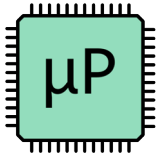
Serial Interfaces: Part 2

Lecture 9

Microprocessor-based Systems (E155)

Prof. Josh Brake



Outline

- Universal Synchronous/Asynchronous Receiver Transmitter
  - Description
  - Activity

Learning Objectives

By the end of this lecture you will be able to

- Explain the tradeoffs between synchronous and asynchronous serial interfaces
- Develop a library for the UART peripheral on the STM32F401RE
- Verify the output using the logic analyzer on the ADALM2000

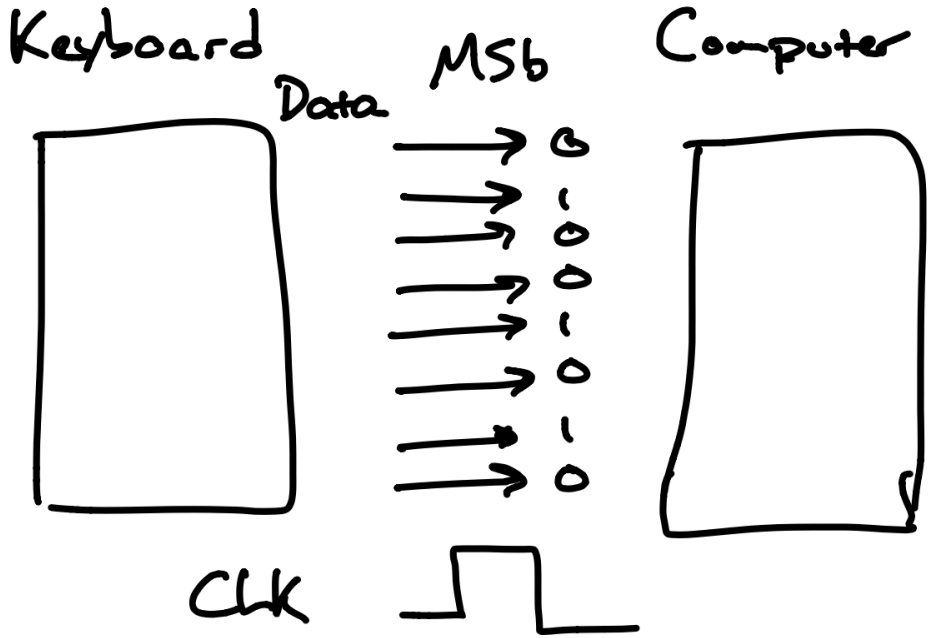
Serial Interfaces Overview

Motivation

- How can we interface a peripheral?

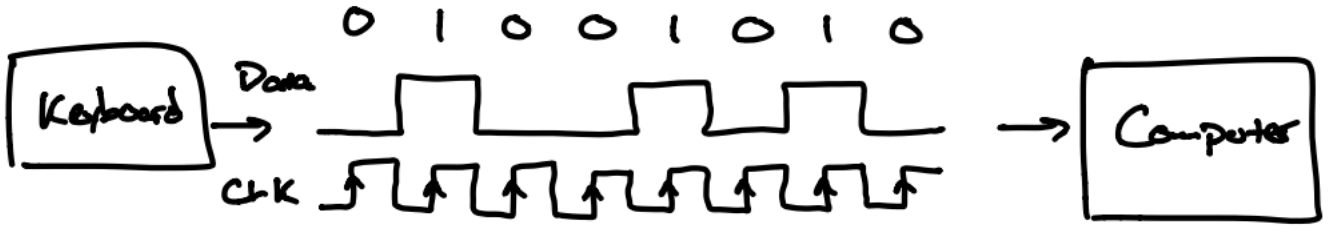
Imagine transmitting a character on a keyboard.

Capital J in ASCII is $74_{10} = 01001010_2$



What if we repackage data in a stream?

- Essential multiplexing in time
- To send N bits, we only need 2 lines (CLK + Data) instead of 9
- Price we pay is time – but often worth it.



What if we don’t want a shared clock?

- Agree on shared data rate
- Sample the incoming data stream at higher frequency to synchronize the input data stream with the reading circuitry
- Add additional bits at the beginning and end of the transmission to signal the bounds of the transmission

Question

What are some downsides of an asynchronous serial interface as compared to a synchronous one?

