Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

*이두이노 – M4

```
#include <Servo.h>

Servo myservo;

int pos = 0;

void setup()
{
  myservo_attach(9);
}

void loop(){
  //put your main code here. to run repeatedly;
  for(pos =0;pos < 180;pos+=1)
  {
    myserv0, write(pos);
    delay(15);
  }

  for(pos = 180; pos>=1;pos-=1)
  {
    myserv0,write(pos);
    delay(15);
  }
}
```

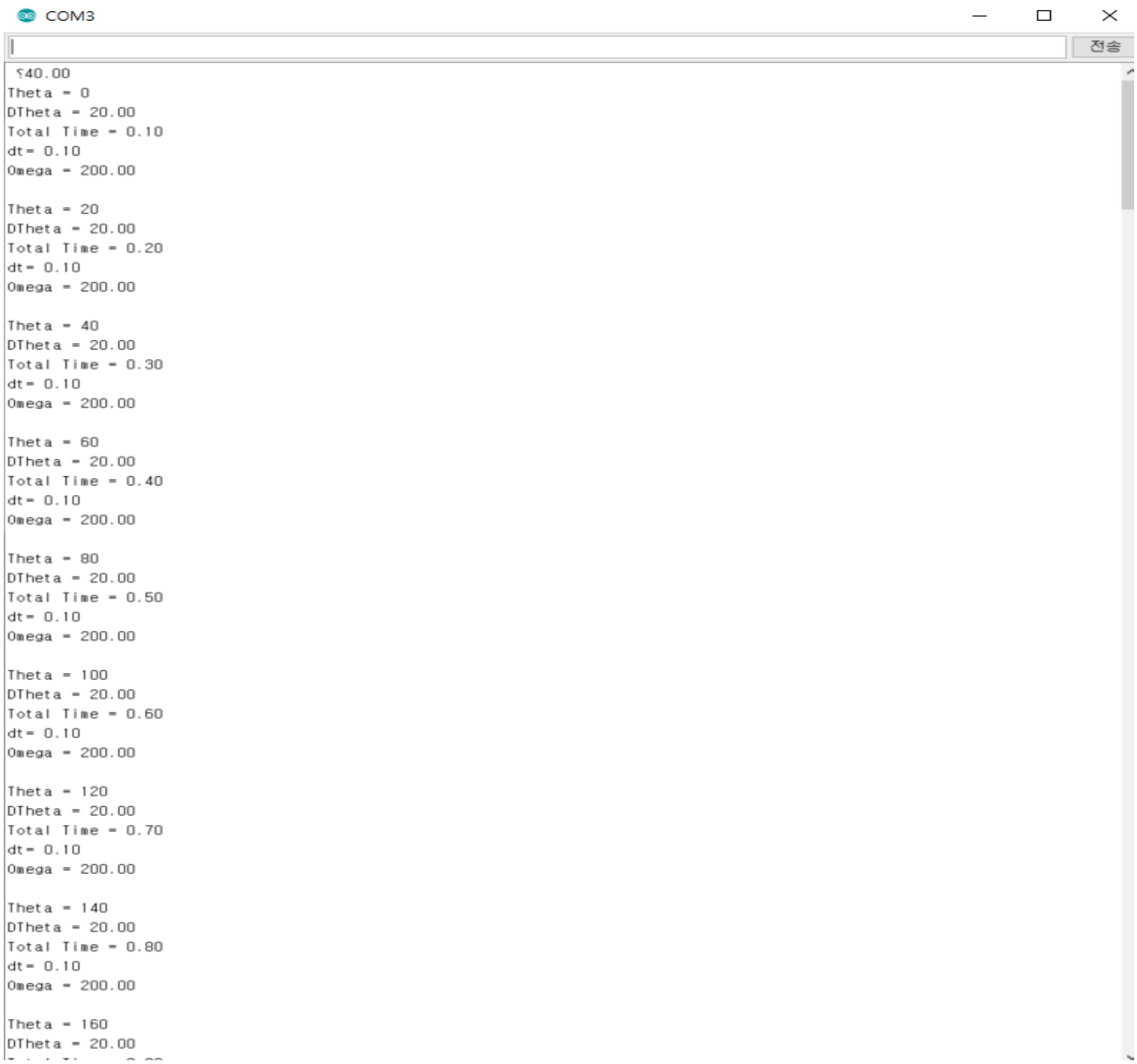
서보 모터는 정확한 각도 회전을 위해 사용된다는 점은 스텝모터와 비슷하지만 구동되는 방식은 전혀 다르다.

서보 모터는 일반적으로 모터 하나만을 가지고 서보라고 하지는 않는다. 서보(Servo)라는 말 자체가 '하인'이라는 말에서 유래되어 따르다, 추종하다 등의 의미를 담고 있다고 하여 제어 명령에 따라 정확한 위치와 속도를 맞출 수 있는 모터를 서보모터라고 한다. 서보모터는 원하는 각도만큼 회전이아니라 위치시킨다고 보면 된다. 제어방법은 DC모터와 같은 PWM이지만, 서보 모터의 PWM은 주파수가 정해져있으며, 듀티비라기보다는 신호의 유지 시간으로 회전 각도가 결정된다.

* 아두이노로 서보모터 제어하기.

