

Chapter 125. 와치독 타이머

마이크로컨트롤러 기반의 시스템을 운용하는 동안 여러 가지 이유로 시스템이 정지한 경우가 있다. 시스템이 정지하였지만 설치된 시스템에 쉽게 접근할 수 없다면 시스템은 작동이 멈춘 상태로 계속 방치되어 있어야 하므로 바람직하지 않다. 이런 경우 사용할 수 있는 것이 바로 와치독 타이머(watchdog timer, WDT)이다. 와치독 타이머의 주요 목적은 프로그램이 정상적인 동작을 수행하지 못하는 경우 시스템을 리셋시킴으로써 프로그램이 다시 시작되도록 하는데 있다.

와치독이라는 단어는 경비견을 가리키며 마이크로컨트롤러에서는 CPU가 정상적으로 동작하고 있는지를 감시하기 위해 사용된다. 타이머는 시간을 측정하는 도구이므로 와치독과 결합되어 일정 시간 내에 정해진 동작이 발생하지 않으면 지키고 있던 경비견이 달려들도록 한다. 경비견이 달려들면 마이크로컨트롤러는 리셋된다. 와치독 타이머를 동작 상태에 놓고 아무런 조치 없이 그대로 둘 경우 일정 시간이 경과하면, 즉 타이머가 만료되면 마이크로컨트롤러가 리셋되므로 정해진 시간 이내에 타이머를 다시 리셋시켜 주어야 한다. ATmega328의 경우 128KHz로 동작하는 내부 오실레이터에 의해 동작하며, 타이머가 만료되는 경우 인터럽트가 발생하거나 마이크로컨트롤러를 리셋하는 두 가지 동작을 수행할 수 있으며 각각 인터럽트 모드와 리셋 모드라고 부른다.

1. 와치독 타이머를 위한 레지스터

와치독 타이머를 사용하기 위해서는 일반적으로 하이 푸트 바이트 중 WDTON(Watchdog Timer Always On) 비트를 영(zero)으로¹⁾ 설정한다. ATmega328의 초깃값은 1로 와치독 타이머를 사용하지 않도록 설정되어 있으며 아두이노 우노의 경우도 동일하다.

와치독 타이머에 의해 마이크로컨트롤러가 리셋된 경우를 어떻게 알아낼 수 있을까? 마이크로컨트롤러가 리셋된 이유는 MCUSR 레지스터를 통해 알아낼 수 있다. MCUSR 레지스터는 MCU Status Register로 마이크로컨트롤러를 리셋시킨 소스에 관한 정보를 가지고 있다. 표 1은 MCUSR 레지스터의 비트를 요약한 것이다.

1) 퓨즈 비트의 경우 세트된 경우 0의 값을, 클리어된 경우 1의 값을 가지며 이는 일반적인 레지스터의 경우와 반대이다.

비트	7	6	5	4	3	2	1	0
	-	-	-	-	WDRF	BORF	EXTRF	PORF
읽기/쓰기	R	R	R	R	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 1. MCUSR 레지스터 구조

비트	설명
WDRF	Watchdog System Reset Flag : 와치독 타이머에 의한 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이 발생하거나 해당 비트에 영(zero)의 값을 기록하는 경우 클리어 된다.
BORF	Brown-out Reset Flag : 브라운아웃에 의한 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이 발생하거나 해당 비트에 영(zero)의 값을 기록하는 경우 클리어 된다.
EXTRF	External Reset Flag : 외부 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이 발생하거나 해당 비트에 영(zero)의 값을 기록하는 경우 클리어 된다.
PORF	Power-on Reset Flag : 파워온 리셋이 발생한 경우 1로 세트된다. 해당 비트에 영(zero)의 값을 기록하는 경우에만 클리어 된다.

표 1. MCUSR 레지스터 비트

표 1에서 알 수 있듯이 ATmega328에는 4 종류의 리셋 소스가 존재한다. 파워온 리셋은 전원이 인가된 후 공급 전압이 파워온 리셋 임계치 이하이면 마이크로컨트롤러가 리셋 상태에 있도록 한다. 전원이 처음 공급되면 오실레이터와 공급 전압이 안정되기까지 시간이 필요하므로 파워온 리셋을 통해 마이크로컨트롤러가 안정된 상태에서 시작할 수 있도록 해준다. 브라운아웃 리셋은 공급 전압이 브라운 아웃 리셋 임계치 아래로 떨어질 때 발생한다. 파워온 리셋과 유사하지만 마이크로컨트롤러의 동작 중에 발생한다는 점에서 차이가 있다. 외부 리셋은 마이크로컨트롤러의 리셋 칩으로 일정 시간보다 긴 리셋 신호가 가해질 때 발생하며, 와치독 리셋은 와치독 타이머에서 발생하는 리셋으로 이 장에서 다루고 있는 내용이다.