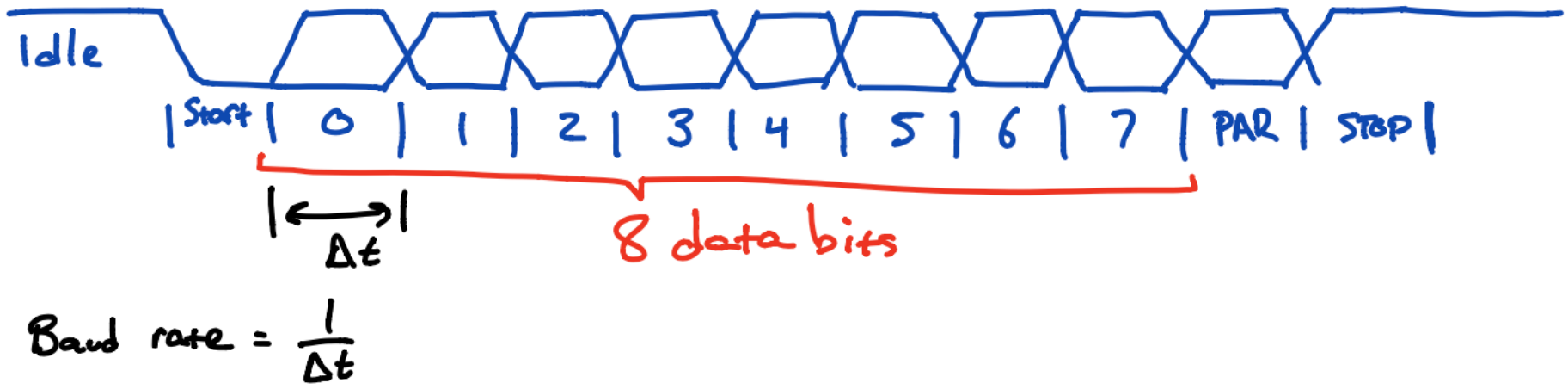
Answer

- Reduced maximum transmission frequency (typically 8x-16x overhead from sampling)
- Wasted bits in each transmission

Universal Synchronous/Asynchronous Receiver Transmitter (USART)

USART Data Frame

- 4 components
  1. Start bit: always logical 0
  2. Data bits: 5-9 bits of data
  3. Parity bit: Option bit with parity of data (i.e., even or odd. Simple error checking)
  4. Stop bit(s): 1-2 bits. Always logical 1.



