Memoria del proyecto de Sistemas Empotrados II 2024/25

Implementación en SYS-BIOS de una tarea para el manejo de un GPS en la plataforma TIVA

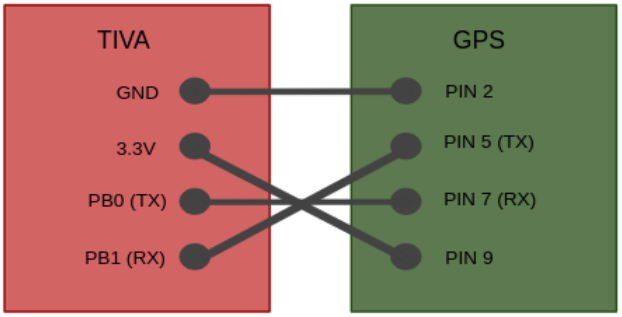
Inés Román Gracia, 820731

Introducción

El proyecto consiste en realizar lecturas a un módulo GPS Fastrax mediante una TIVA TM4C123GXL. El objetivo es implementar un driver de bajo nivel que permita la recepción y transmisión de datos utilizando el protocolo NMEA, empleado por el módulo GPS, utilizando la UART1 de la TIVA. En el estado actual del proyecto, el programa es capaz de recibir los datos enviados por el módulo GPS a través de un manejador de interrupciones que almacena la información en un buffer circular. La TIVA accede a este buffer mediante una tarea periódica que, en cada ejecución, lee los caracteres que almacena el buffer y cuando hay un mensaje completo lo imprime en la consola. A continuación, se detallan los pasos que se he seguido, tanto de conexiones de los componentes como de desarrollo del código para realizar el proyecto.

1. Conexiones físicas

Antes de comenzar a implementar las configuraciones y comunicaciones entre el GPS y la placa, se interconectan los pines de transmisión (TX) y recepción (RX) de ambos dispositivos. Además, se conectaron el suministro de corriente y la conexión a tierra del GPS utilizando la alimentación proporcionada por la placa.



2. Configuración de la UART1

Para poder leer los datos del módulo GPS, necesitamos configurar la UART de una forma adecuada para recibir los datos, configurar una interrupcipon asociada y un manejador de interrupciones. Para ello hemos creado la función *InitUart*:

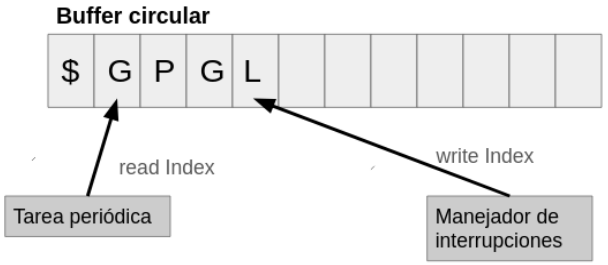
```
Void InitUart(void) {
  SYSCTL_RCGCUART_R = 0x2;  // Activa UART1
  SYSCTL_RCGCGPIO_R = 0x2;  // Activa GPIOB

  GPIO_PORTB_DEN_R = 0x3;  // Habilita PB0 y PB1 como pines digitales
  GPIO_PORTB_AFSEL_R = 0x3; // Habilita funciones hardware en PB0 y PB1
  GPIO_PORTB_AMSEL_R = 0x0; // Deshabilita funcion analagica en PB0 y PB1
  GPIO_PORTB_PCTL_R = 0x11;  // Configura PB0 y PB1 para UART1

  UART1_IBRD_R = 104; // Configura la parte entera del divisor de baudios
  UART1_FBRD_R = 11; // Configura la parte fraccional del divisor de baudios
  UART1_LCRH_R = 0x60; // Configura 8 bits sin paridad, 1 bit de stop
  UART1_CC_R = 0x0; // Selecciona el reloj del sistema para UART1
  UART1_CTL_R = 0x301;  // Habilita UART1, TX y RX
```

```
  UART1_ICR_R = ~0x10; // Limpia las interrupciones previas
  UART1_IM_R = 0x10;  // Habilita interrupcion por recepcion (RX)
  NVIC_EN0_R = 0x40;  // Habilita interrupciones para UART1 (bit 6 en EN0)
}
```

Se activan los módulos UART1 y GPIOB, se habilitan PB0 y PB1 como pines digitales y se establecen parámetros de comunicación como 9600 baudios, 8 bits de datos, sin paridad y 1 bit de parada. Además, se limpian interrupciones previas y habilita interrupciones por recepción de datos en el NVIC.

