HerkuleX를 C로 제어하기

HOVIS

herkulex 라이브러리를 이용해서 HerkuleX에 직접 패킷을 보내고 제어해 봅니다.

H/W 원리

DRC에 장착된 ATmega128에는 USART0, USART1의 두 USART 포트가 있으며 이 중 USART0은 HerkuleX와의 통신에 사용되는 4핀 포트 5개와 병렬적으로 연결되어 있습니다. 제어하고 싶은 HerkuleX를 케이블로 DRC와 연결하면 USART0을 통해서 직접 패킷을 보내고 받을 수 있습니다.

이번 Chapter에서는 HerkuleX와 통신을 하기 때문에 HerkuleX의 레지스터 맵과 프로토콜에 대한 어느 정도의 지식이 필요합니다. 이번 예제에서 사용하는 패킷에 사용된 명령을 요약하면 아래와 같습니다.

이름	CMD	설명
RAM_WRITE	0x03	HerkuleX의 휘발성 레지스터 영역에 데이터를 씁니다.
RAM_READ	0x04	HerkuleX의 휘발성 레지스터 영역의 데이터를 읽습니다.
I_JOG	0x05	HerkuleX의 위치/속도를 제어하여 움직이게 합니다.

이번 예제에서 사용된 휘발성 레지스터 영역의 항목을 요약하면 아래와 같습니다.

이름	주소	크기	설명
Torque Control	52	1	토크 인가 상태를 제어합니다. 0x00 : 자유롭게 움직이는 힘이 풀린 상태 0x40 : 힘은 풀렸으나 돌아가는 데에 저항이 있는 상태 0x60 : 토크가 인가되어 스스로 구동하는 상태
Calibrated Position	58	2	현재 위치와 GPIO 입력 결과를 나타냅니다. 여기서 GPIO란 HerkuleX의 일부 모델에서 오른쪽 통신 포트를 스위치 입력 포트로 사용하는데, 이 때의 입력 값입니다. 15번 비트 : 사용하지 않음 14:13번 비트 : GPIO 입력 값을 나타냅니다. 12:0번 비트 : 현재 HerkuleX의 위치를 나타냅니다.

프로토콜과 레지스터 맵에 대한 더 자세한 내용은 HerkuleX 매뉴얼을 참조하세요.

예제

ID가 253인 DRS-0101 서보 모터를 반복적으로 움직이면서, 위치 값을 읽어서 목표 위치에 실제로 도달하였는지 확인하는 예제입니다.

TIMER0.h

```
#ifndef TIMER0_H_
#define TIMER0_H_

#ifdef __TIMER0_C
	#define TIMER0_EXT
#else
	#define TIMER0_EXT extern
#endif

//타이머를 사용해 시간을 측정할 때의 전역변수
TIMER0_EXT volatile unsigned char gucTimerTick;
//타이머/카운터0의 초기화 함수
TIMER0_EXT void TIMER0_Init(void);

#endif /* TIMER0_H_ */
```

