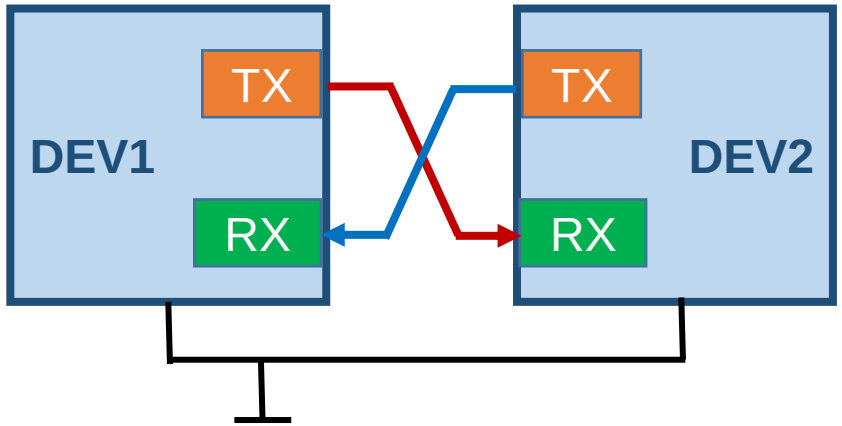
Comunicación serie asíncrona: UART

Índice

- Transmisión serie asíncrona
- UARTs en ESP32
- Librería “serial” de Arduino
- Funciones del entorno ESP-IDF
- Ejercicios

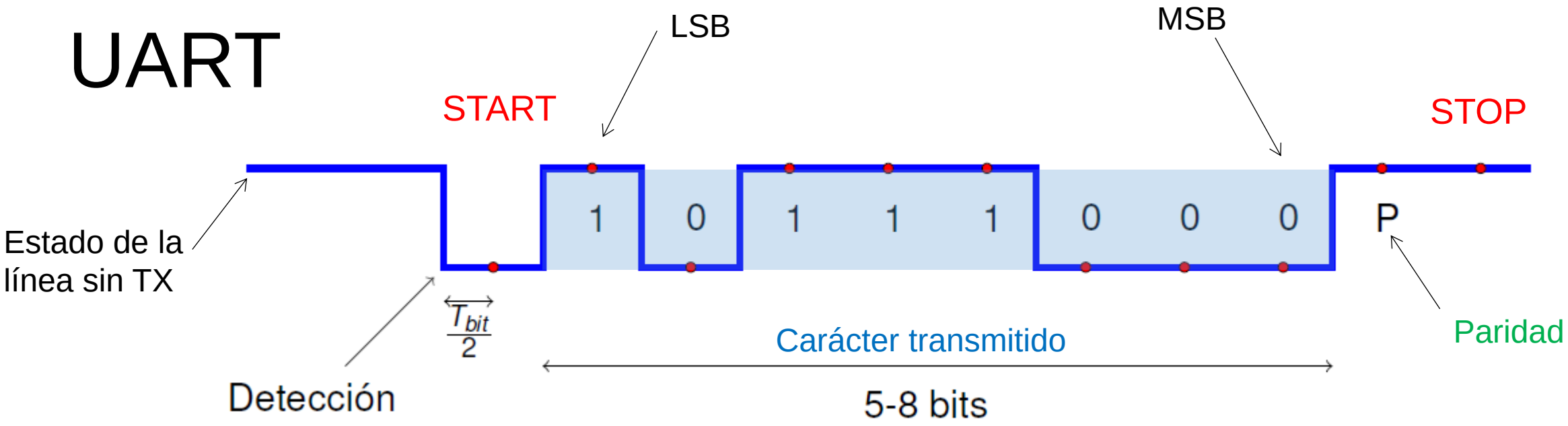
UART: Universal Asynchronous Receiver/Transmitter

- Forma simple de comunicar dos dispositivos con pocos recursos
- Orientada a la transmisión de caracteres
- Transmisión asíncrona: no requiere del envío de un reloj



- Solo requiere de 2 cables (TX y RX)
- La trama empieza con un bit de START, seguido de un carácter y finalizado con un bit de STOP
- El dispositivo TX y el RX deben de trabajar a la misma velocidad de transmisión (baudios)

