USING USART COMMUNICATION PROTOCOL WITH THE STM32F051C6 MICROCONTROLLER

****

1. USART PINS ON THE STM32F051C6

****

	CLOCK SOURCE	TX PIN	RX PIN
USART1	APB2	PA9	PA10
USART2	APB1	PA2	PA3

****

****

2. USART REGISTERS ON THE STM3F051C6

****

REGISTER	FUNCTION
Advanced High-performance Bus Enable Register RCC -> AHBENR	Connect clock to Port A
Mode Register GPIOx -> MODER	Configure Tx and Rx pins to alt function mode
Alternate Function Register GPIOx -> AF[x]	Enable USART capability of selected pins
Advanced Peripheral Bus 1/2 Enable Register RCC -> APBxENR	Connect clock to either USART1 or USART2
Baud Rate Register USARTx -> BRR	Set baud rate
Control Register 1 USARTx -> CR1	Set word length, parity enable and selection, interrupt enable, USART enable, mode enable, etc…
Interrupt & Status Register USARTx -> ISR	Program flow control using USART interrupt event flags
Receive Data Register USARTx -> RDR	Contains the received data
Transmit Data Register USARTx -> TDR	Contains the data to be transmitted

****

****

3. EXAMPLE

****

In this example we write a program to read every two bytes(8-bits) of incoming data via USART1, then increment their values by one and transmit them back. We will use odd parity for error checking.

STEP 1: Pin Selection

In this example the pins have already been selected because the program specifies that we use USART1 and its Tx and Rx pins for USART1 are PA9 and PA10, respectively.

STEP 2: Enable Port A

All USART pins are on Port A, so we now connect clock signal to Port A

```
RCC -> AHBENR |= RCC_AHBENR_GPIOAEN;
```

STEP 3: Configure Tx and Rx pins to alternate function mode

To configure the mode of a pin we use the mode register. Each pin on the STM32F051C6 can be configured to one of the four I/O modes. We want to use USART capability of pins PA9 and PA10, so we have to configure them to alternate function mode.

	MODER bits	
I/O mode	bit 1	bit 0
input	0	0
output	0	1
Alt function	1	0
Analog	1	1

```
GPIOA -> MODER &= ~GPIO_MODER_MODER9_0; // write 0 to MODER bit 0 of PA9
GPIOA -> MODER |= GPIO_MODER_MODER9_1; // write 1 to MODER bit 1 of PA9
GPIOA -> MODER &= ~GPIO_MODER_MODER10_0;
GPIOA -> MODER |= GPIO_MODER_MODER10_1;
```

STEP 4: Select Alternate Function

One pin can have multiple alternate capabilities, so we have to select which one we intend to use. To do this we will consult the following table.

Table 14. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1				COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2				
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3				COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4				
PA4	SPI1_NSS, I2S1_WS	USART2_CK		TSC_G2_IO1	TIM14_CH1			
PA5	SPI1_SCK, I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2				
PA6	SPI1_MISO, I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3		TIM16_CH1	EVENTOUT	COMP1_OUT
PA7	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1	EVENTOUT	COMP2_OUT
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT				
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1				
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2				
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	TSC_G4_IO3				COMP1_OUT
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	TSC_G4_IO4				COMP2_OUT
PA13	SWDIO	IR_OUT						
PA14	SWCLK	USART2_TX						
PA15	SPI1_NSS, I2S1_WS	USART2_RX	TIM2_CH1_ETR	EVENTOUT				

We want to use USART capabilities of pins PA9 and PA10; according to the table, we have to select Alternate Function 1(AF1). After determining which alternate function we want, we have to write to the Alternate Function Register(AFR) to indicate which AF we want to enable. The AFR is organised as follows:

AFRH (AFR[1]) and AFRL (AFR[0])

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

To enable a specific alternate function on a pin, we write to 4-bits of the AFR as indicated by the table below;

	bit 3	bit 2	bit 1	bit 0
AF0	0	0	0	0
AF1	0	0	0	1
AF2	0	0	1	0
AF3	0	0	1	1
AF4	0	1	0	0
AF5	0	1	0	1
AF6	0	1	1	0
AF7	0	1	1	1

So, to enable AF1 on PA9 and PA10, we write:

```
GPIOA -> AFR[1] = 0b0001 << (9-8)*4;
GPIOA -> AFR[1] = 0b0001 << (10-8)*4;
```

STEP 5: Connect APB clock to USART

USART1 and USART2 have different clock signal sources, USART1 uses APB2 clock and USART2 uses APB1 clock. So, for our example we have to connect APB2 clock signal to USART1:

```
RCC -> APB2ENR |= RCC_APB2ENR_USART1EN;
```