와치독 리셋과 브라운아웃에 의한 리셋이 연속으로 발생하는 경우를 생각해 보자. 와치독 리셋이 발생하면 WDRF 비트가 세트되고 이후 브라운아웃 리셋이 발생하면 BORF 비트가 세트되어 리셋 상황을 구별할 수 없게 된다. 따라서 리셋 상황을 알고 싶다면 프로그램의 시작 부분에서 MCUSR 레지스터를 통해 리셋의 원인을 알아내고 반드시 클리어시켜 주어야 한다.

와치독 리셋의 동작은 WDTCR (Watchdog Timer Control Register) 레지스터에 의해 제

어된다. WDTCSR 레지스터의 구조는 그림 2와 같고 각 비트의 의미는 표 2와 같다.

비트	7	6	5	4	3	2	1	0
	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	X	0	0	0

그림 2. WDTCSR 레지스터 구조

비트	설명
WDIF	Watchdog Interrupt Flag : 인터럽트 모드에서 와치독 타이머의 타임아웃이 발생하였을 때 세트된다. 인터럽트 서비스 루틴이 실행되면 자동으로 클리어 된다.
WDIE	Watchdog Interrupt Enable : 와치독 타이머의 타임아웃에 대한 동작을 결정한다. 이외에도 와치독 타이머가 타임아웃 되는 경우 동작에 영향을 미치는 값에는 하이 퓨즈의 WDTON 비트와 WDE 비트가 있다. 세 가지 비트의 조합에 따른 타임아웃 동작은 표 3과 같다.
WDCE	Watchdog Change Enable : WDE와 WDP 비트의 값을 변경하기 위해서는 WDCE 비트를 먼저 세트한 후 변경하여야 한다. WDCE 비트가 세트된 이후에는 4클록 이후 자동으로 클리어 된다.
WDE	Watchdog System Reset Enable : 와치독 타이머 만료에 의한 시스템 리셋을 허용한다. WDE는 MCUSR 레지스터의 WDRF 비트에 의해 오버라이드 된다, 즉 WDRF 비트가 세트되면 WDE 비트는 항상 세트된다. 따라서 WDE 비트를 클리어 하기 위해서는 WDRF 비트를 먼저 클리어하여야 한다.
WDP3..0	Watchdog Timer Prescaler : 와치독 타이머의 분주비를 설정한다. 분주비에 따른 타이머 만료 시간은 표 4와 같다.

표 2. WDTCSR 레지스터 비트

하이 퓨즈 비트	WDTCSR 레지스터 비트		모드	타이머 만료시 동작
WDTON	WDE	WDIE		
1	0	0	정지	없음
1	0	1	인터럽트 모드	인터럽트 처리
1	1	0	리셋 모드	시스템 리셋
1	1	1	인터럽트 실행 후 리셋	인터럽트 처리, 시스템 리셋
0	-	-	리셋 모드	시스템 리셋

표 3. 와치독 타이머 만료시 동작

WDP3	WDP2	WDP1	WDP0	WDT 오실레이터 사이클 수	VCC = 5.0V에서의 만료 시간
0	0	0	0	2K (2,048)	16 ms
0	0	0	1	4K (4,096)	32 ms
0	0	1	0	8K (8,192)	64 ms
0	0	1	1	16K (16,384)	0.125 s
0	1	0	0	32K (32,768)	0.25 s
0	1	0	1	64K (65,536)	0.5 s
0	1	1	0	128K (131,072)	1.0 s
0	1	1	1	256K (262,144)	2.0 s
1	0	0	0	512K (524,288)	4.0 s
1	0	0	1	1024K (1,048,576)	8.0 s
1	0	1	0	-	-
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

표 4. 분주비 설정 및 분주비에 따른 만료시간

2. 와치독 타이머 사용하기

와치독 타이머는 시스템의 오류로 의도하지 않게 와치독 타이머의 설정이 바뀌지 않도록 사용

과정에서 정해진 시간과 순서를 따라야만 한다. WinAVR에서 제공하는 와치독 타이머 기능은 인터럽트 기능을 지원하지 않고 시스템 리셋만을 지원하는 단점이 있기는 하지만 정해진 시간과 순서에 맞게 와치독 타이머 기능을 사용할 수 있도록 해주므로 흔히 사용된다.