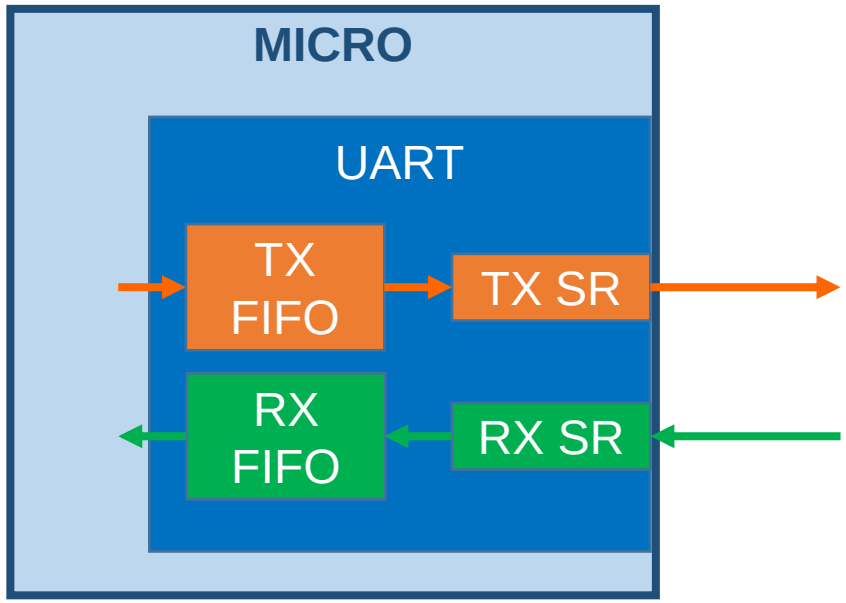
UART

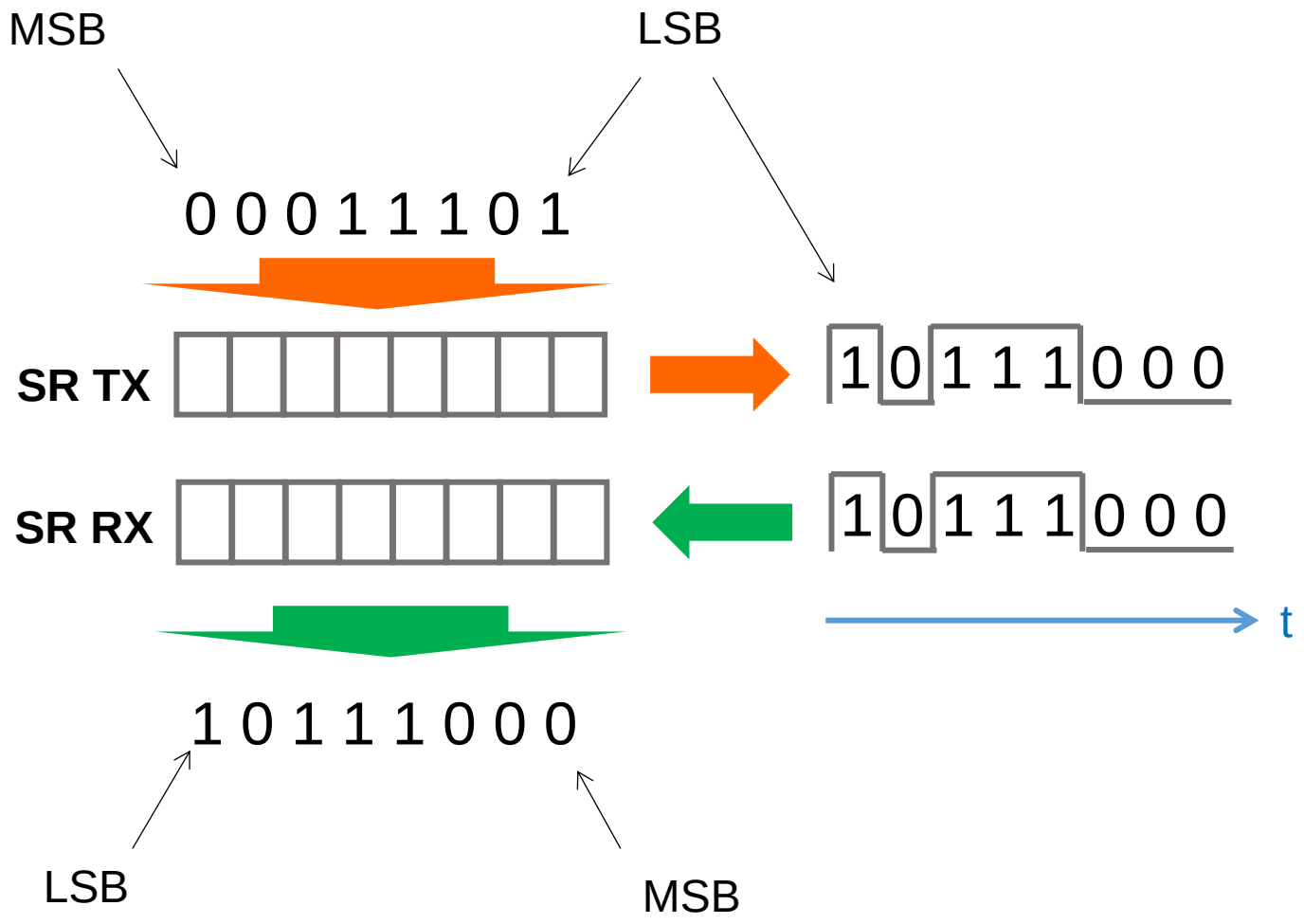


- Sin transmisión la línea está a “1”
- 1 bit de START ⇒ se utiliza para la detección de la trama
- Entre 5 y 8 bits de datos
- Puede incluirse un bit de paridad (“even” u “odd”)
- 1 o varios bits de STOP
- Velocidad de transmission en baudios = 1/Tbit

- Bits transmitidos/seg (*Throughput*) = $\frac{\text{\#bits del carácter transmitido}}{\text{Tiempo de transmisión de la trama}}$

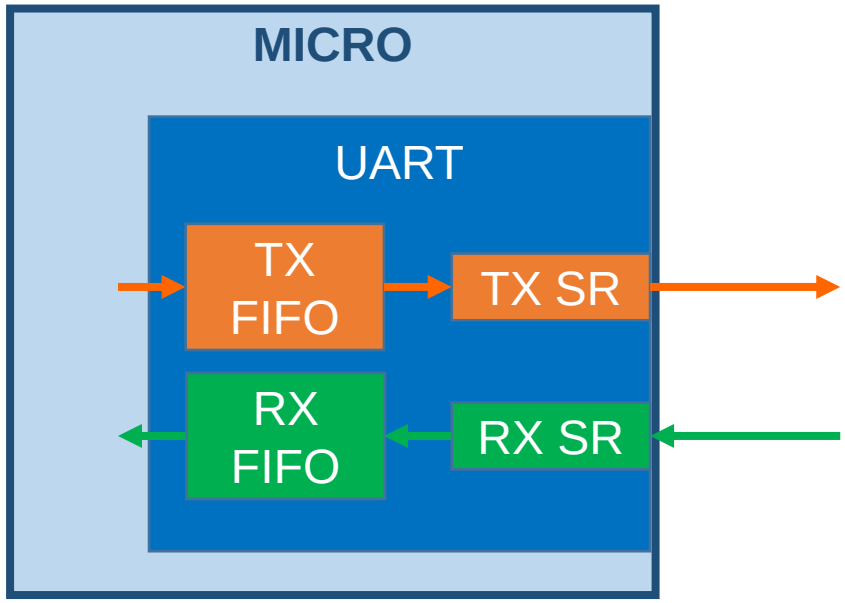
UART





SR: SHIFT REGISTER

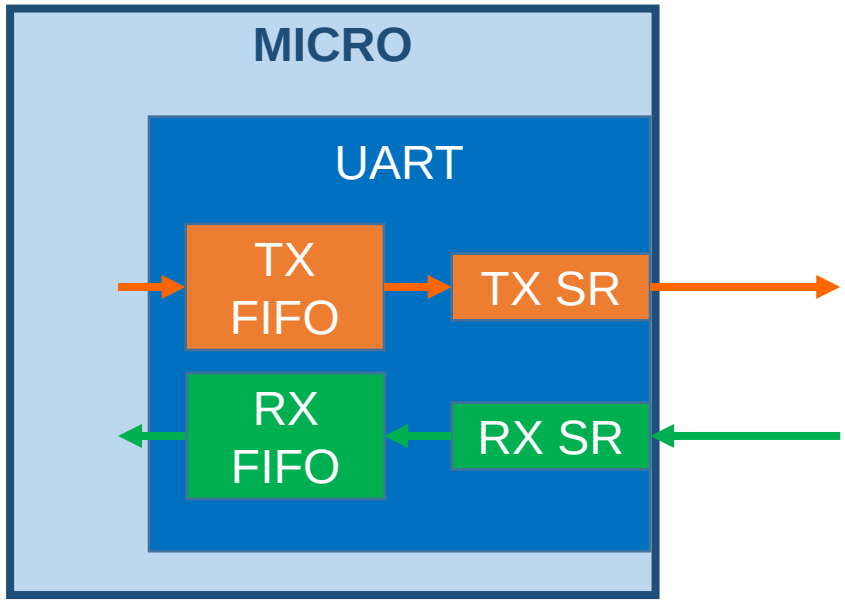
UART



Operaciones básicas:

- ¿FIFO de TX llena?
- Enviar dato a FIFO
- ¿Cuántos datos recibidos?
- Retirar dato de FIFO

UART



Control de flujo HW:

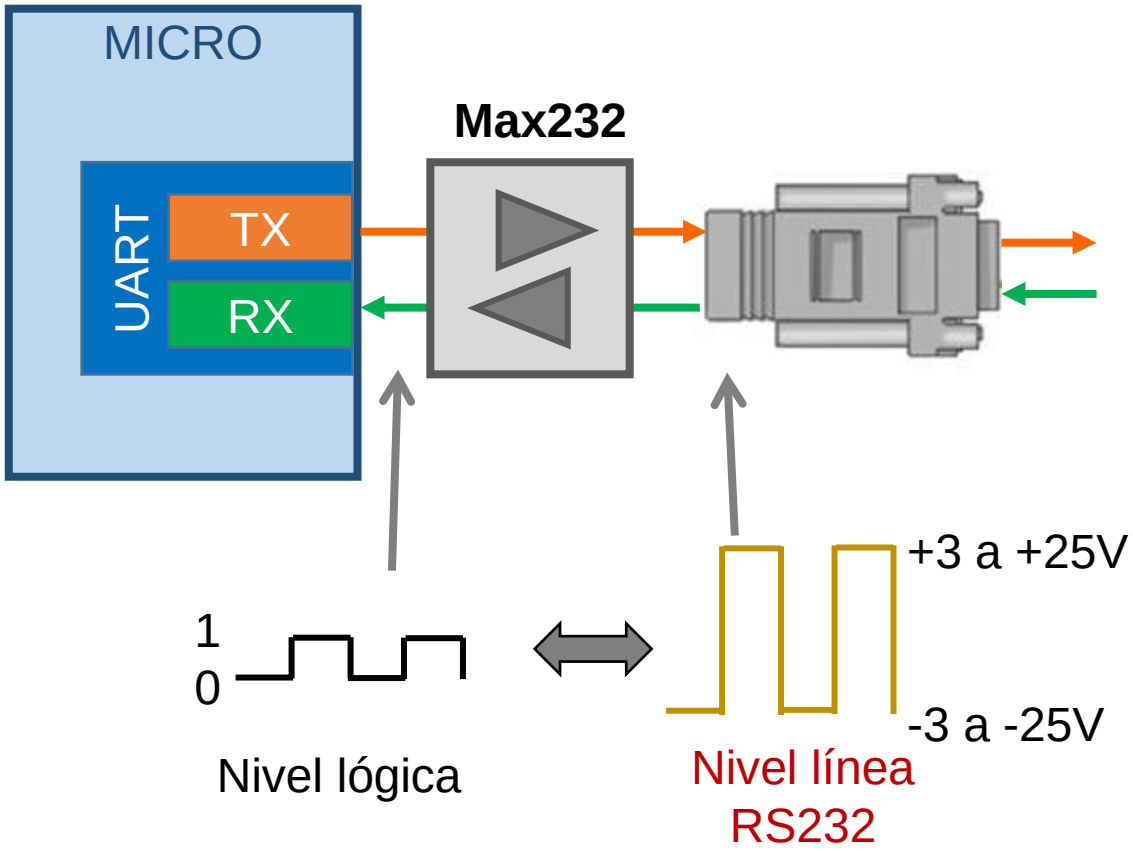
- Requiere de líneas adicionales:
  - RST (Request to send): Solicitud de envío (TX→RX)
  - CTS (Clear to send): Libre para envío (RX→TX)

Control de flujo SW:

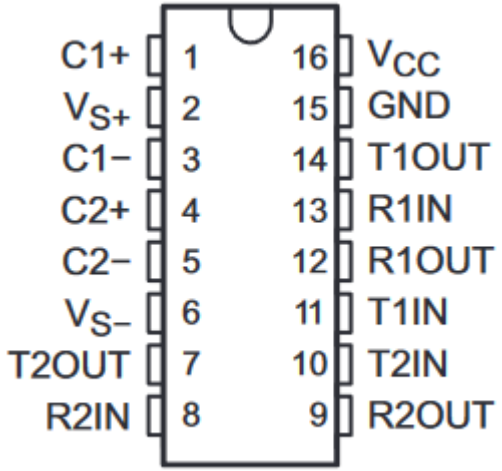
- No requiere de líneas adicionales
- RX y TX se intercambian comandos:
  - Xon: RX FIFO con espacio (ASCII 17 - DC1)
  - Xoff: FIFO llena (ASCII 19 – DC3)

UART

Estándar RS232



MAX232



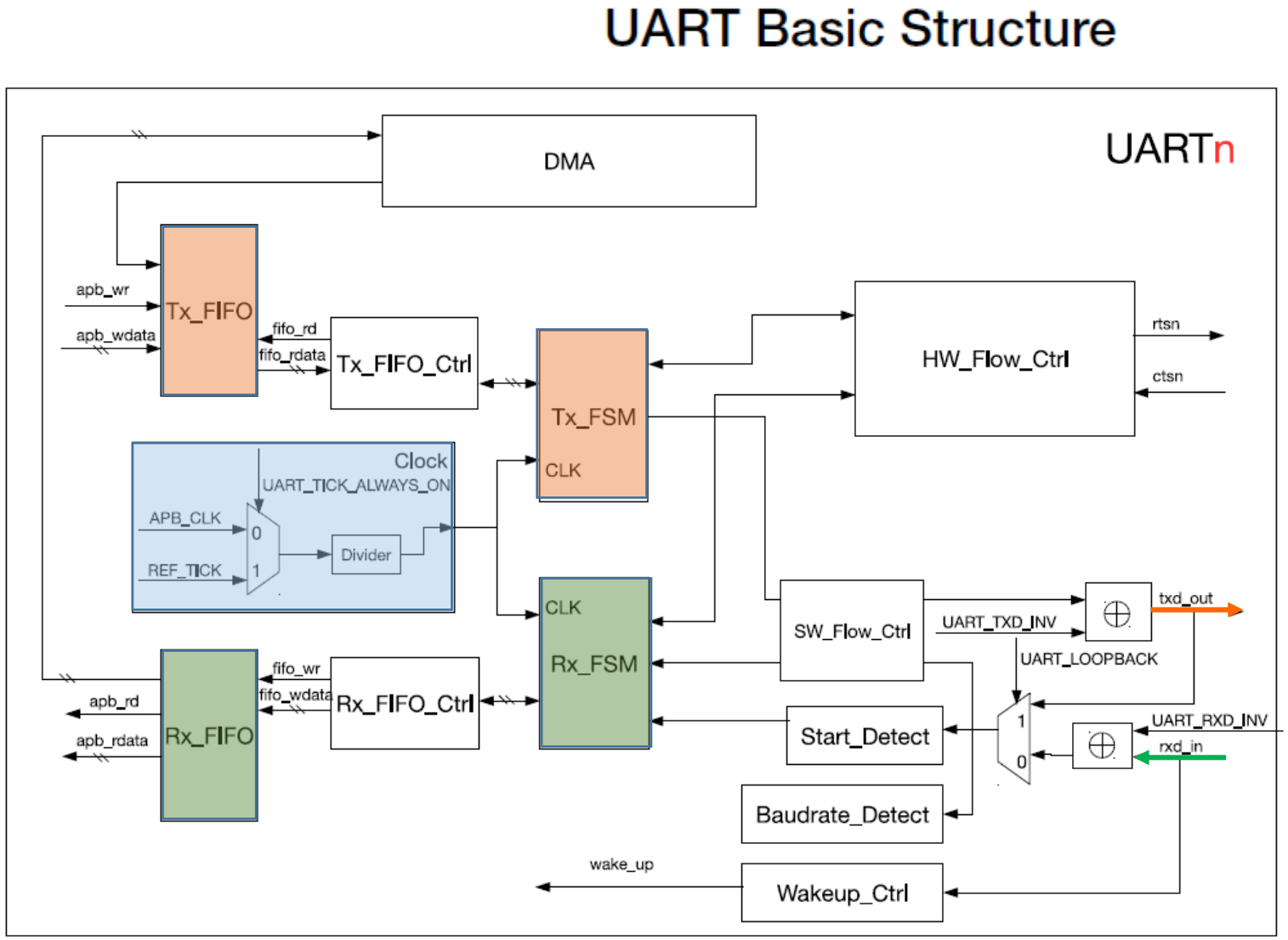
Conectores DB9

Cables USB-RS232

UARTs en ESP32

- ESP32 dispone de 3 UARTs: UART0, UART1 y UART2
- Características de las UARTs:
  - Tasa de baudios programable
  - Disponen de una TX_FIFO y RX_FIFO, comparten una RAM de 1024x8 bits
  - Configurable para TX/RX 5, 6, 7 u 8 bits
  - Bits de STOP configurables: 1, 1.5, 2, 3 o 4 bits
  - Admite bit de paridad
  - Posibilidad de despertar el procesador al recibir un dato
  - Incluye protocolos RS485, IrDA
  - Control de flujo configurable
  - DMA