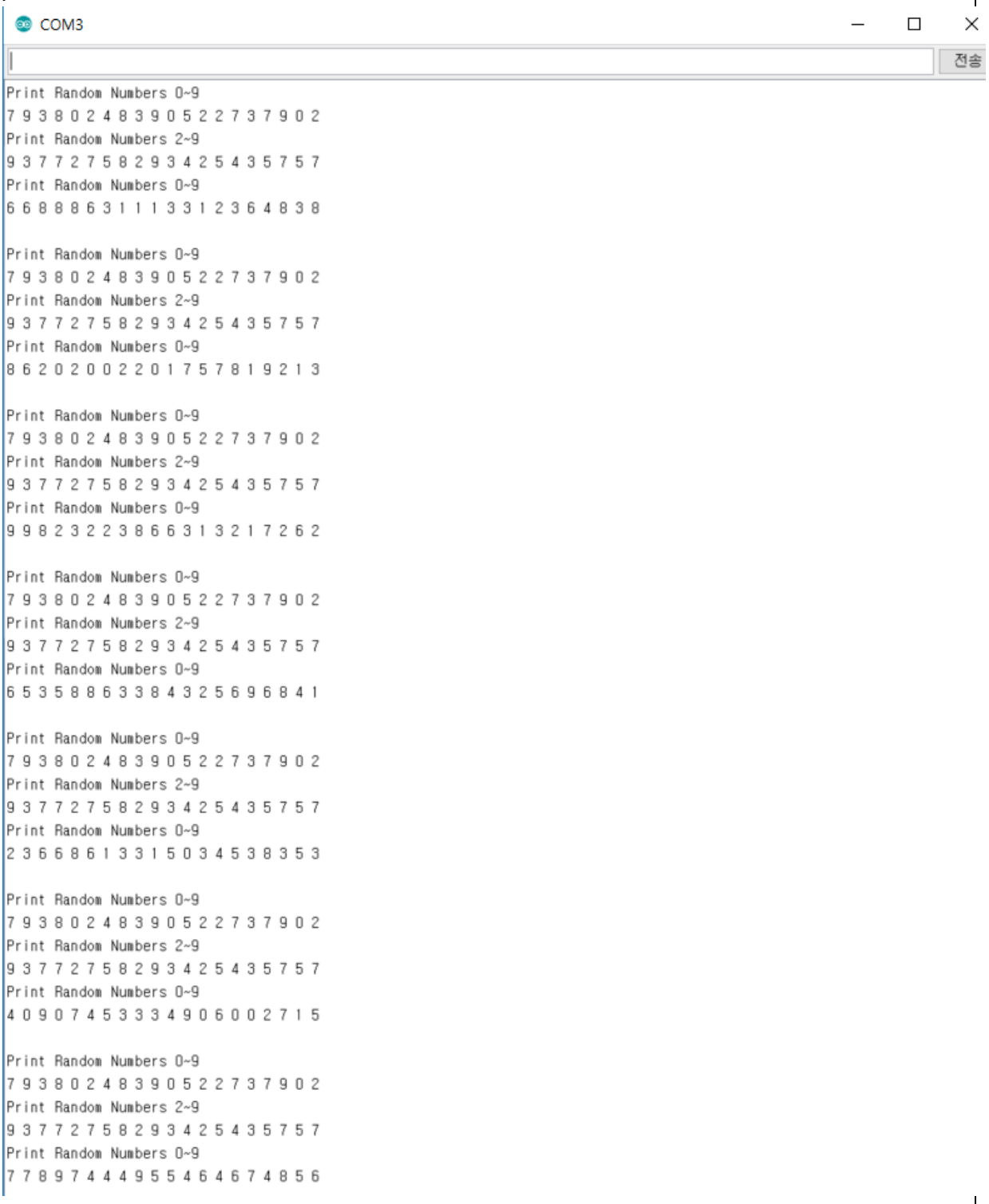
```
#include <Servo.h>

#define DT 100.0          // 0.1 sec
#define DTHETA 20.0     // 각도의 변화량

Servo myservo;            // 서보모터제어하기위한 servo 객체 생성

int theta = 0;
double omega;             // 각속도
double alpha;             // 각가속도
double velocity;          // 속도
double acceleration;      // 가속도

double dt = DT/1000.0;     // 0.1
double time = 0.0;

void setup() {
   Serial.begin(9600);   // Serial.begin(9600) 명령은 시리얼통신을 시작
   myservo.attach(9);   // 서보모터를 9번핀으로 설정
}

void loop() {
     Serial.println((double)(DTHETA / (500.0/1000.0)));
   for(theta = 0; theta < 180; theta += DTHETA)
    {
            myservo.write(theta);   //서보모터를 지정한각도로 회전
            delay(DT);                  //0.1초 delay
            Serial.print("Theta = ");
            Serial.println(theta);
            Serial.print("DTheta = ");
            Serial.println(DTHETA);
            time += dt;                 // 0.1
            Serial.print("Total Time = ");
            Serial.println(time);
            Serial.print("dt= ");
            Serial.println(dt);
            Serial.print("Omega = ");
```

```
            Serial.println((double)(DTHETA) / dt);
            Serial.println();
            //delay(1500);
    }

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



```
int randNumber;

void setup()
{
  Serial.begin(9600);
  Serial.println("Print Random Numbers 0~9");

  for (int i = 0; i < 20; i++)
  {
    randNumber = random(10);  // random이라는 함수에 지정되어있는 값(곱하기 더하고 빼고 어쩌고 저쩌고해서 나온 값이 랜덤한수 0~9까지 나옴. 그래서 항상 같다.
    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); // 마찬가지.
    Serial.print(randNumber);
    Serial.print(" ");
  }

  randomSeed(analogRead(0)); // 지금 시간을 초기값으로 넣어줌으로써 다음 랜덤값이 계속변할 수 있게 해준다.
  Serial.println();
  Serial.println("Print Random Numbers 0~9");

  for (int i = 0; i < 20; i++)
  {
    randNumber = random(10);  // 여기서부터 random값이 계속 바뀌는 것을 볼 수 있다.
    Serial.print(randNumber);
    Serial.print(" ");
  }
  Serial.println();
  Ser
void loop()
```