WinAVR에서 제공하는 와치독 타이머를 사용하기 위해서는 'wdt.h' 파일을 포함시켜야 한다. 'wdt.h' 파일에는 와치독 타이머를 사용하기 위한 3개의 매크로 함수들이 정의되어 있다.

```
wdt_reset();
wdt_disable();
wdt_enable(value);
```

wdt_reset 함수는 와치독 타이머를 리셋하는 함수이다. 와치독 타이머가 만료되기 이전에 타이머를 리셋하지 않으면 마이크로컨트롤러가 리셋된다. wdt_disable 함수는 와치독 타이머를 동작하지 않도록 설정하는 함수이며 wdt_enable 함수는 와치독 타이머를 동작하도록 설정하는 함수이다. 와치독 타이머가 동작하도록 설정하는 경우에는 와치독 타이머의 만료 시간을 설정하여야 하며 만료 시간은 wdt.h 파일에 정의되어 있으며 표 5의 값 중 하나를 지정하면 된다. 표 5에서 알 수 있듯이 상수에 나타나는 시간과 실제 ATmega328에서의 시간이 정확하게 일치하지는 않는다는 점에 유의하여야 한다.

상수	WDP3..0	VCC = 5.0V에서의 만료 시간
WDT0_15MS	0000	16 ms
WDT0_30MS	0001	32 ms
WDT0_60MS	0010	64 ms
WDT0_120MS	0011	0.125 s
WDT0_250MS	0100	0.25 s
WDT0_500MS	0101	0.5 s
WDT0_1S	0110	1.0 s
WDT0_2S	0111	2.0 s
WDT0_4S	1000	4.0 s
WDT0_8S	1001	8.0 s

표 5. 와치독 타이머 만료 시간 상수

먼저 퓨즈 비트에서 와치독 타이머의 설정 상태를 살펴보자. 앞서도 언급한 바와 같이 ATmega328의 초깃값과 아두이노 우노의 초깃값은 와치독 타이머를 사용하지 않는 것으로 퓨즈가 설정되어 있다. 하지만 표 3에서도 알 수 있듯이 퓨즈가 설정되어 있지 않아도 와치독 타이머를 사용할 수 있으며 WinAVR의 와치독 라이브러리 역시 마찬가지이다.