UARTs en ESP32

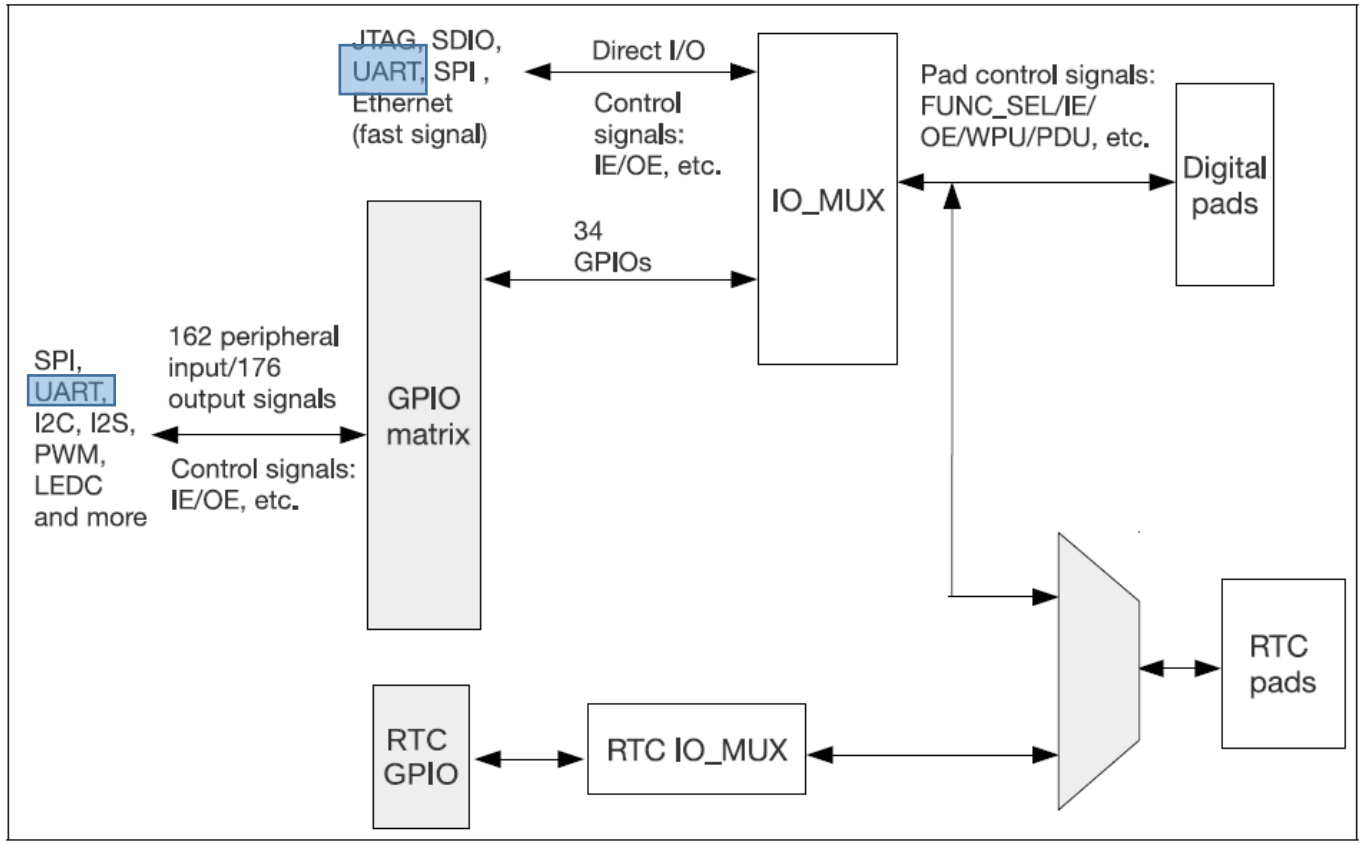


UARTs en ESP32

- Conexiones (por defecto) directas al IO_MUX

UART	RX IO	TX IO	CTS	RTS
UART0	GPIO3	GPIO1	N/A	N/A
UART1	GPIO9	GPIO10	GPIO6	GPIO11
UART2	GPIO16	GPIO17	GPIO8	GPIO7

- Conectable a cualquier pin a través de la GPIO matrix

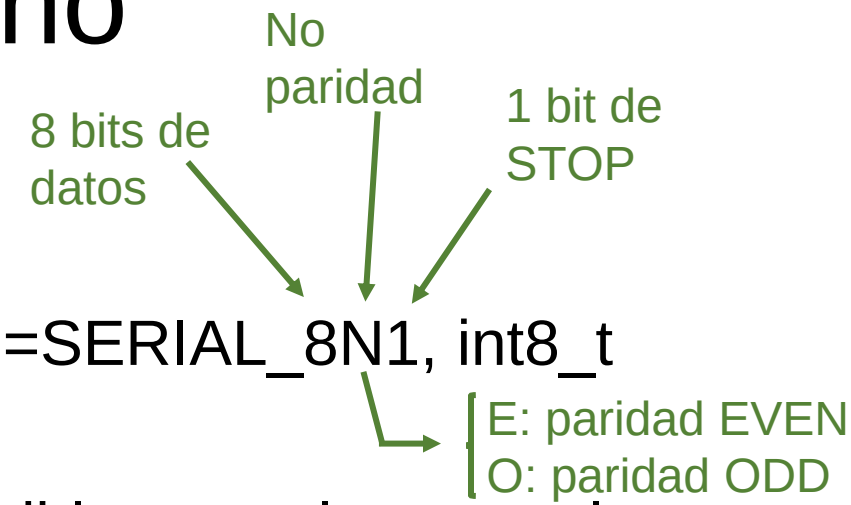


Uso de las UARTs con Arduino

- Disponible con funciones de comunicación “**Serial**”
- Hay que utilizar la clase “HardwareSerial” para declarar el objeto que identifica la UART a utilizar:
  - nombre_objeto(int uart_nr);
- Arduino tiene declarado por defecto el uso de la UART0 (conectada al micro-USB del M5Stack) y la UART1 y 2
  - **HardwareSerial Serial(0);**
  - **HardwareSerial Serial1(1);**
  - **HardwareSerial Serial2(2);**

https://www.arduino.cc/reference/en/language/functions/communication/serial/

Uso de las UARTs con Arduino



- Funciones de comunicación “**Serial**”:
  - begin(): inicializa el puerto serie
    - void begin(unsigned long baud, uint32_t config=SERIAL_8N1, int8_t rxPin=-1, int8_t txPin=-1)
  - available(): obtiene el número de caracteres disponibles para leer por el puerto serie
  - print(): envía los datos como caracteres ASCII
  - println(): envía los datos e incluye el retorno de carro (ASCII 13 o ‘\r’) y el de nueva línea (ASCII 10 o ‘\n’).
  - read(): lee el dato recibido por el puerto serie
  - end(): deshabilita el puerto y libera los pines para otros usos

https://www.arduino.cc/reference/en/language/functions/communication/serial

Uso de las UARTs con ESP-IDF

- Es necesario declarer la **librería** “driver**uart.h**”
- ESP32 dispone de 3 UARTs de libre uso
  - Nombres: UART_NUM_0, UART_NUM_1 y UART_NUM_2
- Configuración y uso de la UART:
  1. Establecer los parámetros de configuración: uart_param_config()
  2. Fijar los pines del puerto (TX, RX, …): uart_set_pin()
  3. Instalar el driver: uart_driver_install()
  4. Establecer la comunicación (leer o escribir datos): uart_write_bytes() y uart_read_bytes()
  5. (Opcional) Configurar la generación de interrupciones o eventos
  6. (Opcional) Eliminar el driver: uart_driver_delete()

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/uart.html

Uso de las UARTs con ESP-IDF

- Establecer los **parámetros de configuración**
  - Utilizar la estructura **uart_config_t**

```
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
    .rx_flow_ctrl_thresh = 120,
    .use_ref_tick = true
};
```

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/uart.html

Uso de las UARTs con ESP-IDF

- Cada UART dispone de diferentes **fuentes de interrupción**, ej:
  - UART_TX_DONE_INT: El TX ha enviado todos los datos de la FIFO
  - UART_SW_XOFF_INT: El RX ha recibido Xon char con uart_sw_flow_con_en = 1
  - UART_SW_XON_INT: El RX ha recibido Xoff char con uart_sw_flow_con_en = 1
  - UART_CTS_CHG_INT: El RX detecta un cambio en CTSn
  - UART_PARITY_ERR_INT: El RX detecta un error de paridad
  - UART_TXFIFO_EMPTY_INT: La FIFO del TX tiene menos datos que la cantidad indicada en tx_mem_cnttxfifo_cnt
  - UART_RXFIFO_FULL_INT: La FIFO tiene más datos que lo indicado en rx_flow_thrhd_h3, rx_flow_thrhd.

Esp32_reference_manual

Uso de las UARTs con ESP-IDF

- Configurar la generación de **interrupciones**
  - Habilitar o deshabilitar la generación de interrupciones: uart_enable_intr_mask(), uart_disable_intr_mask()
  - Liberar y registrar el manejador de la rutina ISR: uart_isr_free(), uart_isr_register()
  - Puesta a cero del flag de la interrupción: uart_clear_intr_status()
  - Configurar interrupción en una UART: uart_intr_config()
    - Utiliza la estructura uart_intr_config_t
- Lectura de datos de la UARTx dentro de la ISR
  - Conocer el status de las interrupciones:
    - status = UARTx.int_st.val;
  - Conocer el número bytes recibidos:
    - num_bytes_rec = UARTx.status.rxfifo_cnt;
  - Leer un byte
    - byte_leido = UARTx0.fifo.rw_byte;

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/uart.html

Uso de las UARTs con ESP-IDF

- La UART se puede utilizar para despertar al micro del “**light sleep**”
  - Cuenta los flancos positivos recibidos en el pin RX y compara la cuenta con un umbral
    - Si la cuenta > umbral ⇒ despierta al micro
  - Los bits de START y STOP también contribuyen al conteo de flancos
    - Ej: enviando la letra ‘a’, ASCII 97(010001101) despierta al micro con el umbral = 3
  - Función para establecer el umbral: el manejador de la rutina ISR: uart_set_wakeup_threshold(uart_port_t uart_num, int wakeup_threshold))
  - Hay que seleccionar REF_TICK como fuente de reloj de la UART:
    - use_ref_tick=true en la configuración uart_config_t
  - Funciones de la librería "esp_sleep.h"
    - Poner el micro en light sleep: esp_light_sleep_start()
    - Habilitar el “wake-up” de la UART: esp_sleep_enable_uart_wakeup(int uart_num)

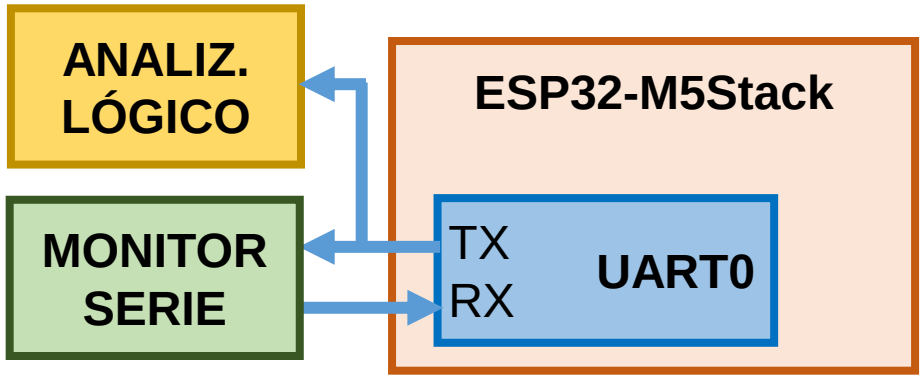
https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/uart.html