3. Comunicaciones TIVA y GPS

La comunicación entre la TIVA y el GPS se realiza utilizando un buffer circular. El manejador de interrupciones (asociado a la UART1) lee los datos recibidos desde el GPS y los almacena en este buffer. La TIVA puede acceder a este buffer mediante una tarea periódica que lee los datos y, cuando recibe un mensaje completo, lo muestra por consola.



3.1. Configuración inicial de las comunicaciones

Antes de iniciar las comunicaciones entre la TIVA y el GPS, es necesario enviar ciertos comandos al módulo GPS para configurar correctamente la comunicación, como ajustar los baudios:

```
"$PMTK220,1000*1F\x0D\x0A"
"$PMTK251,9600*17\x0D\x0A"
"$PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\x0D\x0A"
```

Para poder enviar estos datos al GPS desde la tiva a través de la UART1 se ha creado la función:

```
Void sendGpsCommand(const char *command) {
  while (*command) {  // Mientras haya caracteres en el comando
    // Espera hasta que el buffer de transmisión esté libre
    while ((UART1_FR_R & UART_FR_TXFF) != 0) {}
    UART1_DR_R = *command;  // Envía el siguiente carácter al GPS
      command++;  // Avanza al siguiente carácter del comando
  }
}
```

3.2. Manejador de interrupciones

El manejador de interrupciones *GPSInt* se encarga de gestionar los datos que llegan a la TIVA a través de la UART1 desde el GPS y de almacenarlos en un buffer circular:

```
Void GPSInt(UArg arg) {
   // Verifica si el registro de datos de UART1 tiene datos válidos
   if (!(UART1_DR_R & 0xF00)) {
      char received = UART1_DR_R & 0xFF; // Lee el dato recibido (8 bits)
      UART1_ICR_R |= 0x10; // Limpia la bandera de interrupción RX de UART1
      // Verifica si hay espacio en el buffer para almacenar el dato recibido
      if(((writeIndex + 1) % BUFFER_SIZE) != readIndex) {
          buffer[writeIndex] = received;  // Almacena el dato en el buffer
          writeIndex = (writeIndex + 1) % BUFFER_SIZE; // Avanza el índice
      }
   }
}
```

3.3. Tarea periódica

La tarea periódica ha sido creada de igual forma que las tareas vistas en las prácticas de la asignatura. Además de la función que se desea ejecutar, se ha utilizado un semáforo que permite la ejecución de la función y la función de *release* que es invocada por el reloj cada 100 ms, lo que permitirá a la tarea llevar a cabo su función principal.

La función principal de la tarea periódica es *periodicTaskFunc*. Esta función se ejecuta de forma continua en un bucle infinito, pero solo comienza a ejecutarse cuando el semáforo es liberado por *periodic_release*:

```
 Void periodicTaskFunc(UArg arg0, UArg arg1) {
    char msgBuffer[BUFFER_SIZE];
    int msgIndex = 0;
    for (;;) {
        Semaphore_pend(periodicSem, BIOS_WAIT_FOREVER);
        . . .
```

Una vez que la tarea se ejecuta, comienza a leer los datos almacenados en el buffer circular. Los datos se leen en un bucle while que continuará mientras el índice de lectura (readIndex) no haya alcanzado el índice de escritura (writeIndex), lo que significa que aún hay datos en el buffer:

```
        . . .
    while (readIndex != writeIndex) {
          char received = buffer[readIndex];
          readIndex = (readIndex + 1) % BUFFER_SIZE;
          . . .
```

Finalmente se procesan los datos guardados en el buffer circular. Se crea un buffer local en el que se empiezan a guardar caracteres cuando se detecta el inicio de un mensaje (empiezan por “$”) y acaba cuando se detecta el final de un mensaje (acaba con salto de línea) y, entonces, lo imprime por consola:

```
. . .
if (received == '$') { // Comienzo del mensaje
  msgIndex = 0; // Se empieza a escribir en orden en el buffer local
}

if (msgIndex < BUFFER_SIZE - 1) { // Si el buffer no se desborda
  // Guarda el siguiente caracter en el buffer local
  msgBuffer[msgIndex++] = received;
}

if (received == '\n') { // Final del mensaje
  // Guarda el caracter de finalizacion el el buffer local
  msgBuffer[msgIndex] = '\0';
  // Imprime el mensaje guardado en el buffer local
  System_printf("%s", msgBuffer);
  System_flush();

  msgIndex = 0; // Reinicia el índice (no es necesario)
}
```

4. Observaciones

Es un sistema sencillo con poca carga de cómputo. El GPS manda mensajes aproximadamente cada segundo y la tarea periódica los procesa de forma bastante rápida. Al observar las tareas en el Execution Analysis de Code Composer, se ve como el tiempo de ejecución de la tarea es mínimo: