TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-05-10 (51회차)

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - 정유경

ucong@naver.com

4. cortex R5 PWM 제어하기

모터는 20ms 즉 50Hz로 구동시킬 수 있지만, 현제 HR Clock이 75Mhz로 설정되어 있다.
따라서 펄스 폭 변조(PWM)을 이용하여 Actual LR Time을 20ms로 제어해주어야 한다.
(LED는 75MHz로 구동이 가능하다)
결국 오늘 한 내용은 MUX로 신호선이 어디로 들어갈지 선택해서
FPGA로 CPU의 타이머를 설계한 것이다.
이를 위하여 High End Timer를 사용한다.
High End Timer를 사용하면 아주 정밀한 제어가 가능하다.
(Het에 대한 자세한 설명은 다음을 참고: http://www.ti.com/lit/an/spna069/spna069.pdf)

HET를 사용하기 위해 다음과 같이 프로젝트를 작성한다.

Code Compiler Studio 에서 hetPWM 프로젝트 생성
HAL Code Generator에서 TMS570 LC4357 프로젝트 생성해서 CCS의 hetPWM과 연결
Het1 driver만 enable 시키고
Pin 0에 대해 HR Share 설정하고 Generate Code한다.
(옵션 include 하는 것 잊지말자: "${workspace_loc:/${ProjName}/include}")

Main에 다음과 같은 코드를 작성한다.

```
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_het.h"
```

```c
int main(void)
{
    hetInit(); // initialize het 1 module
    while(1);
    return 0;
}
```

코드를 분석해보자

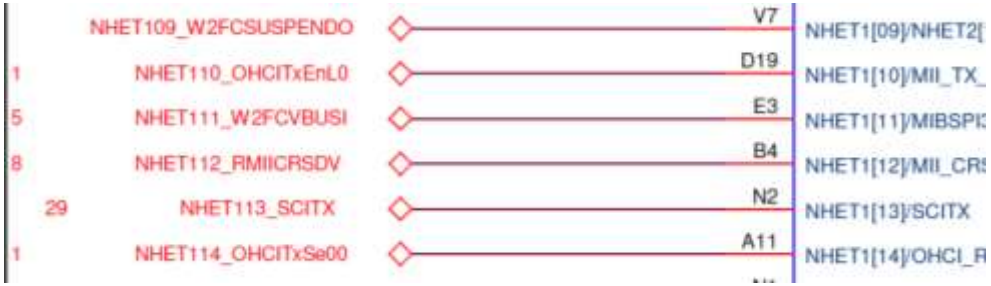
hetInit( ) 함수의 내부를 보면

1. `hetREG1->DOUT = (uint32) 0x00000000U`

HET pin의 출력을 모두 0으로 설정한다.

2. `hetREG1->DIR = (uint32) 0x00000400U`

HETDIR 레지스터(NHET Direction Register)의 10번 비트를 set하여 10번핀을 출력 핀으로 설정한다.
출력핀의 위치는 데이터시트를 보면 확인할 수 있다.
(보드 앞면에 없으면 뒷면을 확인해보자)



3. `hetREG1->PDR = (uint32) 0x00000000U /** - Set HET pins open drain enable */`

`hetREG1->PULDIS = (uint32) 0x00000000U /** - Set HET pins pullup/down enable */`

`hetREG1->PSL= (uint32) 0x00000000U /** - Set HET pins pullup/down select0020*/`

오픈드레인, 풀업/풀다운 저항은 사용하지 않는다.

4. `hetREG1->HRSH= (uint32) 0x0000F00FU  Set HET pins high resolution share`

15	14	13	12	11	10	9	8
HR SHARE31/30	HR SHARE29/28	HR SHARE27/26	HR SHARE25/24	HR SHARE23/22	HR SHARE21/20	HR SHARE19/18	HR SHARE17/16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
HR SHARE15/14	HR SHARE13/12	HR SHARE11/10	HR SHARE9/8	HR SHARE7/6	HR SHARE5/4	HR SHARE3/2	HR SHARE1/0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 23-29. HR Share Control Register (HETHRSH) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reads return 0. Writes have no effect.
15-0	HRSHARE n+1 / n		HR Share Bits Enables the share of the same pin for two HR structures. For example, if bit HRSHARE1/0 is set, the pin HET[0] will then be connected to both HR input structures 0 and 1. **Note:** If HR share bits are used, pins not connected to HR structures (the odd number pin in each pair) can be accessed as general inputs/outputs.
		0	HR Input of HET[n+1] and HET[n] are not shared.
		1	HR Input of HET[n+1] and HET[n] are shared; both measure pin HET[n].