```
{

}
```


COM3
전송
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
6 6 8 8 8 6 3 1 1 1 3 3 1 2 3 6 4 8 3 8
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
8 6 2 0 2 0 0 2 2 0 1 7 5 7 8 1 9 2 1 3
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
9 9 8 2 3 2 2 3 8 6 6 3 1 3 2 1 7 2 6 2
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
6 5 3 5 8 8 6 3 3 8 4 3 2 5 6 9 6 8 4 1
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
2 3 6 6 8 6 1 3 3 1 5 0 3 4 5 3 8 3 5 3
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
4 0 9 0 7 4 5 3 3 3 4 9 0 6 0 0 2 7 1 5
Print Random Numbers 0~9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print Random Numbers 2~9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print Random Numbers 0~9
7 7 8 9 7 4 4 4 9 5 5 4 6 4 6 7 4 8 5 6

-Pi : 각도 계산

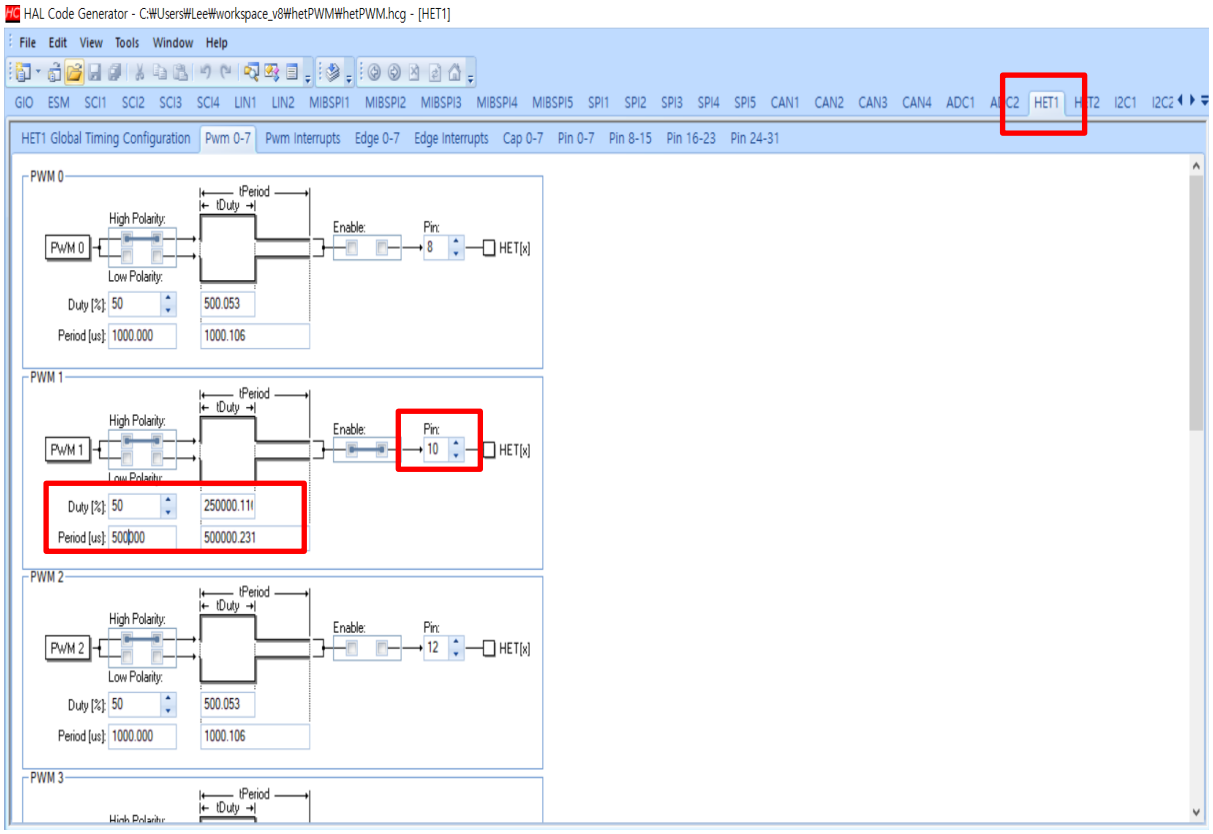
```
#include <Servo.h>

#define DT 50.0
Servo myservo;
double pi = 3.14159265358979323846264338327950288419716939937510582O9;
//정밀도를 높이기 위해 길게 씀
int theta = 0;
double d_theta = 0.0;

double radian = 0.0;
double omega = 0.0;
double d_omega = 0.0;

double alpha = 0.0;
double velocity;
double acceleration;

double dt = DT/1000.0;    //  샘플링타임 0.05초
double time = 0.0;

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

void loop()
{
  while(theta < 180)
  {
    myservo.write(theta);
    delay(DT); //0.05초 기다렸다가

    Serial.print("Theta = ");
    Serial.println(theta);
```

```
    Serial.print("DTheta = ");
    Serial.println(d_theta);

    Serial.print("Radian = ");
    radian = (d_theta / 360)*2*pi;
    Serial.println(radian);
    time += dt;                          //

    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 = ");
    Serial.println(d_omega);

    Serial.print("Veloctity =");
    velocity = 0.01815*omega;
    Serial.println(velocity);

    Serial.print("Acceleration = ");
    acceleration = 0.01815*omega*omega;
    Serial.println(acceleration,10);
    Serial.print("Alpha = ");
    alpha = d_omega / dt;
    Serial.println(alpha);

