Lecture 14: STM32 Serial Communication I

Seyed-Hosein Attarzadeh-Niaki

Review

- Direct Memory Access
  - Channels
  - Transfer modes
  - Registers
  - Configuration

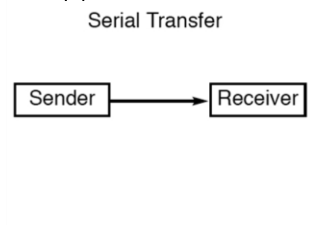
Outline

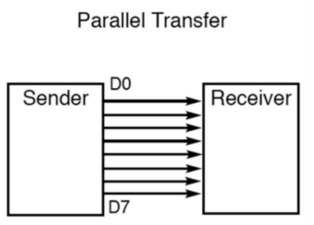
- Serial communication concepts
  - Synchronous and asynchronous serial buses
- STM32F401 universal synchronous asynchronous receiver transmitter (USART)
  - Baud-rate generation
  - Registers
  - Programming with polling and interrupt
- Basic UART problems and solutions

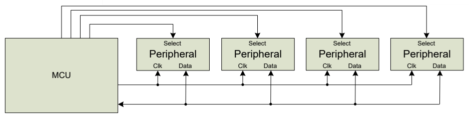
SERIAL COMMUNICATION CONCEPTS

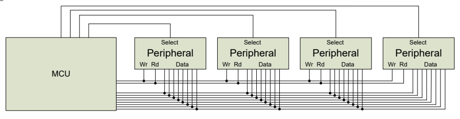
Serial vs. Parallel Communication

- Serial
  - Cheaper
  - Long distance (different cities using a modem)
  - Slow (?)
- Parallel
  - Expensive
  - Short distance
  - Fast
  - No modulation



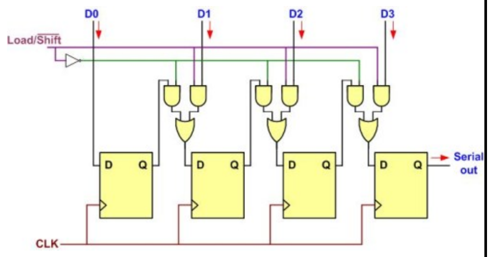


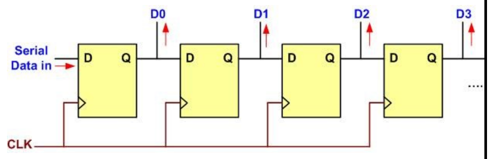




Parallel <-> Serial

- Parallel In Serial Out
- Serial In Parallel Out

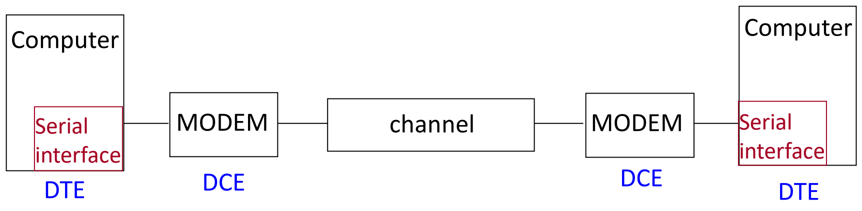




High-Level View of Serial Communication

The sender and receiver need a <u>protocol</u> to make sense of data

- e.g., how the data is packed, how many bits constitute a character, when the data begin and end

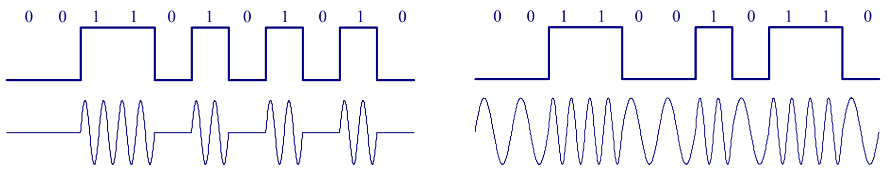


DTE- Data Terminal Equipment, usually a computer.

DCE- Data Communication Equipment, usually a *modem*.

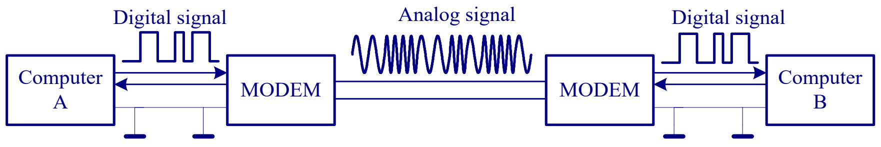
Serial interface –ICs such as 8251A,16550, and 8250, connecting DTE and DCE.

Modulation and Demodulation

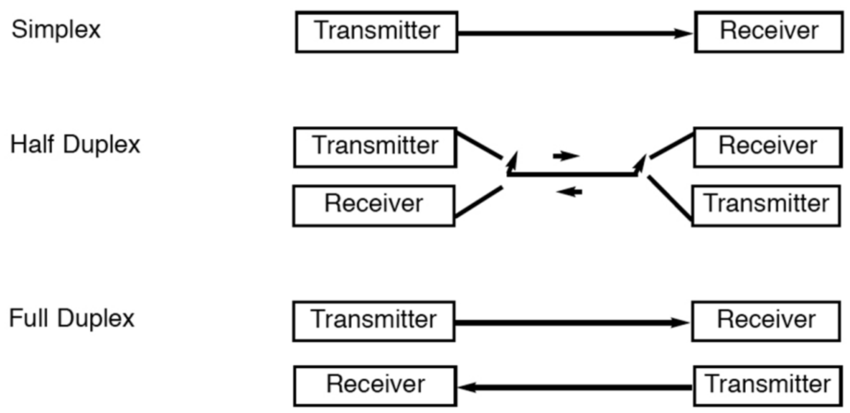


1 and 0 are represented using different amplitude

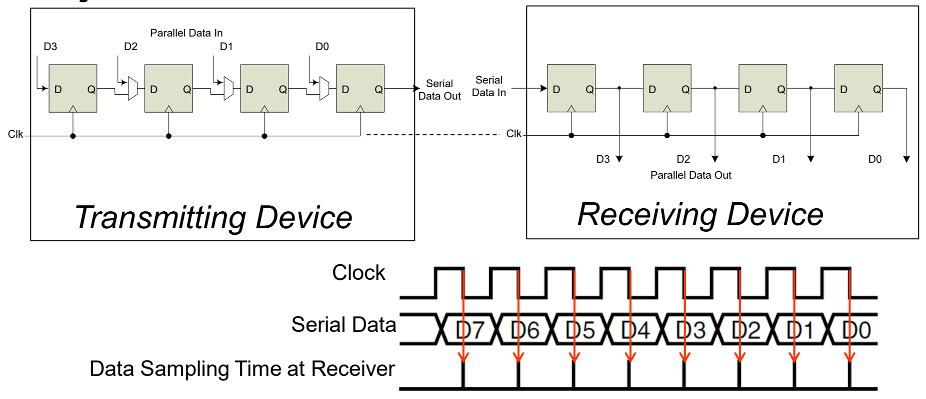
1 and 0 are represented using different frequency