STM32 USART

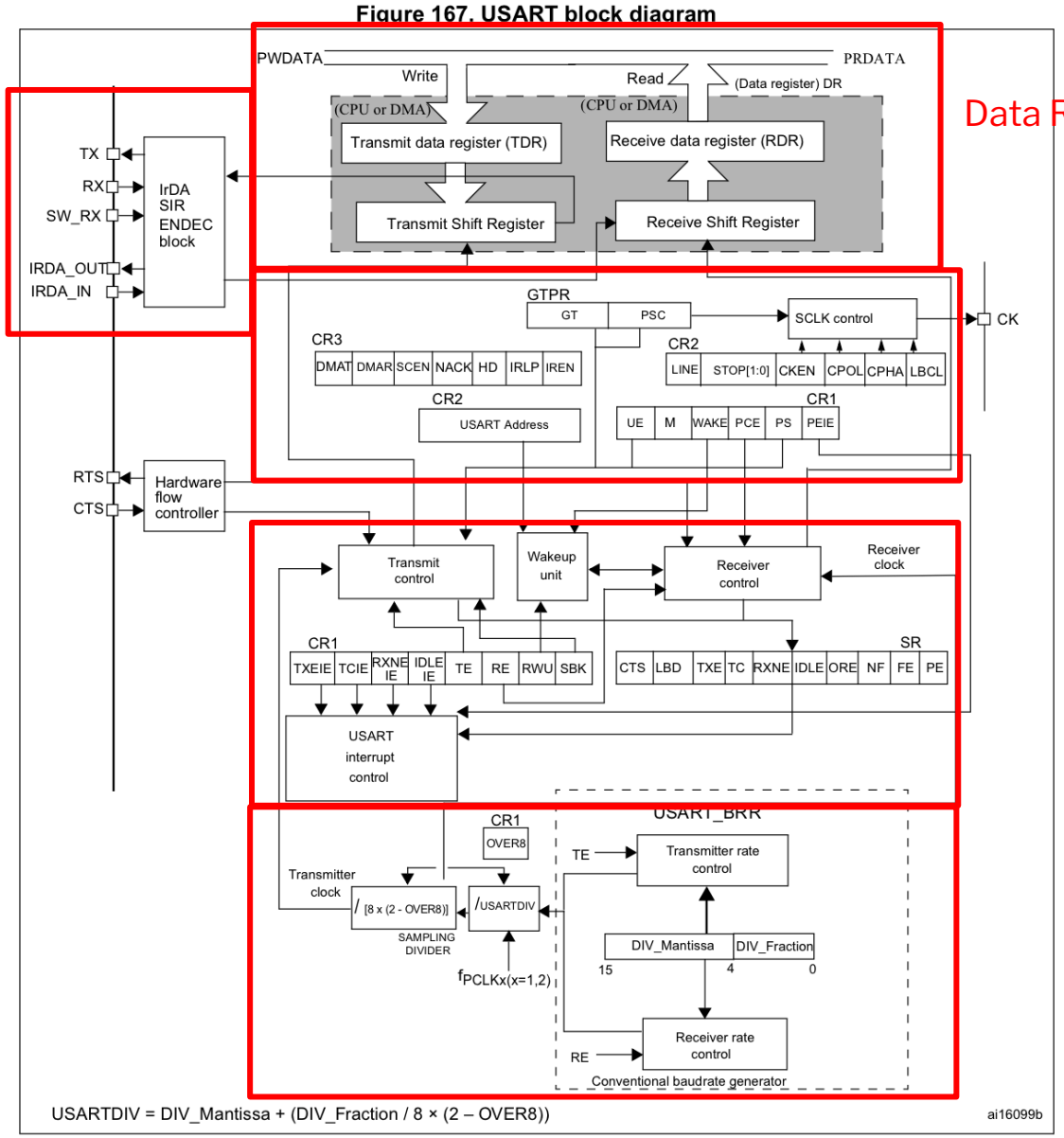
I/O Interface

Data Registers

Configuration

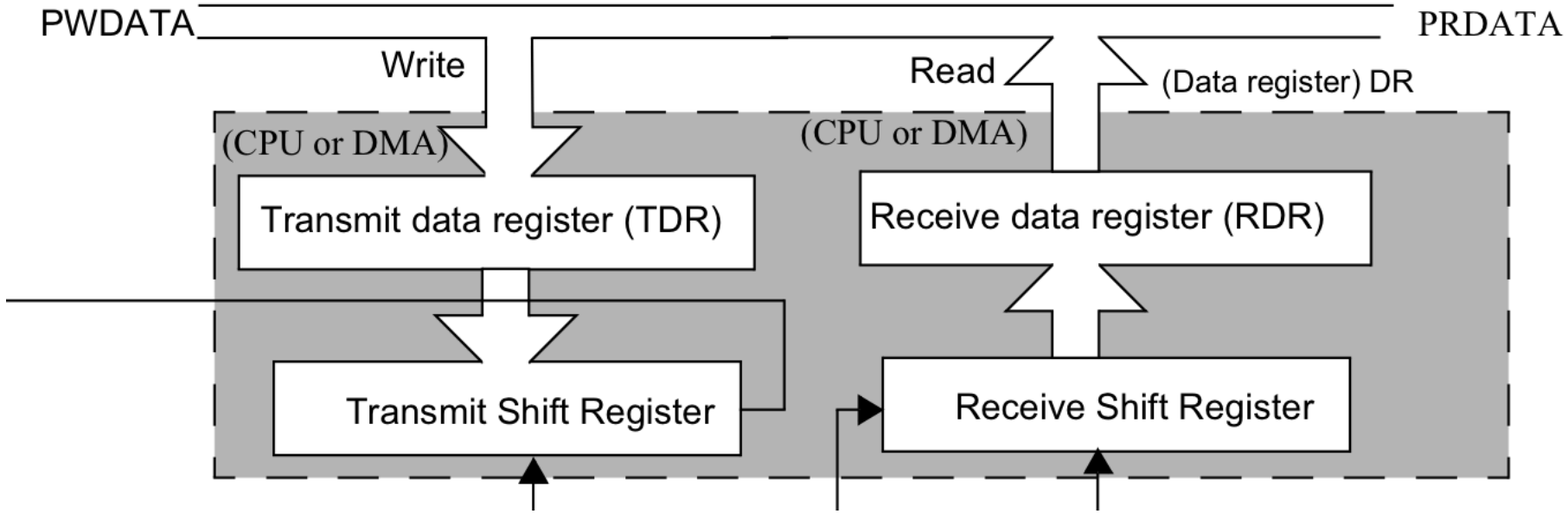
Transmit/Receive Control

Synchronous Clock Config



STM32F401RE Reference Manual p. 509

Data Registers

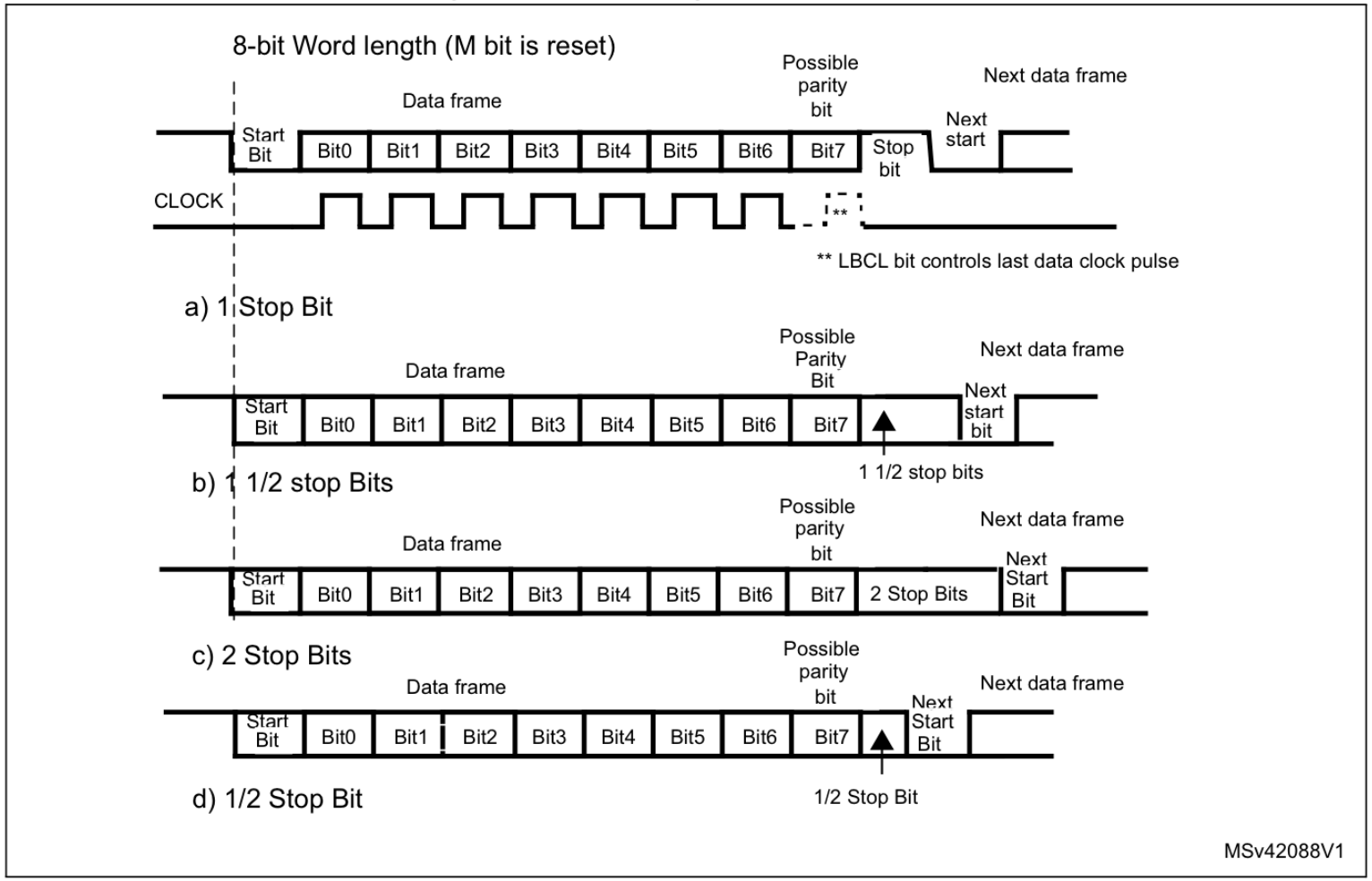


Pins

- TX – transmitted data out from USART
- RX – received data in to USART
- CK – (optional) clock output for synchronous mode
- RTS – Request To Send indicates the USART is ready to receive data (when low)
- CTS – Clear To Send block data transmission at the end of the current transfer when high

