Informe Técnico Nº 2

Puertos de la Computadora

Sofía Escudero, Nael Pighin, Martín Reus
Facultad de Ingeniería y Ciencias Hídricas

3 de julio de 2023

Resumen

En el presente informe se detalla el funcionamiento de los puertos de una computadora y, en particular, del puerto paralelo IEEE 1284 en sus distintos modos de operación y se propone un programa en Assembler que permite leer la información de posición generada por un módulo receptor GPS u-blox MAX-M10M utilizando el protocolo NMEA 0183 y un puerto asincrónico UART.

1. Sección de teoría

1.1. Puertos de una computadora

Un **puerto** es una interfaz a través de la cual se puede enviar y recibir los diferentes tipos de datos entre computadoras, o entre una computadora y sus periféricos.

Los puertos pueden ser usados para conectar diversos tipos de dispositivos, como puede ser un monitor, una cámara web, un parlante, o algún otro periférico, como un receptor de GPS. Debido a la gran diversidad de periféricos que pueden ser conectados, es necesario contar con un dispositivo específico que facilite la comunicación con los mismos. Es aquí donde distintos dispositivos como el receptor-transmisor asíncrono universal (UART) toman protagonismo, para el caso de transmisiones asíncronas, o el receptor-transmisor asíncrono y síncrono universal (USART), que permite ambos tipos de transmisiones.

1.1.1. Clasificaciones

Un **puerto lógico** es una zona o localización de la memoria de la computadora que se asocia con un puerto físico o un canal de comunicación, y que proporciona un espacio para el almacenamiento temporal de la información que se va a transferir entre la localización de memoria y el canal de comunicación. Existen distintos tipos de puertos lógicos según qué periféricos permiten conectar: PCI, puertos USB, puertos de memoria, etc.

Un **puerto físico** es una salida especializada al que se conecta un conector y proporcionan un método para transferir señales entre dispositivos. Hay distintos tipos de puertos físicos según la transferencia de señal: Los *puertos serie* envían y reciben un bit a la vez a través de un par de cables, mientras que los *puertos paralelos* envían y reciben varios bits a través de conjuntos de cables, es decir, se implementa un cable o una vía física para cada bit de datos formando un bus.

1.2. Puerto IEEE 1284

El puerto IEEE 1284 es la norma para el puerto paralelo de los PC que provee una alta velocidad de comunicación bidireccional entre el ordenador y el periférico externo, llegando a ser de 50 a 100 veces mas rápido que el puerto paralelo original con un rendimiento teórico máximo de 4 MB/s. En la practica, el ancho de banda es de alrededor de 2 MB/s dependiendo del hardware utilizado.

Este puerto tiene cinco modos de operación según la forma en la que se transfieren los datos:

1.2.1. Standard Parallel Port

El **Standard Parallel Port** o **Modo de Compatibilidad** es una interfaz unidireccional con unas pocas diferencias con el puerto paralelo original. Este modo es usado exclusivamente por impresoras. Las únicas señales que la impresora puede enviar de vuelta al host son líneas de estado de significado fijo, que simbolizan condiciones comunes de error, como la falta de papel en la impresora.

1.2.2. Nibble mode

El **Nibble mode** o **Modo Nibble** es una interfaz bidireccional que permite transferir nibbles (cuatro bits de dato), utilizando las cuatro líneas de estado del modo compatibilidad para transferir datos. Fue introducido por HP, usándose generalmente como una mejora para el estado de las impresoras.

1.2.3. Byte mode

El **Byte mode** o **Modo byte** es una interfaz bidireccional half-duplex que permite transferir bytes (ocho bits de dato) utilizando las mismas lineas de datos para la otra dirección.

1.2.4. Enhanced Parallel Port

El **Enhanced Parallel Port** o **Modo mejorado** es una interfaz bidireccional half-duplex que permite que los dispositivos como las impresoras, escáneres o dispositivos de almacenamiento puedan hacer transferencias grandes cantidades de datos a alta velocidad a la vez que cambian canales de dirección. Proporcionan un ancho de banda de hasta 2 MB/s, aproximadamente 15 veces la velocidad que alcanza un puerto paralelo normal de comunicación, con mucho menos consumo de la CPU.

