Programación de Sistemas Embebidos

Clase 5 – Serial asincrónico

Programa analítico de la asignatura

UNIDAD 2: Herramientas de desarrollo:
Toolchains (compilador, ensamblador, vinculador) .
Compilación cruzada (cross compiler). Debugger. Analizador de archivos objetos y ejecutables (disassembly). Automatización del ciclo de compilación (make).

UNIDAD 3: E/S de bajo nivel:
Serial asincrónico: UART.

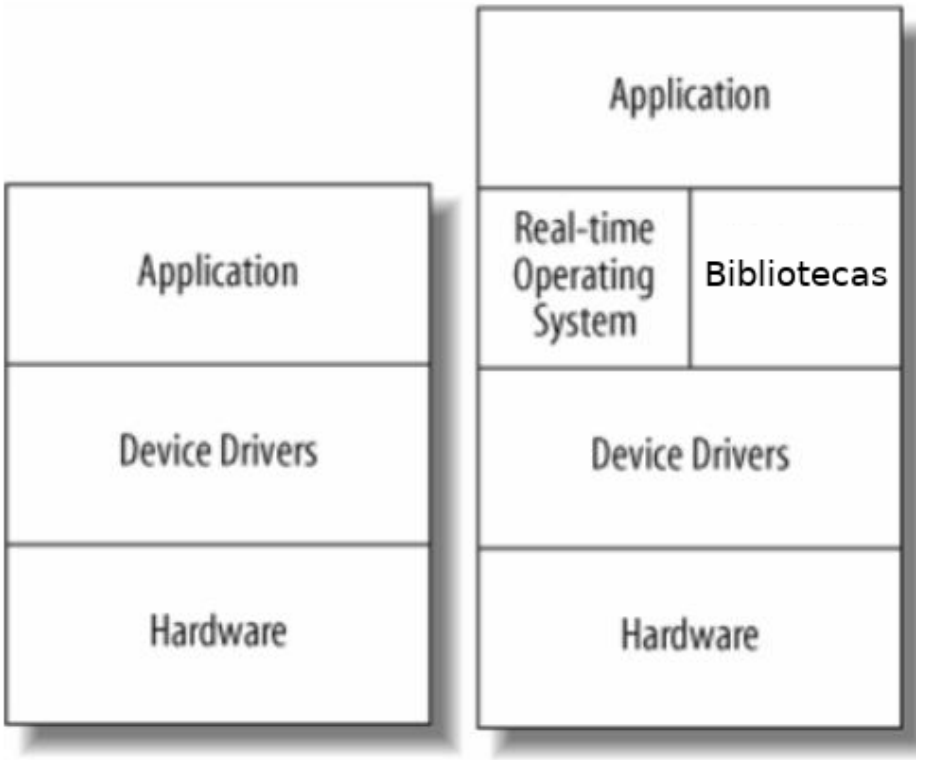
UNIDAD 4: Programación de bajo nivel
Lenguaje C. Programación sobre hardware sin sistema (baremetal).

Temario

- Arquitectura de un sistema (sw) embebido
- Reset e inicialización del software
- Ciclo de vida del desarrollo de software embebido
- Arquitectura del hardware UART en AVR
- Arquitectura de un driver para sistemas embebidos

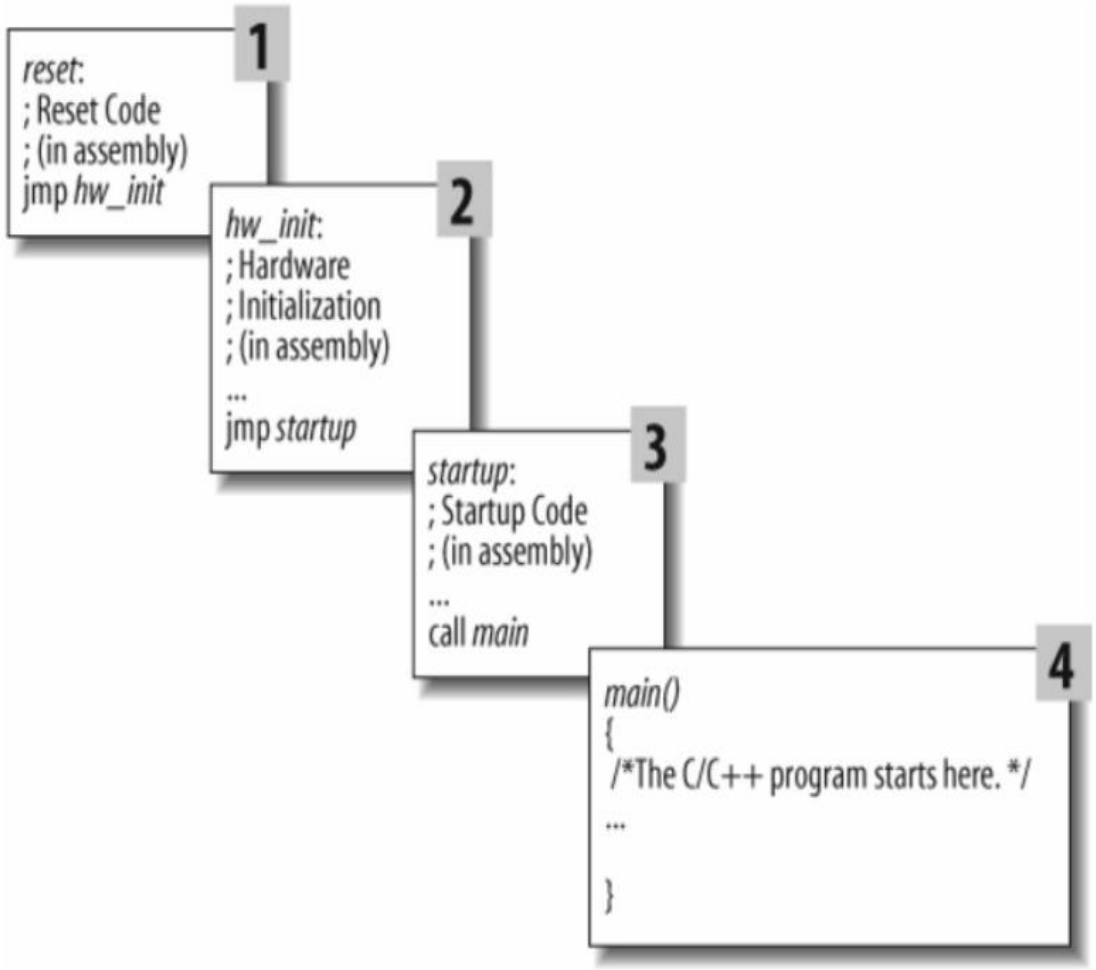
Arquitectura de un sistema (sw) embebido