Ejercicio 1

Utilizar el analizador lógico conectado al pin TX de la UART0 para visualizar las tramas serie que se generan con el siguiente código:

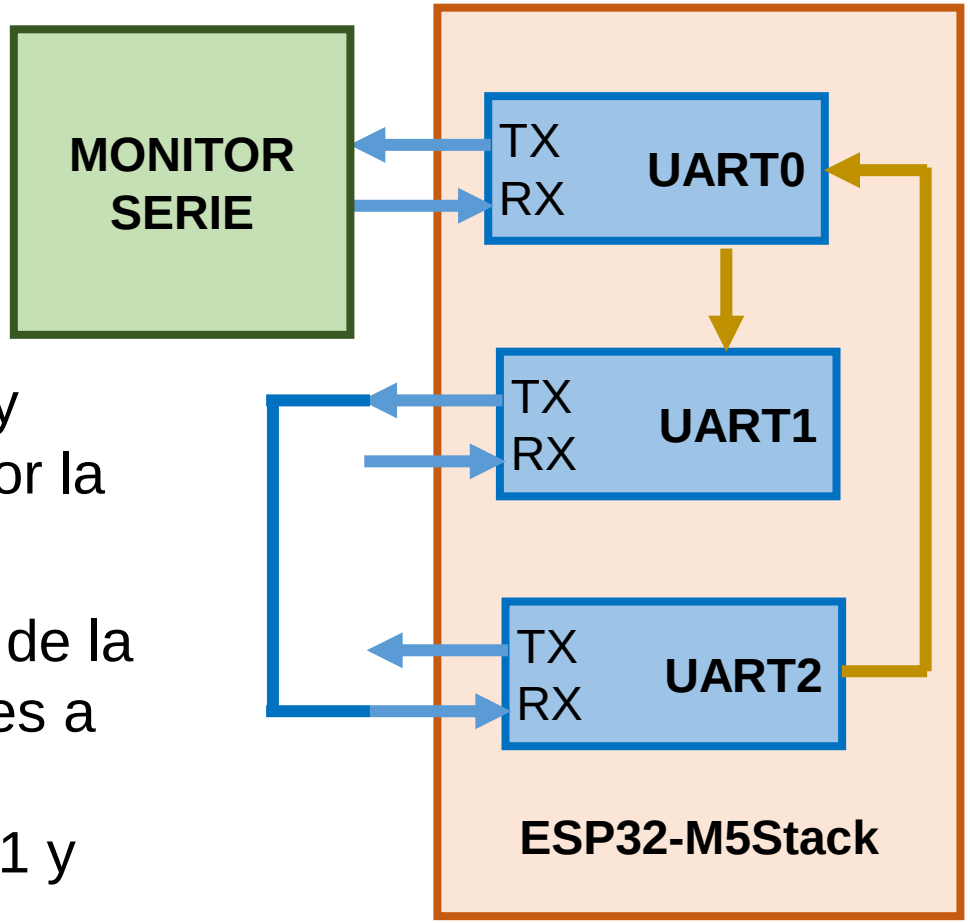
```
void loop() {

int A =135;
  if (Serial.available()>0)
  {
    char ch = Serial.read();
    if(ch=='1') { Serial.print("A"); }
    if(ch=='2') { Serial.println("Hola"); }
    if(ch=='3') { Serial.print(A); }
    if(ch=='4') { Serial.print(A,0); }
    if(ch=='5') { Serial.print(A,BIN); }
  }
}
```



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Ejercicio 2



Realizar un programa que nos permita enviar por la UART1 la cadena de caracteres recibida por la UART0 y enviar por la UART0 la cadena de caracteres recibida por la UART2.

Se conectará la salida TX de la UART1 a la entrada RX de la UART2, de esa forma al enviar una cadena de caracteres a través del monitor serie, esta cadena se escribirá en el monitor serie después de ser retransmitida por la UART1 y recibida por la UART2.

Configurar la transmisión a 115200 baudios de 1 byte con 1 bit de START y uno de STOP.

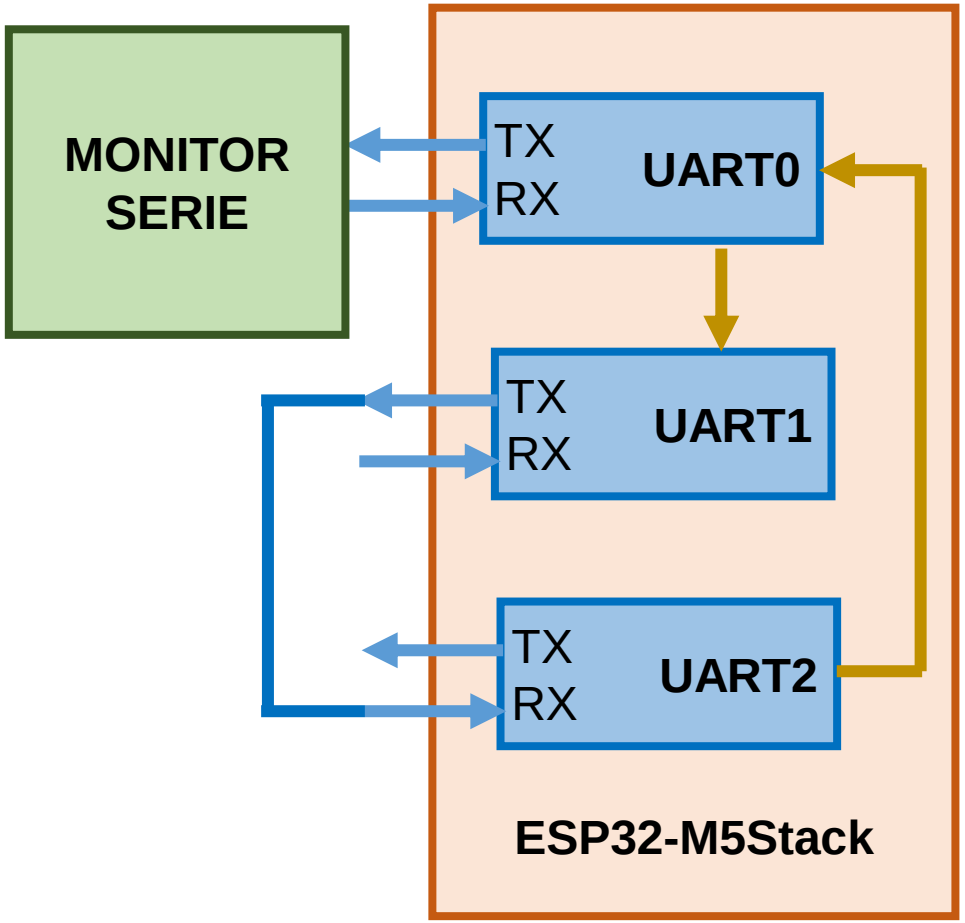
- Pines UART1: TX 21 RX -1, Pines UART2: TX -1 RX 22
- Utilizar las funciones: available(), read() y print()

Ejercicio 3

Completar el código Ejerc_3.ino para repetir el ejercicio 2 utilizando las funciones del ESP32 IDF para configurar y comunicar las 3 UARTs siguiendo el esquema.