Types of Serial Communication



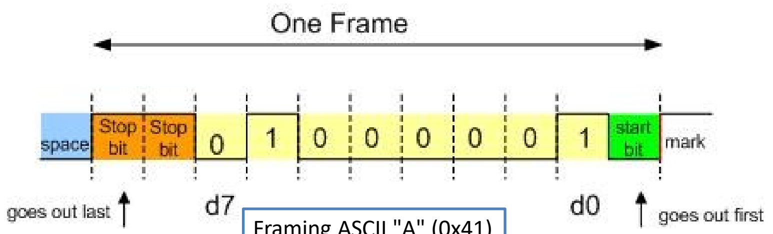
Synchronous Serial Data Transmission



- Use shift registers and a clock signal to convert between serial and parallel formats
- **Synchronous**: an explicit clock signal is transmitted along with the data signal, or clock resynchronization is performed regularly.

Asynchronous Serial Communication: Framing Data, Start and stop bits

- In asynchronous transmission there is no clock with data
- When there is *no* transfer the signal is *high*
- Transmission *begins* with a start (*low*) bit
- LSB first
- Finally 1 *stop* bit (*high*)
- Transmitter and receiver clocks must be reasonably close, since the only timing reference is the start bit

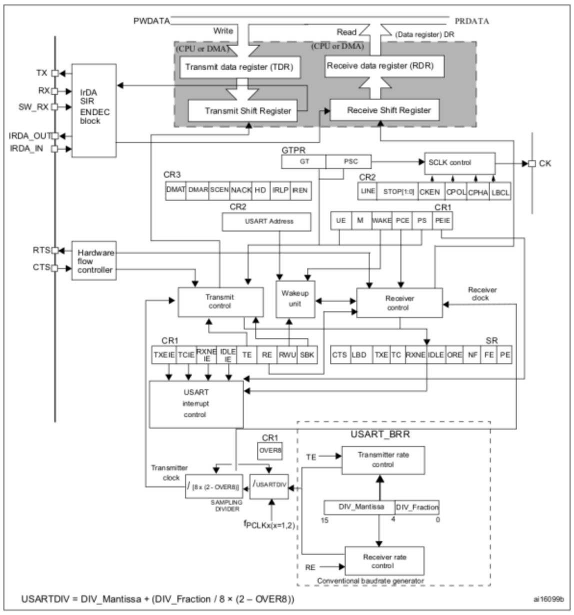


Framing ASCII "A" (0x41)

STM32F401 UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

USART Block Diagram

- Full duplex, asynchronous communications
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance
- Fractional baud rate generator systems
- Programmable data word length (8 or 9)
- Configurable stop bits - support for 1 or 2 stop bits
- Transmitter clock output for synchronous transmission
- Single-wire half-duplex communication
- Configurable multibuffer communication using DMA (direct memory access)
- Separate enable bits for transmitter and receiver
- Transfer detection flags
- Parity control
- Four error detection flags
- Ten interrupt sources with flags
- Multiprocessor communication
- Wake up from mute mode
- Two receiver wakeup modes



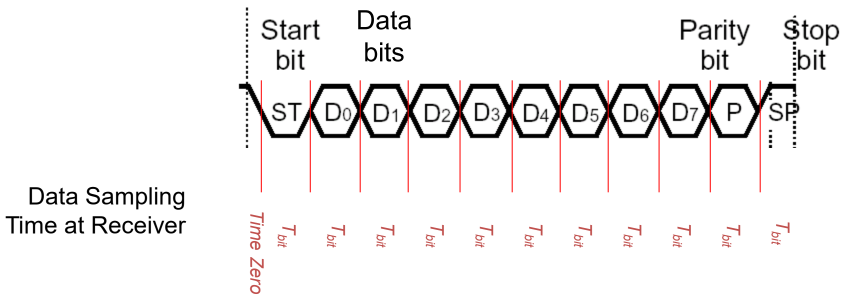
USART

- Like other modules, USART in the STM32F4 families is capable to operate in many modes

USART modes	USART1	USART2	USART3	UART4	UART5	USART6
Asynchronous mode	X	X	X	X	X	X
Hardware flow control	X	X	X	NA	NA	X
Multibuffer communication (DMA)	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous	X	X	X	NA	NA	X
Smartcard	X	X	X	NA	NA	X
Half-duplex (single-wire mode)	X	X	X	X	X	X
IrDA	X	X	X	X	X	X
LIN	X	X	X	X	X	X

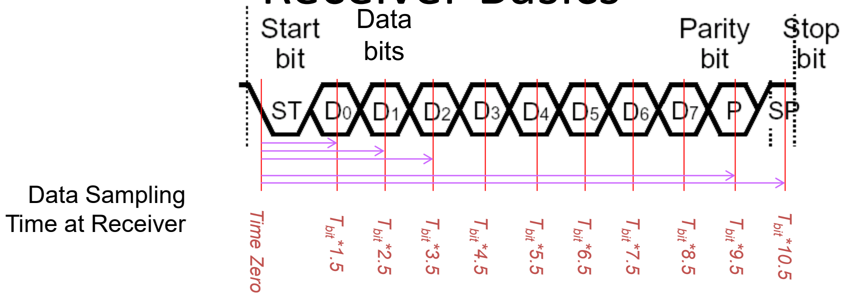
- We focus on the Asynchronous mode, which is sometimes referred as Universal asynchronous receiver/transmitter (UART).