- Aplicación de bucle infinito con drivers
- Aplicación con drivers e interrupciones
- Aplicación con drivers (e interrupciones) y núcleo de tiempo real
- Combinación de las anteriores junto a Bibliotecas



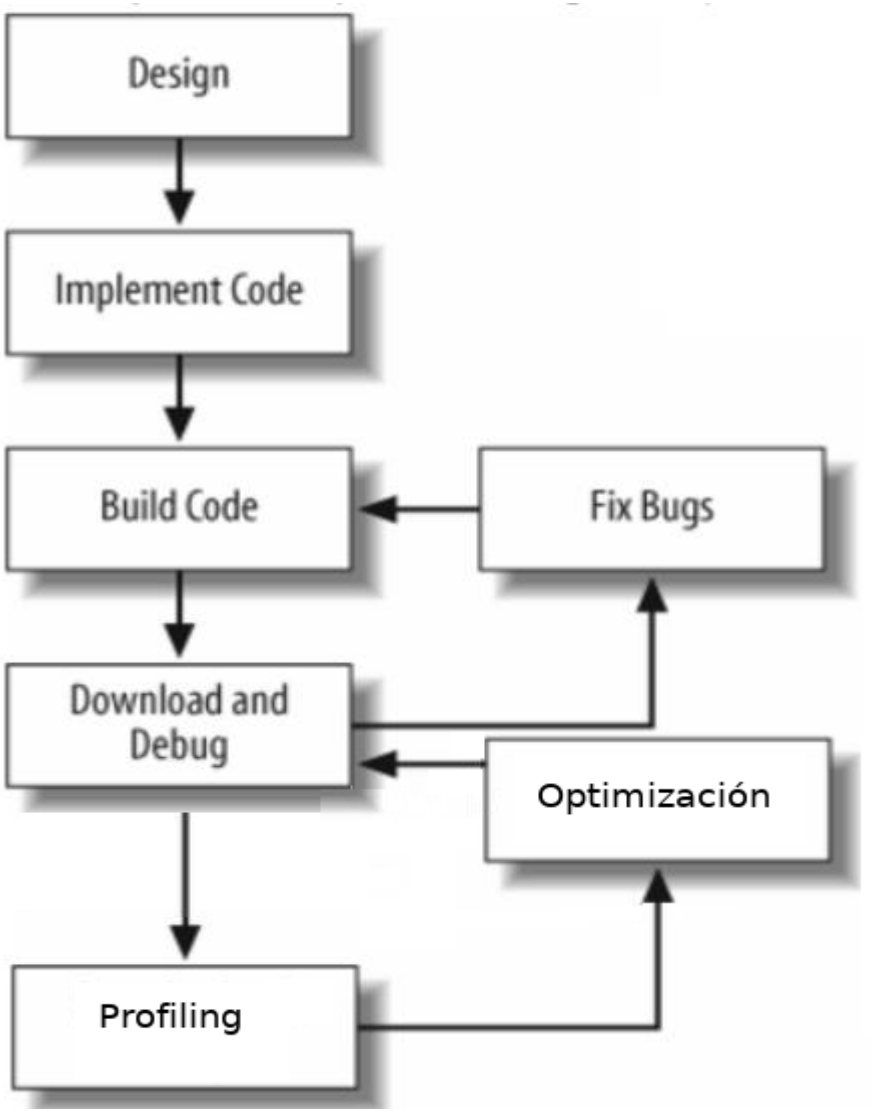
Reset e inicialización del software

- Tabla de vectores
- Deshabilitar interrupciones
- Copiar codigo de ROM a RAM
- Copiar variables globales de ROM a RAM
- Inicializar en RAM variables
- Establecer el tope de la pila
- Llamar a main