Utilizar las funciones:

- uart_param_config()
- uart_set_pin()
- uart_driver_install()
- uart_read_bytes()
- uart_write_bytes()



Ejercicio 4

Realizar un programa que reciba desde el monitor serie los caracteres correspondientes a un número decimal de varios dígitos, obtenga su valor decimal y lo muestre por el monitor serie.

Ejercicio 5

Realizar un programa que detecte el envío desde el monitor serie del carácter ‘A’ o ‘D’ y a continuación su valor decimal y le asigne dicho valor la variable “addr” o “data” según se haya detectado ‘A’ o ‘D’, respectivamente. Mostrar el valor asignado a “addr” o “data” por el monitor serie.

Incluir mensajes de error que indique si el carácter recibido no es válido si es distinto de ‘A’ o ‘D’ o de si el valor decimal de “addr” o “dato” es mayor de 255.

Referencias

- **Tutorial Serial Communication:** https://learn.sparkfun.com/tutorials/serial-communication
- **Tutorial Comunicación serie Arduino:** https://aprendiendoarduino.wordpress.com/tag/rs232/
- **UART-ESP32-IDF tutorial:** http://www.lucadentella.it/en/2017/11/06/esp32-26-uart/
- **Librería Serial de Arduino:** https://www.arduino.cc/reference/en/language/functions/communication/serial/
- **Arduino Tutorial: Comunicación serie con Arduino,** https://aprendiendoarduino.wordpress.com/2016/07/02/comunicacion-serie-arduino/
- **Kolban's book on ESP32**
- **ESP-IDF Programming Guide, API Reference\Peripherals\UART:** https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/uart.html

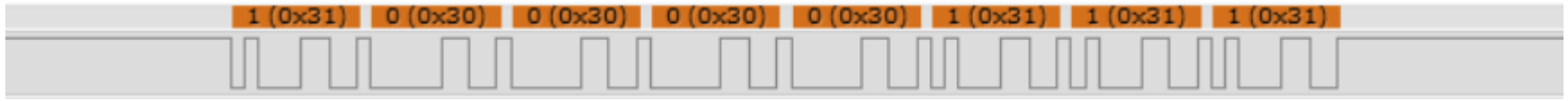
Serial.print()

```
int A = 335;
```

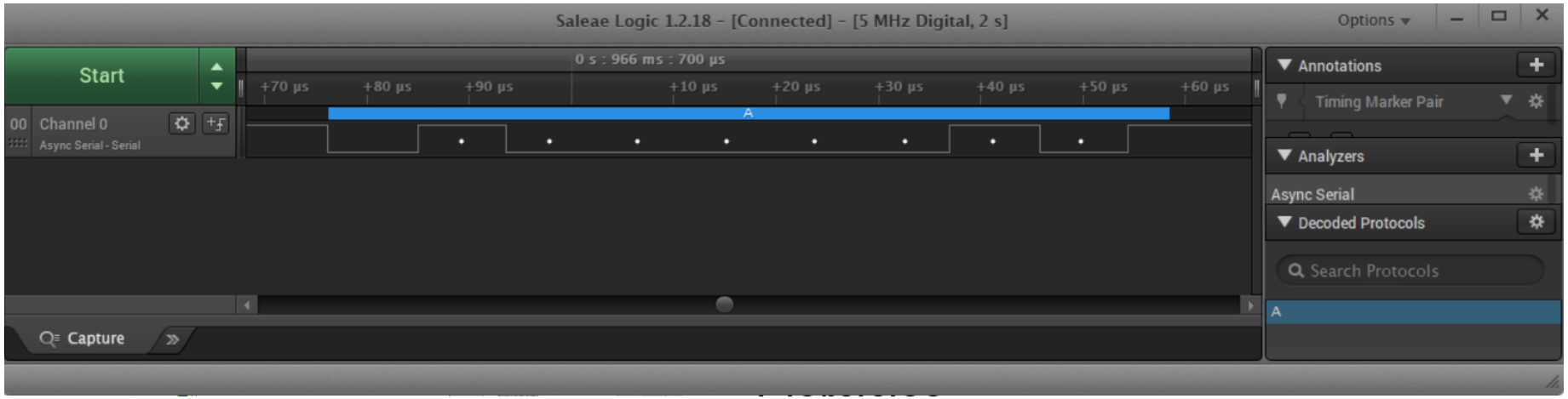
Serial.print(A)

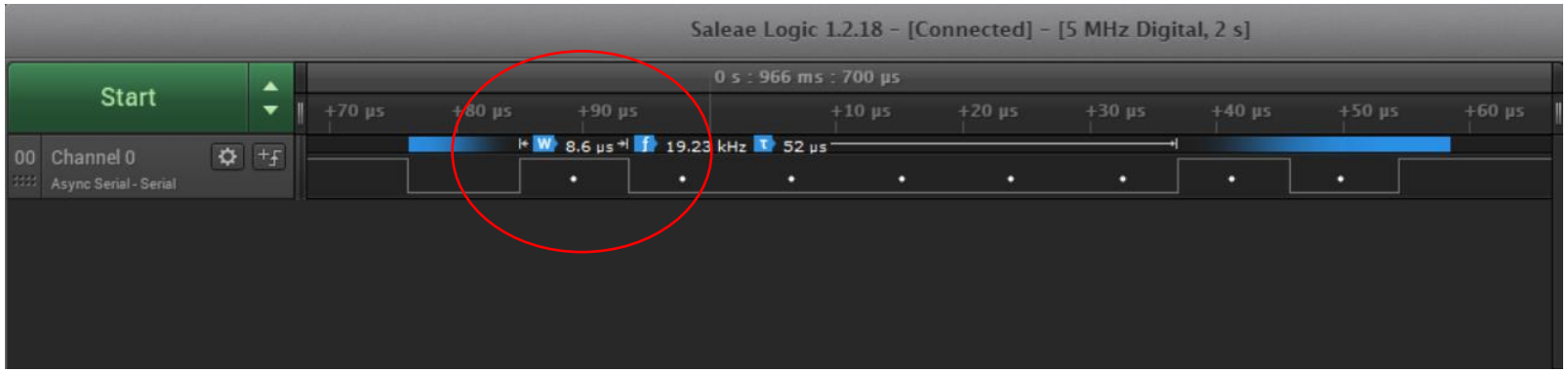
Serial.print(A,0)