Transmitter Basics



- If no data to send, keep sending 1 (stop bit) – idle line
- When there is a data word to send
  - Send a 0 (start bit) to indicate the start of a word
  - Send each data bit in the word (use a shift register for the transmit buffer)
  - Send a 1 (stop bit) to indicate the end of the word

Receiver Basics

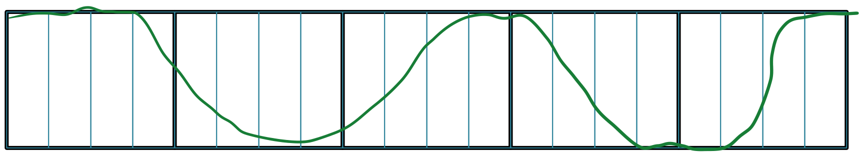


- Wait for a falling edge (beginning of a Start bit)
  - Then wait ½ bit time
  - Do the following for as many data bits in the word
    - Wait 1 bit time
    - Read the data bit and shift it into a receive buffer (shift register)
  - Wait 1 bit time
  - Read the bit
    - if 1 (Stop bit), then OK
    - if 0, there’s a problem!

How the STM32F4 works

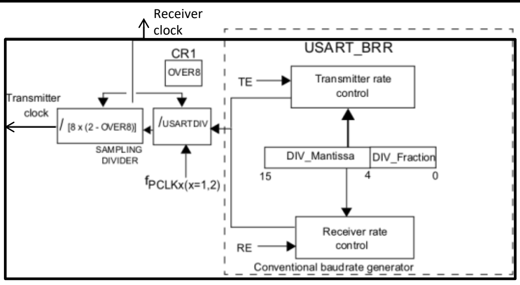
- Transmitter and receiver must agree on several things (protocol)
  - Order of data bits
    - LSB will be transmitted first
  - Number of data bits
    - Can be configured as 8 or 9 bits
  - What a start bit is (1 or 0)
    - 0
  - What a stop bit is (1 or 0)
    - 1
    - Configurable length (0.5, 1, 1.5 or 2)
  - How long a bit lasts
    - Software programmable phase and polarity
- Many of them are configurable as well

Input Data Oversampling



- When receiving, UART oversamples incoming data line
  - Extra samples allow voting, improving noise immunity
  - Better synchronization to incoming data, improving noise immunity
- FTM32F4 provides configurable oversampling rate of either 8 or 16 times the baud rate clock
- Two voting method: Single sample in the center or majority vote of the three samples in the center

Fractional Baud Rate Generation



- Both Rx and Tx are set to the same baud rate as programmed in the Mantissa and Fraction values of USARTDIV
- In standard USART (SPI mode included)
  - Baud rate $= \frac{f_{ck}}{8\times(2-OVER8)\times USARTDIV}$
  - USARTDIV = DIV_Mantissa + DIV_FRACTION/(8 × (2-OVER8))
    - OVER8=0 -> fractional part coded on 4 bits and programmed by DIV_fraction[3:0]
    - OVER8=1 -> fractional part coded on 3 bits and programmed by DIV_fraction[2:0]
- Example: derive USARTDIV value from USART_BRR (OVER8=0)
  - DIV_Mantissa=0d27, DIV_Fraction=0d12 (USART_BRR=0x1BC)
  - Mantissa=0d27, Fraction=12/16=0d0.75 (left shift by 4 bits)
  - Therefore the USARTDIV=0d27.75
- Check Sec. 19.3.4 of the reference manual for more examples

Using the UART

Transmitter

- Configure the GPIO
  - AF mode, fast speed
- Enable the USART (Set UE in CR1)
- Define word length (M bit in CR1)
- Program the stop bits (CR2)
- Using DMA? (DMAT in CR3)
- Parity Check? (PCE/PS in CR1)
- Configure Baud Rate (USART_BRR)
- First transmission (Set TE in CR1)
- Write data to send (DR)
- Repeat writing data to send (DR)
- For ending the transmission, wait until the last frame is complete (when TC=1)

Receiver

- Configure the GPIO
  - AF mode, fast speed
- Enable the USART (Set UE in CR1)
- Define word length (M bit in CR1)
- Program the stop bits (CR2)
- Using DMA? (DMAR in CR3)
- Parity Check?(PCE/PS in CR1)
- Configure Baud Rate (USART_BRR)
- Begin waiting for start bit (Set RE in CR1)
- If RXNE is set, then the data has been received and can be read
- Interrupt if RXNEIE bit is set

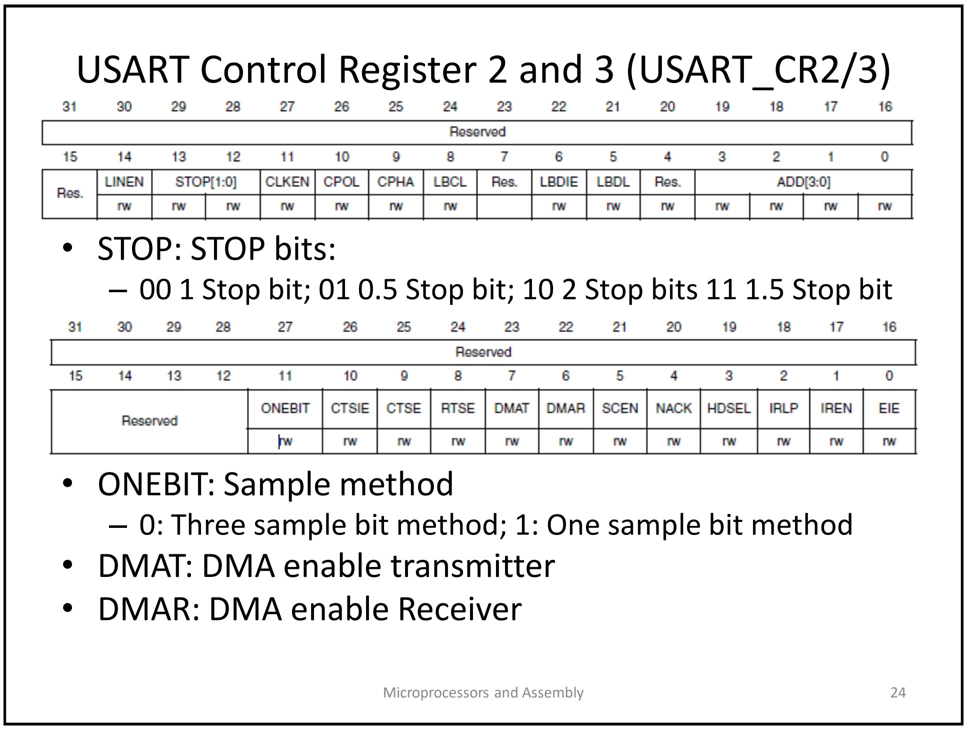
Error calculation for programmed baud rates at f_{PCLK} = 8 MHz or f_{PCLK} = 12 MHz, oversampling by 16