Data framing

Figure 169. Configurable stop bits



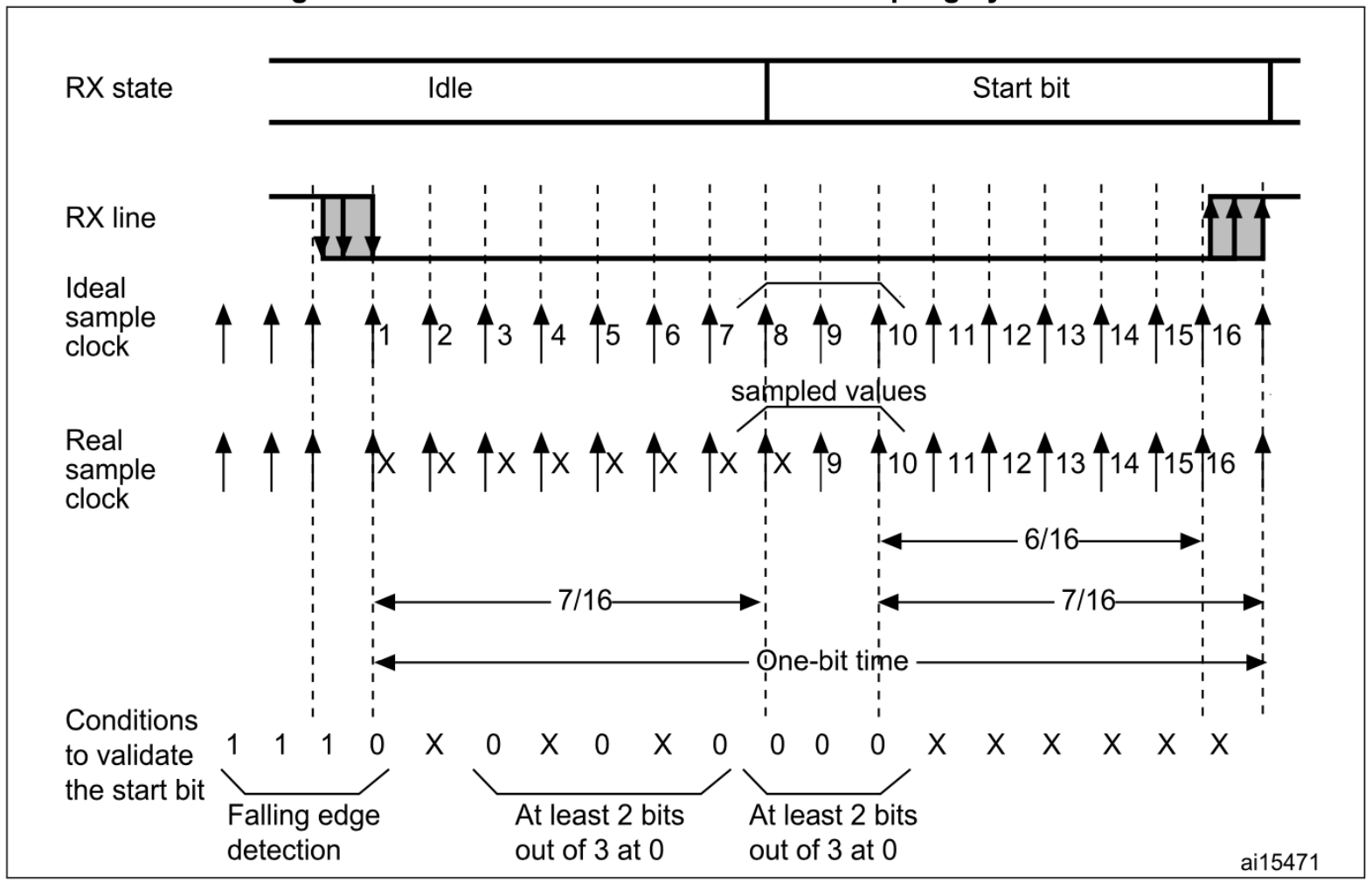
Reference Manual p. 512

Error Flags

- Overrun – new byte in the holding reg before the old one was read out
- Frame – didn’t get the stop bit(s) we expected
- Parity – calculated parity doesn’t match parity bit

Receiver

Figure 171. Start bit detection when oversampling by 16 or 8



Reference Manual p. 514

USART registers: Status Register

- UART Status Register
  - TXE – transmit data register empty (0 if data is not transferred to the shift register, 1 if it is)
  - TC – transmission complete flag
  - RXNE – read data register not empty (0 if data has not been received, 1 if it is ready to be read)
  - FE – framing error
  - PE – parity error

19.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

USART registers: Data Register

- Used for both reads and writes
- Max 9-bit data value DR[8:0]

USART registers: Baud Rate Register

19.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

USART registers: Control register 1

- M: word length 8 or 9 data bits
- PCE: parity control enable
- TE: transmitter enable
- RE: receiver enable

19.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

USART registers: Control register 2

- STOP: 2 bit field, number of stop bits (0.5, 1, or 2)
- Various clock control (if using in synchronous mode)

19.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Character Reception

1. Enable the USART with UE=1 in USART_CR1
2. Program the M bit in USART_CR1 to define word length
3. Program the number of stop bits in USART_CR2
4. (optional): Enable DMA
5. Select the desired baud rate in USART_BRR
6. Set the RE bit in USART_CR1

Wait for RXNE bit to go from 0 (no data received) to 1 (data received).
Then you can read out the data from the data register