1.2.5. Extended Capability Port

El **Extended Capaility Port** o **Modo Capacidades Extendidas** es una interfaz bidireccional halfduplex similar al modo mejorado con la excepción de que las implementaciones de la PC usan acceso directo a memoria para gestionar las transferencias, lo que proporciona una velocidad de transferencia de datos mayor que la del modo mejorado, al permitir que el hardware del ISA del acceso directo a memoria y la interfaz del puerto paralelo manejen la tarea de transferir los datos, en lugar de la CPU. Proporciona hasta 2.5 MB/s de ancho de banda.

2. Sección de práctica

2.1. Consigna

Utilizando un procesador RISC-V, se escribió un programa en Assembler que permite leer la información de posición generada por un módulo receptor GPS u-blox MAX-M10M utilizando el protocolo NMEA 0183 y un puerto serie asincrónico UART.

2.2. Implementación

```
# -------------- REGISTROS -------------- #
.data
    #Estado y control de recepcion posicion 0x10000
    RCxSTA: .byte 0

    #Estado y control de transmision posicion 0x10001
    TXxSTA: .byte 0

    #8 bits superiores del divisor, posicion 0x10002
    SPxBRGH: .byte 0

    #8 bits inferiores del divisor, posicion 0x10003
    SPxBRGL: .byte 0

    #Registro de recepcion de datos posicion 0x10004
    RCxREG: .byte 0

    #Registro de transmision de datos posicion 0x10005
    TXxREG: .byte 0

    #Control de Baud Rate posicion 0x10006
    BAUDxCON: .byte 0

# Los comandos del protocolo NMEA utilizados para
# configurar el módulo tienen 4 partes principales:
#
#
```

```
# 1. Cadena de caracteres de comienzo: $PMTK.
#
# 2. Tres caracteres entre 000-999 para seleccionar el
#    comando a realizar.
#
# 3. Seccion llamada 'DataField' de longitud variable
#    donde se eligen los parámetros de cada comando.
#
# 4. Carácter '*' que marca la finalizacion del DataField
#    y otros dos últimos carácteres de checksum.



# Comando de acknowledge (comando 001)

    enviar_ack: .asciz "$PMTK001,500,3*35"
    .align 0



# Comando para seleccionar la frecuencia con la que se
# enviarán los datos de salida del modulo (comando 314).
# Los parámetros corresponden a la frecuencia con que
# se envía cada tipo de salida. La frecuencia puede
# configurarse con valores de 0-5.
#
# El cuarto parámetro es el que corresponde a la salida
# $GPGGA (nuestro caso), así que dejamos los demás en 0.

    frecuencia_salida: .asciz "$PMTK314
       ,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29"
    .align 0


# Comando para aumentar el baud rate (comando 251)
# El parámetro a elegir es el baud rate que se desea
# utilizar (puede ser 4800,9600,14400,19200,38400,57600
# o 115200).

    nuevo_baudrate: .asciz "$PMTK251,19200*22"
    .align 0


# El formato GPGGA tiene la siguiente forma:
#   $GPGGA,hhmmss.dd,xxmm.dddd,<N|S>,yyymm.dddd,
#       <E|W>,v,s,d.d,h.h,M,g.g,M,a.a,xxxx*hh<CR><LF>
#
# Las coordenadas de la latitud corresponden al campo
# xxmm.dddd (después de la segunda coma), mientras que
```

```
# las coordenadas de la longitud corresponden al campo
# yyymm.dddd (después de la cuarta coma).

    CHECK_GPGGA: .asciz "$GPGGA"
    .align 0

.text


# -------------- CONFIGURACION -------------- #

#Se configura registro de control y estado de transmision
configTXxSTA:
            # Los bits que nos interesa setear son
            # BRGH en 0 -> para bajos baud rates.
            # SENDB en 0 -> porque la transmision
                comienza deshabilitada.