Oversampling by 16 (OVER8=0)							
Baud rate7		f_{PCLK} = 8 MHz			f_{PCLK} = 12 MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	416.6875	0	1.2 KBps	625	0
2	2.4 KBps	2.4 KBps	208.3125	0.01	2.4 KBps	312.5	0
3	9.6 KBps	9.604 KBps	52.0625	0.04	9.6 KBps	78.125	0
4	19.2 KBps	19.185 KBps	26.0625	0.08	19.2 KBps	39.0625	0
5	38.4 KBps	38.462 KBps	13	0.16	38.339 KBps	19.5625	0.16
6	57.6 KBps	57.554 KBps	8.6875	0.08	57.692 KBps	13	0.16
7	115.2 KBps	115.942 KBps	4.3125	0.64	115.385 KBps	6.5	0.16
8	230.4 KBps	228.571 KBps	2.1875	0.79	230.769 KBps	3.25	0.16
9	460.8 KBps	470.588 KBps	1.0625	2.12	461.538 KBps	1.625	0.16
10	921.6 KBps	NA	NA	NA	NA	NA	NA
11	2 MBps	NA	NA	NA	NA	NA	NA
12	3 MBps	NA	NA	NA	NA	NA	NA

USART Control Register 1 (USART_CR1)

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

• OVER8:	0 oversampling by 16/; 1 oversampling by 8
• UE:	Write 1 to enable
• M:	Word length, 0: 8 data bits; 1: 9 data bits
• WAKE:	Wakeup method
• PCE:	Parity enabled with 1
• PS:	Odd parity with 1, even parity with 0
• TE:	Transmitter enable
• RE:	Receiver enable



USART Control Register 2 and 3 (USART_CR2/3)

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

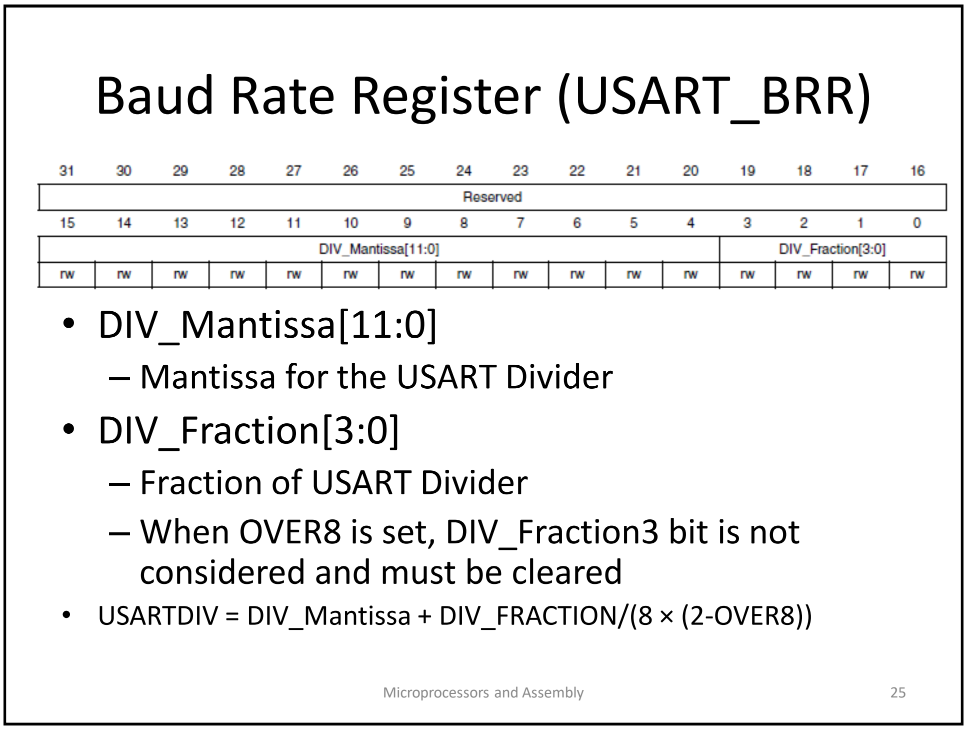
- STOP: STOP bits:
  - 00 1 Stop bit; 01 0.5 Stop bit; 10 2 Stop bits 11 1.5 Stop bit

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				ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- ONEBIT: Sample method
  - 0: Three sample bit method; 1: One sample bit method
- DMAT: DMA enable transmitter
- DMAR: DMA enable Receiver

Microprocessors and Assembly 24



Baud Rate Register (USART_BRR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

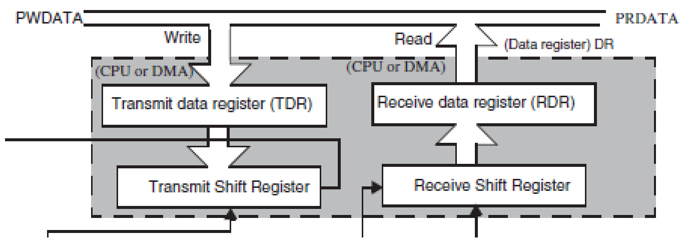
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- DIV_Mantissa[11:0]
  - Mantissa for the USART Divider
- DIV_Fraction[3:0]
  - Fraction of USART Divider
  - When OVER8 is set, DIV_Fraction3 bit is not considered and must be cleared
- USARTDIV = DIV_Mantissa + DIV_FRACTION/(8 × (2-OVER8))

Microprocessors and Assembly 25

Data Register

- DR[8:0]: Data value
  - The data to be transmitted or the data received, depending on whether it is read from or written to



- The TDR register provides the parallel interface between the internal bus and the output shift register .
- The RDR register provides the parallel interface between the input shift register and the internal bus.

USART Status Register (USART_SR)

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

- TXE: Transmit data register empty
- TC: Transmission complete.
- RXNE: Read data register not empty
- IDLE: IDLE line detected
- ORE: Receive overrun. Received data has overwritten previous data in receive buffer
- NF: Noise flag. Receiver data bit samples don't agree.
- FE: Framing error. Received 0 for a stop bit, expected 1.
- PE: Parity error. Incorrect parity received.