USART Activity

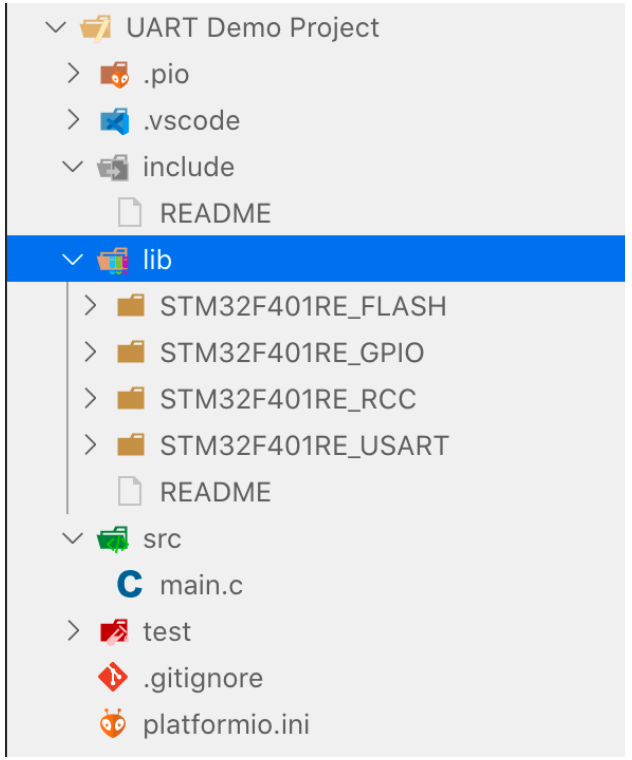
Activity

- Configure the USART as an UART to transmit serial data
  - Read user manual and develop a bullet list outline of how to configure the peripheral
  - Write USART library
    - Finish STM32F401RE_USART.h and STM32F401RE_USART.c.
  - Configure in common 8N1 mode
    - 8 data bits
    - No parity bit
    - 1 stop bit
    - Operate at 9600 baud (9.6 Kbps)
    - Assume clock is configured at 84 MHz with flash latency also configured
  - Create simple main function to transmit a string of your choice over the UART and visualize it on the ADALM2000 logic analyzer.

Code available on course Github: https://github.com/joshbrake/E155_FA2020/tree/master/L09/demo_src

Setup

- Create new project in PlatformIO (choose CMSIS for platform)
- Copy files from Github into the project
  - Overwrite `platformio.ini` file
  - Put `main.c` in `src/`
  - Put libs in lib subfolder. These are linked in with the `lib_dir` environment variable in `platformio.ini`




```
[env:genericSTM32F401RE]
platform = ststm32
board = genericSTM32F401RE
framework = cmsis
lib_dir = "./lib"
debug_tool = stlink
upload_protocol = stlink
```