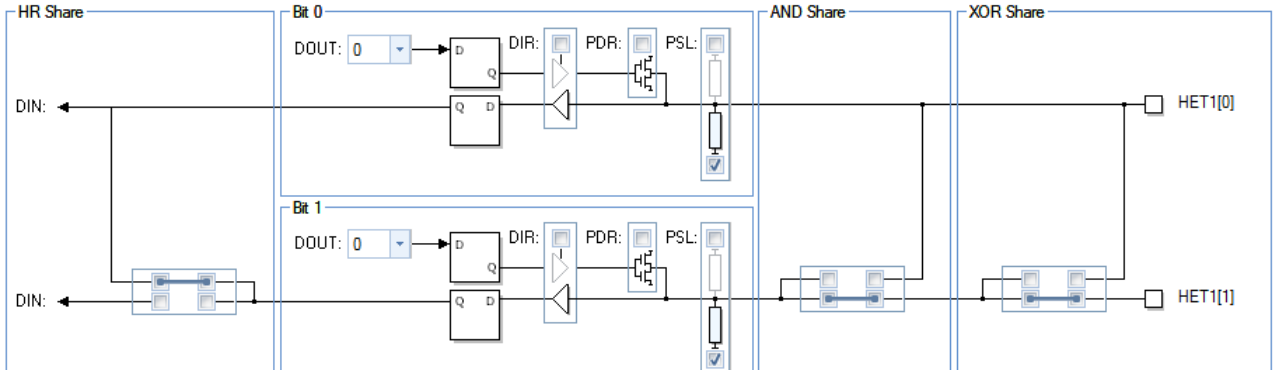
12~15번 0~3번 비트를 1로 지정하고 있다. 이때의 의미는 다음과 같다.

two HR structures to share the same pin for ***input capture** only

따라서 HET pin0~7, Het pin 24~31에서 각각의 n과 n+1 pin에 대해 share하고 있음을 확인할 수 있다.

공유하지 않은 HET pin에 대해서는 GPIO로 동작이 가능하다.

다음 그림처럼 HAL Code Generator에서 확인이 가능하다




```
5.hetREG1->AND  /** - Set HET pins AND share */
hetREG1->XOR /** - Set HET pins XOR share */
```

and 나 xor로 공유하는 핀들은 없다

```
6. hetREG1->PFR = (uint32)((uint32) 6U << 8U) | ((uint32) 0U);
/** - Setup prescaler values (Loop resolution prescaler, High resolution prescaler) */
```

Table 23-17. Prescale Factor Register (HETPFR) Field Descriptions

Bit	Field	Value	Description
31-11	Reserved	0	Reads return 0. Writes have no effect.
10-8	LRPFC		Loop-Resolution Pre-scale Factor Code. LRPFC determines the loop-resolution prescale divide rate (lr).
		0	/1
		1h	/2
		2h	/4
		3h	/8
		4h	/16
		5h	/32
		6h	/64

prescaler값을 설정한다. HETPFR레지스터의 9,10번 비트를 set해주고 있다.

따라서 LR Prescale 값이 64분주라는 의미이다.

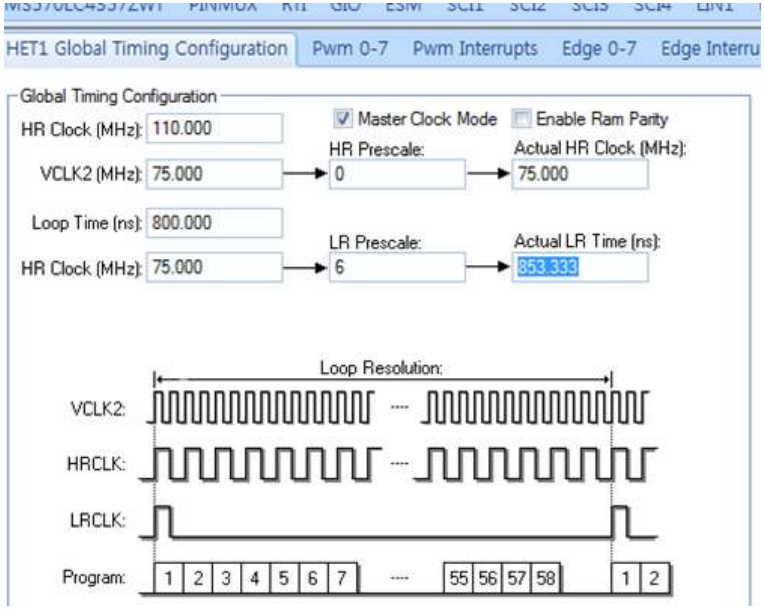
(75Mhz를 64분주한 값의 역수를 취해주면 0.853333333sec 즉, 853.333ns)

