AVR – HW4

임베디드스쿨1기

Lv1과정

2020. 10. 09

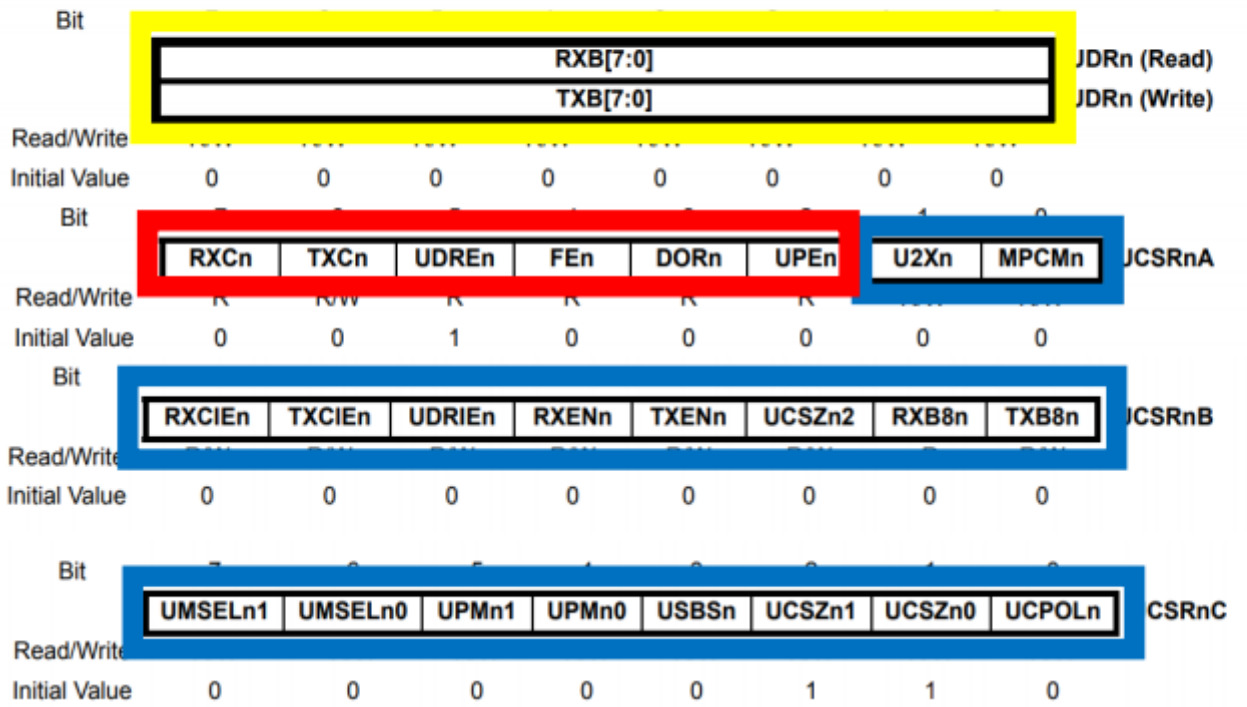
강경수

1. REVIEW

▣ UART REVIEW

2020.10.08 KSS

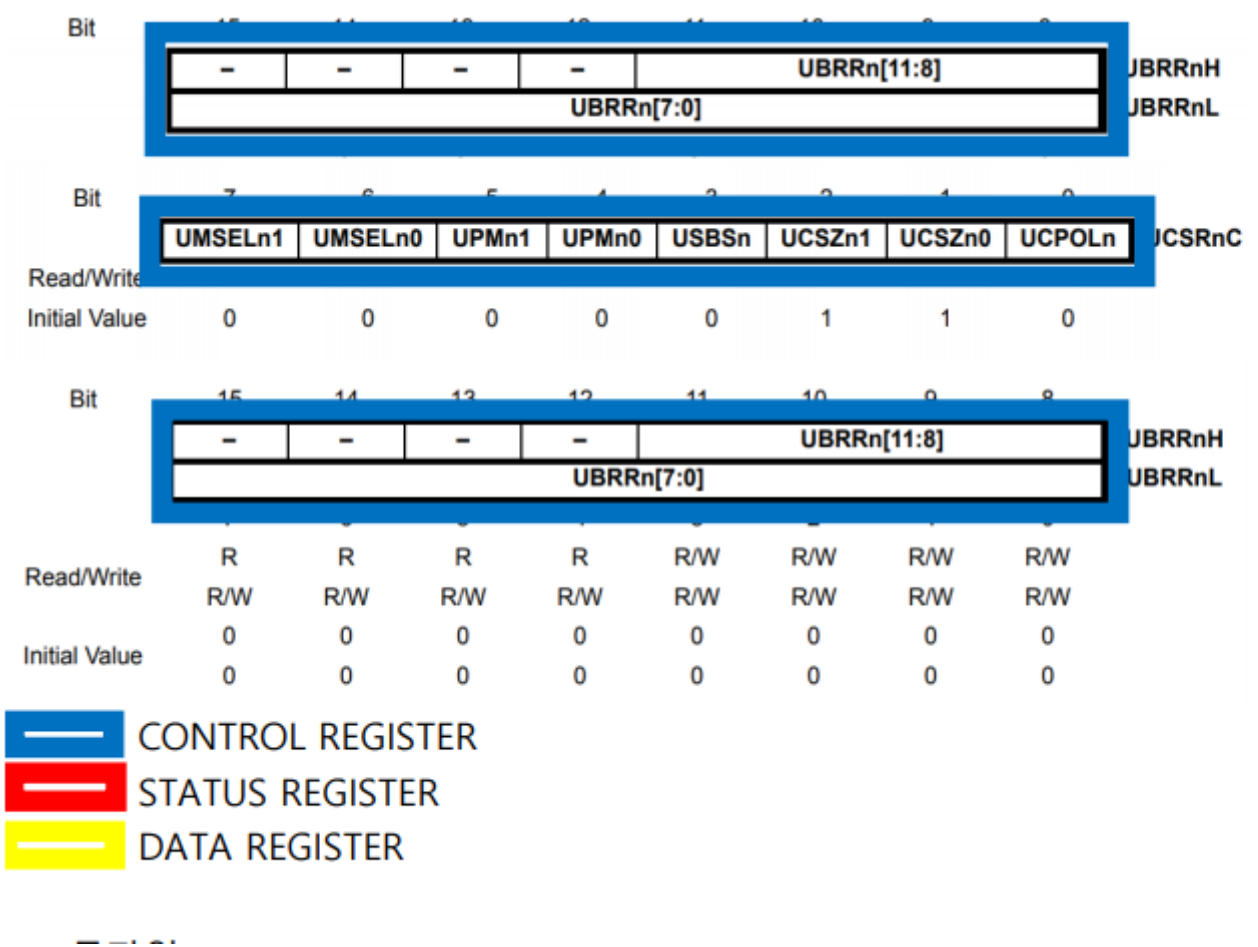
1. UART 레지스터 정리



RX,TX데이터 저장

RX TX 완료 FLAG

1. REVIEW


Bit
–
–
–
–
UBRRn[11:8]
UBRRnH
UBRRn[7:0]
UBRRnL
Bit
UMSELn1
UMSELn0
UPMn1
UPMn0
USBSn
UCSZn1
UCSZn0
UCPOLn
UCSRnC
Read/Write
Initial Value
0 0 0 0 0 1 1 0
Bit
–
–
–
–
UBRRn[11:8]
UBRRnH
UBRRn[7:0]
UBRRnL
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
CONTROL REGISTER
STATUS REGISTER
DATA REGISTER

1. REVIEW

2. 문자열

- 문자열은 Data section에 저장되며 아래와 같은 배열구조를 갖는다. 따라서 uart string송신시 while(*t_data != '₩0') 과 갖는 경우가 많다. 별도의 문자열 선언없이 "(문장)" 큰따옴표로 묶어 사용 가능하다. 이때 "(문자)"의 data type은 unsgiend char* 이다. 또한 배열 시작주소를 의미. 이 문자열 저장영역은 COMPILE시에 할당 된다.

H	E	L	L	O	W	O	R	L	D	NULL

각 글자에는 ASCII코드에 해당되는 HEX값이 저장 돼 있다.

1. REVIEW

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <string.h>

#define cbi(PORTX, BitX) (PORTX &= ~(1<<BitX))
#define sbi(PORTX, BitX) (PORTX |= (1<<BitX))

#define UART_BUFLEN 10
static int usartTxChar(char, FILE*);

void UART_INIT(void){
    sbi(UCSR0A, U2X0); //U2X0 = 1 --> Baudrate 9600 = 207

    UBRR0H = 0x00;
    UBRR0L = 207; //Bauddrate 9600

    UCSR0C |= 0x06; //1stop bit, 8bit data

    sbi(UCSR0B, RXEN0);  //enable receiver and transmitter
    sbi(UCSR0B, TXEN0);
}
```

1. REVIEW