Serial.print(A,BIN)

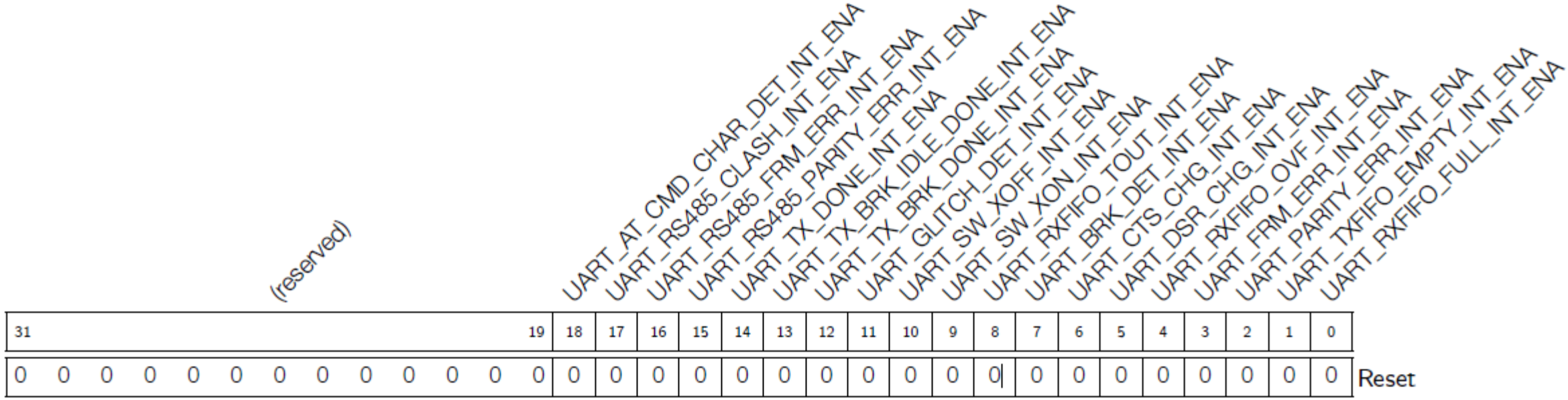


Ejercicio 1


Saleae Logic 1.2.18 - [Connected] - [5 MHz Digital, 2 s]
Options
Start
0 s : 966 ms : 700 µs
+70 µs
+80 µs
+90 µs
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
+60 µs
00 Channel 0
Async Serial - Serial
A
Annotations
Timing Marker Pair
Analyzers
Async Serial
Decoded Protocols
Search Protocols
A
Capture

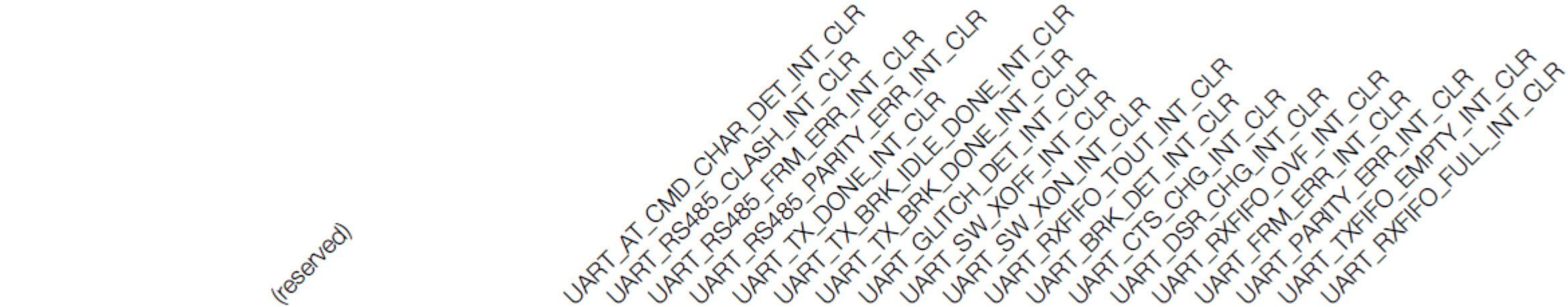

Saleae Logic 1.2.18 - [Connected] - [5 MHz Digital, 2 s]
Start
0 s : 966 ms : 700 µs
+70 µs
+80 µs
+90 µs
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
+60 µs
00 Channel 0
Async Serial - Serial
W 8.6 µs
f 19.23 kHz
t 52 µs

Register 13.4: UART_INT_ENA_REG (0xC)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(reserved)													UART_AT_CMD_CHAR_DET_INT_ENA	UART_RS485_CLASH_INT_ENA	UART_RS485_FRM_ERR_INT_ENA	UART_RS485_PARITY_ERR_INT_ENA	UART_TX_DONE_INT_ENA	UART_TX_BRK_IDLE_DONE_INT_ENA	UART_TX_BRK_DONE_INT_ENA	UART_GLITCH_DET_INT_ENA	UART_SW_XOFF_INT_ENA	UART_SW_XON_INT_ENA	UART_RXFIFO_TOUT_INT_ENA	UART_BRK_DET_INT_ENA	UART_CTS_CHG_INT_ENA	UART_DSR_CHG_INT_ENA	UART_RXFIFO_OVF_INT_ENA	UART_FRM_ERR_INT_ENA	UART_PARITY_ERR_INT_ENA	UART_TXFIFO_EMPTY_INT_ENA	UART_RXFIFO_FULL_INT_ENA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																															

Register 13.5: UART_INT_CLR_REG (0x10)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(reserved)													UART_AT_CMD_CHAR_DET_INT_CLR	UART_RS485_CLASH_INT_CLR	UART_RS485_FRM_ERR_INT_CLR	UART_RS485_PARITY_ERR_INT_CLR	UART_TX_DONE_INT_CLR	UART_TX_BRK_IDLE_DONE_INT_CLR	UART_TX_BRK_DONE_INT_CLR	UART_GLITCH_DET_INT_CLR	UART_SW_XOFF_INT_CLR	UART_SW_XON_INT_CLR	UART_RXFIFO_TOUT_INT_CLR	UART_BRK_DET_INT_CLR	UART_CTS_CHG_INT_CLR	UART_DSR_CHG_INT_CLR	UART_RXFIFO_OVF_INT_CLR	UART_FRM_ERR_INT_CLR	UART_PARITY_ERR_INT_CLR	UART_TXFIFO_EMPTY_INT_CLR	UART_RXFIFO_FULL_INT_CLR
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																															