HR Clock은 분주없이 VCLK2의 75Mhz를 그대로 사용한다. (HET과 기존 보드의 동기를 매칭시키는 용도로 보인다)

966p 참고

High Resolution Clock: The high resolution clock is the smallest time increment with which a pin can change it's state or can be measured in the case of input signals.
Loop Resolution Clock: The loop resolution clock defines the timebase for executing all instructions in a N2HET program.

HAL Code Generator에서 확인 가능하다. (HR Clock = Loop Time / LR Time임을 볼 수 있다)



7. `hetREG1->PCR = (uint32) 0x00000005U /** - Parity control register - Enable/Disable Parity check */`
패리티 체크를 Disable한다. HET으로 전송하는 과정에서 비트 에러를 검출하지 않겠다는 의미인 것 같다.

8. `(void)memcpy((void *)hetRAM1, (const void *)het1PROGRAM, sizeof (het1PROGRAM));`
`/* memcpy des , src, size */ /** - Fill HET RAM with opcodes and Data */`
hetRAM1에 het1PROG를 복사하여 명령어와 데이터를 전송한다.
HETRAM의 위치는 다음과 같다. `#define hetRAM1 ((hetRAMBASE_t *)0xFF460000U)`

MIBADC1 RAM	Parity	0xFF3E0000	0xFF3FFFFF	8	n/a
NHET2 RAM	Parity	0xFF440000	0xFF45FFFF	15	n/a
NHET1 RAM	Parity	0xFF460000	0xFF47FFFF	3	n/a
HET TU2 RAM	Parity	0xFF4C0000	0xFF4DFFFF	16	n/a

Reference Manual을 확인해보면, HET의 명령어는 독특하게도 96bit이지만(ARM의 명령어는 모두 4바이트였음), 128bit로 정렬된다고 되어있다.
다시 말해서 명령어는 3개의 32bit 필드(프로그램, 제어, 데이터필드)와 128bit경계를 맞춰주기 위한 1개의 32bit reserved필드로 구성된다.
다음과 같은 이유로 위와 같은 설정을 해준 것이 아닌가 싶다.

> **NOTE:** The external host interface supports any access size for reads, but only 32-bit writes to the N2HET RAM are supported. Reserved addresses should not be accessed, the result of doing so is indeterminate.

`hetINSTRUCTION_t` 구조체의 내부를 보면 위에 설명된 내용을 확인할 수 있다.

```
123*   This type is used to access a HET Instruction.
124*/
125typedef volatile struct hetInstructionBase
126{
127    uint32 Program;
128    uint32 Control;
129    uint32 Data;
130    uint32   rsvd1;
131} hetINSTRUCTION_t;
132
```

HET의 12바이트 명령어들은 다음과 같이 따로 정의되어 있다.

이 명령어들을 HET RAM으로 복사하는 초기화 과정을 거친 후 HET에서 프로그램이 실행된다.

23.6 Instruction Set

23.6.1 Instruction Summary

Table 23-73 presents a list of the instructions in the N2HET instruction set. The pages following describe each instruction in detail.

Table 23-73. Instruction Summary

Abbreviation	Instruction Name	Opcode	Sub-Opcode	Cycles[1]
ACMP	Angle Compare	Ch	-	1
ACNT	Angle Count	9h	-	2
ADCNST	Add Constant	5h	-	2
ADC	Add with Carry and Shift	4h	CP[25:23] = 011, C5 = 1	1-3

```
70*   Het program running after initialization.
71*/
72
73static const hetINSTRUCTION_t het1PROGRAM[58U] =
74{
75    /* CNT: Timebase
76    *       - Instruction                 = 0
77    *       - Next instruction            = 1
78    *       - Conditional next instruction = na
79    *       - Interrupt                   = na
80    *       - Pin                         = na
81    *       - Reg                         = T
82    */
83    {
84        /* Program */
85        0x00002C80U,
86        /* Control */
87        0x01FFFFFU,
88        /* Data */
89        0xFFFFFF80U,
90        /* Reserved */
91        0x00000000U
92    },
93    /* PWCNT: PWM 0 -> Duty Cycle
94    *         - Instruction                 = 1
95    *         - Next instruction            = 2
```