herkulex.h

```
#ifndef HERKULEX_H_
#define HERKULEX_H_

//////////////////////////////프로토콜에 대한 선언//////////////////////////////
//각 항목의 인덱스
#define PROTOCOL_SIZE_IDX		2
#define PROTOCOL_ID_IDX			3
#define PROTOCOL_CMD_IDX		4
#define PROTOCOL_CS1_IDX		5
#define PROTOCOL_CS2_IDX		6
#define PROTOCOL_DATA_IDX		7

//헤더 관련
#define HEADER				0xFF

//SIZE 관련
#define MIN_PACKET_SIZE			7
#define MIN_ACK_PACKET_SIZE		9
#define MAX_PACKET_SIZE			223
#define MAX_DATA_SIZE				(MAX_PACKET_SIZE-MIN_PACKET_
SIZE)
```

```
//ID 관련
#define MAX_ID                          0xFD
#define BROADCAST_ID                    0xFE

//CMD 관련 - Request Packet
#define CMD_EEP_WRITE                   0x01
#define CMD_EEP_READ                    0x02
#define CMD_RAM_WRITE                   0x03
#define CMD_RAM_READ                    0x04
    #define CMD_RW_DATA_ADDR_IDX        7
    #define CMD_RW_DATA_LEN_IDX         8
#define CMD_I_JOG                       0x05
    #define CMD_I_JOG_STRUCT_SIZE       5
    #define CMD_I_JOG_MAX_DRS           (MAX_DATA_SIZE/CMD_I_JOG_STRUCT_
SIZE)
#define CMD_S_JOG                       0x06
    #define CMD_S_JOG_STRUCT_SIZE       4
    #define CMD_S_JOG_MAX_DRS           (MAX_DATA_SIZE/CMD_S_JOG_
STRUCT_SIZE)
#define CMD_STAT                        0x07
#define CMD_ROLLBACK                    0x08
#define CMD_REBOOT                      0x09

#define CMD_MIN                         (CMD_EEP_WRITE)
#define CMD_MAX                         (CMD_REBOOT)

//CMD 관련 - ACK Packet
#define CMD_ACK_MASK                    0x40

#define CMD_EEP_WRITE_ACK               (CMD_EEP_WRITE|CMD_ACK_MASK)
#define CMD_EEP_READ_ACK                (CMD_EEP_READ|CMD_ACK_MASK)
#define CMD_RAM_WRITE_ACK               (CMD_RAM_WRITE|CMD_ACK_
MASK)
#define CMD_RAM_READ_ACK                (CMD_RAM_READ|CMD_ACK_MASK)
#define CMD_I_JOG_ACK                   (CMD_I_JOG|CMD_ACK_MASK)
#define CMD_S_JOG_ACK                   (CMD_S_JOG|CMD_ACK_MASK)
#define CMD_STAT_ACK                    (CMD_STAT|CMD_ACK_MASK)
#define CMD_ROLLBACK_ACK                (CMD_ROLLBACK|CMD_ACK_MASK)
#define CMD_REBOOT_ACK                  (CMD_REBOOT|CMD_ACK_MASK)

#define CMD_ACK_MIN                     (CMD_EEP_WRITE_ACK)
#define CMD_ACK_MAX                     (CMD_REBOOT_ACK)

//CheckSum 관련
#define CHKSUM_MASK                     0xFE
```

```
//////////////////////////////프로토콜 구조체//////////////////////////////
typedef struct DrsJog
{
        unsigned int    uiValue : 15;
        unsigned int    reserved : 1;
}DrsJog;

typedef struct DrsSet
{
        unsigned char   ucStopFlag : 1;
        unsigned char   ucMode : 1;
        unsigned char   ucLedGreen : 1;
        unsigned char   ucLedBlue : 1;
        unsigned char   ucLedRed : 1;

unsigned char   ucJogInvalid : 1;
        unsigned char   reserved : 2;
}DrsSet;

typedef struct DrsIJog
{
        DrsJog          stJog;
        DrsSet          stSet;
        unsigned char   ucId;
        unsigned char   ucPlayTime;
}DrsIJog;

typedef struct DrsSJog
{
        DrsJog          stJog;
        DrsSet          stSet;
        unsigned char   ucId;
}DrsSJog;

typedef struct DrsIJogData
{
        DrsIJog         stIJog[CMD_I_JOG_MAX_DRS];
}DrsIJogData;

typedef struct DrsSJogData
{
        unsigned char   ucPlayTime;
        DrsSJog         stSJog[CMD_S_JOG_MAX_DRS];
}DrsSJogData;

typedef struct DrsRWData
{
        unsigned char   ucAddress;
```

```
unsigned char    ucLen;
        unsigned char    ucData[MAX_DATA_SIZE-2];
}DrsRWData;

typedef union DrsData
{
        unsigned char    ucData[MAX_PACKET_SIZE-MIN_PACKET_SIZE];

        DrsRWData        stRWData;
        DrsIJogData      stIJogData;
        DrsSJogData      stSJogData;
}DrsData;

typedef struct
{
        unsigned char    ucHeader[2];
        unsigned char    ucPacketSize;
        unsigned char    ucChipID;
        unsigned char    ucCmd;
        unsigned char    ucCheckSum1;
        unsigned char    ucCheckSum2;
        DrsData          unData;
}DrsPacket;

//////////////////////////////수신 상태 결과 값////////////////////////////////
enum{
        DRS_RXWAITING,
        DRS_RXCOMPLETE,
        DRS_HEADERNOTFOUND,
        DRS_INVALIDSIZE,
        DRS_UNKNOWNCMD,
        DRS_INVALIDID,
        DRS_CHKSUMERROR,
        DRS_RXTIMEOUT
}DrsRxStatus;

#ifdef __HERKULEX_C
        #define HERKULEX_EXT
#else
        #define HERKULEX_EXT extern
#endif

//HerkuleX를 제어하기 위해 초기화하는 함수
HERKULEX_EXT void hklx_Init(unsigned long ulBaudRate);
//HerkuleX로 패킷을 보내는 함수
HERKULEX_EXT void hklx_SendPacket(DrsPacket stPacket);
//HerkuleX로부터 패킷을 받는 함수
HERKULEX_EXT unsigned char hklx_ucReceivePacket(DrsPacket *pstPacket);

#endif /* HERKULEX_H_ */
```