    Serial.println();

    d_theta = random(1,11);
    theta += d_theta;
  }
  for(theta = 180; theta >= 1; theta -= random(1,11))
```

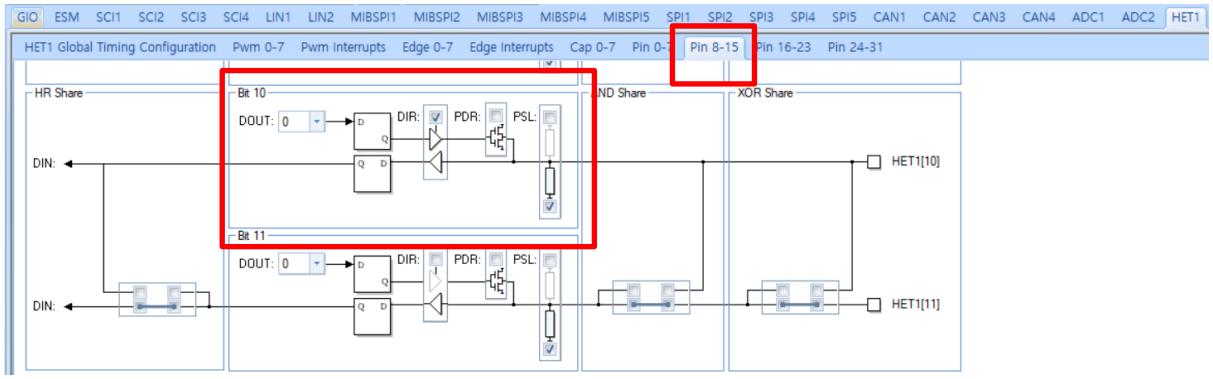
```
  {
    myservo.write(theta);
    delay(DT);
  }
}
```

Cortex R5 - PWM 제어

-HAL code generator



➔ PWM1 , Pin 10번으로 연결 pin 10번 을본다. Duty 50, period 1000 으로 설정.



➔ Pin 10 -> direct 로 연결

F5 generate code

-CCS

Main code

```
69 int main(void)
70 {
71 /* USER CODE BEGIN (3) */
72       hetInit();
73       while(1);
74 /* USER CODE END */
75
76     return 0;
77 }
78
```

➔ 실험 결과: LED가 계속 켜져있는 것을 볼 수 있다. 껐다켜졌다 하는 것이지만, 눈으로 확인불가.. 주기 : 0.001초

그러나, Duty: 50% period: 500ms 로 설정했을 때는 0.5초마다 깜빡이는 것을 볼 수 있었다.

주기: 0.5초

➔ 이번에는 Duty 를 바꿔보았다. 20으로 설정했을 때 LED가 꺼져있는 시간이 더 길었지만, 80으로 설정했을 때는 LED가 켜져있는 시간이 더 길었다.

!! 이제 Code를 분석해보자. !!

1. hetInit

```
/** - Set HET pins default output value */
1.1  hetREG1->DOUT = 0x00000000U
```

23.4.20 N2HET Data Output Register (HETDOUT)

N2HET1: offset = FFF7 B854h; **N2HET2:** offset = FFF7 B954h

Figure 23-75. N2HET Data Output Register (HETDOUT)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HETDOUT															
R/W-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HETDOUT															
R/W-0															

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

Table 23-36. N2HET Data Output Register (HETDOUT) Field Descriptions

Bit	Field	Value	Description
31-0	HETDOUT[n]		Data out write. Writes to this bit will only take effect when the pin is configured as an output. The current logic state of the pin will be displayed by this bit even when the pin state is changed by writing to HETDSET or HETDCLR.
		0	Pin HET[n] is at logic low (0).
		1	Pin HET[n] is at logic high (1) if the HETPDR[n] bit = 0 or the output is in high-impedance state if the HETPDR[n] bit = 1.

: pint HET[n] 은 logic low(0)으로 설정한다. 초기값을 모두 0으로 초기화 한다.

```
/** - Set HET pins direction */
1.2 hetREG1->DIR = 0x00000400U   //10번 bit
```

23.4.18 NHET Direction Register (HETDIR)

N2HET1: offset = FFF7 B84Ch; **N2HET2:** offset = FFF7 B94Ch

Figure 23-73. N2HET Direction Register (HETDIR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HETDIR															
R/W-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HETDIR															
R/W-0															

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

Table 23-34. N2HET Direction Register (HETDIR) Field Descriptions

Bit	Field	Value	Description
31-0	HETDIR[n]		Data direction of NHET pins
		0	Pin HET[n] is an input (and its output buffer is tristated).
		1	Pin HET[n] is an output.

NOTE: Table 23-9 shows how the register bits of DIR, PULDIS and PULSEL are affecting the N2HET pins.

:pin HET[10]을 output으로 설정하겠다. 나머지는 input.

```
/** - Set HET pins open drain enable */
1.3 hetREG1->PDR = 0x00000000U
```

23.4.23 N2HET Open Drain Register (HETPDR)

Values in this register enable or disable the open drain capability of the data pins.
N2HET1: offset = FFF7 B860h; **N2HET2:** offset = FFF7 B960h

Figure 23-78. N2HET Open Drain Register (HETPDR)

31	16
HETPDR	
R/W-0	
15	0
HETPDR	
R/W-0	

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

Table 23-39. N2HET Open Drain Register (HETPDR) Field Descriptions

Bit	Field	Value	Description
31-0	HETPDR[n]		Open drain control for HET[n] pins
		0	The pin is configured in push/pull mode.
		1	The pin is configured in open drain mode. The HETDOUT register controls the state of the output buffer:
			HETDOUT[n] = 0 The output buffer of pin HET[n] is driven low.
			HETDOUT[n] = 1 The output buffer of pin HET[n] is tristated.

: 0번이면 push/pull mode, 1번이면 Open drain mode 로 동작.

BJT, MOSFET → 전자회로

