UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Enginyeria de Sistemes TIC

Interrupcions

Dispositius Programables — Enginyeria de Sistemes TIC

Jordi Bonet Francisco del Águila

4 de desembre de 2022

Índex

1 Objectiu

L'objectiu d'aquesta pràctica conèixer com s'han de fer servir les interrupcions amb l'AVR.

2 Introducció teòrica

2.1 Interrupcions a l'AVR

La descripció detallada del funcionament de les interrupcions es troba a l'apartat 7.7 i l'apartat 12 de [**?**]. Tal i com s'ha explicat a classe, el concepte d'interrupció es pot entendre com una crida a una subrutina amb la diferència de que qui crida no és l'instrucció **call** o equivalents a dins del programa principal, sinó que és el propi maquinari perifèric de la CPU que provoca el salt a la rutina d'interrupció.

Es poden habilitar o no habilitar les interrupcions, per fer-ho de forma global (per tant afecta a totes) existeixen les instruccions **sei** (habilitar) i **cli** (deshabilitar). També es pot controlar independent cada mòdul. En aquest cas s'ha de mirar els registres de control de cada mòdul que trobem definits al mapa d'entrada / sortida.

La manera de definir rutines d'atenció a la interrupció en assemblador és molt simple. S'ha de fer servir el símbol especial **___vector_XX** on **XX** indica la posició del vector d'interrupció definit a l'apartat 12.4 pàg. 67 de [**?**]. Únicament s'ha de tenir la precaució de que la posició **0** correspon al **RESET**, per tant, per definir per exemple la interrupció de recepció de USART s'ha de definir **___vector_18**. Aquests símbols s'han de definir com símbols globals, així el lincador els interpretarà correctament, al igual que el símbol especial **main**, que és on comença el nostre programa.

2.2 Comunicació serie asíncrona

Una de les formes que tenen els computadors per comunicar-se els uns amb els altres és amb un perifèric anomenat **USART** [**?**]. Aquest perifèric permet una comunicació de tipus serie, per tant fa servir dues línies de comunicació, una per a la recepció i l'altra per a la transmissió. Aquest tipus de comunicació generalment es fa en mode asíncron, per tant no hi ha cap tipus de rellotge que reguli la mateixa comunicació. En una comunicació serie, la informació en forma de bits viatja un darrere de l'altra, per aquest motiu la quantitat de cables és mínima.

En el cas de l'Arduino la interfície de comunicació USART que es fa servir per comunicar amb l'ordinador personal està formada per la mateixa interfície USB. El port USB queda configurat emulant el comportament d'una USART. Per tant, el cable USB que fem servir per alimentar i per programar l'Arduino també conté la comunicació USART amb l'ordinador.

2.3 Programari de comunicació

L'ordinador que es connecta amb l'Arduino necessita d'un terminal que permeti la interacció entre l'usuari i el dispositiu amb el que es comunica. La funció d'aquest terminal és permetre que tot el que l'usuari tecleja sigui transmès al dispositiu connectat i també permet que allò que el dispositiu envia cap a l'ordinador pugui ser visualitzat per la pantalla. Una eina d'aquest tipus podria ser **picocom**.

Per instal · lar aquesta aplicació en una distribució tipus Debian s'ha d'executar la comanda

```
sudo aptitude install picocom
```

Per saber com funciona aquesta aplicació es pot fer ús del seu manual: *man picocom*. La comanda mínima per executar-lo és

```
picocom /dev/dispositiu_serie
```

En aquest cas agafa els valors de configuració per defecte (9600 bps, 8 bits de dades, sense paritat, 1 bit de stop, ...)

2.4 Mòdul USART

La majoria d'AVR, i en particular el ATmega328p que fem servir a la plataforma Arduino conté un mòdul USART configurable de diferents maneres. La descripció detallada d'aquest mòdul es troba a l'apartat 20, pàg 178 del manual de referència del ATmega328p [**?**].

La configuració que es fa servir per a la comunicació amb l'ordinador és en mode asíncron, amb un rellotge intern i sense habitar el mode de multiprocessador.

La velocitat a la que poden anar els bits, al igual que el format del missatge són paràmetres configurables tant per part de l'Arduino com per part de l'ordinador. La comunicació serà possible si els dos dispositius tenen la mateixa configuració. En general aquesta comunicació per defecte queda definida per 8 bits de dades, 1 bit de parada i no fer servir la paritat. Aquesta serà la configuració que es farà servir tant a l'Arduino com en l'ordinador.

2.5 Codificació ASCII

Un estàndard molt simple per poder codificar en binari els caràcters d'un text és la codificació ASCII [**?**]. Aquesta codificació en el seu format més simple consisteix en una taula de 7 bits. Amb 7 bits només es poden codificar 128 possibles valors per tant el conjunt de possibles caràcters és bastant reduït. Ja que la majoria de sistemes digitals treballen amb unitats de 8 bits (1Byte), la taula ASCII també està definida amb 8 bits. Això augmenta el nombre de caràcters al doble (256 valors), quedant definida la taula ASCII estesa.

