Práctica 2. Implementación de un driver básico para la transmisión de datos a través de la UART.

1. Objetivo

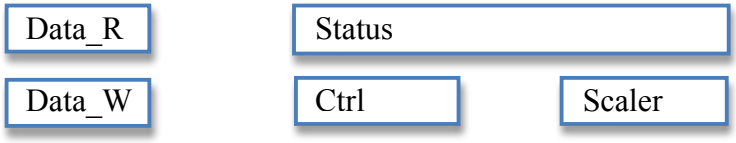
El objetivo de la siguiente práctica es el de implementar el software de control (***driver***) de un dispositivo de entrada/salida. El dispositivo elegido forma parte del conjunto de interfaces estándar disponibles en la configuración del simulador TSIM2 del procesador LEON2 que se está utilizando en este laboratorio. Concretamente se trata de la **UART**-A (Universal Asynchronous Receiver-Transmiter-A), que permite la **comunicación serie asíncrona** con otros dispositivos, tanto en modo entrada, como salida.

La implementación de un *driver* para un dispositivo de estas características requiere conocer el comportamiento de su **controlador** con un cierto nivel de detalle. Cada uno de los registros del controlador cuenta con una función específica, que debe ser analizada. Por otra parte, un mismo dispositivo puede ser configurado para trabajar de manera distinta y eso determinará el enfoque de diseño del *driver*. Algunas de las preguntas básicas que condicionan este diseño son:

- ¿debe accederse al dispositivo en **modo bloque** o en **modo carater**?
- ¿debe el *driver* poder utilizarse por más de un hilo/proceso?
- ¿Las operaciones sobre el dispositivo implican una **espera activa** o se emplean buffers controlados por interrupciones para evitar ese bloqueo?

En esta práctica se va a comenzar con la implementación de un sencillo *driver* en modo carácter para la UART-A del LEON3. El *driver* cuenta con una función que permite simplemente transmitir un caracter a través de este dispositivo con una **espera activa**, de manera que el hilo/proceso permanecerá en la función hasta que el caracter se haya transmitido. La práctica además plantea un ejercicio relacionado con la implementación de rutinas auxiliaries empleadas en funciones de formateo y redirección de la información hacia un dispositivo de salida (tipo *printf*).

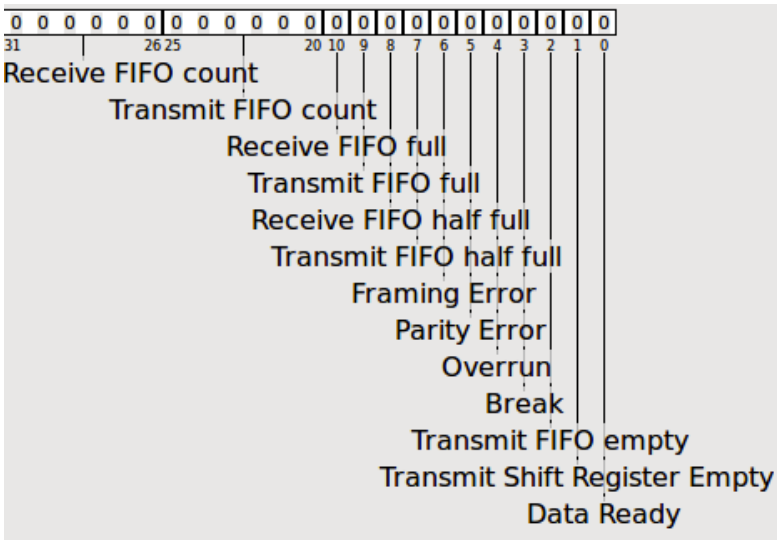
El controlador de la UART-A está formado internamente por 5 registros. Dos registros de datos, **Data_R** y **Data_W**, que permiten, respectivamente, leer el dato recibido, y escribir el dato que se desea transmitir. Un registro denominado **Status** que proporciona información sobre el estado actual del dispositivo. El registro **Ctrl**, que permite controlar el dispositivo y, finalmente, el registro de **Scaler**, que permite configurar la velocidad a la que trabajará el puerto. La siguiente figura muestra un esquema de los registros internos del controlador.



La configuración de interfaces que presenta el simulador TSIM2 ubica los regitros de datos, control y estado del controlador de este dispositivo en el rango de direcciones de memoria 0x80000100-0x800010C.

Los registros **Data_R** y **Data_W** son accesibles a través de la misma dirección 0x80000100. El controlador internamente selecciona **Data_R** en las operaciones de lectura y **Data_W** en las operaciones de escritura. El registro **Status** está localizado en la dirección 0x80000104, mientras que **Ctrl** y **Scaler** se ubican respectivamente, en 0x80000108 y 0x8000010C.

El control de la UART que se va a realizar en esta práctica requiere conocer únicamente el comportamiento del bit ***Transmit FIFO Full*** (**TFF**) del registro **Status** (ver figura siguiente)**.** Este bit nos indica si la cola del transmisor está llena (valor de ***TFF*** = 1). El algoritmo a emplear para transmitir va consistir simplemente en comprobar el valor del bit ***TFF*** antes de escribir en el registro **Data_W**. En caso de que valga 1 (cola llena), se esperará en un bucle durante un cierto tiempo a que el bit cambie de valor. En caso de que después de ese tiempo ***TFF*** permanezca a 1, se considerará que hay un error y que no es posible transmitir el dato.