```
/** - Set HET pins pullup/down enable */
1.4 hetREG1->PULDIS = 0x00000000U;
```

23.4.24 N2HET Pull Disable Register (HETPULDIS)

Values in this register enable or disable the pull-up/-down functionality of the pins.
N2HET1: offset = FFF7 B864h; **N2HET2:** offset = FFF7 B964h

Figure 23-79. N2HET Pull Disable Register (HETPULDIS)

31	16
HETPULDIS	
R/W-n	
15	0
HETPULDIS	
R/W-n	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; n is device dependent, see device specific data sheet

Table 23-40. N2HET Pull Disable Register (HETPULDIS) Field Descriptions

Bit	Field	Value	Description
31-0	HETPULDIS[n]		Pull disable for N2HET pins
		0	The pull functionality is enabled on pin HET[n].
		1	The pull functionality is disabled on pin HET[n].

NOTE: See device data sheet for which pins provide programmable pullups/pulldowns.
Table 23-9 shows how the register bits of HETDIR, HETPULDIS, and HETPSL are affecting the N2HET pins.

:pin HET[31~0] = pin pull 을 eable 하겠다. → push/.pull 중에 pull up..

/** - Set HET pins pullup/down select */

1.5 hetREG1->PSL = 0x00000000U;

23.4.25 N2HET Pull Select Register (HETPSL)

Values in this register select the pull-up or pull-down functionality of the pins.
N2HET1: offset = FFF7 B868h; **N2HET2:** offset = FFF7 B968h

Figure 23-80. N2HET Pull Select Register (HETPSL)

31	16
HETPSL	
R/W-0	

15	0
HETPSL	
R/W-0	

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

Table 23-41. N2HET Pull Select Register (HETPSL) Field Descriptions

Bit	Field	Value	Description
31-0	HETPSL[n]		Pull select for NHET pins
		0	The pull down functionality is enabled if corresponding bit in HETPULDIS is 0.
		1	The pull up functionality is enabled if corresponding bit in HETPULDIS is 0.

NOTE: See device data sheet for which pins provide programmable pullups/pulldowns.

Table 23-9 shows how the register bits of HETDIR, HETPULDIS and HETPSL are affecting the N2HET pins.

The information of this register is also used to define the pin states after a parity error:

After a parity error all N2HET pins, which are

1. Defined as output pins in the HETDIR register
2. Not defined as open drain pins (with the HETPDR register)
3. Selected with the HETPPR register, will remain outputs, but automatically change their levels in the following way:
   - If the HETPSL register specifies 0 for the pin, it will switch to low level.
   - If the HETPSL register specifies 1 for the pin, it will switch to high level.

This behavior is independent of the value, which register HETPULDIS specifies for the corresponding pin.

: Pull down 사용

/** - Set HET pins high resolution share */

1.6 hetREG1->HRSH = 0x0000F00F; // 0~3 , 12~15

23.4.13 HR Share Control Register (HETHRSH)

N2HET1: offset = FFF7 B834h; **N2HET2:** offset = FFF7 B934h

Figure 23-68. HR Share Control Register (HETHRSH)

31	16
Reserved	
R-0	

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

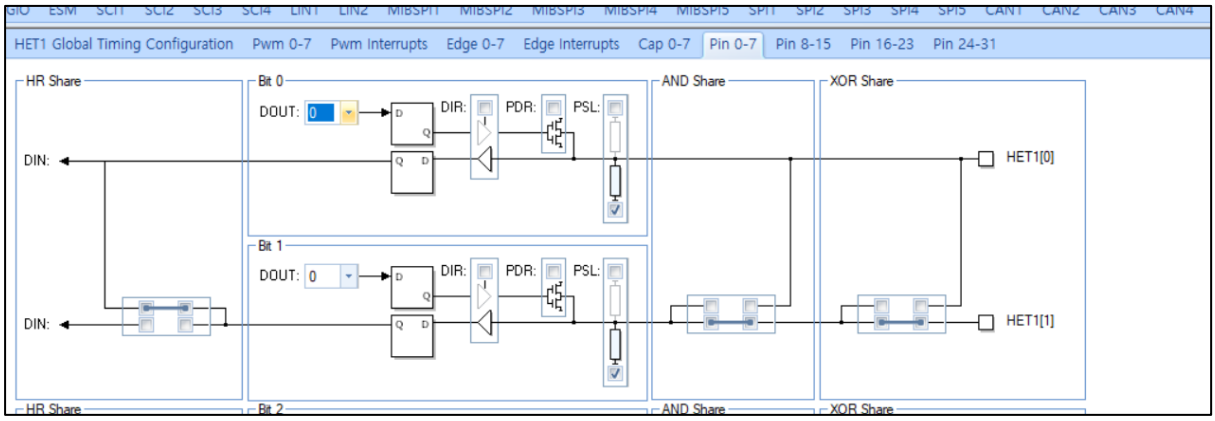
7	6	5	4	3	2	1	0
HR SHARE15/14	HR SHARE13/12	HR SHARE11/10	HR SHARE9/8	HR SHARE7/6	HR SHARE5/4	HR SHARE3/2	HR SHARE1/0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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].

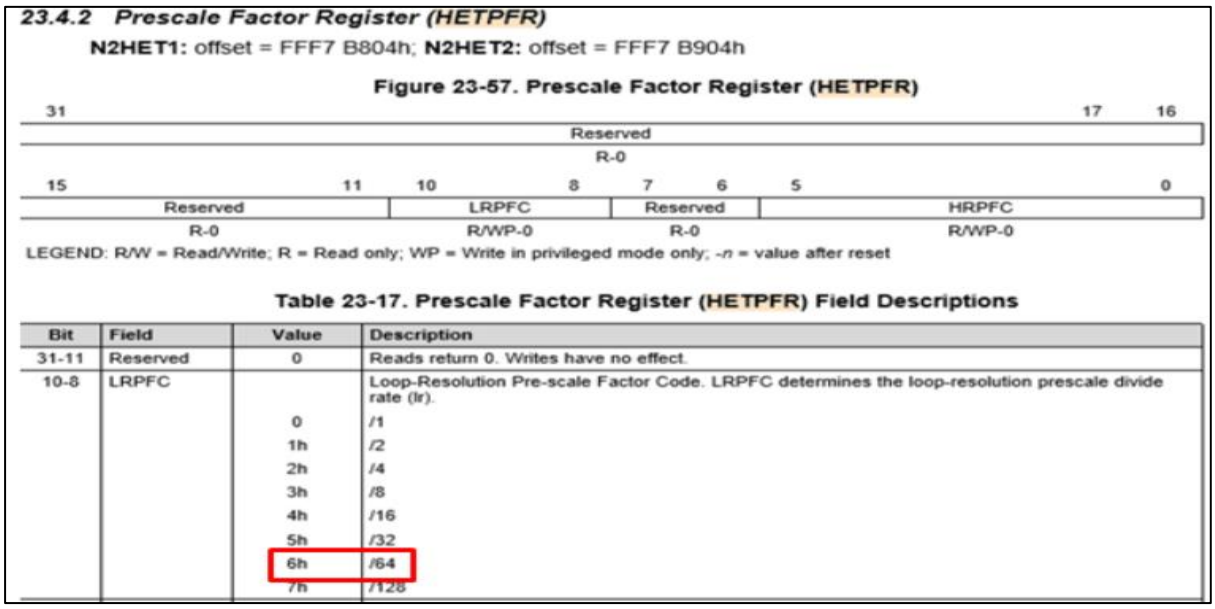
: SHARE 31~24,7~0 각각 서로 신호선을 공유하고 있다. -> WHY?? 핀 10을 연결



HR share bit 0,1 이 연결되어 있는 것을 볼 수 있다. 허나 다른 비트들도 연결되어 있는데, 뭘까..? 이해가 안된다 31~24, 7~0 HR share 선만 공유한다는 것??

..