La codificació binària que es fa servir per transmetre un caràcter a través d'una comunicació basada en un dispositiu USART és la codificació ASCII. Per exemple, la transmissió de la lletra **A** queda codificada amb els bits **0x41**.

Quan s'escriu un programa en assemblador i es vol fer referència a qualsevol codi de la taula ASCII no cal escriure el seu valor numèric, sinó que es pot fer referència a aquest caràcter escrivint-lo entre comes simples.

LDS r16 , 'A'

3 Exemple de programa

El següent programa, *exemple.S*, implementa una comunicació serie de manera que l'Arduino, un cop s'ha inicialitzat tot correctament, es manté sense fer res en un bucle infinit. Quan es produeix la condició de que ha rebut un byte es genera la interrupció i executa la rutina d'atenció a la interrupció, rep aquest byte i simplement es limita a tornar-lo a transmetre. Aquest tipus de comportament és el que s'anomena fer un eco.

```
.set DDRB_o , 0x4
.equ PORTB_o , 0x5
PORTD_o = 0x0b
DDRD_o = 0x0a

UDR0 = 0xc6
UBRR0H = 0xc5
UBRR0L = 0xc4
UCSR0C = 0xc2
UCSR0B = 0xC1
UCSR0A = 0xC0

.global main
.global __vector_18

/* rutina de transmissió de byte, el valor a transmetre està al registre r16 */
tx: lds  r17,UCSR0A
    sbrs r17,5
    rjmp tx
    sts UDR0,r16
    ret

/* defineixo la rutina d'interrupció
per recepció de byte a la USART */
__vector_18:
    lds r16,UDR0
    call tx
    reti

main:
    /* set baud rate a 9600*/
    ldi r16, 0
    sts UBRR0H,r16
    ldi r16, 103
    sts UBRR0L,r16

    /* set frame format */
    /* el valor de reset dels registres ja és correcte:
```

```
asíncron, sense paritat, 1 stop, 8 dades,
velocitat normal, comunicació no multiprocessor */
    /*assegurem el que volem encara que en reset ja ho sigui*/
    ldi r16, 0b00100000
    sts UCSR0A, r16

    ldi r16, 0b00000110
    sts UCSR0C, r16

    /* enable rx, tx, amb interrupció de rx */
    ldi  r16, 0b10011000
    sts UCSR0B,r16

    /* configuració dels pins, és possible que no calgui */
    ldi r16,0b00000010
    out DDRD_o,r16

    /*habilitem interrupcions */
    sei

/*el bucle principal no fa res*/
loop: rjmp loop
    ret
```

4 Estudi previ

1. Llegiu detalladament l'apartat 7.7 i 12 de [**?**]

2. Assembla *exemple.S* i a partir del fitxer *exemple.elf* des-assembla el fitxer generant *exemple.disam*. Descriu detalladament tot el codi que apareix justificant exactament que fa cada grup d'instruccions màquina.

3. Fes un programa fent servir la interrupció de recepció de la USART de manera que quan detecti exactament la seqüència de lletres *"led"* encengui el led. Amb qualsevol altre seqüència el led s'ha de mantenir apagat. La resposta de l'Arduino cap a l'ordinador a cada pulsació hauria de ser el nombre de lletres que queden per teclejar fins aconseguir la seqüència *"led"*. Implementeu aquest programa com una màquina d'estats. Dibuixeu el graf corresponent, definiu els diferents estats, declareu qui mantindrà el valor de l'estat del sistema. Anomena'l *prac6_1.S*.

4. Quina avantatge suposa fer servir la interrupció de transmissió? Teniu en compte que el propi microcontrolador sap en quin moment vol transmetre.

5. Fes un programa fent servir interrupcions de recepció i transmissió que quan detecti que s'ha polsat la lletra n o N respongui amb els caràcters corresponents als nombres 0,1,2,3,4,5,6,7,8,9. Quan es rebi qualsevol altre valor, respongui amb el caràcter N. Anomena'l *prac6_2.S*.

5 Treball pràctic

El treball al laboratori consisteix en la comprovació pràctica de les tasques de l'estudi previ. A continuació teniu algunes comandes del *Toolchain* de GNU, imprescindibles per passar els codis al microcontrolador.

```
avr-gcc -mmcu=atmega328p -o a.elf a.s

avr-objcopy --output-target=ihex a.elf a.hex

avrdude -c arduino -P /dev/ttyACM0 -p m328p -U flash:w:a.hex:i
```

5.1 Tasques

1. Assembla *exemple.S* i comprova el seu funcionament.
2. Comprova el correcte funcionament de *prac6_1.S*.
3. Comprova el correcte funcionament de *prac6_2.S*.

Referències

[USART] Dispositiu USART; `http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter`

[ATmega328p] Atmel. ATmega328P datasheet; `https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf`

[ASCII] Taula de caràcters ASCII; `http://en.wikipedia.org/wiki/ASCII`