Ciclo de vida del desarrollo de software embebido

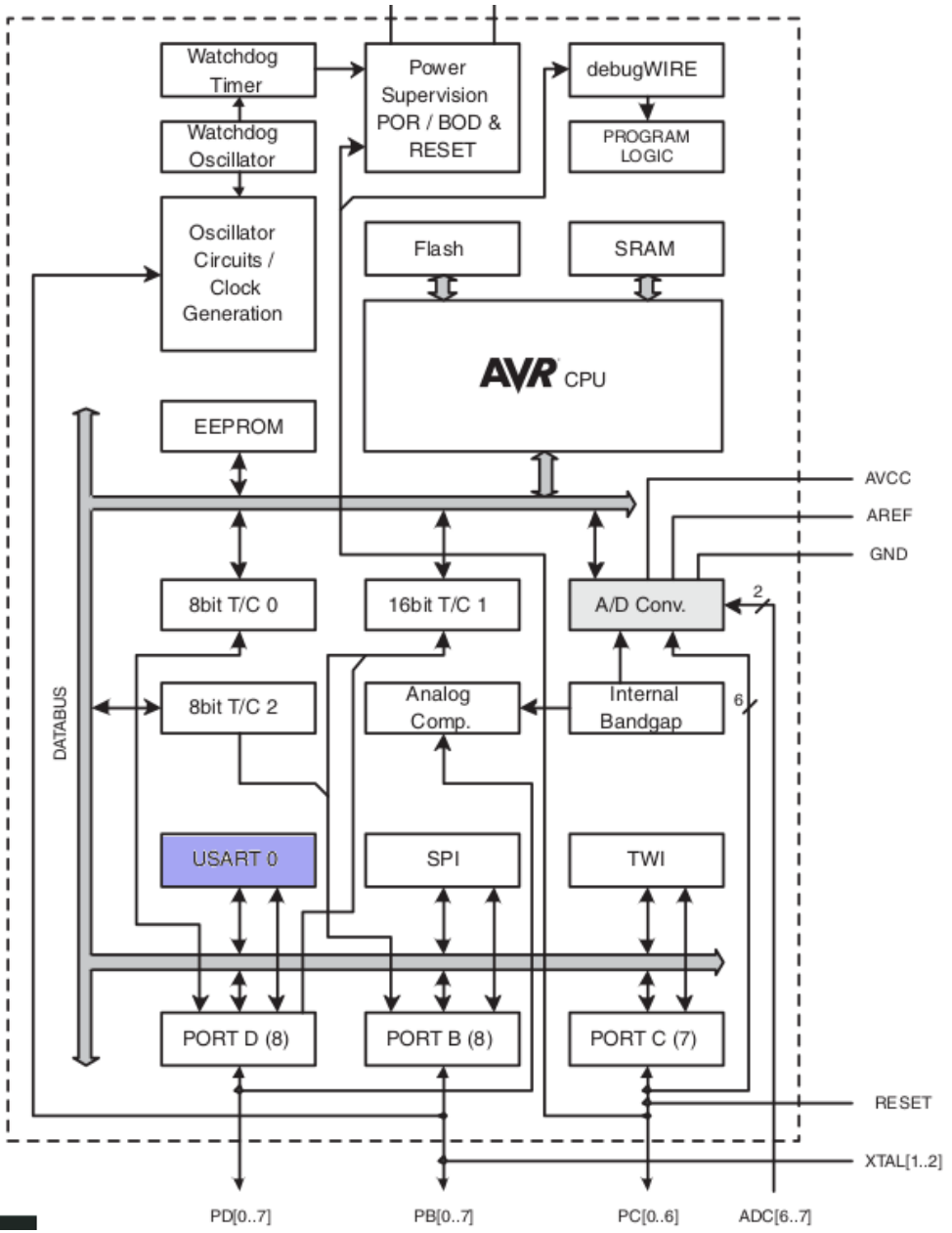
- En las etapas de diseño se contemplan distintas variantes del hardware
- En las etapas de profiling y optimización se decide el hardware final



UART: Arquitectura del hardware en AVR

- ¿Qué es un UART?
- ¿Qué significa asincrónico o sincrónico?