Software for Polled Serial Comm.

```
void usart2_init(uint32_t pclk, uint32_t baudrate){
    uint32_t temp     =   0x00;
    uint32_t integer =   0x00;
    uint32_t fraction=   0x00;
    integer   =   ((25*pclk*1000000)/(4*baudrate));
    temp      =   (integer/100)<<4;
    fraction  =   integer-(100*(temp>>4));
    temp     |=   ((((fraction*16)+50)/100))&((uint8_t) 0x0F);

    RCC->AHB1ENR |=  RCC_AHB1ENR_GPIOAEN;
    RCC->APB1ENR |=  RCC_APB1ENR_USART2EN;
    GPIOA->MODER |=  GPIO_MODER_MODER2_1;//Pin2 mode AF
    //GPIOA->PUPDR|=GPIO_PUPDR_PUPDR2_0;//Pull up
    GPIOA->OSPEEDR|= GPIO_OSPEEDER_OSPEEDR2_1;

    GPIOA->AFR[0]|=  0x00000700;//Set the AF to AF7(USART1~3);

    SET_BIT(RCC->AHB1RSTR,RCC_APB1RSTR_USART2RST);
    CLEAR_BIT(RCC->AHB1RSTR,RCC_APB1RSTR_USART2RST);

    USART2->CR1   |=  USART_CR1_UE;
    USART2->BRR    =  temp;
    USART2->CR1   |=  USART_CR1_TE;
}
```

Software for Polled Serial Comm.

```
void usart2_send(uint16_t data) {
    USART2->DR=(data&(uint16_t)0x01ff);
    while(!READ_BIT(USART2->SR, USART_SR_TC)){}
}

uint16_t usart6_receive() {
    while(!READ_BIT(USART6->SR, USART_SR_RXNE)){}
    return (uint16_t)(USART6->DR & (uint16_t)0x01FF);
}
```

- Example Transmitter & Receiver

Transmitter

```
while(1) {
    usart2_send(data);
}
```

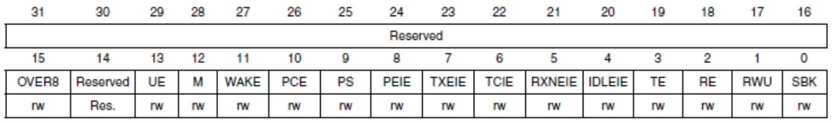
Receiver

```
while (1) {
    echo=usart6_receive();
}
```

Interrupt-Driven Serial Communication

- Use interrupts
- First, initialize peripheral to generate interrupts
- Second, create single ISR with three sections corresponding to the cause of interrupt
  - Transmitter
  - Receiver
  - Error

USART Control Register 1 (USART_CR1)



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

- TCIE: Transmission complete interrupt enable
- RXNEIE: RXNE interrupt enable, this means whenever overrun error or read data register is not empty (ORE=1 or RXNE=1), it will assert the interrupt.

Interrupt Initialization (transmitter)

```
void Init_UART2(uint32_t pclk,uint32_t baud_rate) {
    …
    NVIC_SetPriority(USART2_IRQn, 0);
    NVIC_ClearPendingIRQ(USART2_IRQn);
    NVIC_EnableIRQ(USART2_IRQn);

    USART2->CR1   |= USART_CR1_TCIE;
    …
}
```

Interrupt Handler

- Transmitter

```
void UART2_IRQHandler(void) {
    NVIC_ClearPendingIRQ(USART2_IRQn);
    usart2_send(echo-1);
    …
}
```

- Receiver

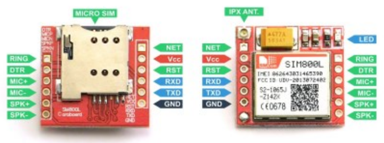
```
void UART2_IRQHandler(void) {
    …
    if (READ_BIT(USART2->SR, USART_SR_RXNE)) {
        // received a character
    }
}
```

Interrupt Handler: Error Cases

```
void UART2_IRQHandler(void) {
    …
    if (READ_BIT(USART2->SR, USART_SR_ORE){
            // handle the error
            ...
            // clear the flag
            ...
    }
}
```

Example Application: SIM800 GSM Module

- Can transmit Voice, SMS, USSD, and data information
- Embedded TCP/UDP protocol
- Many interfaces including serial
- AT cellular command interface



Example AT commands:

- ATA: Answer an Incoming Call.
- AT+GSN: Request TA Serial Number Identification (IMEI)
- AT+CMGS: Send SMS Message

USB to UART Interface

- PCs haven't had external asynchronous serial interfaces for a while, so how do we communicate with a UART?
- USB to UART interface
  - USB connection to PC
  - Logic level (0-3.3V) to microcontroller's UART (not RS232 voltage levels)

Example chips/modules:

- FT232
- PL2303
- CP2101

Building on Asynchronous Comm.

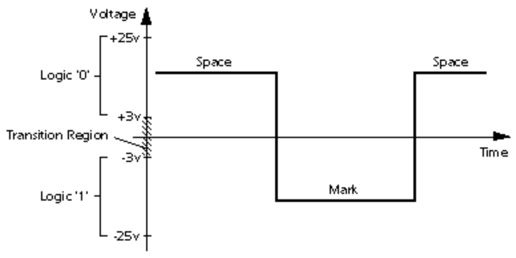
Problem #1

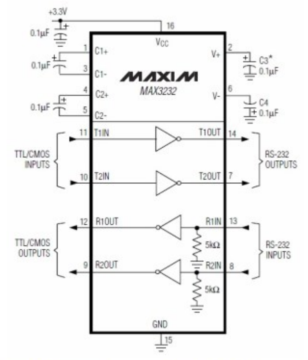
- Logic-level signals (0 to 1.65 V, 1.65 V to 3.3 V) are sensitive to noise and signal degradation

Problem #2

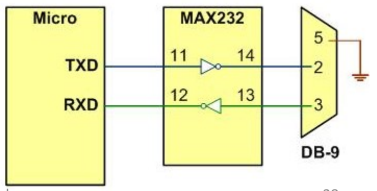
- Point-to-point topology does not support a large number of nodes well
  - Need a dedicated wire to send information from one device to another
  - Need a UART channel for each device the MCU needs to talk to
  - Single transmitter, single receiver per data wire

