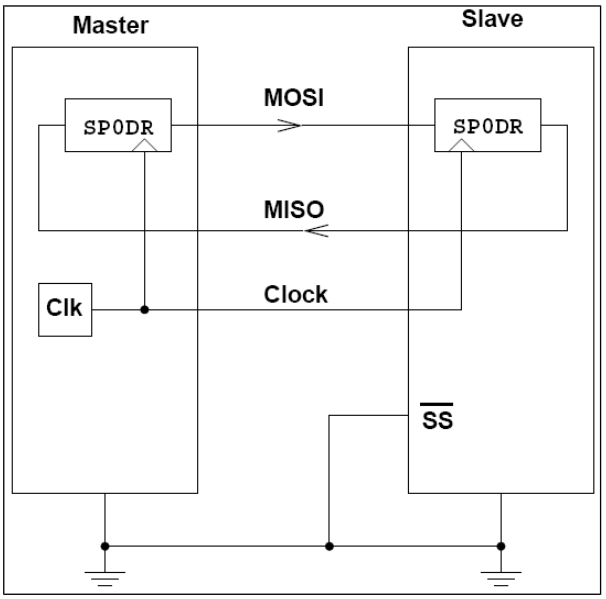
1. El hc12 tiene una **interfase serial sincrona,** en el hc12 es llamada **Interfase Periferica Serial** (**SPI**)
2. Si un hc12 genera el reloj usado para la transferencia de datos sincronos este esta operando en **Modo Maestro**.
3. Si un hc12 usa un reloj externo para la transferencia de datos sincronos este esta operando en **Modo Esclavo.**
4. Si dos Hc12 se comunican uno con otro usando sus SPI's deben ser configurados uno como maestro y otro como esclavo.
5.

Establece comunicación con dispositivos periféricos tales como registros de corrimiento, display de cristal líquido, convertidores analógicos digitales, de forma directa, lo anterior hace que el puerto pueda funcionar como maestro o como esclavo.



MAESTRO **ESCLAVO**

Otro diagrama para la comprensión es presentado a continuación:



MISO: Master In – Slave Out
MOSI: Master Out – Slave In

CONFIGURACION:

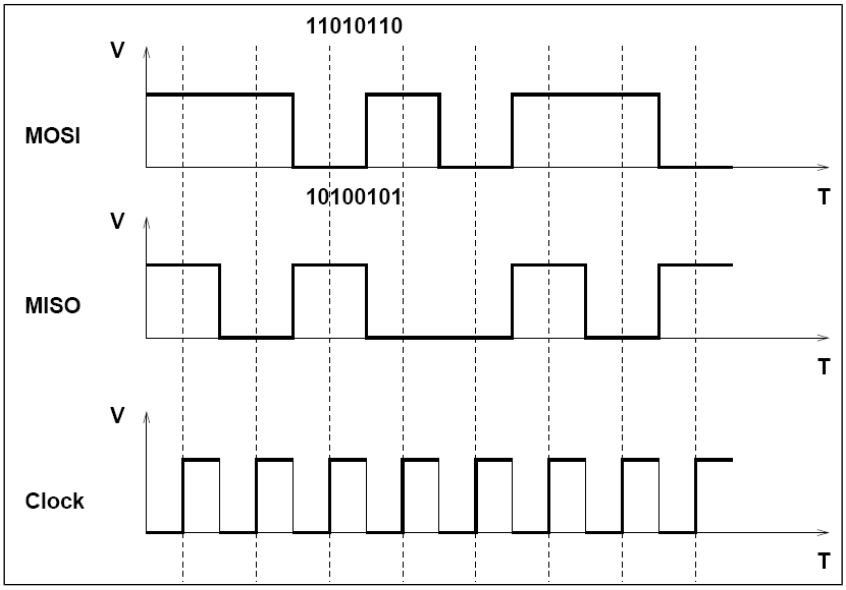
MAESTRO:

MISO → línea maestro de datos de entrada
MOSI → línea maestra de datos de salida

ESCLAVO:

MISO → línea esclava de datos de salida
MOSI → línea esclava de datos de entrada.