HerkuleXCtrl.c

```
//CPU 클럭 설정
#define F_CPU 16000000

//ATmega128의 레지스터 등이 정의되어 있음
#include <avr/io.h>
//interrupt를 사용하는 데 필요한 정보들이 정의되어 있음
#include <avr/interrupt.h>
//_delay_ms 등 딜레이 함수가 정의되어 있음
#include <util/delay.h>

//herkulex 라이브러리의 헤더 파일을 포함
#include <herkulex.h>
#include <TIMER0.h>

//DRS-0101의 Calibrated Position 정보를 사용하기 위한 구조체
typedef struct DrsCaliPos
{
	int		iPosition : 13;
	unsigned int	uiGPIO1 : 1;
	unsigned int	uiGPIO2 : 1;
	unsigned int	reserved : 1;
}DrsCaliPos;

//Calibrated Position 정보를 쉽게 다루기 위한 공용체
typedef union DrsUnionCaliPos
{
	DrsCaliPos	stCaliPos;
	unsigned int	uiCaliPos;
}DrsUnionCaliPos;

int main(void)
{
	//보낼 패킷과 받을 패킷이 저장될 패킷 구조체 선언
	DrsPacket stSendPacket, stRcvPacket;
	//Calibrated Position을 저장할 공용체 변수 선언
	DrsUnionCaliPos unCaliPos;
	//사용할 변수들 선언
	unsigned char ucResult;
	int iPos = 400;

	//포트 C의 입출력 방향 설정
	DDRC = 0b01111111;
	//포트 C의 출력 값 설정
	PORTC = 0b01111111;
```

```
//전체 인터럽트를 비활성화
	cli();

	//HerkuleX를 사용하기 위해 초기화
	hklx_Init(115200);

	//전체 인터럽트를 활성화
	sei();

	//Torque Control에 0x60을 써서 토크를 거는 패킷 구성
	stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
	stSendPacket.ucChipID = 253;
	stSendPacket.ucCmd = CMD_RAM_WRITE;
	stSendPacket.unData.stRWData.ucAddress = 52;
	stSendPacket.unData.stRWData.ucLen = 1;
	stSendPacket.unData.stRWData.ucData[0] = 0x60;

	//패킷 보내기
	hklx_SendPacket(stSendPacket);

	while(1)
	{
		//목표 위치 변수 값 변경
		if(iPos == 400){
			iPos = 624;
		}
		else{
			iPos = 400;
		}

		//DRS-0101을 움직일 I_JOG 패킷 구성
		stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_
SIZE;
		stSendPacket.ucChipID = BROADCAST_ID;
		stSendPacket.ucCmd = CMD_I_JOG;
		stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = iPos;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 1;
		stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
		stSendPacket.unData.stIJogData.stIJog[0].ucId = 253;
		stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

		 //패킷 보내기
		hklx_SendPacket(stSendPacket);

		//다 움직일 때 까지 1초 대기
		_delay_ms(1000);
```

```
        //Calibrated Position을 읽어올 RAM_READ 패킷 구성
                stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2;
                stSendPacket.ucChipID = 253;
                stSendPacket.ucCmd = CMD_RAM_READ;
                stSendPacket.unData.stRWData.ucAddress = 58;
                stSendPacket.unData.stRWData.ucLen = 2;

                //패킷 보내기
                hklx_SendPacket(stSendPacket);

                //TIMEOUT 까지 기다릴 시간 설정. 2*15 = 30(ms)
                gucTimerTick=15;
                while(1){
                        //패킷을 받는 함수를 호출해 결과를 ucResult에 저장
                        ucResult = hklx_ucReceivePacket(&stRcvPacket);
                        //결과 값이 DRS_RXWAITING이 아니면 빠져나옴
                        if(ucResult != DRS_RXWAITING){
                                break;
                        }
                        //30ms가 지나서 gucTimerTick이 0이 되면 빠져나옴
                        if(gucTimerTick==0){
                                ucResult = DRS_RXTIMEOUT;
                                break;
                        }
                }

                //패킷 수신이 정상적으로 완료 시
                if(ucResult == DRS_RXCOMPLETE){
                        //받은 데이터 2바이트를 공용체 변수에 저장
                        unCaliPos.uiCaliPos = stRcvPacket.unData.stRWData.ucData[0] |
                                        (stRcvPacket.unData.stRWData.ucData[1]<<8);

                        //목표 위치와 5 이내로 근접했으면 LED 모두 끔
                        if((iPos - unCaliPos.stCaliPos.iPosition) > -5 &&
                          (iPos - unCaliPos.stCaliPos.iPosition) < 5){
                                PORTC = 0b01111111;
                        }
                        //목표 위치의 5 밖으로 나갔으면 LED 모두 켬
                        else{
                                PORTC = 0b00000000;
                        }
                }
                //패킷 수신이 정상적으로 이루어지지 않았을 때 LED 모두 켬
                else{
                        PORTC = 0b00000000;
                }
        }

        //함수의 형태에 맞게 정수값 반환
        return 1;
}
```