Solution to Noise: Higher Voltages

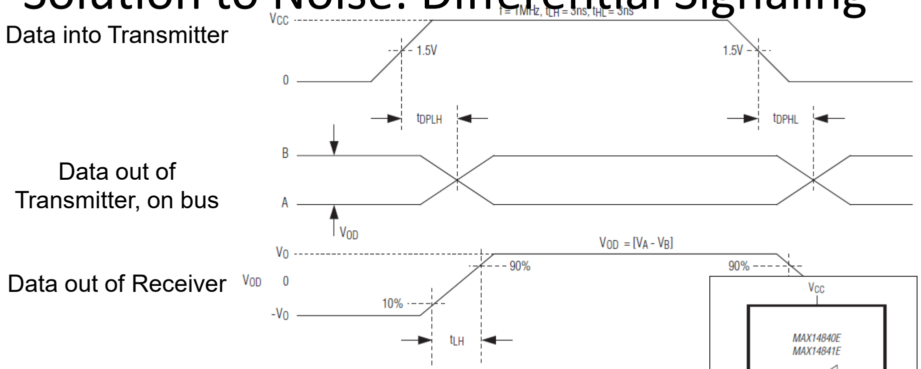


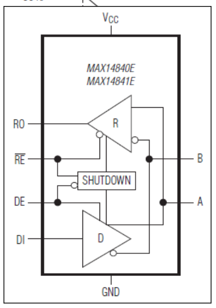


- Use higher voltages to improve noise margin: +3 to +15 V, -3 to -15 V
- Example IC (Maxim MAX3232) uses charge pumps to generate higher voltages from 3.3V supply rail
  - Has two sets of line drivers



Solution to Noise: Differential Signaling





- Use differential signaling
  - Send two signals: Buffered data (A), buffered complement of data (B)
  - Receiver compares the two signals to determine if data is a one (A > B) or a zero (B > A)

Solutions to Poor Scaling

Approaches

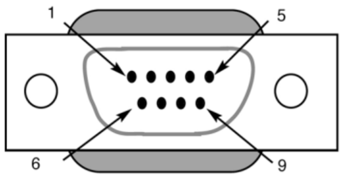
- Allow one transmitter to drive multiple receivers (multi-drop)
- Connect many transmitters and receivers to same data line (multi-point network).
  - Need to add a medium access control (MAC) technique so all nodes can share the wire

Example Protocols

- RS-232: higher voltages, point-to-point
- RS-422: higher voltages, differential data transmission, multi-drop
- RS-485: higher voltages, multi-point

RS232 Standard

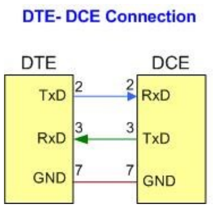
- Developed in 1960 and updated in 1969
- Logic 1 : -3 to -25 volt
- Logic 0 : 3 to 25 volt
- MAX232 converts signals from TTL level to RS232 level
- The baud rate of both sides must match
- PC standard baud rates
  – 2400-4800-9600-14400-19200-28800-33600-57600-115200

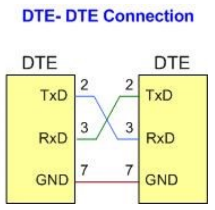


Pin	Description
1	Data carrier detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

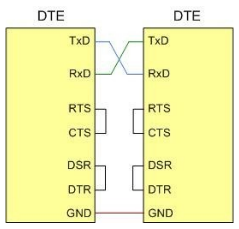
RS232 Standard

- DTE-DCE and DTE-DTE Connections

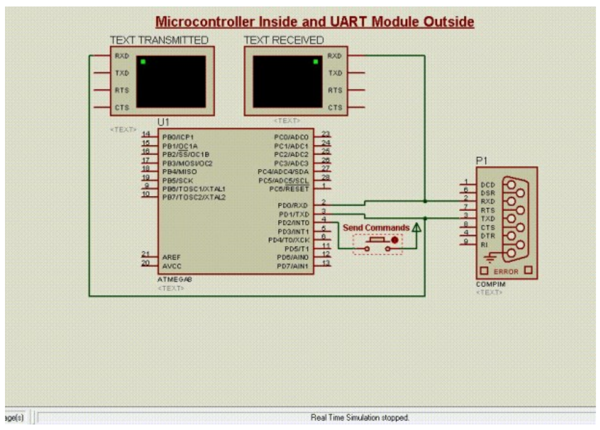




- Null modem connection with flow control signals



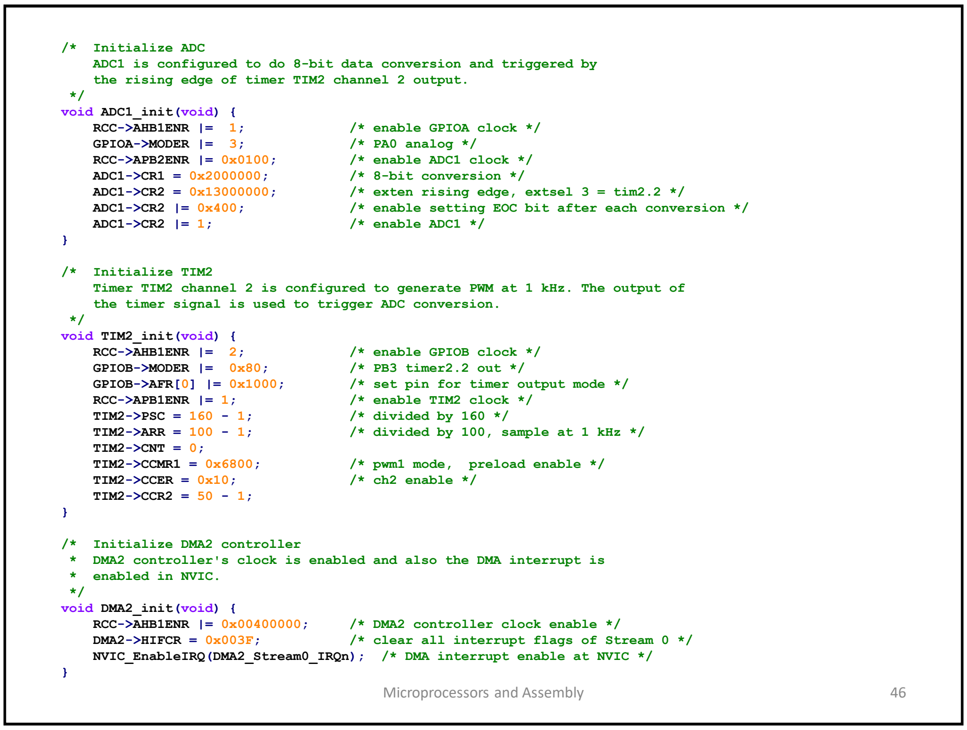
Virtual Terminal & Virtual COM Port in Proteus Simulator