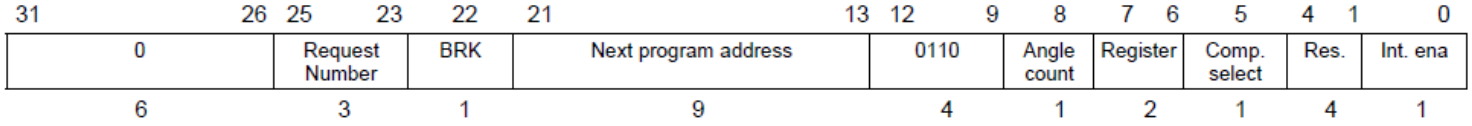
Het1PROGRAM 구조체 배열의 첫번째 요소는 CNT이다. 다음과 같은 값들로 셋팅된다.
CNT 명령어는 가상의 타이머를 정의한다.

```
/* Program */      0x00002C80U, // 10 1100 1000 0000
/* Control */      0x01FFFFFU, // 1 1111 1111 1111 1111 1111 1111
/* Data */         0xFFFFFF80U, // 1111 1111 1111 1111 1111 1111 1000 0000
/* Reserved */     0x00000000U
```

5. CNT 명령어 해석하기

1062p의 23.6.2 Abbreviations, Encoding Formats and Bits를 참고하여 해석할 수 있다.

Figure 23-134. CNT Program Field (P31:P0)



프로그램필드는 P13, [P11:P10] P7번 비트가 셋팅되어 있다.

Request Number Defines the number of the request line (0,1,..,7) to trigger either the HTU or the DMA. Default: 0 0설정

BRK Defines the software breakpoint for the device software debugger. Default: OFF 0설정

Next Program Address Defines the program address of the next instruction in the program flow. This value may be a label or an 9-bit unsigned integer. Default: Current instruction + 1

[P21:P13] 에 1이 지정되어 있으므로 다음 명령어는 Het1PROGRAM 구조체 배열의 두번째 요소인 PWCNT이다.

Request Allows to select between no request (NOREQ), request (GENREQ) and quiet request(QUIET).

0이므로 NOREQ

[P11:P10] 은 0110으로 이 명령어를 특정지을 수 있는 Opcode가 들어있다. (Ref Manual 1060p에서 확인가능)

P8이 0이므로 time mode이다.

Register Register select: Selects the register for data comparison and storage Default: No register (None)

P7이 1이므로 R,S 레지스터가 선택된다.

comp_mode* Specifies the compare mode. This field is used with the 64-bit move instructions. Thisfield ensures that the sub-opcodes are moved correctly. Default: ECMP

0이므로 ECMP

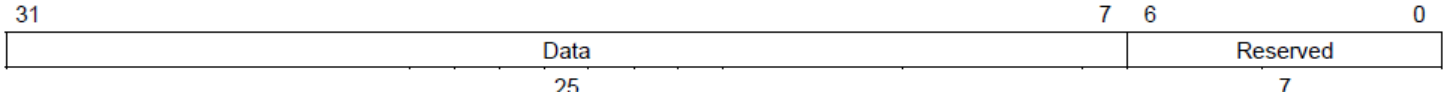
Figure 23-135. CNT Control Field (C31:C0)

컨트롤필드는 [C24:C0] 비트가 1로 셋팅되어 있다.

Maxcount: 데이터필드에서 허용가능한 max count값 (25-bit integer value)

나머지는 모두 0이므로

Figure 23-136. CNT Data Field (D31:D0)



데이터 필드는 31~7번비트에 모두 1이 셋팅되어 있다.

명령어가 실행될 때 1씩 증가되는 25-bit integer value이다. (Default: 0)

control field에 지정되어 있는 Max count값에 도달하면 카운터는 리셋된다.

(time mode (angle count bit [P8] = 0)에서는 1cycle, angle mode (angle count bit [P8] = 1)에서는 2cycle이 필요하다.)

PMW Input Capture Mode란

Input Mode중 PWM Input Capture모드는 Pulse의 Rising, Falling Edge때 CNT값을 CCR1, 2 또는 CCR3, 4에 각각 저장(Capture)하여 채널 입력 신호의 펄스이나 PWM 주기를 측정할 수 있다. 또한 이러한 Capture동작때 Interrupt가 발생한다.