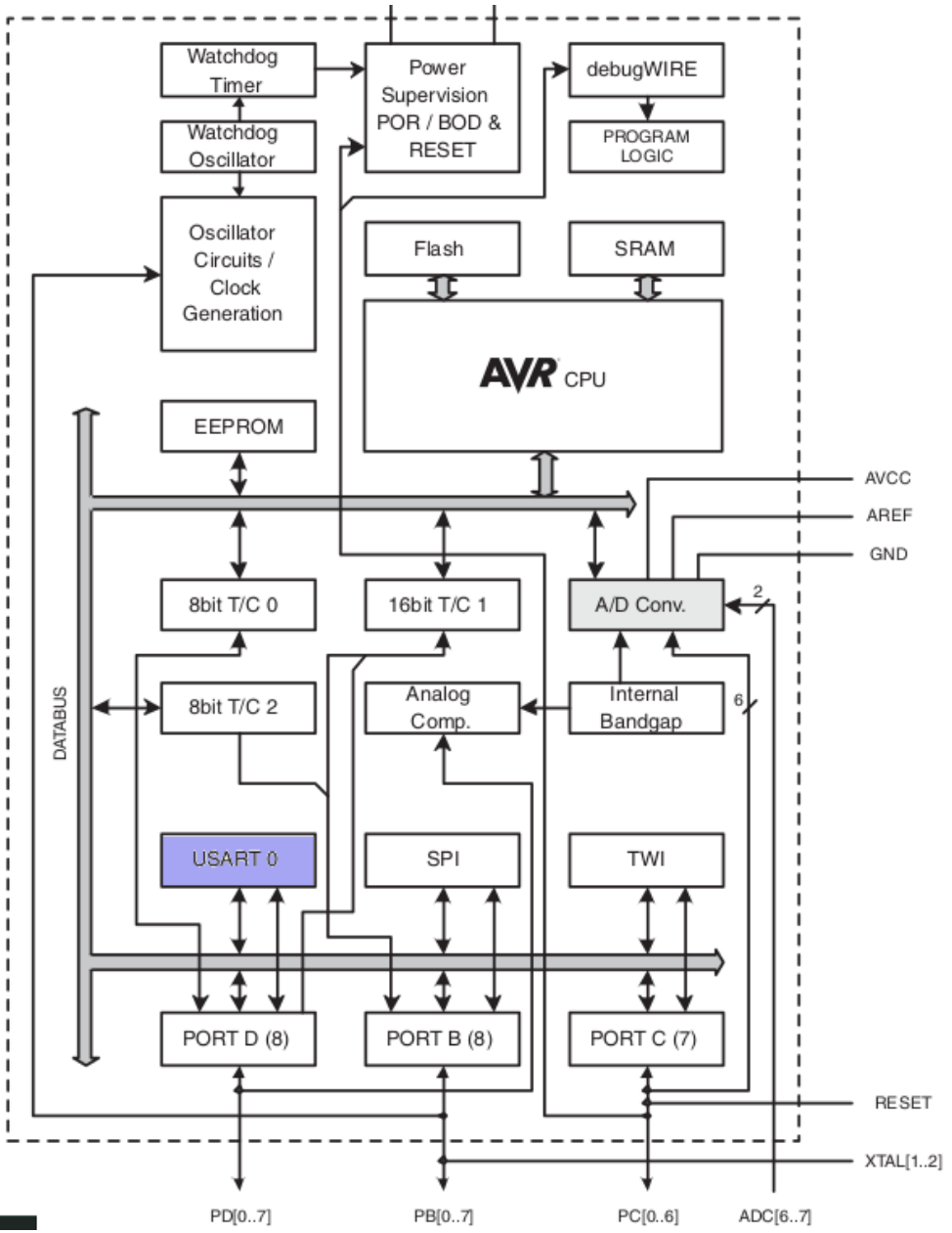
  "The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device."



UART: Arquitectura del hardware en AVR

Componentes (source: wikipedia):

- input and output shift registers
- transmit/receive control
- read/write control logic
- transmit/receive buffers (optional)
- system data bus buffer (optional)
- First-in, first-out (FIFO) buffer memory (optional)
- Signals needed by a third party DMA controller (optional)
- Integrated bus mastering DMA controller (optional)