            # SYNC en 0 -> porque la UART se usará en
                modo asíncrono.
            # TXEN en 0 -> porque la transmision comienza
                 deshabilitada.
            # TX9 en 0 -> porque la transmision será en
                modo 8 bits.
        # que corresponden a los bits 2, 3, 4, 5 y 6
            respectivamente, los demás bits pueden tomar
            cualquier valor

    la a1, TXxSTA
    andi a2,a2,0x81     # 1000 0001
    sb a2, (a1)


#Se configura registro de control y estado de recepcion
configRCxSTA:
        # Los bits que nos interesa setear son
            # SPEN en 0 -> porque el puerto serie
                comienza deshabilitado.
            # RXEN en 0 -> porque la recepcion comienza
                deshabilitada.
            # RX9 en 0 -> porque la recepcion será en
                modo 8 bits.
        # que corresponden a los bits 4, 6 y 7
            respectivamente, los demás bits pueden tomar
            cualquier valor

    la a1, RCxSTA
    andi a2,a2,0x23     # 0010 0011
    sb a2, (a1)
```

```
#Se configura el registro de control de Baud Rate
configBaudRate:
        # Los bits que nos interesa setear son:
            # ABDEN en 0 -> para deshabilitar el modo
               auto-deteccion.
            # WUE en 0 -> para deshabilitar la suspension
                del modulo.
            # BRG16 en 0 -> porque se utiliza el
               generador en modo 8 bits.
            # SCKP en 1 -> para definir que los datos se
               reciben en el flanco ascendente del clk.
        # que corresponden a los bits 0, 1, 3 y 4
           respectivamente, los demás bits pueden tomar
           cualquier valor.

    la a1, BAUDxCON      # Baud rate: 9600
    andi a2,a2,0x10      # 00010000
    sb a2, (a1)

# Suponiendo que se tiene una frecuencia de oscilacion de
    18.432 MHz, el valor que debe setearse en SPxBRG (
   formado por su parte inferior y superior) para
   conseguir un baud rate de 9600 será 29.

    la a1, SPxBRGL
    addi a2, zero, 29    # Valor de la parte inferior del
       divisor
    sb a2, (a1)

    la a1, SPxBRGH
    add a2, zero, zero  # Valor de la parte superior del
       divisor
    sb a2, (a1)

#Se habilita el registro de transmision
enableTXxREG:

    # Para transmision, debe activarse el bit 3 del
    # registro TXxREG
    la a1, TXxREG
    lb a0, (a1)
    ori a0, a0, 0x8 # 00001000
    sb a0, (a1)


#Se habilita el registro de recepcion
enableRCxREG:
```

```
    # Para recepcion deben activarse los  bits 7 y 4
    # del registro RCxREG
    la a1, RCxREG
    lb a0, (a1)
    ori a0, a0, 0x90 # 10010000
    sb a0, (a1)


# -------------- ACKNOWLEDGE -------------- #

# Se testea la comunicacion enviando un comando de
# acknowledge.
#
# En particular, el GPS utiliza el comando PMTK000
# y, para la transmision de mensajes, se utilizan
# 8 bits de datos, 0 bits de paridad y 1 bit de stop.

    la a0, enviar_ack    # Mensaje de acknowledge

acknowledge_loop:

    lb a1, (a0)
    beq a1, zero, acknowledge_continuar
    slli a1,a1,1         # Se agrega el bit de stop
    sb a1, TXxREG

    # Operacion NOP para esperar a que el dato se reciba
    add zero,zero,zero

    addi a0, a0, 4
    j acknowledge_loop

acknowledge_continuar:


# ---------- SELECCION DE TIPO DE DATO ---------- #

# Se seelecciona el tipo de dato que queremos recibir con
# el comando:
# $PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29,
# donde el cuarto bit encendido indica un formato GPGGA.

    # Frecuencia de salida configurada