Example

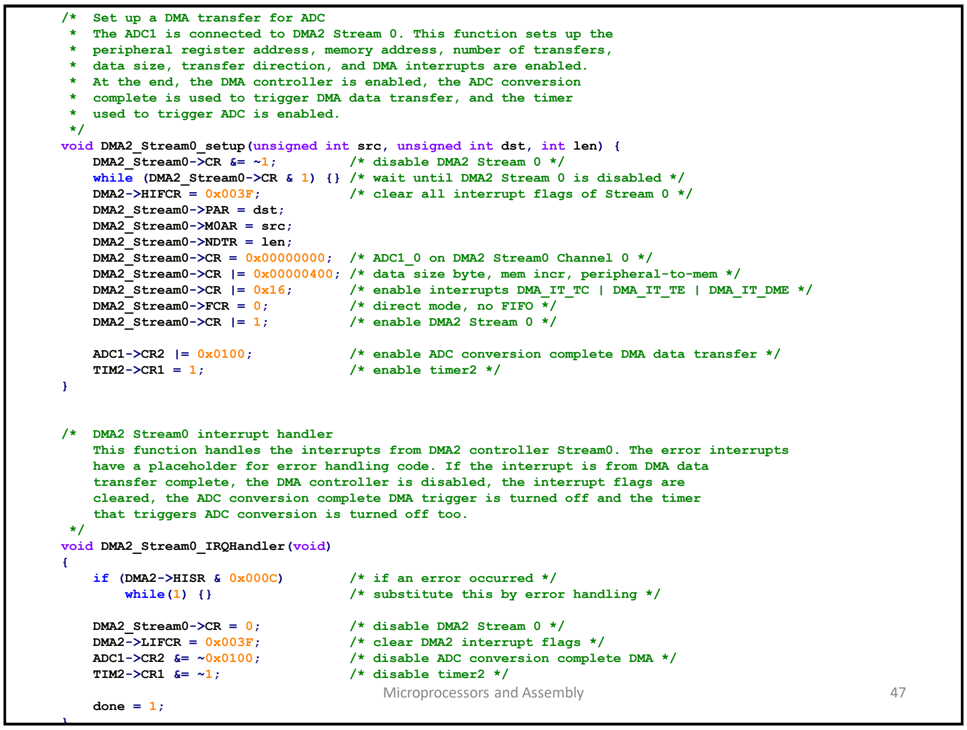
- Set up the timer TIM2 channel 2 to trigger ADC1 to convert the analog input channel 0.
- The output of the ADC1 is transferred to the buffer in memory by DMA
- Once the buffer if full, the DMA is stopped
- That data are converted to decimal ASCII numbers and sent to USART2 to be displayed on the console
- A global variable, done, is used by the DMA transfer complete interrupt handler to signal the other part of the program that a buffer full of data conversion is done

```c
#include "stm32f4xx.h"
#include "stdio.h"

#define ADCBUFSIZE 64

void USART2_init (void);    /* Initialize UART pins, Baudrate */
void DMA2_init(void);       /* Initialize DMA2 controller */
void DMA2_Stream0_setup(unsigned int src, unsigned int dst, int len); /* set up a DMA transfer for ADC1 */
void TIM2_init(void);       /* initialize TIM2 */
void ADC1_init(void);       /* setup ADC */

int done = 1;
char adcbuf[ADCBUFSIZE];    /* buffer to receive DMA data transfers from ADC conversion results */
char uartbuf[ADCBUFSIZE * 5];   /* buffer to hold ASCII numbers for display */

int main(void) {
    int i;
    char* p;

    USART2_init();
    DMA2_init();
    TIM2_init();
    ADC1_init();

    while(1) {
        done = 0;               /* clear done flag */
        /* start a DMA of ADC data transfer */
        DMA2_Stream0_setup((uint32_t)adcbuf, (uint32_t)&(ADC1->DR), ADCBUFSIZE);
        while (done == 0) {}     /* wait for ADC DMA transfer complete */

        /* convert the ADC data into decimal ASCII numbers for display */
        p = uartbuf;
        for (i = 0; i < ADCBUFSIZE; i++) {
            sprintf(p, "%3d ", adcbuf[i]);
            p += 4;
        }

        /* send the ADC numbers through USART2 to the console terminal */
        for (i = 0; i < p - uartbuf; i++)
        {
            while (!(USART2->SR & 0x40))  {}
            USART2->DR = uartbuf[i];
        }
    }
}
```