Cuando el **MAESTRO** inicia una transferencia 8 ciclos de reloj se generan sobre el pin **SCLK**. Tanto como en el **MAESTRO** como en el **ESCLAVO** el dato se adquiere en un flanco del ciclo de reloj y se muestrea en el ciclo opuesto. El dato cargado en el registro de corrimiento de 8 bits, sale por el pin **MOSI** hacia el **ESCLAVO**, mientras otro dato sale por la terminal **MISO** del **ESCLAVO.**



	7	6	5	4	3	2	1	0	
SPOCR1	0	1	0	1	0	0	0	0	$00D0

Al reset

*para este curso se tomo la configuración que se indica.

SP0CR1	**SPIE**	**SPE**	SWOM	**MSTR**	**CPOL**	**CPHA**	**SSOE**	**LSBF**	**0x00D0**
SP0CR2	0	0	0	0	PUPS	RDS	0	**SPC0**	**0x00D1**
SP0BR	0	0	0	0	0	**SPR2**	**SPR1**	**SPR0**	**0x00D2**
SP0SR	**SPIF**	WCOL	0	MODF	0	0	0	0	**0x00D3**
SP0DR	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**0x00D5**
DDRS	**DDS 7**	**DDS6**	**DDS5**	**DDS4**	**DDS3**	**DDS2**	**DDS1**	**DDS0**	**0x00D7**

Explicación:

$00D0	SPI Control Register 1 (SP0CR1) See page 308.	Read:	SPIE	SPE	SWOM	MSTR	CPOL	CPHA	SSOE	LSBF
		Write:								
		Reset:	0	0	0	0	0	0	0	0

Read: Anytime
Write: Anytime

SPIE — SPI Interrupt Enable Bit
0 = SPI interrupts are inhibited.
1 = Hardware interrupt sequence is requested each time the SP
or MODF status flag is set.

SPE — SPI System Enable Bit
0 = SPI internal hardware is initialized and SPI system is in a low-power disabled state.
1 = PS4–PS7 are dedicated to the SPI function.

When MODF is set, SPE always reads 0. SP0CR1 must be written part of a mode fault recovery sequence.

SWOM — Port S Wired-OR Mode Bit

Controls not only SPI output pins but also the general-purpose outp pins (PS4–PS7) which are not used by SPI.
0 = SPI and/or PS4–PS7 output buffers operate normally.
1 = SPI and/or PS4–PS7 output buffers behave as open-drain outputs.

MSTR — SPI Master/Slave Mode Select Bit
0 = Slave mode
1 = Master mode

$00D1	SPI Control Register 2 (SP0CR2) See page 310.	Read:	0	0	0	0	PUPS	RDS	0	SPC0
		Write:								
		Reset:	0	0	0	0	1	0	0	0

PUPS — Pullup Port S Enable Bit

0 = No internal pullups on port S

1 = All port S input pins have an active pullup device. If a pin is programmed as output, the pullup device becomes inactive.

RDS — Reduce Drive of Port S Bit

0 = Port S output drivers operate normally.

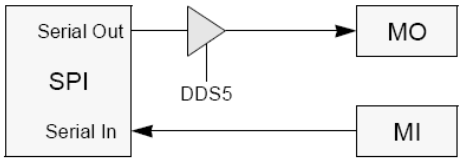
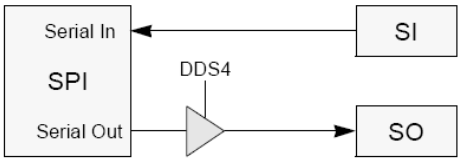
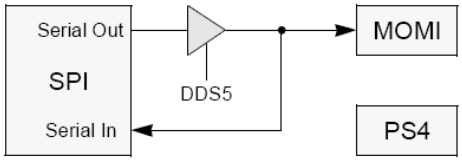
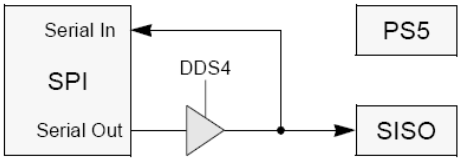
1 = All port S output pins have reduced drive capability for lower power and less noise.

SPC0 — Serial Pin Control 0 Bit

This bit decides serial pin configurations with MSTR control bit.