```c
unsigned char UART_receive(void)
{
   while(!(UCSR0A & (1<<RXC0))); //wait for data to be received
   return UDR0;   //get and return received data from buffer
}

void UART_transmit(unsigned char data)
{
   while(!(UCSR0A & (1<<UDRE0)));  //wait for empty transmit buffer
   UDR0 = data;   //put data into buffer, sends the data
}

void UART_string_transmit(char *string)
{
   while(*string != '\0')
   {
      UART_transmit(*string);
      string++;
   }
}

void UART_PRINT(char *name, long val)
{
   char debug_buffer[UART_BUFLEN] = {'\0'};
```

1. REVIEW

```c
int main(void)
{
    /* Replace with your application code */

   FILE* fpStdio = fdevopen(usartTxChar, NULL); //printf 를 사용하기위한 F

   UART_INIT();
   UART_string_transmit("uart_init\n");

    while (1)
    {
      printf("Hello, Double! %f\r\n", 10.205);//\r\n== linux에서 enter
      UART_PRINT("kyungsoo age", 26);
      _delay_ms(1000);

    }
   return 0;
}

int usartTxChar(char ch, FILE *fp){     //시리얼 통신을 stdout으로 초기화
   while(!(UCSR0A & (1<<UDRE0)));
   UDR0  = ch;
   return 0;
}
```

1. REVIEW

▣ UART REVIEW

2020.10.08 KSS

```
#include <avr/io.h>

#define F_CPU 16000000UL

#define sbi(PORTX, BitX) (PORTX |= (1<<BitX))
#define cbi(PORTX, BitX) (PORTX &= ~(1<<BitX))

#include <avr/io.h>

uint16_t adcValue = 0;

#define ADC_REG 0x78;
```

ADC_REG 즉 0x78 을 void형
포인터로 반환함.

1. REVIEW

```
 volatile struct adc *const adc = (void*) ADC_REG;
struct adc
{
    union
    {
        struct {
                uint8_t adc_l;
                uint8_t adc_h;
            };
            uint16_t adc;
        };
        uint8_t adcsr_a;
        uint8_t adcsr_b;
        uint8_t admux;
    };

void adcInit(void){

    sbi(adc->admux, REFS0);
    sbi(adc->adcsr_a,ADPS0);
    sbi(adc->adcsr_a,ADPS1);
    sbi(adc->adcsr_a,ADPS2);
    sbi(adc->adcsr_a,ADEN);

}
```

익명 구조체 및 UNION을 통한 REGISTER I/O MAPPING

AVR CORE또한 litte endian이라 하위 비트부터 할당 됨을 알 수 있다.

adc struct는 총 40bit이다.

이때 이 값은 adc 구조체 포인터 변수 adc에 할당됨.

이 adc를 선언해 줄때 핵심 volatile로 선언하여야

register I/O MAPPING에 있어서 컴파일러가 함부러 최적화 하지 않고 사용시마다 메모리에 접근하게 끔 한다.

clockwise rule에 따라서 해석하면 adc는 상수이며 (선언이후에 값 변경 불가능, datatype은 volatile strcut adc 이다. 즉 이 adc에 접근할경우 컴파일시 함부러 최적화 하면 안되고 adc pointer가 가리키는 값에 항상 접근하여야 한다.

1. REVIEW

```
uint16_t readADC(uint8_t channel)
{
    adc->admux &=  0xF0;
    adc->admux |= channel;

    sbi(adc->adcsr_a,ADSC);
    while(adc->adcsr_a & (1<<ADSC));

    return adc->adc;
}
```

adcsr_a (8bit) 중 ADSC (#define ADSC 6) 즉
*(0x7A) |= (1<< 6); 이라고 볼 수 있다.

내부 1.1V 레퍼런스 전압 사용

channel을 통해 adc채널중 하나 선택.
atmega328p 는 아래와같은 ADC채널 존재.

(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
(0x79)	ADCH	ADC data register high byte							
(0x78)	ADCL	ADC data register low byte							

Single Ended Input
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
ADC6
ADC7
ADC8(1)

1. REVIEW

▣ const와 volatile에 대한 이해

2020.10.08 KSS

- const 상수 즉 어떤값을 선언과 동시에 초기화할때 이후에는 변경하지 못하게 한다.

ex) int const a =10;

a = 10으로 선언된 이후 이 값은 변경이 불가능하다.

- 복잡한 const 사용법

ex) const int *const a; 시계방향rule에 따라서 변수이름부터 시계 방향으로 해석

즉 a 는 const이며 또한 int형 pointer이다. 이때 int* 또한 consts이다.

a의 값 즉 변수값 고정이되고 포인터의 값도 바꿀 수 없다.