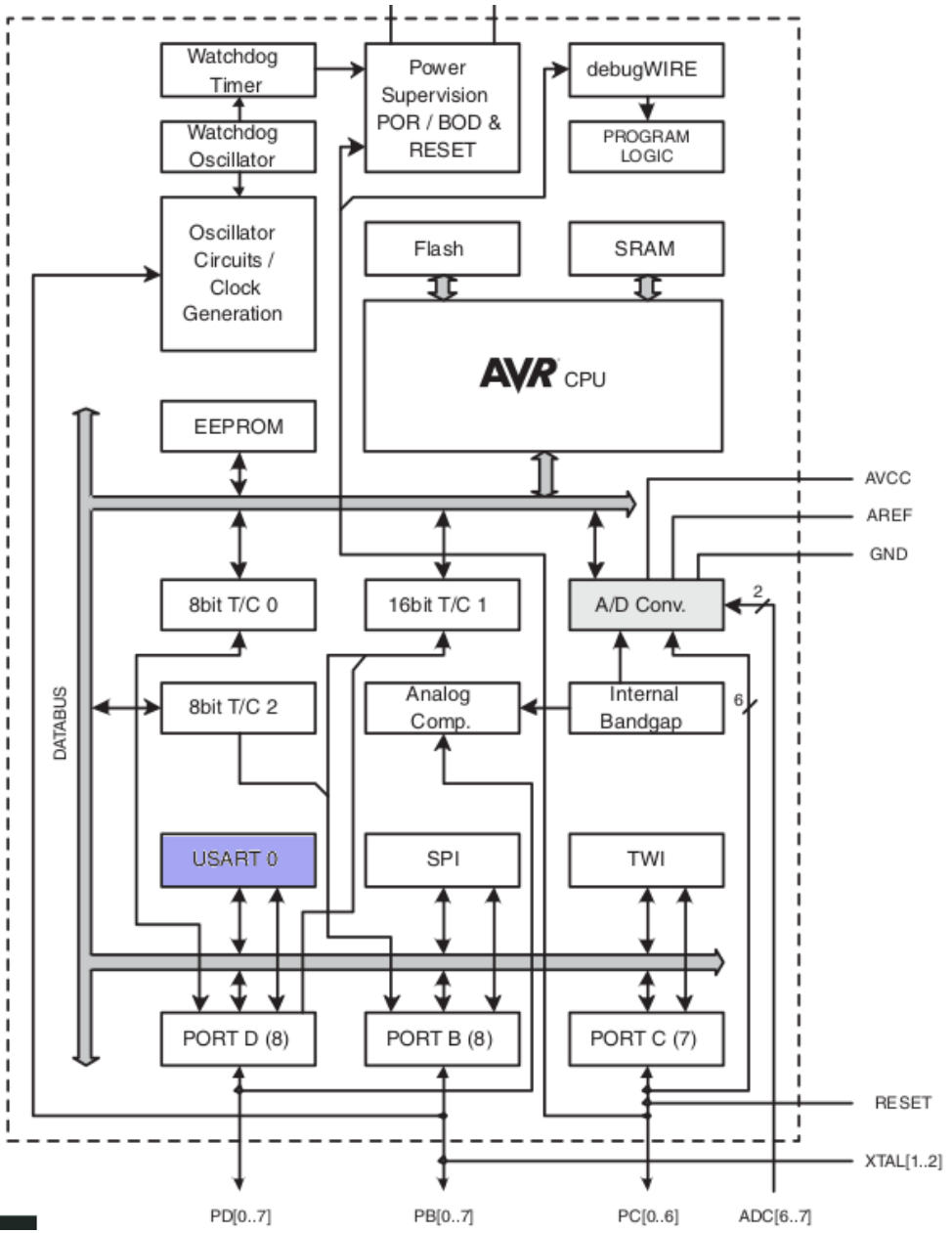


UART: Arquitectura del hardware en AVR

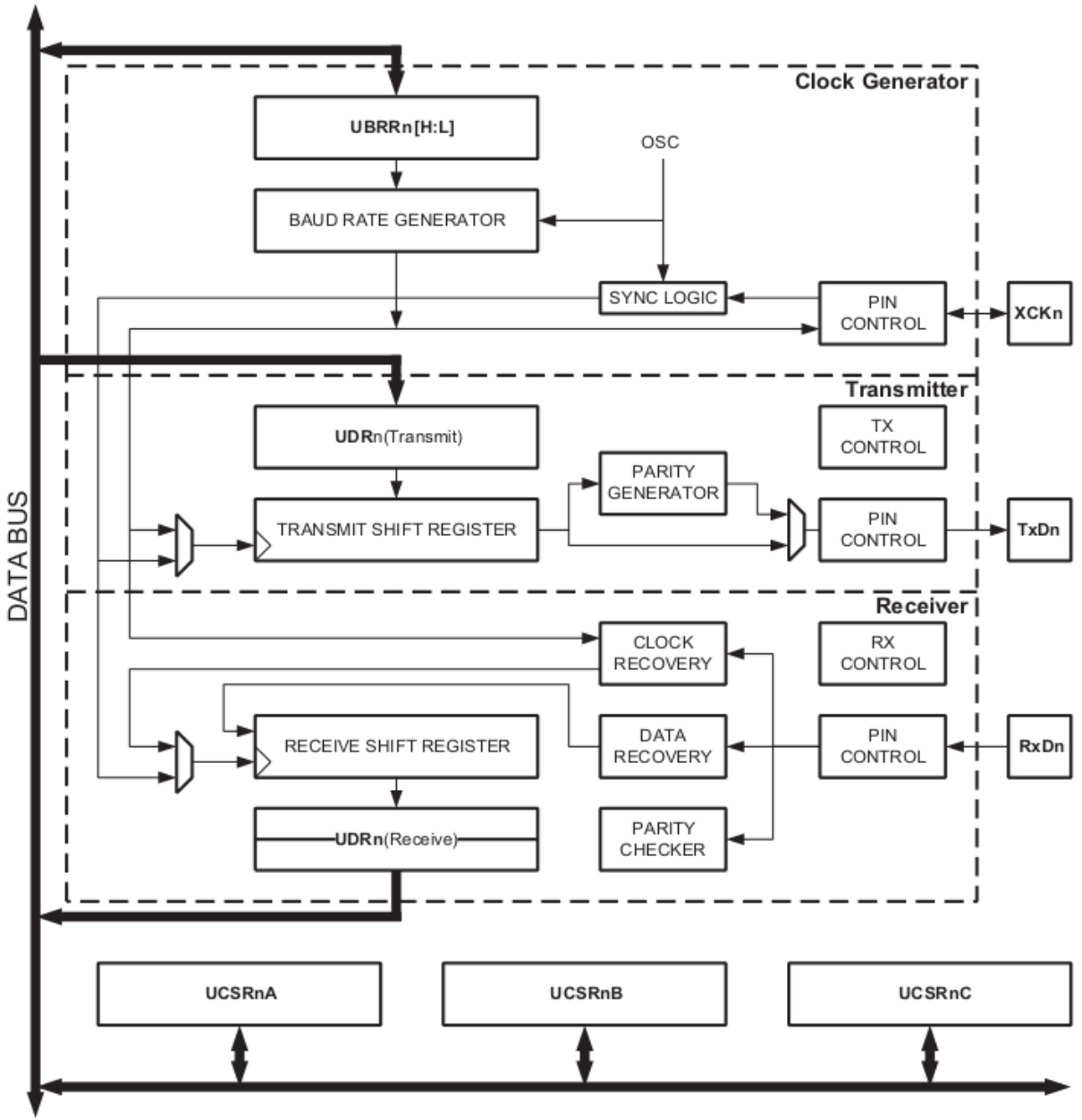
- El funcionamiento es sencillo y el hardware necesario es básico (barato)
- Todos los microcontroladores y microprocesadores suelen contar con un dispositivo UART

Usos:

- Comunicación entre un MCU y PC
- Comunicación entre un MCU y MCU
- Comunicación entre un MCU y dispositivo serial
- Muy util para debug, programación, etc



UART: Arquitectura del hardware en AVR



UART: Arquitectura del hardware en AVR

Registros:

- UBRRn[h:l] : 16 bits. Baud rate
- UCSRnA, UCSRnB, UCSRnC: 8 bits. Status y Control
- UDRn (Rx/Tx): 8bits. Datos

UART: Arquitectura del hardware en AVR

Registros: UBRRn[h:l] : 16 bits. Baud rate

Table 20-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$

20.11.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UART: Arquitectura del hardware en AVR

Registros: UCSRnA, UCSRnB, UCSRnC: 8 bits. Status y Control

20.11.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXCn: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buff (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushec consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Cc interrupt (see description of the RXCIEn bit).