Pin Mode		SPC0[1]	MSTR	MISO[2]	MOSI[3]	SCK[4]	$\overline{SS}$[5]
#1	Normal	0	0	Slave out	Slave in	SCK in	$\overline{SS}$ in
#2			1	Master in	Master out	SCK out	$\overline{SS}$ I/O
#3	Bidirectional	1	0	Slave I/O	General-purposeI/O	SCK in	$\overline{SS}$ in
#4			1	General-purposeI/O	Master I/O	SCK out	$\overline{SS}$ I/O

1. The serial pin control 0 bit enables bidirectional configurations.
2. Slave output is enabled if DDS4 = 1, SS = 0, and MSTR = 0. (#1, #3)
3. Master output is enabled if DDS5 = 1 and MSTR = 1. (#2, #4)
4. SCK output is enabled if DDS6 = 1 and MSTR = 1. (#2, #4)
5. $\overline{SS}$ output is enabled if DDS7 = 1, SSOE = 1, and MSTR = 1. (#2, #4)

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0	 SWOM enables open-drain output.	 SWOM enables open-drain output.
Bidirectional Mode SPC0 = 1	 SWOM enables open-drain output. PS4 becomes GPIO.	 SWOM enables open-drain output. PS5 becomes GPIO.

$00D2 SPI Baud Rate Register (SP0BR) See page 311.

	7	6	5	4	3	2	1	0
Read:	0	0	0	0	0	SPR2	SPR1	SPR0
Write:								
Reset:	0	0	0	0	0	0	0	0

Read: Anytime
Write: Anytime

At reset, E clock divided by 2 is selected.

SPR2–SPR0 — SPI Clock (SCK) Rate Select Bits

These bits are used to specify the SPI clock rate.

Table 14-4. SPI Clock Rate Selection

SPR2	SPR1	SPR0	E Clock Divisor	Frequency at E Clock = 4 MHz	Frequency at E Clock = 8 MHz
0	0	0	2	2.0 MHz	4.0 MHz
0	0	1	4	1.0 MHz	2.0 MHz
0	1	0	8	500 kHz	1.0 MHz
0	1	1	16	250 kHz	500 kHz
1	0	0	32	125 kHz	250 kHz
1	0	1	64	62.5 kHz	125 kHz
1	1	0	128	31.3 kHz	62.5 kHz
1	1	1	256	15.6 kHz	31.3 kHz

$00DB	Port S Pullup/Reduced Drive Register (PURDS) See page 316.	Read:	0	RDPS2	RDPS1	RDPS0	0	PUPS2	PUPS1	PUPS0
		Write:								
		Reset:	0	0	0	0	0	0	0	0

Read: Anytime
Write: Anytime

RDPS2 — Reduce Drive of PS7–PS4
0 = Port S output drivers for bits 7–4 operate normally.
1 = Port S output pins for bits 7–4 have reduced drive capability for lower power and less noise.

RDPS1 — Reduce Drive of PS3 and PS2
0 = Port S output drivers for bits 3 and 2 operate normally.
1 = Port S output pins for bits 3 and 2 have reduced drive capability for lower power and less noise.