```
// Setup prescaler values, Loop resolution prescale, High resolution prescaler
1.7  hetREG1->PFR = (uint32)((uint32) 6U << 8U)   // 9,10번 핀
```



23.4.2 Prescale Factor Register (HETPFR)

N2HET1: offset = FFF7 B804h; **N2HET2:** offset = FFF7 B904h

Figure 23-57. Prescale Factor Register (HETPFR)

31 – 17	16	15 – 11	10 – 8	7 – 6	5 – 0
Reserved		Reserved	LRPFC	Reserved	HRPFC
R-0		R-0	R/WP-0	R-0	R/WP-0

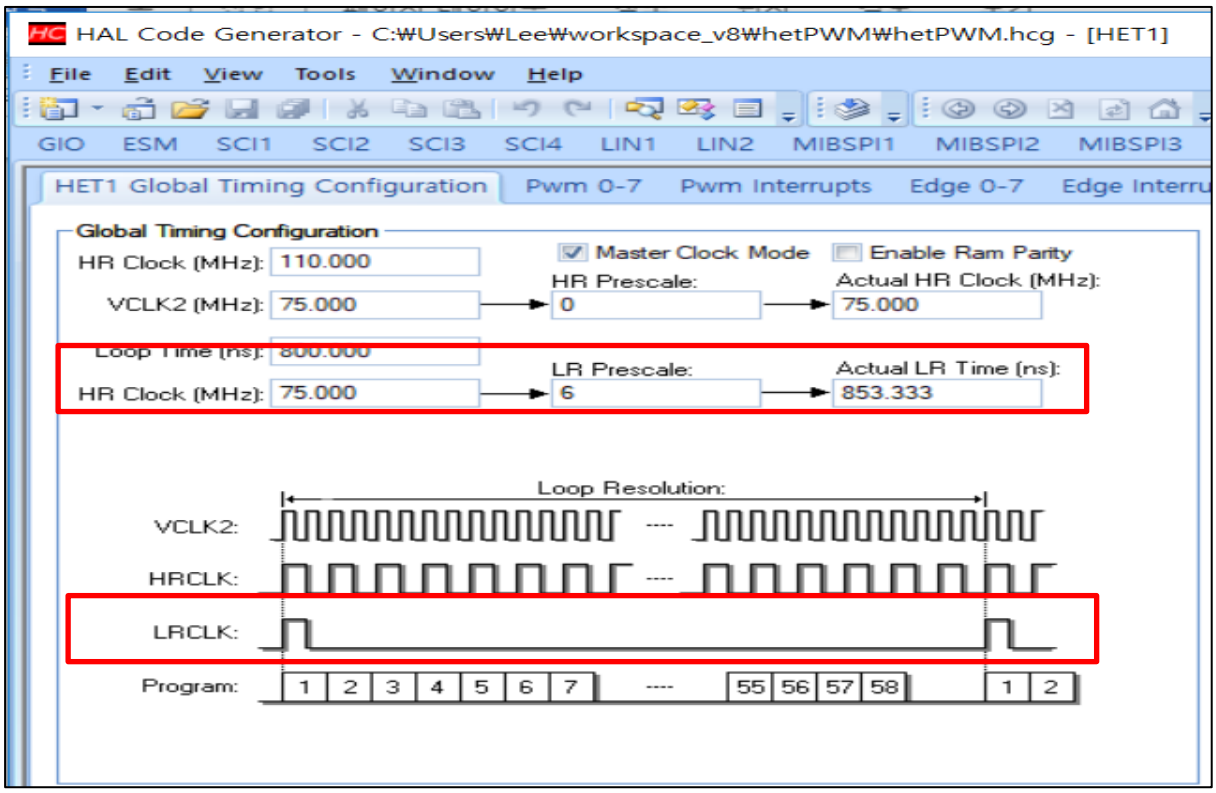
LEGEND: R/W = Read/Write; R = Read only; WP = Write in privileged mode only; -*n* = value after reset

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
		7h	/128

: 프리스케일 LRPFC : Loop – Resolution Pre-scale Factor Code. 64분주 함

-HCG HET1 Timing 을 보자.



:LR CLK이 느리게 가는 것을 볼 수 있다.