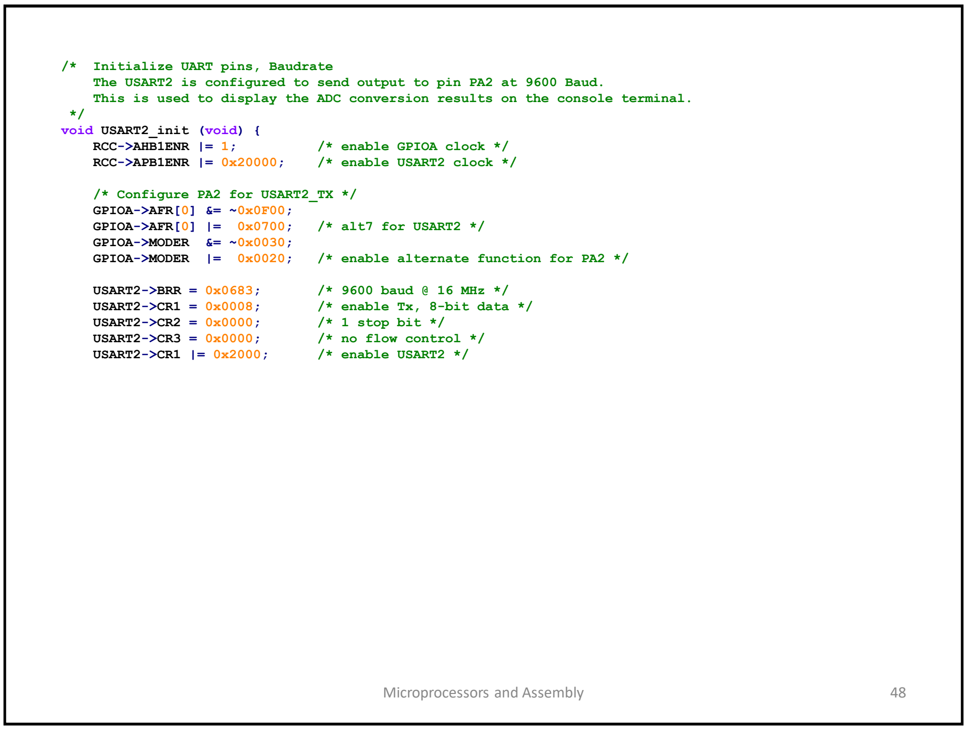
```
/*  Initialize ADC
    ADC1 is configured to do 8-bit data conversion and triggered by
    the rising edge of timer TIM2 channel 2 output.
 */
void ADC1_init(void) {
    RCC->AHB1ENR |=  1;              /* enable GPIOA clock */
    GPIOA->MODER |=  3;              /* PA0 analog */
    RCC->APB2ENR |= 0x0100;          /* enable ADC1 clock */
    ADC1->CR1 = 0x2000000;           /* 8-bit conversion */
    ADC1->CR2 = 0x13000000;          /* exten rising edge, extsel 3 = tim2.2 */
    ADC1->CR2 |= 0x400;              /* enable setting EOC bit after each conversion */
    ADC1->CR2 |= 1;                  /* enable ADC1 */
}

/*  Initialize TIM2
    Timer TIM2 channel 2 is configured to generate PWM at 1 kHz. The output of
    the timer signal is used to trigger ADC conversion.
 */
void TIM2_init(void) {
    RCC->AHB1ENR |=  2;              /* enable GPIOB clock */
    GPIOB->MODER |=  0x80;           /* PB3 timer2.2 out */
    GPIOB->AFR[0] |= 0x1000;         /* set pin for timer output mode */
    RCC->APB1ENR |= 1;               /* enable TIM2 clock */
    TIM2->PSC = 160 - 1;             /* divided by 160 */
    TIM2->ARR = 100 - 1;             /* divided by 100, sample at 1 kHz */
    TIM2->CNT = 0;
    TIM2->CCMR1 = 0x6800;            /* pwm1 mode,  preload enable */
    TIM2->CCER = 0x10;               /* ch2 enable */
    TIM2->CCR2 = 50 - 1;
}

/*  Initialize DMA2 controller
 *  DMA2 controller's clock is enabled and also the DMA interrupt is
 *  enabled in NVIC.
 */
void DMA2_init(void) {
    RCC->AHB1ENR |= 0x00400000;      /* DMA2 controller clock enable */
    DMA2->HIFCR = 0x003F;            /* clear all interrupt flags of Stream 0 */
    NVIC_EnableIRQ(DMA2_Stream0_IRQn);  /* DMA interrupt enable at NVIC */
}
```

Microprocessors and Assembly 46



```
/*  Set up a DMA transfer for ADC
 *  The ADC1 is connected to DMA2 Stream 0. This function sets up the
 *  peripheral register address, memory address, number of transfers,
 *  data size, transfer direction, and DMA interrupts are enabled.
 *  At the end, the DMA controller is enabled, the ADC conversion
 *  complete is used to trigger DMA data transfer, and the timer
 *  used to trigger ADC is enabled.
 */
void DMA2_Stream0_setup(unsigned int src, unsigned int dst, int len) {
    DMA2_Stream0->CR &= ~1;          /* disable DMA2 Stream 0 */
    while (DMA2_Stream0->CR & 1) {} /* wait until DMA2 Stream 0 is disabled */
    DMA2->HIFCR = 0x003F;            /* clear all interrupt flags of Stream 0 */
    DMA2_Stream0->PAR = dst;
    DMA2_Stream0->M0AR = src;
    DMA2_Stream0->NDTR = len;
    DMA2_Stream0->CR = 0x00000000;  /* ADC1_0 on DMA2 Stream0 Channel 0 */
    DMA2_Stream0->CR |= 0x00000400; /* data size byte, mem incr, peripheral-to-mem */
    DMA2_Stream0->CR |= 0x16;       /* enable interrupts DMA_IT_TC | DMA_IT_TE | DMA_IT_DME */
    DMA2_Stream0->FCR = 0;          /* direct mode, no FIFO */
    DMA2_Stream0->CR |= 1;          /* enable DMA2 Stream 0 */

    ADC1->CR2 |= 0x0100;            /* enable ADC conversion complete DMA data transfer */
    TIM2->CR1 = 1;                  /* enable timer2 */
}


/*  DMA2 Stream0 interrupt handler
    This function handles the interrupts from DMA2 controller Stream0. The error interrupts
    have a placeholder for error handling code. If the interrupt is from DMA data
    transfer complete, the DMA controller is disabled, the interrupt flags are
    cleared, the ADC conversion complete DMA trigger is turned off and the timer
    that triggers ADC conversion is turned off too.
 */
void DMA2_Stream0_IRQHandler(void)
{
    if (DMA2->HISR & 0x000C)        /* if an error occurred */
        while(1) {}                 /* substitute this by error handling */

    DMA2_Stream0->CR = 0;           /* disable DMA2 Stream 0 */
    DMA2->LIFCR = 0x003F;           /* clear DMA2 interrupt flags */
    ADC1->CR2 &= ~0x0100;           /* disable ADC conversion complete DMA */
    TIM2->CR1 &= ~1;                /* disable timer2 */

    done = 1;
}
```

Microprocessors and Assembly 47



```
/*  Initialize UART pins, Baudrate
    The USART2 is configured to send output to pin PA2 at 9600 Baud.
    This is used to display the ADC conversion results on the console terminal.
 */
void USART2_init (void) {
    RCC->AHB1ENR |= 1;          /* enable GPIOA clock */
    RCC->APB1ENR |= 0x20000;    /* enable USART2 clock */

    /* Configure PA2 for USART2_TX */
    GPIOA->AFR[0] &= ~0x0F00;
    GPIOA->AFR[0] |=  0x0700;   /* alt7 for USART2 */
    GPIOA->MODER  &= ~0x0030;
    GPIOA->MODER  |=  0x0020;   /* enable alternate function for PA2 */

    USART2->BRR = 0x0683;       /* 9600 baud @ 16 MHz */
    USART2->CR1 = 0x0008;       /* enable Tx, 8-bit data */
    USART2->CR2 = 0x0000;       /* 1 stop bit */
    USART2->CR3 = 0x0000;       /* no flow control */
    USART2->CR1 |= 0x2000;      /* enable USART2 */
```