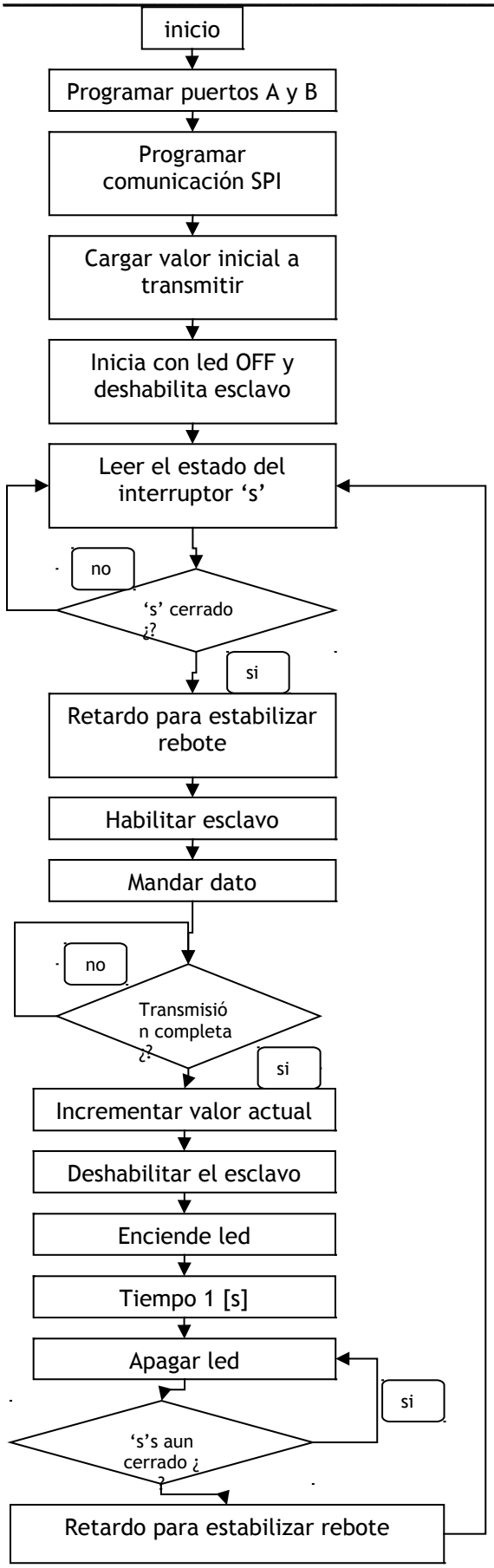
RDPS0 — Reduce Drive of PS1 and PS0
0 = Port S output drivers for bits 1 and 0 operate normally.
1 = Port S output pins for bits 1 and 0 have reduced drive capability for lower power and less noise.

PUPS2 — Pullup Port S Enable PS7–PS4
0 = No internal pullups on port S bits 7–4.
1 = Port S input pins for bits 7–4 have an active pullup device. If a pin is programmed as output, the pullup device becomes inactive.

COMUNICACIÓN SERIE

Desarrollar un programa que mande un dato de µC por su terminal serial síncrona cada vez que se cierra el interruptor ‘s’. Al termino de cada dato mandado encender el led ‘L’ durante un segundo y esperar el próximo cierre del interruptor.

(poner diagrama)