- **Bit 6 – TXCn: USART Transmit Complete**

This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and ther new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The T can generate a Transmit Complete interrupt (see description of the TXCIEn bit).

UART: Arquitectura del hardware en AVR

Registros: UCSRnA, UCSRnB, UCSRnC: 8 bits. Status y Control

20.11.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DOR UPEn Flags.

- **Bit 3 – TXENn: Transmitter Enable n**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port oper the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Tr Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer ov TxDn port.

UART: Arquitectura del hardware en AVR

Registros: UCSRnA, UCSRnB, UCSRnC: 8 bits. Status y Control

20.11.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 20-8.

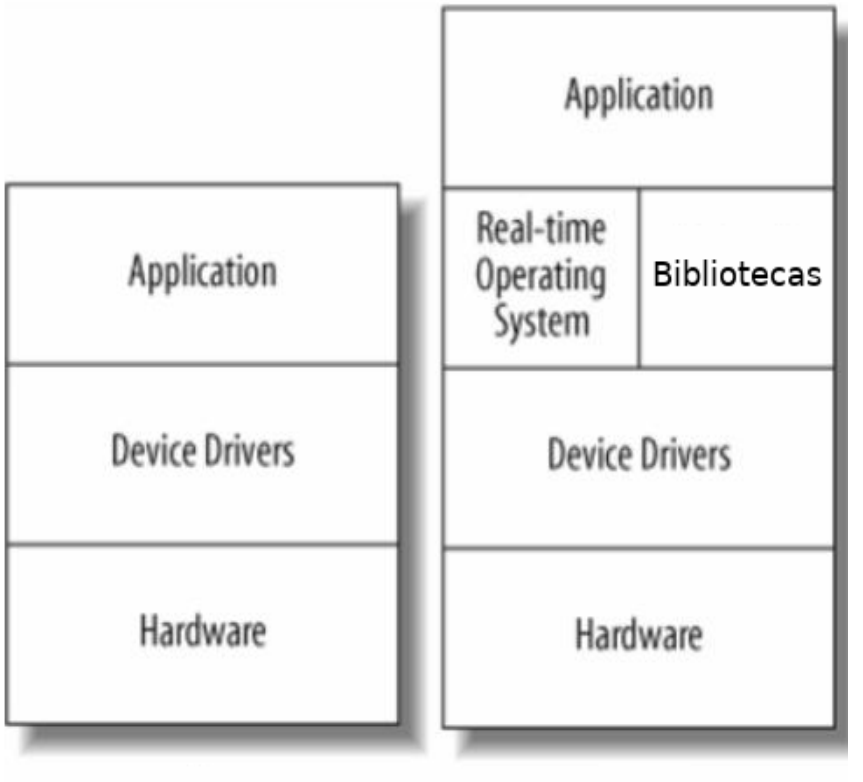
Table 20-8. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART

UART: Arquitectura de un driver para sistemas embebidos (software)

serial_uart.c
serial_uart.h

- Definir una estructura que represente los registros de estado, control y datos del periférico
- Crear la rutina de inicialización (init)
- Escribir las rutinas de ENTRADA y SALIDA
  - Con E/S programada
  - Con E/S via interrupciones (rutina de atención de interrupciones)



- Escribir una aplicación que verifique (utilice) el driver de manera básica (main.c

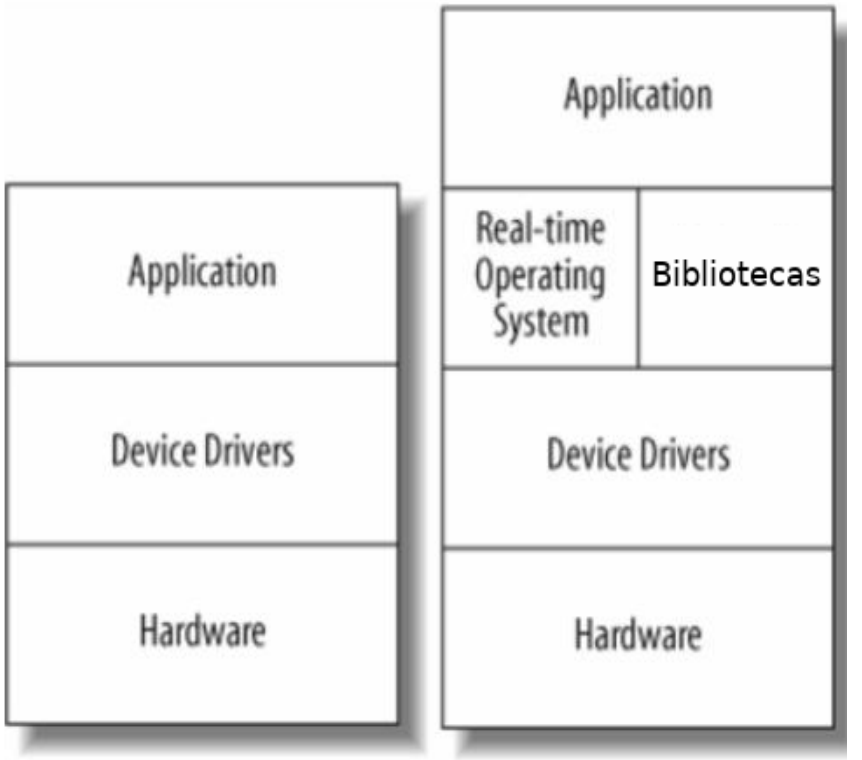
- IDEA CLAVE: Ocultar el Hardware completamente (main no debería conocer detalles de los registros ni del funcionamiento interno del periférico). Permite portar la aplicación a otras plataformas.