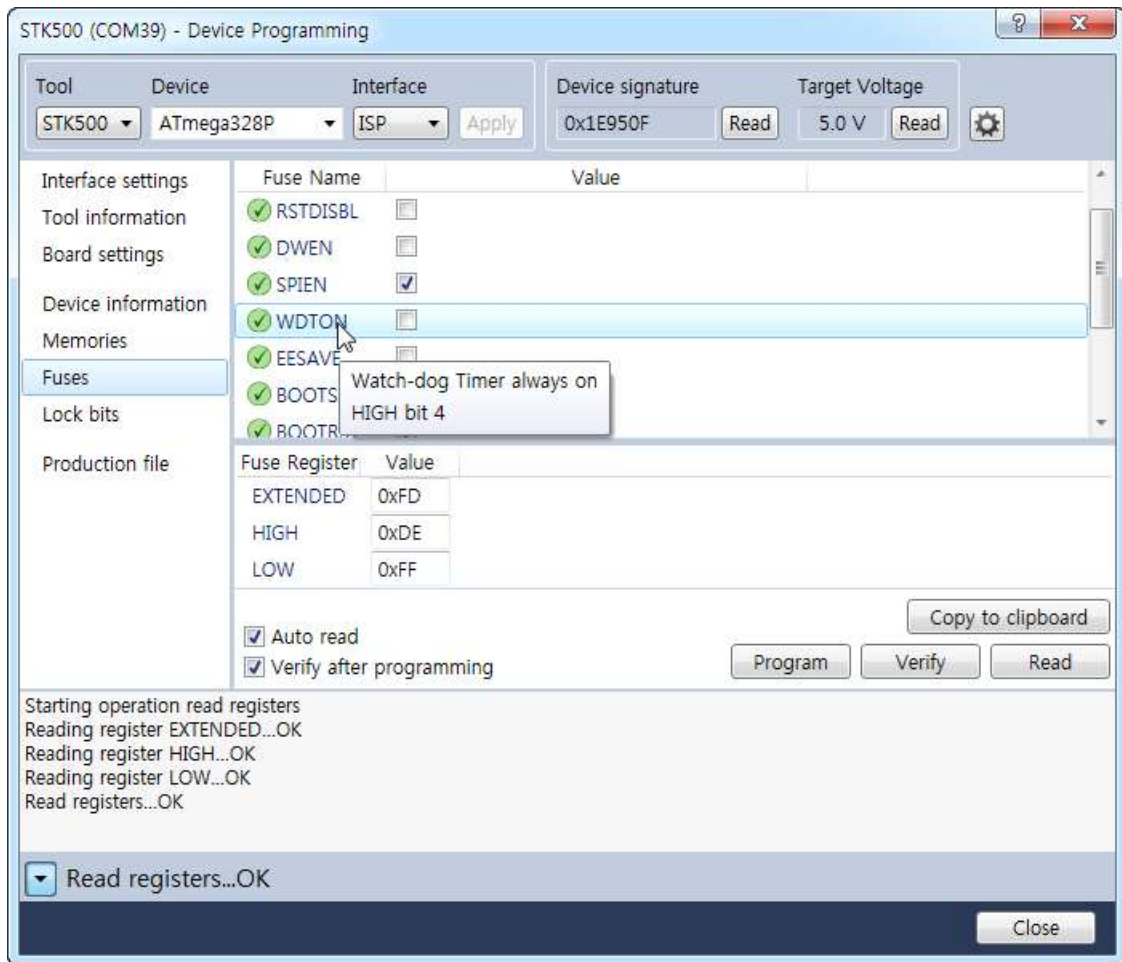


그림 3. 와치독 퓨즈 비트 설정

코드 1은 8초 후에 만료되는 와치독 타이머를 설정하고 와치독 타이머 리셋을 시행하지 않아 8초 후에 시스템이 리셋되는 예이다.

코드 1. 와치독 타이머 만료

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include "UART.h"

int main(void)
{
    wdt_enable(WDTO_8S);           // 왓치독 타이머 만료 시간을 8초로 설정
    UART_INIT();                   // UART 통신 초기화
    UART_printString("** Initialization **\n");

    uint16_t count = 0;

    while(1)
    {
        _delay_ms(1000);           // 1초에 한번 카운트 출력

        count++;
        UART_printString("Count : ");
        UART_print16bitNumber(count);
        UART_transmit('\n');
    }
}
```

그림 4는 코드 1의 실행 결과로 8초 후에 마이크로컨트롤러가 리셋되어 초기화 메시지가 다시 출력되는 것을 확인할 수 있다.

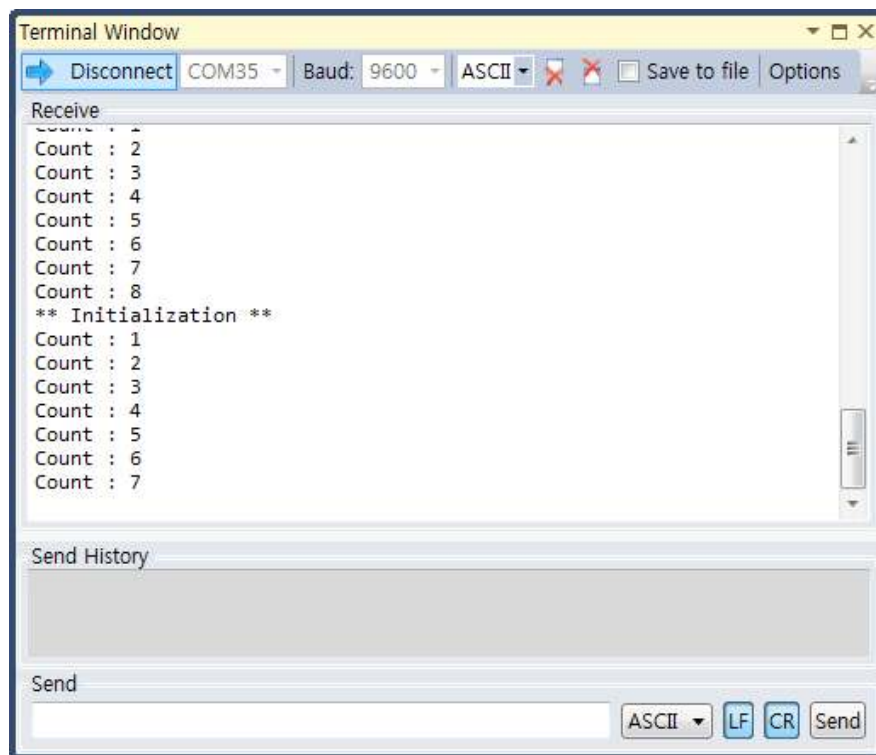


그림 4. 코드 1 실행 결과

코드 1에서 컨트롤러가 리셋되지 않도록 하기 위해서는 8초 이내에 wdt_reset 함수를 호출해 주면 된다. 코드 2는 코드 1에서 매초 wdt_reset 함수를 호출하는 함수를 추가한 것으로 카운터가 정상적으로 동작한다면 마이크로컨트롤러가 리셋되지 않는다는 것을 확인할 수 있다.

코드 2. 와치독 타이머

```

#define F_CPU 16000000L
#include <avr/io.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include "UART.h"

int main(void)
{
    wdt_enable(WDTO_8S);           // 와치독 타이머 만료 시간을 8초로 설정
    UART_INIT();                  // UART 통신 초기화
    UART_printString("** Initialization **\n");
  
```

```
uint16_t count = 0;

while(1)
{
    _delay_ms(1000);                // 1초에 한번 카운트 출력

    count++;
    UART_printString("Count : ");
    UART_print16bitNumber(count);
    UART_transmit('\n');
    wdt_reset();                    // 왓치독 타이머 리셋
}
}
```

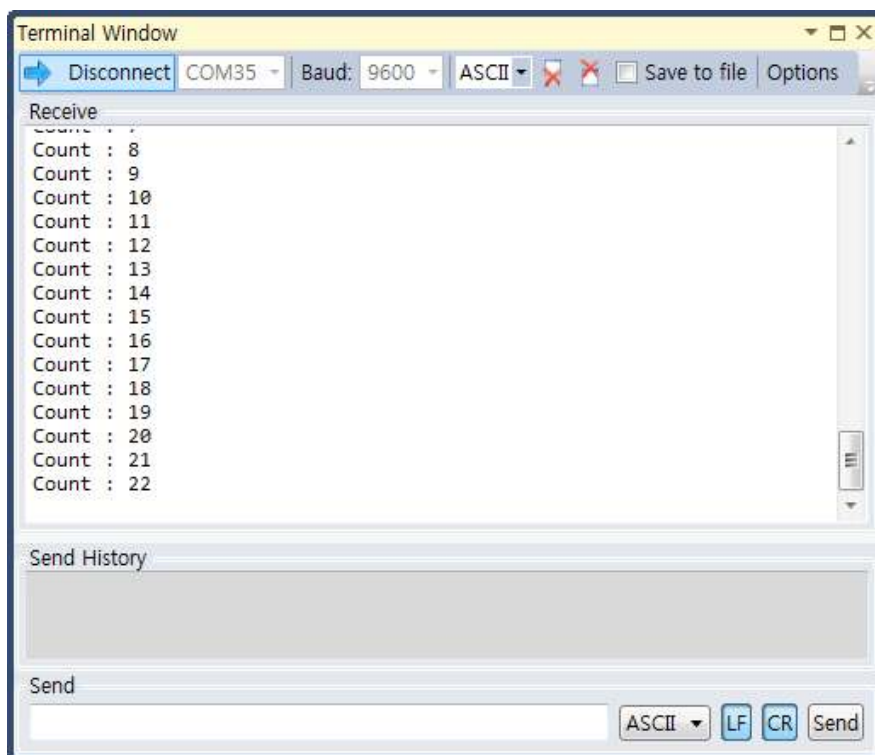


그림 5. 코드 2 실행 결과

3. 왓치독 타이머 사용하기 - 아두이노

아두이노에서도 WinAVR의 와치독 타이머 기능을 사용할 수 있으며 기본적인 사용 방법은 위의 예에서도 동일하다. 코드 3은 와치독 타이머를 리셋하지 않아 8초 간격으로 마이크로컨트롤러가 리셋되는 예이며, 코드 4는 loop 함수에서 와치독 타이머를 리셋하여 마이크로컨트롤러가 리셋되는 것을 방지한 예이다.

코드 3. 와치독 타이머 만료 - 아두이노

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

int count = 0;

void setup(){
    wdt_enable(WDTO_8S);           // 와치독 타이머 만료 시간을 8초로 설정
    Serial.begin(9600);            // 직렬 통신 초기화
    Serial.println("** setup function **");
}

void loop(){
    while(1){                      // 1초에 한번 카운트 출력
        delay(1000);
        count++;
        Serial.println(String("Count : ") + count);
    }
}
```

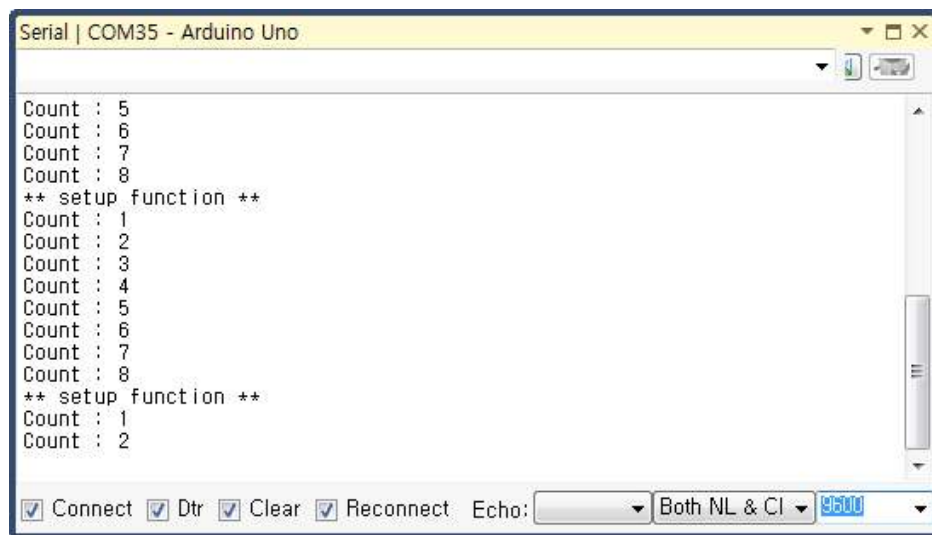


그림 6. 코드 3 실행 결과

코드 4. 와치독 타이머 - 아두이노

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

int count = 0;

void setup(){
    wdt_enable(WDTO_8S);           // 와치독 타이머 만료 시간을 8초로 설정
    Serial.begin(9600);           // 직렬 통신 초기화
    Serial.println("** setup function **");
}

void loop(){
    delay(1000);
    count++;
    Serial.println(String("Count : ") + count);    // 1초에 한번 카운트 출력
    wdt_reset();                                   // 와치독 타이머 리셋
}
```

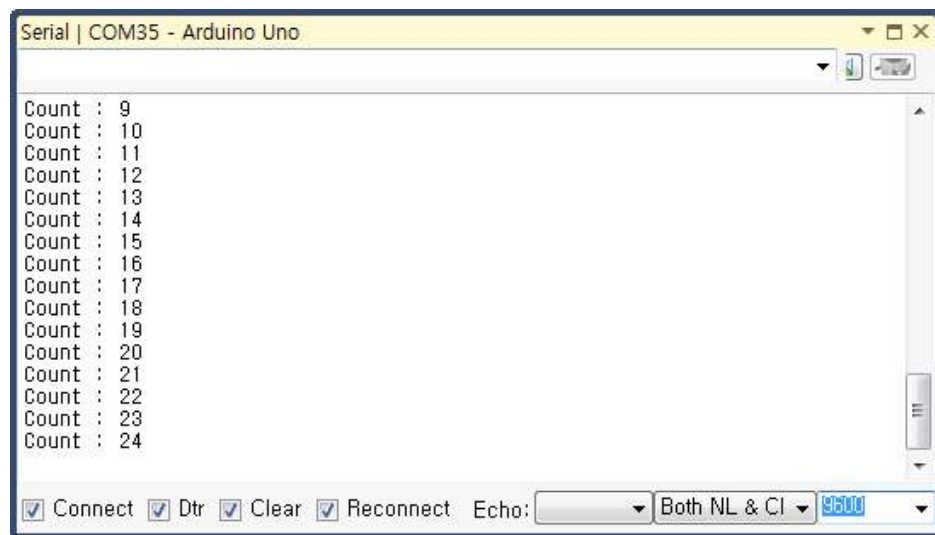


그림 7. 코드 4 실행 결과