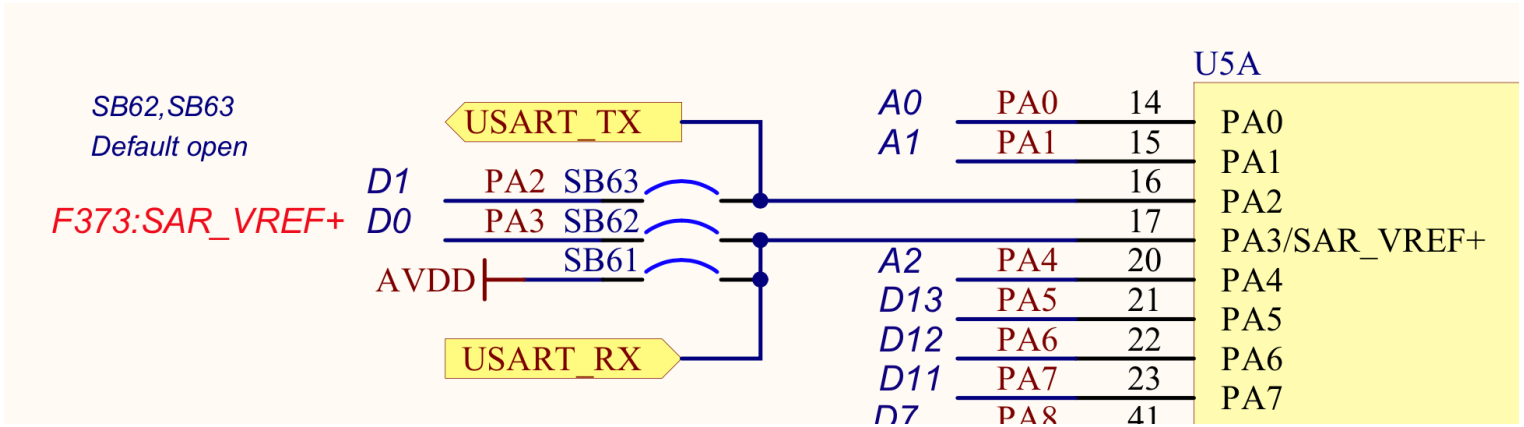
USART2 Wiring on Nucleo-64

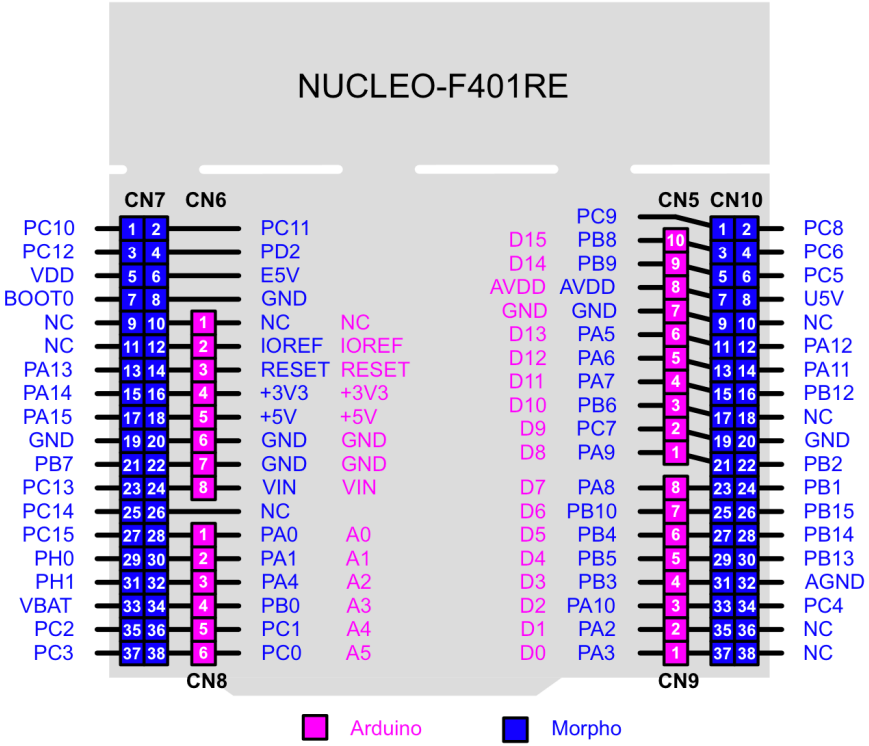
6.8 USART communication

The USART2 interface available on PA2 and PA3 of the STM32 microcontroller can be connected to ST-LINK MCU, ST morpho connector or to Arduino connector. The choice can be changed by setting the related solder bridges. By default the USART2 communication between the target STM32 and ST-LINK MCU is enabled, in order to support virtual COM port for Mbed™ (SB13 and SB14 ON, SB62 and SB63 OFF). If the communication between the target STM32 PA2 (D1) or PA3 (D0) and shield or extension board is required, SB62 and SB63 should be ON, SB13 and SB14 should be OFF. In such case it is possible to connect another USART to ST-LINK MCU using flying wires between ST morpho connector and CN3. For instance on NUCLEO-F103RB it is possible to use USART3 available on PC10 (TX) and PC11 (RX). Two flying wires need to be connected as follow:

- PC10 (USART3_TX) available on CN7 pin 1 to CN3 pin RX
- PC11 (USART3_RX) available on CN7 pin 2 to CN3 pin TX