UART: Arquitectura de un driver para sistemas embebidos (software)

serial_uart.c
serial_uart.h

- Definir una estructura que represente los registros de estado, control y datos del periférico

```
typedef struct
{
    uint8_t status_control_a;    /* ucsr0a USART Control and Status A */
    uint8_t status_control_b; /* ucsr0b USART Control and Status B */
    uint8_t status_control_c; /* ucsr0c USART Control and Status C */
    uint8_t _reserved; /* espacio sin utilizar */
    uint8_t baud_rate_l; /* ubrr0l baud rate low */
    uint8_t baud_rate_h; /* ubrr0h baud rate high */

    uint8_t data_es;    /* udr0 i/o data */

} volatile uart_t;

/* puntero a la estructura de los registros del periferico */
volatile uart_t *puerto_serial = (uart_t *) (0xc0);
```



UART: Arquitectura de un driver para sistemas embebidos (software)

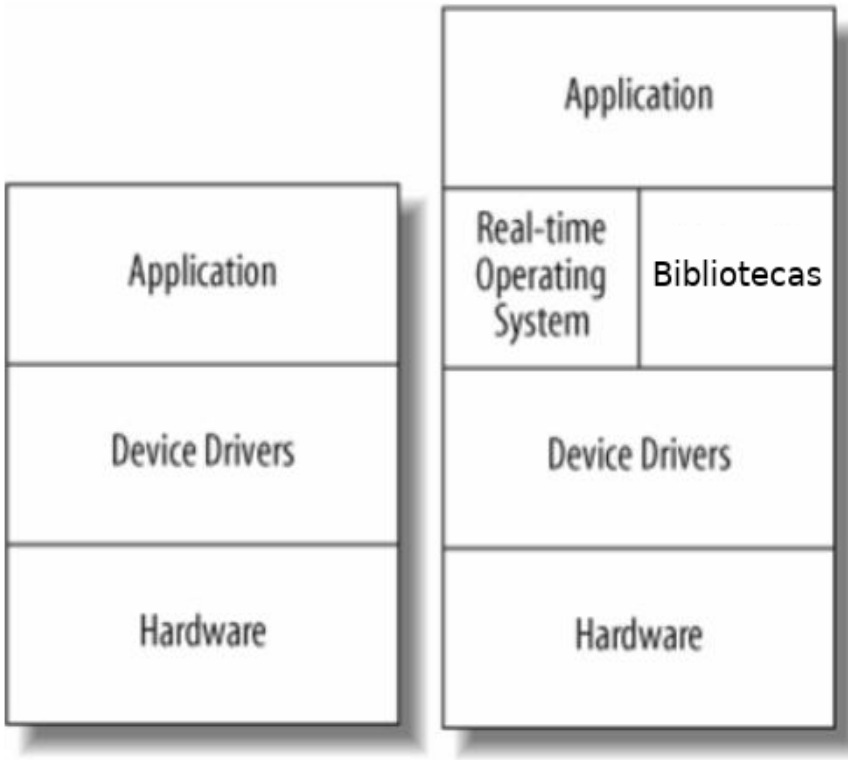
serial_uart.c
serial_uart.h

- Crear la rutina de inicialización (init)

  Ejemplo: Establecer los registros de control para una comunicación UART de 8bits, sin bit de paridad ni bit de stop. 9600 baudios/s

  ```
  void serial_init() {

      /* Escribir una rutina de inicializacion */
      /* El manual del atmega328p tiene un ejemplo. Adecuarla a C y
         la estructura de datos */

      /* Configurar los registros High y Low con BAUD_PRESCALE */

      puerto_serial->baud_rate_h = (unsigned char) (BAUD_PRESCALE>>8);
      puerto_serial->baud_rate_l = (unsigned char) (BAUD_PRESCALE);
      /* Configurar un frame de 8bits, con un bit de paridad y bit de stop */
      puerto_serial->status_control_c = (unsigned char) (INIT);

      /* Activar la recepcion y transmicion */
      puerto_serial->status_control_b = (unsigned char) (EN_RX_TX);

  }
  ```

- ¿Qué valores hay que colocar en las macros BAUD_PRESCALE, INIT e EN_RX_TX?