```
//- Parity control register,- Enable/Disable Parity check
1.8  hetREG1->PCR = (uint32) 0x00000005U;  // 0,2번 핀
```

23.4.26 Parity Control Register (HETPCR)

N2HET1: offset = FFF7 B874h; **N2HET2:** offset = FFF7 B974h

Figure 23-81. Parity Control Register (HETPCR)

31															16
Reserved															
R-0															

15						9	8	7		4	3		0
Reserved							TEST	Reserved			PARITY_ENA		
R-0							R/WP-0	R-0			R/WP-5h		

LEGEND: R/W = Read/Write; R = Read only; WP = Write in privileged mode only; -*n* = value after reset

Table 23-42. Parity Control Register (HETPCR) Field Descriptions

Bit	Field	Value	Description
31-9	Reserved	0	Reads return 0. Writes have no effect.
8	TEST		Test Bit. When this bit is set, the parity bits are mapped into the peripheral RAM frame to make them accessible by the CPU.
		0	Read: Parity bits are not memory mapped. Write: Disable mapping.
		1	Read: Parity bits are memory mapped. Write: Enable mapping.
7-4	Reserved	0	Reads return 0. Writes have no effect.
3-0	PARITY_ENA		Enable/disable parity checking. This bit field enables or disables the parity check on read operations and the parity calculation on write operations. If parity checking is enabled and a parity error is detected the N2HET_UERR signal is activated.
		5h	Read: Parity check is disabled. Write: Disable checking.
		Others	Read: Parity check is enabled. Write: Enable checking.

NOTE: It is recommended to write Ah to enable error detection, to guard against soft errors flipping PARITY_ENA to a disable state.

: 패리티 체크를 끄고 있다. 에러검사 X

```
/** - Fill HET RAM with opcodes and Data */
    /*SAFETYMCUSW 94 S MR:11.1,11.2,11.4 <APPROVED> "HET RAM Fill from the table -
Allowed as per MISRA rule 11.2" */
    /*SAFETYMCUSW 95 S MR:11.1,11.4 <APPROVED> "HET RAM Fill from the table -
Allowed as per MISRA rule 11.2" */
    /*SAFETYMCUSW 95 S MR:11.1,11.4 <APPROVED> "HET RAM Fill from the table -
Allowed as per MISRA rule 11.2" */
1.9  (void)memcpy((void *)hetRAM1, (const void *)het1PROGRAM, sizeof(het1PROGRAM));
```

Het -> memcpy() 분석 , PWM control 하기 위한 값들을 hetRAM1 에 셋팅하고 있다.

```
(void)memcpy((void *)hetRAM1, (const void *)het1PROGRAM, sizeof(het1PROGRAM));
```

헤더	string.h
형태	**void** *memcpy(**void** *s1, **const void** *s2, size_t n);
인수	**void** *s1 복사될 메모리의 포인터 **void** *s2 복사할 메모리의 포인터 size_t size복사할 바이트 갯 수
반환	**void** *s1 포인터를 반환하며 실패하면 NULL을 반환한다.

```
#define hetRAM1 ((hetRAMBASE_t *)0xFF460000U)
```

→ 파악할 껀덕지가 없다.

-> 이 주소가 뭐하는 놈인지 찾아보자.

```
static const hetINSTRUCTION_t het1PROGRAM[58U] =
{
    /* CNT: Timebase
    *       - Instruction                  = 0
    *       - Next instruction             = 1
    *       - Conditional next instruction = na
    *       - Interrupt                    = na
    *       - Pin                          = na
    *       - Reg                          = T
    */
    {
        /* Program */
        0x00002C80U,
        /* Control */
        0x01FFFFFFU,
        /* Data */
        0xFFFFFF80U,
        /* Reserved */
        0x00000000U
    },
    /* PWCNT: PWM 0 -> Duty Cycle
    *         - Instruction                  = 1
    *         - Next instruction             = 2
    *         - Conditional next instruction = 2
    *         - Interrupt                    = 1
    *         - Pin                          = 8
    */
    {
        /* Program */
        0x000055C0U,
        /* Control */
        (0x00004006U | (uint32)((uint32)8U << 8U) |
```

N2HET1 Base Address	N2HET2 Base Address	Memory
0xFF46_0000	0xFF44_0000	N2HET Instruction RAM (Program/Control/Data)
0xFF46_2000	0xFF44_2000	N2HET Parity RAM

: hetRAM1 (0Xff460000u)에 progam, control, data, reserved 를 복사한다.

23.2.2 N2HET RAM Organization

The N2HET RAM is organized into two sections. The first contains the N2HET program itself. The second contains parity protection bits for the N2HET program.

Each N2HET instruction is 96-bits wide but aligned to a 128-bit boundary. Instructions consist of three 32-bit fields: Program, Control, and Data. Instructions are separated by a fourth unimplemented address to force alignment to 128-bit boundaries.

The integrity of the N2HET program can be protected by Parity. Parity protection is enabled through the N2HET Parity Control Register (HETPCR).

Table 23-1 shows the base addresses for N2HET RAM and N2HET Parity RAM.

: data sheet를 찾아봤을 때, N2HET Instruction RAM(Program/Control/Data) 로 각각 N2HET 명령어는 96bit, 12byte용량을 갖지만, Cortex-R5 ARM코어는 32비트 프로세서이다. 따라서 기본 버스 크기인 1 워드는 32비트로 4바이트이다. 그렇기 때문에 128bit, 16byte로 저장하여 4Byte의 2의 n승의 크기로 저장하는 것이 좋다. 128비트로 정렬.

Table 23-2. N2HET RAM Bank Structure

N2HET Address	Host CPU or DMA Address Space				
Instruction	Program Field Address	Control Field Address	Data Field Address	Reserved Address	N2HET RAM Bank
000h	XX0000h	XX0004h	XX0008h	XX000Ch	A
001h	XX0010h	XX0014h	XX0018h	XX001Ch	B
002h	XX0020h	XX0024h	XX0028h	XX002Ch	C
003h	XX0030h	XX0034h	XX0038h	XX003Ch	D
004h	XX0040h	XX0044h	XX0048h	XX004Ch	A
:	:	:	:	:	:
03Fh	XX03F0h	XX03F4h	XX03F8h	XX03FCh	D
040h	XX0400h	XX0404h	XX0408h	XX040Ch	A
:	:	:	:	:	:
1FFh	XX1FF0h	XX1FF4h	XX1FF8h	XX1FFCh	D

: N2HET RAM메모리 구성으로 N2HET instruction, program, control, data 의 이상한 값들이 들어있다. 이게 뭘까..?