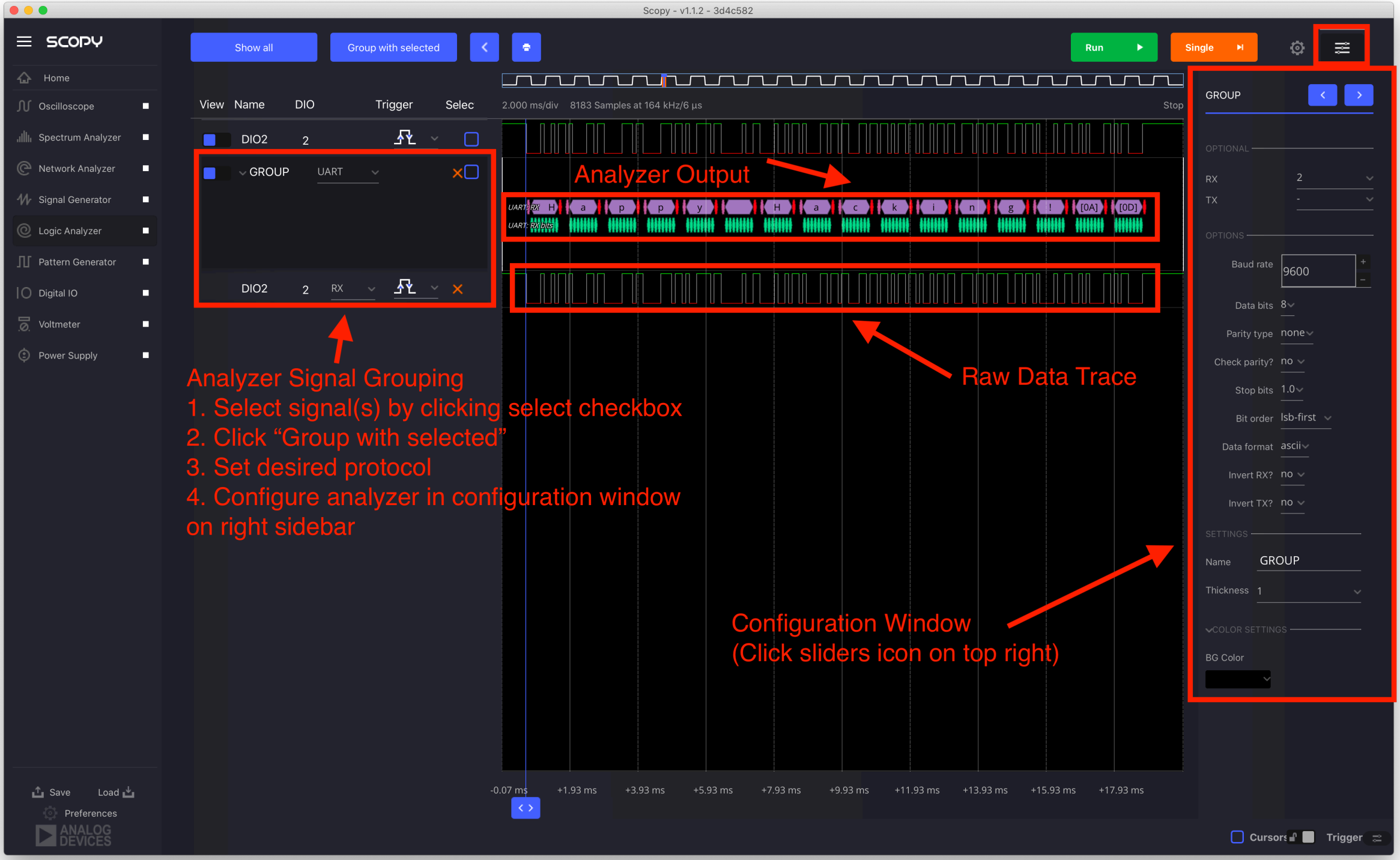
USART2 Wiring





Bonus: Receiving Serial Input over USB

- Windows: Download Putty or your favorite serial terminal
  - Select appropriate serial port (`COM<X>`; check device manager to find X)
- Mac/Linux: Can use `screen`
  - To connect to a serial port with screen, type:
    `screen <teletype_device> <baud_rate>`
  - `<teletype_device>` usually something like `/dev/tty.usbmodem144303` (can use `ls /dev/tty*` to list available options)


Scopy - v1.1.2 - 3d4c582
SCOPY
Home
Oscilloscope
Spectrum Analyzer
Network Analyzer
Signal Generator
Logic Analyzer
Pattern Generator
Digital IO
Voltmeter
Power Supply
Save
Load
Preferences
ANALOG DEVICES
Show all
Group with selected
Run
Single
View Name DIO Trigger Selec
DIO2 2
GROUP UART
DIO2 2 RX
2.000 ms/div 8183 Samples at 164 kHz/6 µs
Stop
Analyzer Output
UART: RX
UART: RX BITS
H a p p y H a c k i n g ! [0A] [0D]
Raw Data Trace
Analyzer Signal Grouping
1. Select signal(s) by clicking select checkbox
2. Click “Group with selected”
3. Set desired protocol
4. Configure analyzer in configuration window
on right sidebar
Configuration Window
(Click sliders icon on top right)
-0.07 ms +1.93 ms +3.93 ms +5.93 ms +7.93 ms +9.93 ms +11.93 ms +13.93 ms +15.93 ms +17.93 ms
GROUP
OPTIONAL
RX 2
TX -
OPTIONS
Baud rate 9600
Data bits 8
Parity type none
Check parity? no
Stop bits 1.0
Bit order lsb-first
Data format ascii
Invert RX? no
Invert TX? no
SETTINGS
Name GROUP
Thickness 1
COLOR SETTINGS
BG Color
Cursors
Trigger

Solution

```
void initUSART(uint8_t USART_ID){
...

USART->CR1.UE = 1; // Enable USART
USART->CR1.M = 0; // M=0 corresponds to 8 data bits
USART->CR2.STOP = 0b00; // 0b00 corresponds to 1 stop
bit
USART->CR1.OVER8 = 0; // Set to 16 times sampling freq

// Set baud rate to 9600 KBps
// Tx/Rx baud = (f_CK)/(8*(2-OVER8)*USARTDIV) = Tx/Rx
baud = (f_CK)/(16*USARTDIV)
// f_CK = 84e6 Hz
// USARTDIV = 546.875 should be in BRR
// 546 = 0x0222
// 0.875 = 7/8 = 0b1110
// DIV_Mantissa = 0x222
// DIV_Fraction = 0b1110
USART->BRR.DIV_Fraction = 0b1110;
USART->BRR.DIV_Mantissa = 546;
USART->CR1.TE = 1; // Enable transmission
USART->CR1.RE = 1; // Enable reception
```

```
void sendChar(uint8_t USART_ID, uint8_t data){
USART_TypeDef * USART = id2Port(USART_ID);

USART->DR.DR = data;
while(!USART->SR.TC);
}
```

```
#define USART_ID USART2_ID

int main(void) {
// Configure flash and clock
configureFlash();
configureClock();

// Initialize USART
initUSART(USART_ID);
uint8_t msg[28] = "Happy Hacking!\n\r";

while(1){
  uint8_t i = 0;
  do {
    sendChar(USART_ID, msg[i]);
    i += 1;
  } while (msg[i]);
  delay_ms(2000);
  }
}
```

Summary

By the end of this lecture you will be able to

- Explain the tradeoffs between synchronous and asynchronous serial interfaces
- Develop a library for the UART peripheral on the STM32F401RE
- Verify the output using the logic analyzer on the ADALM2000

Lab 4 Questions

- Main learning outcome is configuring and using SPI peripheral – don't spend lots of time generating precise sine/square waves
  - One approach: use a timer and a loop to generate and send a new output every x seconds where x is a reasonable fraction of a cycle (maybe 100 samples per cycle?)

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8