    la a0, frecuencia_salida

select_loop:

    lb a1, (a0)
    beq a1, zero, select_continuar
```

```
    # Se agrega el bit de stop
    slli a1,a1,1

    sb a1, TXxREG

    # Operacion NOP para esperar a que el dato se reciba
    add zero,zero,zero

    addi a0, a0, 4
    j select_loop

select_continuar:

# -------------- AUMENTAR BAUD RATE -------------- #

 # Para poder recibir los datos en tiempo real,
 # se debe aumentar el baud rate:

# Del GPS:
la a0, nuevo_baudrate
cambiarBaudRate_loop:
    lb a1, (a0)
    beq a1, zero, cambiarBaudRate_continuar

    # Se agrega el bit de stop
    slli a1,a1,1

    sb a1, TXxREG

    # Operacion NOP para esperar a que el dato se
    # reciba
    add zero,zero,zero

    addi a0, a0, 4
    j cambiarBaudRate_loop

cambiarBaudRate_continuar:


# Del puerto UART:

# Nuevo divisor debe ser 14 para lograr baud rate
# de 19200
la a1, SPxBRGL
addi a2, zero, 14
sb a2, (a1)
```

```
# -------------- COMPROBACION DE INICIO -------------- #

# Se comprueba que la cadena, la cual se recibe desde la
# posicion de memoria del registro RCxREG, comience con
# $GPGGA.
#
# Se reserva el registro s4 para almacenar la suma de los
# caracteres ASCII, que se utilizará luego para comparar
# con el checksum de los datos recibidos y determinar su
# validez.

comprobar_inicio:

    # Direccion de palabra $GPGGA para comparar
    la t0,CHECK_GPGGA

    # Direccion de datos recibidos
    la t1,RCxREG

    # Reinicio la suma de caracteres para checksum
    add s4, zero, zero

comprobar_loop:

    lb t3, (t0)

    # Si termina la cadena $GPGGA, entonces coinciden
    # y se puede comenzar a recibir los datos.
    beq t3, zero, recibir_inicio

    # Se guarda el dato del puerto en t2
    lw t2, (t1)

    # Se quita el bit de stop
    srli t2,t2,1

    #  Se vuelve a empezar si no son iguales
    bne t3,t2, comprobar_inicio

    add s4,s4, t2
    addi t0, t0, 1

    j comprobar_loop


# -------------- RECEPCION DE DATOS -------------- #

recibir_inicio:
```

```
    # Caracter * (ASCII: 42) indica inicio del checksum
    lb t3, 42

    # Lugar de memoria donde se guardan los datos
    lw t0, 0x112000

recibir_loop:

    # Se guarda el dato del puerto
    lw t2, (t1)

    # Se quita el bit de stop
    srli t2,t2,1

    # Si se encuentra *, comienza el checksum
    beq t3, t2, comprobar_checksum

    sb t2, (t0)
    add s4,s4, t2
    addi t0, t0, 1

    j recibir_loop

# El checksum consiste en un algoritmo del cual se
# obtiene una suma de verificacion para validar la
# integridad de los datos.
#
# El resultado se compara con la suma de caracteres
# ASCII, almacenada en el registro s4 en la seccion
# de comprobacion de datos.

comprobar_checksum:
    # Parte alta
    lw s1, (t1)
    # Parte baja
    lw s2, (t1)

    # Se quita el bit de stop en ambas
    srli s1,s1,1
    srli s2,s2,1

    # Se desplaza 4 bits a la izquierda la parte alta
    slli s1,s1,4

    # Se suman parte alta y baja
    add s3, s1, s2
```

```
    # Se comprueba que la suma y el checksum sean
    # iguales, si lo son entonces se finaliza
    beq s3, s4, finalizar


# Si se llega a esta seccion, los datos no coincidieron y
# se descartan.

error:
    # Exit del programa retornando 1
    addi a0, zero, 1
    addi a7, zero, 93 # Retorna el codigo que haya en a0
    ecall

finalizar:
    # Exit del programa retornando 0
    addi a7, zero, 10
    ecall
```