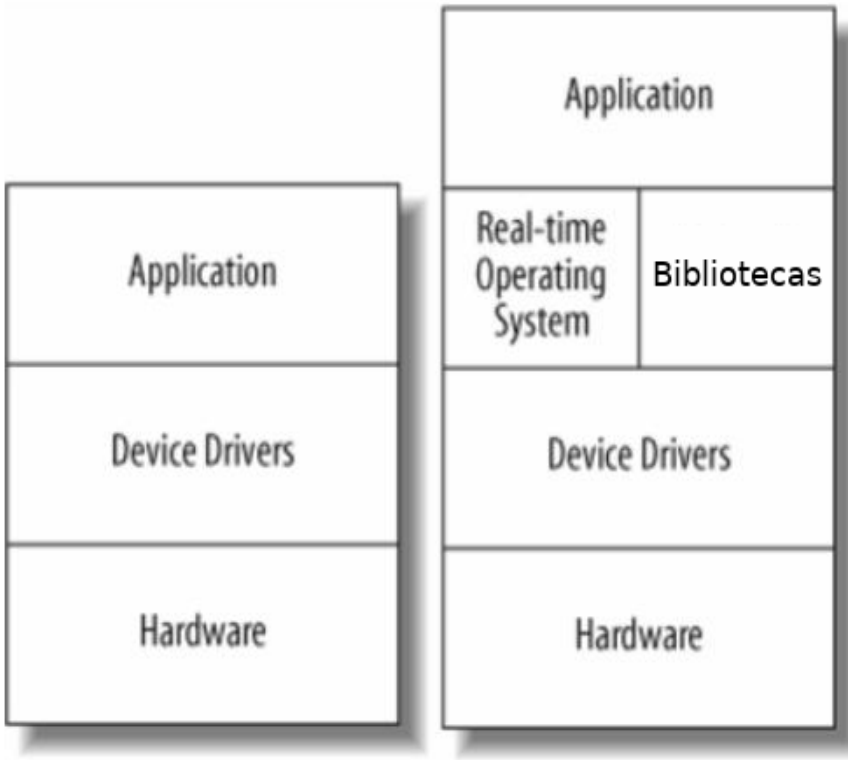


UART: Arquitectura de un driver para sistemas embebidos (software)

serial_uart.c
serial_uart.h

- Escribir las rutinas de ENTRADA y SALIDA
  - Con E/S programada
  - Con E/S via interrupciones (rutina de atención de interrupciones)

```
char serial_get_char(void)
{
   /* Wait for the next character to arrive. */
   /* Completar con E/S programada */
   while(!((puerto_serial->status_control_a) & (EN_RX)));

   /* devolver el valor que se encuentra en el registro de datos de E/S */
   return (puerto_serial->data_es);

}

char put_char(void)
{

}
```

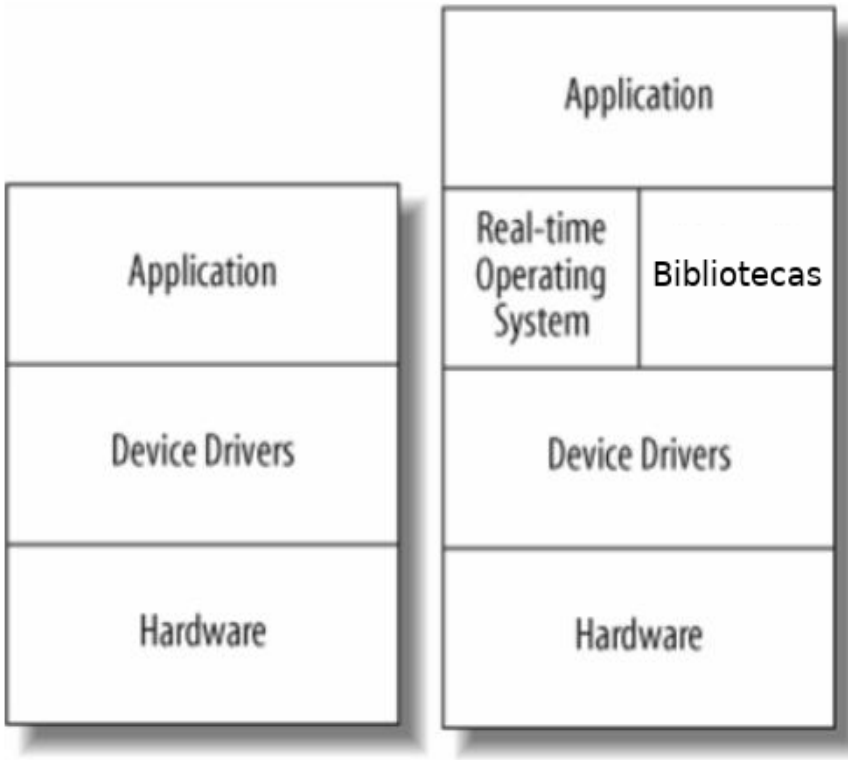


UART: Arquitectura de un driver para sistemas embebidos (software)

serial_uart.c
serial_uart.h

- Escribir una aplicación que utilice el driver de manera básica (main.c)

```
main()
{
   serial_init()

   put_char('H');
   put_char('o');
   put_char('l');
   put_char('a');
   put_char(' ');
   put_char('m');
   put_char('u');
   put_char('n');
   put_char('d');
   put_char('o');
   put_char('\n');
}
```



- Expandir el driver para enviar y recibir cadenas de caracteres y que utilicen put_char y get_char.

```
serial_put_str(char * cadena);
char * serial_get_str();
```

Bibliografía

- Programming Embedded SystemS in C and C++, Michael Barr, O'Reilly, 1999, ISBN: 1-56593-354-5
- Designing Embedded Hardware, John Catsoulis, O'Reilly, 2003, ISBN: 1-596-00362-5
- Hoja de datos atmega328p, Atmel.
- Esquemático arduino pro mini
- Apunte: Driver serial