```
Code: (void)memcpy((void *)hetRAM1, (const void *)het1PROGRAM, sizeof(het1PROGRAM))
```

일단 hetRAM1의 주소가 뭘 뜻하는지 찾아봤더니, N2HET의 구성은 PROGRAM,CONTROL,DATA 라고 했다. het1PROGRAM안에는 pwm에 관한 뭔가가 들어있었다. CNT, PWCNT, DJZ, PWCNT (시간,듀티비,주기) 등등.. 이자식들이 뭔가 살펴보자. 우선 CNT

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	C[25:23] = 011, C5 = 1	1-3
ADD	Add and Shift	4h	C[25:23] = 001, C5 = 1	1-3
ADM32	Add Move 32	4h	C[25:23] = 000, C5 = 1	1-2
AND	Bitwise AND and Shift	4h	C[25:23] = 010, C5 = 1	1-3
APCNT	Angle Period Count	Eh	-	1-2
BR	Branch	Dh	-	1
CNT	Count	6h	-	1-2
DADM64	Data Add Move 64	2h	-	2
DJZ	Decrement and Jump if -zero	Ah	P[7:6] = 10	1
ECMP	Equality Compare	0h	C[6:5] = 00	1
ECNT	Event Count	Ah	P[7:6] = 01	1
MCMP	Magnitude Compare	0h	C[6] = 1	1
MOV32	Move 32	4h	C[5] = 0	1-2
MOV64	Move 64	1h	-	1
OR	Bitwise OR	4h	C[25:23] = 100, C5 = 1	1-3
PCNT	Period/Pulse Count	7h	-	1
PWCNT	Pulse Width Count	Ah	P[7:6] = 11	1
RADM64	Register Add Move 64	3h	-	1
RCNT	Ratio Count	Ah	P[7:6] = 00, P[0] = 1	3
SBB	Subtract with Borrow and Shift	4h	C[25:23] =110, C[5] = 1	1-3
SCMP	Sequence Compare	0h	C[6:5] = 01	1
SCNT	Step Count	Ah	P[7:6] = 00, P[0] = 0	3
SHFT	Shift	Fh	C[3] = 0	1
SUB	Subtract and Shift	4h	C[25:23] = 101, C[5] = 1	1-3
WCAP	Software Capture Word	Bh	-	1
WCAPE	Software Capture Word and Event Count	8h	-	1
XOR	Bitwise Exclusive-Or and Shift	4h	C[25:23] = 111, C[5] = 1	1-3

23.6.3.8 CNT (Count)

Syntax

```
CNT {
[brk={OFF | ON}]
[next={label | 9-bit unsigned integer}]
[reqnum={3-bit unsigned integer}]
[request={NOREQ | GENREQ | QUIET}]
[angle_count={OFF | ON}]
[reg={A | B | T | NONE}]
[comp ={EQ | GE}]
[irq={OFF | ON}]
[control={OFF | ON}]
max={25-bit unsigned integer}
[data={25-bit unsigned integer]
}
```

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

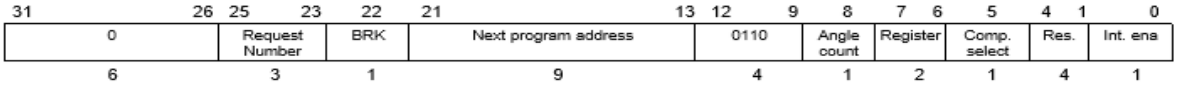


Figure 23-135. CNT Control Field (C31:C0)

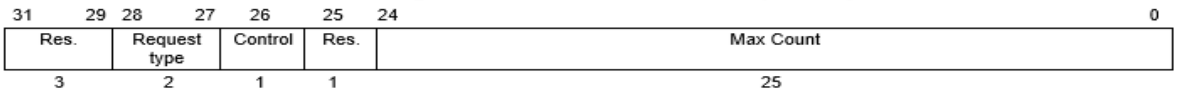
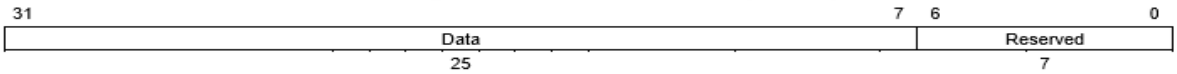


Figure 23-136. CNT Data Field (D31:D0)



1. CNT : Timer 전용 기계어

```
static const hetINSTRUCTION_t het1PROGRAM[58U] =
{
    /* CNT: Timebase
    *       - Instruction                 = 0
    *       - Next instruction            = 1
    *       - Conditional next instruction = na
    *       - Interrupt                   = na
    *       - Pin                         = na
    *       - Reg                         = T
    */
    {
        /* Program */
        0x00002C80U,          //  7,10,11,13
        /* Control */
        0x01FFFFFFU,          //  0~24
        /* Data */
        0xFFFFFF80U,          //  7~31
        /* Reserved */
        0x00000000U
    },
```

- **Program : 0x00002C80 -> 7,10,11,13 bit ON**

Register : 10 (7 bit) -> A,B,T 의 범용레지스터를 사용한다고함. NONE이 있으므로 4개 레지스터일 수도 있다. 이를 가지고 계산을 해준다는 것.

0110 : 0110 (10~12 bit) -> instruction CNT를 자신을 가르킴.

Next program address : 1 (13 bit) -> 다음에 실행할 프로그램의 주소, 지금은 배열 0번째이고 next는 c코드에서는 PWCNT로 넘어가게 된다. 나중에, Next program address에 0을 지정하면 다시 처음으로 돌아오게 된다.

- **Control**

Max Count : 1 (24~0) -> 최대 25bit integer까지 설정가능하고, 24bit 정수값을 지정한다.

- **Data**

Data : 1 (31~7) -> 카운터 역할을 하는 25 비트 정수 값을 지정한다. 기본값은 0이다.