CH1의 Input Capture시 그 동작 절차는 다음과 같다.

- CH1의 Falling Edge에 CC1R에 Capture되게 한다.
- CH1의 Rising Edge에는 CC2R에 Capture가 발생되게 한다.
- 채널1, 2, 3, 4 가 각각 IC1, 2, 3, 4 신호를 발생 시키게 한다. 또한 반대로 CH1, CH30이 IC2, IC4를, CH 2, CH4가 IC 1, IC3을 발생 시키게도 할 수 있다.
- Input Filter는 4개의 채널로 특정 값을 Masking하여 Capture를 하도록 한다. 사용하지 않으므로 0으로 하여 모든 상태를 통과 시키면 된다.

1. 아두이노 서보모터돌리기

```
#include <Servo.h>
#define DT   100.0          // Delay Time 100ms
#define DTHETA   20.0       // 각도 변화량 20
Servo myservo;              // 서보를 하나 만든다
int theta = 0;              // 초기값
double omega;               // 각속도
double alpha;               // 각가속도
double velocity;            // 속도
double acceleration;        // 가속도
double dt = DT / 1000.0;  // 0.1ms
double time = 0.0;          // 누적시간
void setup()        // 초기화 부분, 한번만 실행
{
```

```
  Serial.begin(9600);       // UART 통신속도, 일반적으로 9600 Baud Rate 사용
  myservo.attach(9);        // myservo(사용할 servo이름)를 9번핀으로 컨트롤 (9번핀을  PWM핀으로 쓴다)
}

void loop()         // 주처리부분, 반복실행
{
  Serial.println((double)(DTHETA/ (500.0/1000.0)));    // 시리얼 모니터로 2xDTHETA 출력
  for(theta=0; theta < 180; theta +=DTHETA) // theta를 DTHETA만큼 증가시키면서 루프를 돈다
  (theta+=1 할때보다 속도가 빠르다)
  {
    myservo.write(theta);             // 서보모터의 각도를 theta로 설정
    delay(DT);                        // 100ms 지연
    Serial.print("Theta=");           // 시리얼 모니터로 출력한다
    Serial.println(theta);            // 0
    Serial.print("DTheta=");
    Serial.println(DTHETA);           // 20
    time+=dt;                         // 0.1ms 증가 (sampling time, 미소시간변화량)
    Serial.print("Total time = ");
    Serial.println(time);
    Serial.print("dt=");              // 0.1ms
    Serial.println(dt);
    Serial.print("Omega =");          // 각속도 오메가는 DTHETA/dt, 즉 미소시간에 대한 위상변화량
    Serial.println((double)(DTHETA)/dt);      // 20 / 0.1ms = 200 (but 각속도의 단위는 rad/sec이므로 코드수정이 필요하다)
    Serial.println();
    // delay(1500);          // 1500ms 지연
   }
```

```
    for(theta=180; theta >=1; theta -=DTHETA)
    {
      myservo.write(theta);
      delay(DT);
    }
}
```

시리얼 모니터로 다음과 같은 출력을 확인할 수 있다.

40.00 Theta=0 DTheta=20.00 Total time = 0.10 dt=0.10 Omega =200.00 Theta=20 DTheta=20.00 Total time = 0.20 dt=0.10 Omega =200.00 Theta=40	Theta=80 DTheta=20.00 Total time = 0.50 dt=0.10 Omega =200.00 Theta=100 DTheta=20.00 Total time = 0.60 dt=0.10 Omega =200.00 Theta=120 DTheta=20.00	Theta=160 DTheta=20.00 Total time = 0.90 dt=0.10 Omega =200.00

DTheta=20.00 Total time = 0.30 dt=0.10 Omega =200.00 Theta=60 DTheta=20.00 Total time = 0.40 dt=0.10 Omega =200.00	Total time = 0.70 dt=0.10 Omega =200.00 Theta=140 DTheta=20.00 Total time = 0.80 dt=0.10 Omega =200.00	

2. 아두이노 Random Number

```
int randNumber;
void setup()
{
   Serial.begin(9600);
   Serial.println("Print Random Numbers 0~9");
   for(int i=0; i<20 ;i++)
   {
     randNumber = random(10);      // 0~9의 난수발생
     Serial.print(randNumber);
     Serial.print(" ");
   }

   Serial.println();
```

```
  Serial.println("Print Random Numbers 2~9");

  for(int i=0; i<20 ;i++)
  {
    randNumber = random(2,10);  // 2~9의 난수발생
    Serial.print(randNumber);
    Serial.print(" ");
  }

randomSeed(analogRead(0)); // 아날로그 A0 핀의 값을 읽어 이를 난수의 seed로 사용
Serial.println();
Serial.println("Print Random Numbers 0~9");

for(int i=0; i<20; i++)
{
  randNumber = random(10);
  Serial.print(randNumber);
  Serial.print(" ");
}
Serial.println(" ");
Serial.println(" ");
}

void loop()
{
}
```