Descripción del registro Status de la UART-A

2. Creación de proyecto para LEON3

Creación de un nuevo proyecto (*Empty Project*) denominado prac2 cuyo ejecutable sea para la plataforma Sparc Bare C. En ese proyecto crear dos subdirectorios **include** y **src**. En el directorio **include** crear el archivo leon3_types.h con la siguiente redefinición de los tipos básicos para la arquitectura del proecesador LEON3.

```
#ifndef LEON3_TYPES_H
#define LEON3_TYPES_H

typedef unsigned char         byte_t;
typedef unsigned short int    word16_t;
typedef unsigned int          word32_t;
typedef unsigned long int     word64_t;


typedef signed char           int8_t;
typedef signed short int      int16_t;
typedef signed int            int32_t;
typedef signed long int       int64_t;

typedef unsigned char         uint8_t;
typedef unsigned short int    uint16_t;
typedef unsigned int          uint32_t;
typedef unsigned long int     uint64_t;

#endif
```

3. Implementación de funciones básicas para la transmisión a través de la UART-A del LEON3.

Creación en el directorio **include** el archivo *leon3_uart.h* con el siguiente contenido:

```
#ifndef LEON3_UART_H_
#define LEON3_UART_H_

#include "leon3_types.h"

int8_t leon3_putchar(char c);

#endif /* LEON3_UART_H_*/
```

Creación en el directorio **src** el archivo *leon3_uart.c* con el siguiente contenido en el que se declara la estructura UART_regs que permite acceder a los registros de la UART de LEON3.

```
#include "leon3_uart.h"

//Estructura de datos que permite acceder a los registros de la
//UART de LEON3

struct UART_regs
{
      /** \brief UART  Data Register */
      volatile uint32_t Data;       /* 0x80000100 */
      /** \brief UART  Status Register */
      volatile uint32_t Status;     /* 0x80000104 */
      /** \brief UART  Control Register */
      volatile uint32_t Ctrl;       /* 0x80000108 */
      /** \brief UART  Scaler Register */
      volatile uint32_t Scaler;     /* 0x8000010C */
};
```

En la estructura se utiliza el atributo **volatile** que fuerza a que el compilador no optimice la comprobación de los valores que toman esos campos. Esto permite que se puedan modificar vía hardware, y el software compruebe siempre el valor que tienen, sin considerar si acaba de asignarse a un valor constante. En los siguientes dos ejemplos se ve la diferencia de declarar una variable como **volatile** o no.

```
uint8_t aux;
uint8_t i;

aux=0;

if(aux>0){

      // El optimizador no incluye este código, ni el código de
      // comprobación de aux, puesto que acaba de asignarse a 0

}else{


}
```

```
volatile uint8_t aux;
uint8_t i;

aux=0;

if(aux>0){

      // El optimizador SI incluye este código, y el de la
      // comprobación de aux, puesto que aux es volatile

}else{


}
```

Añadir al archivo *leon3_uart.c* el define `LEON3_UART_TFF` que va a actuar de máscara del bit ***Transmit FIFO Full*** del registro de Status. Recordar que para que una transmisión pueda iniciarse es necesario que el bit enmascarado por `LEON3_UART_TFF` **NO** esté a 1.

```
//!  LEON3 UART A Transmit FIFO is FULL
#define LEON3_UART_TFF (0x200)
```

Añadir al archivo *leon3_uart.c* la declaración del puntero `pLEON3_UART_REGS`. Completar el código para que este puntero apunte a la dirección 0x80000100, que es la dirección donde se encuentra ubicados los registros de la UART-A

```
struct   UART_regs * pLEON3_UART_REGS=  //COMPLETAR
```

Completar la función leon3_putchar, (ubicada igualmente en *leon3_uart.c)* para que, una vez que se ha comprobado que el bit TFF del registro de ***Status*** ha dejado de valer v 1 (lo que indica que se puede realizar una nueva transmisión), se escriba el carácter c pasado por parámetro en el registro ***Data*** la UART. Si en la espera ha habido un timeout el caracter no se escribirá y la función retornará un valor distinto de 0.

```
int8_t leon3_putchar(char c)
{

	uint32_t write_timeout=0;

	while(
			//COMPLETAR.

			&&(write_timeout < 0xAAAAA)
		){
			write_timeout++;

	} //Esperar hasta que TFF es 0


	if(write_timeout <  0xAAAAA)
		//COMPLETAR. Escribir el carácter en el registro Data


	return (write_timeout ==  0xAAAAA);

}
```

4. Primera versión del programa principal.

Crear el archivo main.c con el siguiente código de prueba del driver básico y comprobar que la salida es

```
p2
```

```
#include "leon3_uart.h"

int main()
{

	leon3_putchar('p');
	leon3_putchar('2');
	leon3_putchar('\n');
	return 0;

}
```

5. Rutinas de formateo y redirección hacia la salida.

Añadir al proyecto los archivos leon3_bprint.h (al directorio include) y leon3_bprint.c (al directorio src). Implementar en estos archivos las siguientes rutinas auxiliares de formateo de información y redirección hacia el puerto serie.

```
//transmite a través del puerto serie la cadena de caracteres
//pasada por parámetro.

int8_t leon3_print_string(char* str);  //ver **

//transmite a través del puerto serie el entero transformado en
//cadena de caracteres.

int8_t leon3_print_uint8(uint8_t i);     // ver ***
```

** (Tened en cuenta para la implementación de `leon3_print_string` que el último caracter de una cadena es el '\0')
*** (Tened en cuenta para la implementación de `leon3_print_uint8,` que un entero de 8 bits tiene un valor máximo de 255, por lo que para su transformación bastará con calcular 3 caracteres: el de las centenas, el de las decenas y el de las unidades)

Comprobar que su implementación es correcta mediante el siguiente programa principal que debe dar como salida:

```
cadena
55
```

```
#include "leon3_uart.h"
#include "leon3_bprint.h"

int main()
{
      char * pchar="cadena\n";
      leon3_print_string(pchar);
      leon3_print_uint8(55);
      return 0;

}
```