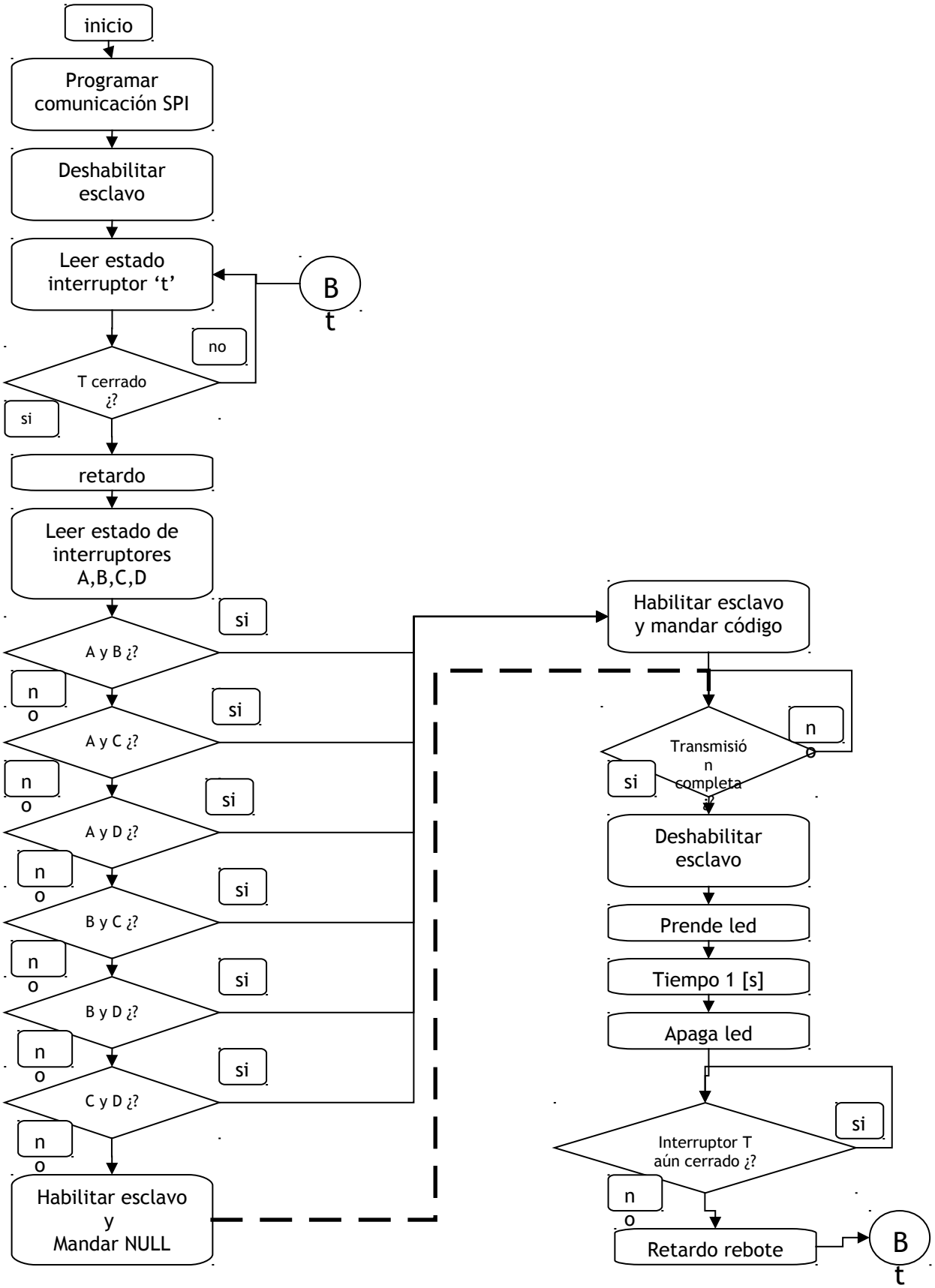

inicio
Programar puertos A y B
Programar comunicación SPI
Cargar valor inicial a transmitir
Inicia con led OFF y deshabilita esclavo
Leer el estado del interruptor ‘s’
no
‘s’ cerrado ¿?
si
Retardo para estabilizar rebote
Habilitar esclavo
Mandar dato
no
Transmisió n completa ¿?
si
Incrementar valor actual
Deshabilitar el esclavo
Enciende led
Tiempo 1 [s]
Apagar led
si
‘s’s aun cerrado ¿ ?
Retardo para estabilizar rebote

```
PORTA   EQU   $0000
DDRA    EQU   $0002
PORTB   EQU   $0001
DDRB    EQU   $0003
SPOCR1  EQU   $00D0
SPOCR2  EQU   $00D1
SPOBR   EQU   $00D2
SPOSR   EQU   $00D3
SPODR   EQU   $00D5
VALSPI  EQU   $4000

        LDX   #$0000
        BCLR  DDRA,X,$80
        BSET  DDRB,X,$48
        BSET  SPOCR1,X,$50
        BCLR  SPOCR2,X,$FF      ;programar SPI

        LDAA  #$02
        STAA  SPOBR,X,$02       ;velocidad de transmisión

        BSET  PORTB,X,$48       ;inicilizar con led off y deshabilita esclavo
ESP     BRCLR PORTA,X,$80,ESP   ; leer
        JSR   REBOTE50          ;retardo para estabilizar rebote
        BCLR  PORTB,X,$08       ;habilitar esclavo
        LDAA  VALSPI            ;manda dato
        STAA  SPODR,X           ;manda dato
AQUI    BCLR  SPOSR,X,$80,AQUI  ;lee si la transmisión se ha completado
        INC   VALSPI            ;incrementa dato, el cual se enviará después del
                                cierre de 's'.

        BSET  PORTB,X,$08       ;deshabilitar esclavo
        BCLR  PORTB,X,$40       ;encender led
        LDAA  #$14              ;14h=20d, 20*50[ms]=1[s]
DLYLP   JSR   REBOTE50          ;retraso de 1 [s]
        DECA                    ;retraso de 1 [s]
        BNE   DLYLP             ;retraso de 1 [s]

        BSET  PORTB,X,$40       ;apagar led
AUN     BRSET PORTA,X,$80,AUN   ;lee si aun esta cerrado 's'
        JSR   REBOTE50          ;retardo para estabilizar rebote
        BRA   ESP
```

Serial síncrona

Se tienen 5 interruptores, uno de contacto momentáneo ‘t’, y 4 de contactos sostenido A, B, C, y D, alambrados como se indica en la figura. Cada vez que el interruptor ‘t’ se cierra, el microcontrolador debe analizar el estado de los interruptores A,B,C y D. y mandar hacia el dispositivo periférico vía el puerto de comunicación serie síncrona los códigos ASCII de las letras que corresponden al estado cuando 2 de los 4 interruptores están cerrados, prender el led ‘L’ durante 1 [s] y esperar otro cierre del interruptor ‘t’. Cuando el estado de los interruptores no corresponda al descrito anteriormente, el mC debe mandar el carácter NULL, prender el led durante 1 [s] y esperar el próximo cierre del interruptor ‘t’.

(pegar figura)


inicio
Programar comunicación SPI
Deshabilitar esclavo
Leer estado interruptor 't'
Bt
no
T cerrado ¿?
si
retardo
Leer estado de interruptores A,B,C,D
si
A y B ¿?
no
si
A y C ¿?
no
si
A y D ¿?
no
si
B y C ¿?
no
si
B y D ¿?
no
si
C y D ¿?
no
Habilitar esclavo y Mandar NULL
Habilitar esclavo y mandar código
Transmisión completa ¿?
no
si
Deshabilitar esclavo
Prende led
Tiempo 1 [s]
Apaga led
Interruptor T aún cerrado ¿?
si
no
Retardo rebote
Bt

```
PORTA           EQU     $0000
```

```
DDRA    EQU   $0002
PORTB   EQU   $0001
DDRB    EQU   $0003
SPOCR1  EQU   $00D0
SPOCR2  EQU   $00D1
SPOBR   EQU   $00D2
SPOSR   EQU   $00D3
SPODR   EQU   $00D5

        LDX   #$0000
        LDX   #$0000
        BCLR  DDRA,X,$80
        BSET  DDRB,X,$48
        BSET  SPOCR1,X,$50
        BCLR  SPOCR2,X,$FF      ;programar SPI
        LDAA  #$02              ;velocidad de transmisión
        STAA  SPOBR,X           ;velocidad de transmisión

        BSET  PORTB,X,$48       ;deshabilitar esclavo
ESPERA  BRCLR PORTA,X,$80,ESPERA ;lee estado del interruptor 'T'
        JSR   REBOTE50          ;retardo para estabilizar rebote

        LDAA  PORTA,X           ;leer estado de interruptores.

        ANDA  #$0F              ;toma los 4 primeros bits para su lectura.
        CMPA  #$OC              ;interrupores A y B
        BEQ   AB                ; salta a subrutina AB

        CMPA  #$OA
        BEQ   AC

        CMPA  #$O9
        BEQ   AD

        CMPA  #$O6
        BEQ   BC

        CMPA  #$O5
        BEQ   BD

        CMPA  #$O3
        BEQ   CD

        BCLR  PORTB,X,$08       ;habilitar esclavo
        CLRA                    ;cargar A con ceros
        STAA  SPODR,X           ;mandar NULL

TRCO    BRCLR SPODR,X,$80,TRCO  ;verifica si la transmisión esta completa
DESH    BSET  PORTB,X,$40       ;deshabilitar esclavo
        BCLR  PORTB,X,$40       ;prender led
        LDAA  #$14              ;14h=20d, 20*50[ms]=1[s]
DLYLP   JSR   REBOTE50          ;retraso de 1 [s]
```

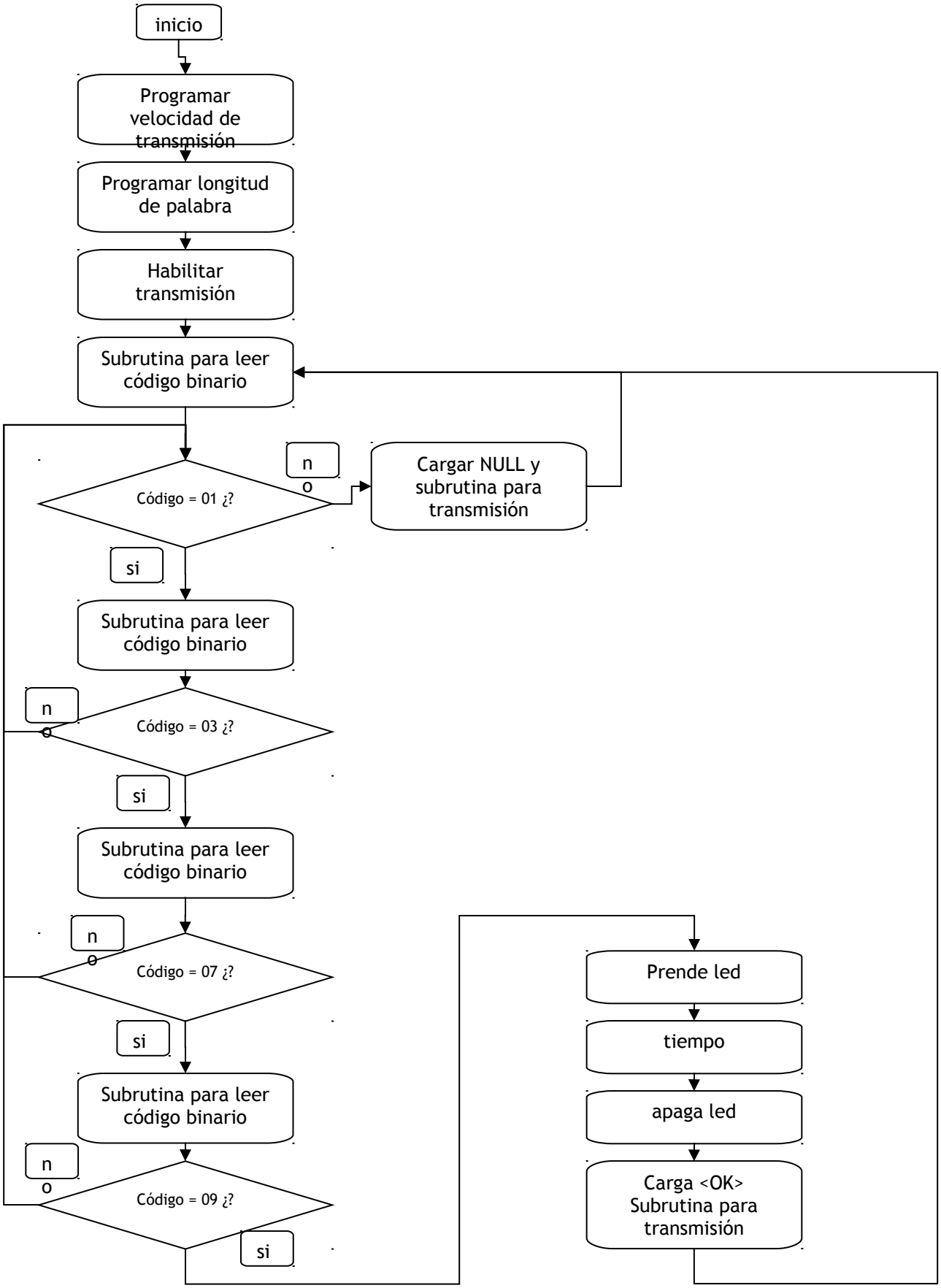
```
            DECA                        ;retraso de 1 [s]
            BNE     DLYLP               ;retraso de 1 [s]

            BSET    PORTB,X,$40         ; apaga led
AUN         BRCLR   PORTB,X,$80,AUN     ;leer si 'T' aún esta cerrado
            JSR     REBOTE50            ; tiempo para estabilizar el rebote
            BRA     ESPERA              ;regresa a leer el estado del interruptor 'T'

AB          LDAA    #$41
            LDAB    #$42
            BRA     TRANS

AC          LDAA    #$41
            LDAB    #$43
            BRA     TRANS

AD          LDAA    #$41
            LDAB    #$44
            BRA     TRANS

BC          LDAA    #$42
            LDAB    #$43
            BRA     TRANS

BD          LDAA    #$
            LDAB    #$
            BRA     TRANS

CD          LDAA    #$43
            LDAB    #$44

TRANS       BCLR    PORTB,X,$08         ;habilitar esclavo
            STAA    SPODR,X             ;transmite los datos del acumulador A
TRC1        BRCLR   SPOSR,X,$80,TRC1    ;verifica transmisión fue completa enviada de A
            STAB    SPODR,X             ;transmite los datos del acumulador B
TRC2        BRCLR   SPOSR,X,$80,TRC2    ;verifica si transmisión fue completa enviada de
                                        B
            BRA     DESH                ;salta a deshabilitar el esclavo

REBOTE50
```

EJEMPLO COMUNICACIÓN SERIAL ASINCRONA

Una terminal transmite en forma serial asíncrona grupos de 8 bits en código binario puro, desarrolle un programa que encienda un led por un tiempo de 5 [s], cada vez que el microcontrolador recibe la secuencia 01,03,07,09, y mande el mensaje hacia la terminal, si la secuencia no es la correcta, el mC debe mandar el mensaje NULL en código ASCII.

Supóngase que se recibe la siguiente secuencia 02, 03, **01, 03, 07, 09**, 08, 07, 01, **01, 03, 07, 09**, 01, 02, 05, **01, 03, 07, 09,**

El mC solo enviara el mensaje cuando detecte la secuencia.


inicio
Programar velocidad de transmisión
Programar longitud de palabra
Habilitar transmisión
Subrutina para leer código binario
Código = 01 ¿?
no
Cargar NULL y subrutina para transmisión
si
Subrutina para leer código binario
Código = 03 ¿?
no
si
Subrutina para leer código binario
Código = 07 ¿?
no
si
Subrutina para leer código binario
Código = 09 ¿?
no
si
Prende led
tiempo
apaga led
Carga <OK> Subrutina para transmisión

```
SCOBDL    EQU    $00C1
SCOCR1    EQU    $00C2
SCOCR2    EQU    $00C3
SCOSR1    EQU    $00C4
SCODRL    EQU    $00C7

          LDX    #$0000
          LDAA   #$34          ;velocidad de transmisión
          STAA   SCOBDL        ;velocidad de transmisión
          CLRA                 ;programar longitud de palabra
          STAA   SCOCR1        ;programar longitud de palabra
          LDAA   #$0C          ;habilitar transmisión
          STAA   SCOCR2        ;habilitar transmisión

OTRO      JSR    GETDATA       ; subrutina para leer código binario

VUELVE    CMPA   #$01
          BNE    NULL
          JSR    GETDATA       ; subrutina para leer código binario
          CMPA   #$03
          BNE    VUELVE        ;si no es 03, vuelve a verificar si es 01
          JSR    GETDATA       ; subrutina para leer código binario
          CMPA   #$07
          BNE    VUELVE        ;si no es 07, vuelve a verificar si es 01
          JSR    GETDATA       ; subrutina para leer código binario
          CMPA   #$09
          BNE    VUELVE        ;si no es 09, vuelve a verificar si es 01

          BSET   PORTA,X,$80   ;enciende LED
          JSR    TIEMPO        ;tiempo de retardo
          BCLR   PORTA,X,$80   ;apaga LED

          LDAA   #$4F          ;carga o
          JSR    SENDATA       ; envia o
          LDAA   #$48          ;carga K
          JSR    SENDATA       ;envia K
          BRA    OTRO          ; regresa a verificar otra secuencia

NULL      LDAA   #$00          ;00 es null
          JSR    SENDATA       ; envia null
          BRA    OTRO          ; regresa a verificar otra secuencia

GETDATA   BRCLR  SCOSR1,X,$20,GETDATA  ;subrutina de obtención de dato
          LDAA   SCODRL,X
          RTS

SENDATA   BRCLR  SCOSR1,X,$80,SENDATA  ;subrutina de envío de dato
          STAA   SCODRL,X

TIEMPO                         ; subrutina de tiempo de retardo
```