Random()함수의 경우, 주어진 인자값의 개수만큼 난수가 발생한다.
random(max), random(min, max)으로 사용하게 되는데 random(100)이면 0을 포함한 0~99사이의 난수가 발생하게 된다.
하지만 이 랜덤은 미리 정해져 있는 무작위 숫자의 연속을 순서대로 출력하는 의사 난수(pseudo random number)이다.
따라서 아두이노의 리셋버튼을 누르거나 다시 시작하면 똑 같은 순서의 랜덤값을 출력한다.
Randomseed() 함수를 이용해서 이 문제를 해결할 수 있다.
Randomseed는 연속된 무작위 숫자의 어느곳부터 차례로 불러올지 결정한다.
함수의 인자값으로는 analogread(0)를 사용하였다.
analogread함수는 아날로그 핀의 값을 읽어오는 함수인데, analogread(0)또는 analogread(A0)는 0번 핀의 값을 읽어오겠다는 의미이다.
그런데 현재 구성한 회로에서는 아날로그 0번핀에 아무것도 연결하지 않았다. 이때 핀은 플로팅 상태가 된다.
이렇게 별도의 연결이 없이 아날로그 핀의 값을 읽어오게 되면 0~1023중 무작위의 값을 반환하게 된다.
3. 아두이노 수치미분(극한을 적용하지 않고 미소시간단위에 대해서 미분함)

```
#include <Servo.h>
#define DT 50.0              // 50ms
Servo myservo;
double pi = 3.14159265358979323846264338327950288419716939937510582O9;
int theta =0;
double d_theta = 0.0;        // 각도 변화량

double radian = 0.0;         // 호도법에 (pi/180)을 곱한 결과값 즉, 라디안 값으로 변경하여 저장할 변수
double omega = 0.0;          // 각속도
double d_omega = 0.0;        // 각속도 변화량

double alpha = 0.0;          // 각가속도
double velocity;             // v = rw (속도는 변위벡터와 각속도의 곱으로 나타낼 수 있다)
```

```
double acceleration;          // a = vw

double dt = DT / 1000.0;  // 0,05ms의 샘플링 타임
double time = 0.0;            // 전체 누적 시간

void setup()
{
      Serial.begin(9600);
      randomSeed(analogRead(0));
      myservo.attach(9);
}

void loop()
{
   while(theta < 180)
   {
      myservo.write(theta);
      delay(DT);

      Serial.print("Theta =");
      Serial.println(theta);    // 초기값 0 // 15
      Serial.print("DTheta =");
      Serial.println(d_theta); // 초기값 0 // 5
      Serial.print("Radian = ");
      radian = (d_theta / 360)*2*pi;
      Serial.println(radian);
```

```
  time += dt;               //+0.05ms
  Serial.print("Total Time = ");
  Serial.println(time);
  Serial.print("dt = ");
  Serial.println(dt);
  Serial.print("Omega= ");
  d_omega = (radian/dt) - omega;
  omega = radian /dt;
  Serial.println(omega);
  Serial.print("DOmega = ");       // -값: 반대방향으로 토크가 전달된다
  Serial.println(d_omega);
  Serial.print("Velocity = ");
  velocity = 0.01815 * omega;    // 모터의 실제 회전반경: 0.01815
  Serial.println(velocity);
  Serial.print("Acceleration = ");
  acceleration = 0.01815 * omega * omega;
  Serial.println(acceleration, 10); // 십진수로 출력한다는 의미 (but, 10지정하지 않아도 디폴트가 십진수)
  Serial.print("Alpha = ");
  alpha = d_omega / dt;
  Serial.println(alpha);
  // 토크는 각가속도와 관성모멘트의 곱이므로 관성모멘트 I =mr^2 를 이용하면 토크를 구할 수 있다
  Serial.println();

  d_theta = random(1,11);
  theta += d_theta;
}
```

```
    for(theta = 180; theta >=1; theta -= random(1,11))
    {
      myservo.write(theta);
      delay(DT);
    }
}
```