예제설명

먼저 HerkuleXCtrl.c에서 #include문을 사용해 포함시켜주는 헤더 파일에 대한 설명을 합니다.

```
#ifndef TIMER0_H_
#define TIMER0_H_

…

#endif /* TIMER0_H_ */
```

```
#define __TIMER0_C
        #include “TIMER0.h”
#undef __TIMER0_C
```

(TIMER0.c의 일부분)

제일 위 코드는 모든 전역 변수와 함수 원형을 한번씩만 선언해주기 위한 부분입니다. 그 아래에 있는 TIMER0.c의 일부분에서 보듯이, TIMERT0.c에서 TIMER0.h를 #include문으로 포함할 때는 __TIMERT0_C를 #define으로 미리 선언하고 포함합니다. 그러면 TIMER0.c에서 TIMER0.h를 컴파일 할 때에는

```
volatile unsigned char gucTimerTick;
void TIMER0_Init(void);
```

로 컴파일 합니다. 변수와 함수 앞의 extern은 다른 소스 파일에 선언된 변수나 함수를 현재 소스 파일에서 접근하거나 호출할 때에 사용됩니다. TIMER0.c외의 다른 파일에서 TIMER0.c의 변수와 함수를 사용하기 위해 extern을 붙여서 컴파일 하는 것입니다.

지금까지 설명한 컴파일러 지시자에 관련된 내용은 TIMER0.h말고도 다른 헤더 파일에서도 모두 똑같이 적용되므로, 이후 다른 헤더 파일에서는 따로 언급하지 않을 것입니다.