STEP 6: Set transmission speed (SET BEFORE ENABLING USART)

The standard transmission speed for UART protocol is 9600 character transmissions per second. We will use this value for our example since it was not otherwise specified. To set the baud rate, write the value of the expression below to the Baud Rate Register(BRR)

$$\frac{f_{CLK}}{\text{baud rate}}$$

At reset, the STM32F051C6 clock frequency is 48MHz, therefore, we write:

```
USART1 -> BRR = 48000000/9600;
```

STEP 7: Set word length

Our program reads incoming data in discrete lengths of 8 data bits, so we have to set word length to 8-bits. To do this, we write 1 to bit M of Control Register 1(CR1):

```
USART1 -> CR1 |= USART_CR1_M;
```

STEP 7: Parity control configurations

Our program uses odd parity for error checking, so we have to enable parity control and select odd parity. To this we write 1 to PCE and PS bits of CR1:

```
USART1 -> CR1 |= USART_CR1_PCE; // enable parity control
USART1 -> CR1 |= USART_CR1_PS; // select odd parity error checking
```

OR `USART1 -> CR1 |= USART_CR1_PCE | USART_CR1_PS;`

STEP 8: Configure interrupts

Our program has to know when data is received to read the data in pars as requested. So, we need an interrupt event every time a byte is received to keep count of transmissions. To do this, we write 1 to the Read data Not Empty Interrupt Enable(RXNEIE) bit in CR1. Enabling the RXNEIE bit will allow hardware to generate interrupt whenever read data register is not empty(i.e data is received).

```
USART1 -> CR1 |= USART_CR1_RXNEIE;
```

STEP 9: USART mode

Our program needs to read and transmit data, so it needs to work in both transmit and receive mode. To enable transmission we write 1 to bit TE in CR1 and to enable data read we write 1 to bit RE in CR1:

```
USART1 -> CR1 |= USART_CR1_RE | USART_CR1_TE;
```

STEP 10: Enable USART

To enable USART1, we write 1 to bit UE in CR1:

```
USART1 -> CR1 |= UART_CR1_UE;
```

STEP 11: Enable USART Interrupt function

We have to enable USART1 interrupt function and its write Interrupt Service Routine(ISR) code to ensure our program reads data in 2 byte pairs.

```
NVIC_EnableIRQ(USART1_IRQn); // enable USART1 interrupt function

void USART1_IRQHandler(void){
      // ISR code
}
```

STEP 12: Write code for main function

****

****

4. EXAMPLE CODE

****

```c
/* LIBRARIES */
#inlcude <stm32f0xx.h>
#include <stdbool.h>

/* GLOBAL VARIABLES */
uint8_t Data[] = {0,0};
uint8_t counter = 0;
bool data_recieved = false;

/* FUNCTION DECLARATIONS */
void init_USART(void);

/* MAIN FUNCTION */
void main(void)
{
	init_USART();

	while(1){
		if(data_recieved == true){
			USART1 -> TDR = Data[0] + 1;
			while( (USART1 -> ISR & USART_ISR_TXE) == 0);
			USART1 -> TDR = Data[1] + 1;
			while( (USART1 -> ISR & USART_ISR_TXE) == 0);
			counter = 0;
			data_recieved = false;
		}
	}
}
```

```c
/* FUNCTION DEFINITION */
void init_USART(void){
	RCC -> AHBENR |= RCC_AHBENR_GPIOAEN;
	GPIOA -> MODER |= (GPIO_MODER_MODER9_1 | GPIO_MODER_MODER10_1);
	GPIOA -> AFR[1] = ( 0b0001 << (9-8)*4 | 0b0001 << (10-8)*4 );

	RCC -> APB2ENR |= RCC_APB2ENR_USRT1EN;
	USART1 -> BRR = 480000000/9600;
	USART1 -> CR1 |= USART_CR1_M;
	USART1 -> CR1 |= ( USART_CR1_PSE | USART_CR1_PS );

	USART1 -> CR1 |= USART_CR1_RXNEIE;

	USART1 -> CR1 |= ( USART_CR1_RE | USART_CR1_TE );
	USART1 -> CR1 |= USART_CR1_UE;

	NVIC_EnableIRQ(USART1_IRQn);
}

/* INTERRUPT FUNCTION DEFINITION */
void USART1_IRQHandler(void){
	Data[counter] = USART1 -> RDR;

	if( counter >= 1 ){
		data_recieved = true;
	}

	counter++;
}
*********************************************************************************************
```