TIMER0.h의 전역변수 gucTimerTick은 Chapter 5에서 사용한 변수와 같이, 0이 아닐 경우 2ms마다 1씩 줄어드는 변수입니다. TIMER0_Init() 함수는 TIMER0의 초기화 함수로, TCNT0을 초기 값으로 설정하고 타이머/카운터0 오버플로우 인터럽트를 활성화 하고, 지정된 분주비로 타이머/카운터0을 시작합니다.

herkulex.h

herkulex.h에서는 HerkuleX를 제어하는 데에 필요한 프로토콜에 대한 선언문과 구조체 등이 정의되어 있습니다. 처음의 프로토콜에 대한 선언 부분에서는 패킷을 이루는 헤더, SIZE, ID, CMD, Check Sum에 관련된 여러 가지 정보가 선언되어 있습니다.

그 다음에는 각 CMD에 따른 패킷 구조체가 정의되어 있어서, 사용자가 자신이 보낼 패킷의 내용을 손쉽게 구성할 수 있도록 되어 있습니다. DrsJog은 I_JOG와 S_JOG 패킷에서 HerkuleX가 이동할 위치/속도 정보가 들어간 구조체이고, DrsSet은 마찬가지로 I_JOG와 S_JOG 패킷에서 여러 설정 정보가 들어간 구조체입니다. DrsIJog와 DrsSJog는 각각 I_JOG와 S_JOG 패킷에서 서보 모터에 하나에 해당하는 정보가 들어간 구조체이며, DrsIJogData와 DrsSJogData는 서보 모터 전체에 해당하는 정보를 가집니다.

DrsRWData는 EEP_WRITE, EEP_READ, RAM_WRITE, RAM_READ의 네 가지 명령에서 사용합니다. 데이터의 첫 바이트는 레지스터 주소, 두 번째 바이트는 읽거나 쓸 데이터 길이, 세 번째 바이트부터는 쓸 데이터(WRITE 명령에만 해당)가 들어가는 패킷 형식을 구조체로 나타냈습니다.

DrsIJogData, DrsSJogData, DrsRWData 모두가 패킷에서 Optional Data 부분을 나타내는 구조체들이므로, DrsData라는 공용체로 묶어서 같은 메모리 영역을 CMD의 종류에 따라서 다르게 접근할 수 있도록 합니다.

마지막으로 DrsPacket은 헤더, SIZE, ID, CMD, Check Sum과 Optional Data를 모두 포함하는 패킷 전체에 대한 구조체입니다. 우리는 HerkuleXCtrl.c에서 DrsPacket 변수를 선언해서 패킷을 구성하고, 보내고, 받고, 받은 패킷을 해석합니다.

```
enum{
        DRS_RXWAITING,
        DRS_RXCOMPLETE,
        DRS_HEADERNOTFOUND,
        DRS_INVALIDSIZE,
        DRS_UNKNOWNCMD,
        DRS_INVALIDID,
        DRS_CHKSUMERROR,
        DRS_RXTIMEOUT
}DrsRxStatus;
```

패킷 구조체 다음에는 hklx_ucReceivePacket에서 반환되는 수신 상태 결과 값이 열거형 변수(enum) DrsRxStatus로 선언되어 있습니다.

```
HERKULEX_EXT void hklx_Init(unsigned long ulBaudRate);
HERKULEX_EXT void hklx_SendPacket(DrsPacket stPacket);
HERKULEX_EXT unsigned char hklx_ucReceivePacket(DrsPacket *pstPacket);
```

위 코드는 herkulex.h에서 선언된 함수들입니다. hklx_Init()은 말그대로 HerkuleX를 제어하기 위해서 ATmega128을 초기화하는 함수입니다. 타이머/카운터0과 USART0을 초기화합니다. hklx_SendPacket()은 HerkuleX로 패킷을 보내는 함수입니다. SIZE, ID, CMD, Optional Data를 DrsPacket 변수에 입력하고 hklx_SendPacket()을 호출하면, 이 함수에서 stPacket에 헤더와 체크섬을 입력한 후 USART0으로 송신합니다. Hklx_ucReceivePacket()은 HerkuleX로부터 패킷을 받아오는 함수입니다. USART0의 수신 버퍼 내용을 검사해서, 온전한 패킷을 모두 받았으면 pstPacket에 복사하고 DRS_RXCOMPLETE를 반환합니다. 그렇지 않은 경우에는 현재 상황에 맞는 DrsRxStatus의 다른 값을 반환합니다.

HerkuleXCtrl.c

```
#define F_CPU 16000000
```

#define 문은 코드 전처리 명령 중 하나로, 앞으로 F_CPU라는 값이 나오면 16000000으로 대체한다는 뜻입니다. DRC의 ATmega128은 16MHz의 클럭 속도로 동작합니다. 클럭 속도에 비례해서 명령 처리 속도가 달라지므로, 이 명령을 통해서 이 프로그램이 동작할 속도를 명시해 주어야 delay 함수 등이 올바르게 동작합니다.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

avr/io.h 헤더 파일에는 DDRX, PORTX 레지스터 등 AVR 제어를 위해 필요한 레지스터 등이 정의되어 있고, avr/interrupt.h 헤더 파일에는 인터럽트를 사용하기 위한 함수들이 정의되어 있습니다. util/delay.h 헤더 파일에는 _delay_ms()등 자주 사용하는 딜레이 함수가 정의되어 있으므로 #include 문을 사용하여 포함시켜 줍니다.

```
#include <herkulex.h>
#include <TIMER0.h>
```

herkulex.h 헤더 파일에는 패킷을 구성하고 보내기 위한 선언들과 구조체, 그리고 함수가 정의되어 있으므로 꼭 포함해야 합니다. 그리고 gucTimerTick을 사용해 패킷의 Timeout을 판정하기 위하여, TIMER0.h도 포함시킵니다.

```
typedef struct DrsCaliPos
{
        int             iPosition : 13;
        unsigned int    uiGPIO1 : 1;
        unsigned int    uiGPIO2 : 1;
        unsigned int    reserved : 1;
}DrsCaliPos;

typedef union DrsUnionCaliPos
{
        DrsCaliPos      stCaliPos;
        unsigned int    uiCaliPos;
}DrsUnionCaliPos;
```

앞서 언급했듯이, HerkuleX 레지스터 맵의 Calibrated Position은 위치 정보와 GPIO 입력 정보를 포함하고 있습니다. 우리가 필요한 것은 위치정보인데, 이렇게 되면 매번 Calibrated Position 값을 읽어올 때 마다 GPIO 입력 정보도 같이 읽혀오기 때문에 비트 연산을 통해서 위치 값만 남겨야 합니다. 이런 불편함을 해소하기 위해서 선언하는 것이 위 구조체와 공용체입니다. DrsCaliPos는 Calibrated Position의 각 비트가 어떻게 사용되는지를 나타내는 구조체이며, DrsUnionCaliPos는 이 구조체와 unsigned int를 같이 묶어서, Calibrated Position의 두 바이트 정보를 DrsCaliPos와 unsigned int의 두 방법으로 접근 가능하도록 했습니다.

```
DrsPacket stSendPacket, stRcvPacket;
DrsUnionCaliPos unCaliPos;
unsigned char ucResult;
int iPos = 400;
```

메인 함수에서 사용할 변수들을 선언합니다. DrsPacket 구조체 타입으로 보낼 때 쓸 변수와 받을 때 쓸 변수를 선언하고, 받은 데이터를 처리하기 위해 DrsUnionCaliPos 공용체 타입으로 unCaliPos를 선언합니다. 그리고 hklx_ucReceivePacket()의 결과 값이 저장될 ucResult를 선언하고, 모터의 목표 위치인 iPos를 선언하고 400으로 초기화 합니다.

```
DDRC = 0b01111111;
PORTC = 0b01111111;
```

포트 C의 0~6번 핀을 출력으로 설정하고 출력 값을 1로 만듭니다. 그러면 모든 LED가 꺼지게 됩니다.

```
cli();
hklx_Init(115200);
sei();
```

인터럽트에 대한 설정을 바꾸는 중에 인터럽트가 발생해 예상치 못한 오동작이 발생하는 것을 방지하기 위해서 우선 cli() 명령을 통해서 전체 인터럽트를 비활성화 시킵니다. 그 후 hklx_Init 함수를 사용해서 HerkuleX를 제어하기 위한 타이머/카운터0과 USART0을 초기화합니다. 사용할 보드 레이트가 115200이므로 파라미터로 그 값을 사용합니다. 초기화가 끝난 후에는 sei() 명령을 통해서 전체 인터럽트를 활성화 합니다.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 3;
stSendPacket.ucChipID = 253;
stSendPacket.ucCmd = CMD_RAM_WRITE;
stSendPacket.unData.stRWData.ucAddress = 52;
stSendPacket.unData.stRWData.ucLen = 1;
stSendPacket.unData.stRWData.ucData[0] = 0x60;

hklx_SendPacket(stSendPacket);
```

모터가 스스로 움직이게 제어하기 위해서는 모터에 토크가 인가되어야 합니다. 위의 코드가 토크를 인가시키기 위한 RAM_WRITE 패킷을 구성하고, 보내는 부분입니다. 패킷의 SIZE는 기본 MIN_PACKET_SIZE(7바이트)에다가, 주소 + 길이 + 데이터(1바이트)로 총 MIN_PACKET_SIZE+3(10바이트)이며, 대상 모터의 ID가 253이므로 ID는 253입니다. CMD는 CMD_RAM_WRITE(0x03)이며, unData 공용체의 멤버 중 stRWData 구조체의 ucAddress, ucLen, ucData[0]의 값을 각각 52, 1, 0x60으로 만들어줍니다. 그 후에는 hklx_SendPacket() 함수를 호출해 패킷을 보냅니다.

```
if(iPos == 400){
        iPos = 624;
}
else{
        iPos = 400;
}
```

여기서부터는 while문 안의 내용입니다. 모터의 목표 위치를 나타내는 변수인 iPos는 while문을 한번 반복할 때마다 번갈아서 400나 624로 값이 바뀌어야 하므로 위와 같이 값을 바꾸어 줍니다.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + CMD_I_JOG_STRUCT_SIZE;
stSendPacket.ucChipID = BROADCAST_ID;
stSendPacket.ucCmd = CMD_I_JOG;
stSendPacket.unData.stIJogData.stIJog[0].stJog.uiValue = iPos;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucStopFlag = 0;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucMode = 0;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedGreen = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedBlue = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucLedRed = 1;
stSendPacket.unData.stIJogData.stIJog[0].stSet.ucJogInvalid = 0;
stSendPacket.unData.stIJogData.stIJog[0].ucId = 253;
stSendPacket.unData.stIJogData.stIJog[0].ucPlayTime = 50;

hklx_SendPacket(stSendPacket);

_delay_ms(1000);
```

모터가 iPos의 위치로 움직이도록 I_JOG 패킷을 구성하고, 보내는 부분입니다. 패킷의 SIZE는 기본 MIN_PACKET_SIZE(7바이트)에다가, 모터 하나당 데이터가 CMD_I_JOG_STRUCT_SIZE(5바이트)이므로 이 둘을 합친 길이(12바이트)가 됩니다. I_JOG는 여러 모터들에게 한꺼번에 명령을 날리는 것이므로 ID는 BROADCAST_ID(254)입니다. CMD는 CMD_I_JOG(0x05)이며, unData 공용체의 멤버 중 stIJogData 구조체의 stIJog[0]에 정보를 알맞게 넣어줍니다.

stJog의 uiValue에는 목표 위치인 iPos 값을 넣고, 위치 제어 모드이므로 stSet의 ucMode는 0, LED 3개를 다 켜기 위해서 stSet의 ucLedGreen, ucLedBlue, ucLedRed를 모두 1로, uclD는 253으로, ucPlayTime은 50(560ms)으로 설정합니다. 그 후에는 hklx_SendPacket() 함수를 호출해 패킷을 보내고, 움직임이 끝날 때까지 1초간 대기합니다.

```
stSendPacket.ucPacketSize = MIN_PACKET_SIZE + 2;
stSendPacket.ucChipID = 253;
stSendPacket.ucCmd = CMD_RAM_READ;
stSendPacket.unData.stRWData.ucAddress = 58;
stSendPacket.unData.stRWData.ucLen = 2;

hklx_SendPacket(stSendPacket);
```

모터의 움직임이 끝난 후에는 모터의 현재 위치 값을 읽어와서 모터가 실제로 목표 위치에 도달했는지 확인해야 합니다. 위의 코드가 Calibrated Position 값을 읽기 위한 RAM_READ 패킷을 구성하고, 보내는 부분입니다. 패킷의 SIZE는 기본 MIN_PACKET_SIZE(7바이트)에다가, 주소 + 길이로 총 MIN_PACKET_SIZE+2(9바이트)이며, 대상 모터의 ID가 253이므로 ID는 253입니다. CMD는 CMD_RAM_READ(0x04)이며, unData 공용체의 멤버 중 stRWData 구조체의 ucAddress, ucLen의 값을 각각 58, 2로 만들어줍니다. 그 후에는 hklx_SendPacket() 함수를 호출해 패킷을 보냅니다.

```
gucTimerTick=15;
while(1){
        ucResult = hklx_ucReceivePacket(&stRcvPacket);
        if(ucResult != DRS_RXWAITING){
                break;
        }

        if(gucTimerTick==0){
                ucResult = DRS_RXTIMEOUT;
                break;
        }
}
```

Timeout 시간을 설정해놓고 ACK Packet이 들어오기를 기다리는 함수입니다. libherkulex.a 파일에는 타이머/카운터0 오버플로우 인터럽트 서비스 루틴이 포함되어 있기 때문에, HerkuleXCtrl.c에서 gucTimerTick 값을 15로 만들면 이 변수는 타이머/카운터0 오버플로우 인터럽트가 발생하는 2ms마다 1씩 줄어들어서, 약 30ms 후에는 0이 됩니다. 그동안 while문 안에서는 계속 hklx_ucReceivePacket() 함수를 호출해서 ucResult에 그 결과 값을 저장합니다.

ACK 패킷이 들어오기를 기다릴 때에는 결과 값이 DRS_RXWAITING이며, 완성된 패킷을 받았거나 그 외에 잘못된 패킷이 들어온 상황에는 DrsRxStatus 중에서 그에 맞는 상태 값이 반환됩니다. 그렇기 때문에, ucResult가 DRS_RXWAITING이

아닌 경우에는 break문을 사용해서 while문을 빠져 나옵니다. 만약 30ms가 지나서 gucTimerTick이 00이 되었는데도 아직 ucResult가 DRS_RXWAITING일 경우에는, ucResult를 DRS_RXTIMEOUT으로 만들고 while문을 빠져 나옵니다.

```
if(ucResult == DRS_RXCOMPLETE){
        unCaliPos.uiCaliPos = stRcvPacket.unData.stRWData.ucData[0] |
                              (stRcvPacket.unData.stRWData.ucData[1]<<8);

        if((iPos - unCaliPos.stCaliPos.iPosition) > -5 &&
          (iPos - unCaliPos.stCaliPos.iPosition) < 5){
                PORTC = 0b01111111;
        }
        else{
                PORTC = 0b00000000;
        }
}
else{
        PORTC = 0b00000000;
}
```

만약 ucResult가 DRS_RXCOMPLETE로 완전한 ACK 패킷을 돌려받았다면, 받은 RAM_READ의 ACK 패킷 중 Calibrated Position 데이터가 들어 있는 부분을 unCaliPos 공용체 변수의 멤버 uiCaliPos에 저장합니다. 그리고 나서 unCaliPos의 stCaliPos 중 iPosition을 접근하면, 읽어온 데이터에서 위치 정보에 해당하는 하위 13비트만 접근해서 사용할 수 있습니다. if문을 사용해서 목표 위치인 iPos가 현재 위치인 unCaliPos.stCaliPos.iPosition과 5 미만으로 차이 나는지 검사하고, 만약 그렇다면 DRC의 LED를 모두 끕니다. 만약 5 이상 차이가 난다면 LED를 모두 켭니다. 또한, 패킷 에러가 발생해서 온전한 ACK 패킷을 받지 못한 경우에도 LED를 모두 켭니다.