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

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

HALCOGEN 설정 (Polling 방식)

1. VIM CHANNEL 16, 29, 44 활성화.(CAN1 HIGH, CAN1 LOW, CAN1 IF3 활성화)

2. CAN1 탭에서 CAN1 General 서브 탭에서 500kb 설정.

3. CAN Msg 1-8 서브 탭에서 메시지 1 - Activate 활성화, TX 활성화 ID 1 Mask 0x7F0

메시지 2 - Activate 활성화, RX 활성화 ID 2 Mask 0x7F0

```
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_can.h"
#include "HL_esm.h"
#include "HL_sys_core.h"
#include "stdio.h"
#define D_COUNT 8
#define D_SIZE 8
uint32 cnt = 0;
uint32 error = 0;
uint32 tx_done = 0;
uint8 tx_data[D_COUNT] = { 1, 2, 3, 4, 4, 3, 2, 1 };
uint8 rx_data[D_COUNT] = { 0 };
uint32_t checkPackets(uint8_t *src_packet, uint8_t *dst_packet, uint32_t psize);

void delay(int time)
{
    int i;
    for (i = 0; i < time; i++)
        ;
}

int main(void)

{
    canInit();
    //dcan_enable_int();

    printf("start\n");
    canEnableErrorNotification(canREG1);
    //canIoSetDirection(canREG1, canMESSAGE_BOX1, canMESSAGE_BOX2);

    while (1)
    {
        printf("transmit CAN message\n");
        delay(10000000);
        canTransmit(canREG1, canMESSAGE_BOX1, (const uint8 *)&tx_data[0]);

        if (canIsRxMessageArrived(canREG1, canMESSAGE_BOX2))
        {
            canGetData(canREG1, canMESSAGE_BOX2, (uint8 *) &rx_data[0]);
            printf("rx_data : %x\n", *rx_data);
        }
    }
}
```

---

canInit()함수 내부 분석

canReg1->CTL

DCAN CTL (CAN Control Register) = 17, 6, 1, 0 비트 ON

20 - DE3 flag > IF3을 위한 DMA request disable.

19 - DE2 flag > IF2을 위한 DMA request disable.

18 - DE1 flag > IF1을 위한 DMA request disable.

17 - IE1 flag > Interrupt 라인 1 enable. 인터럽트는 DCAN1INT 선언, 해당 라인은 지연되는 인터럽트가 진행중이더라도 라인이 active 상태이다.

13~10 - PMD flag > SECDED enable.

9 - ABO flag > Auto-Bus On disable. -> enable

6 - CCE flag > 설정 변경이 가능하게 함. CPU가 BTR 설정 레지스터에 쓰기 가능하도록 한다.

5 - DAR flag > 자동 재 전송을 enable.

2 - SIE flag off> 스테이터스 변경 인터럽트 disable. WakeUpPnd, RxOk, TxOk, LEC은 DCAN0INT 라인에 인터럽트를 발생시키고 Interrupt 레지스터에 영향을 끼친다.

1 - IE0 flag > Interrupt 라인 0번을 enable. Interrupts는 DCAN0INT 라인을 선언한다.

0 - Init > 초기화 비트, 초기화 모드로 들어간다.

---

canReg1->ES = 0xFFFFFFFF;

DCAN ES (Error and Status Register) = 전부 비트 ON

해당 레지스터는 Read Only가 대부분이라 LEC에 7h를 넣는다.

LEC flag는 에러코드 검출 플래그이며, 7h는 이상 없음을 나타낸다.

---

canReg1->INTMUX[0U]

전부 0 이면 DCAN0INT 라인이 활성화, 전부 1이면 DCAN1INT가 활성화 된다.

---

canReg1->ABORT = 0

해당 레지스터는 VBUS 클록 사이클이 지정된 시간만큼 지나가면 BUS - off가 된다.

```
while ((canREG1->IF1STAT & 0x80U) ==0x80U)
    {
    }
```

IF1/IF2 Command Register (DCAN IF1CMD, DCAN IF2CMD)

해당 레지스터는 4개의 char형 변수로 나타내 진다.

```
uint8 rsvd9;
uint8 IF1CMD
uint8 IF1STAT;
uint8 IF1NO;
```

순서대로 DCAN IF1CMD의 Reserved[31:24], IF1CMD[23:26], IF1STAT[15:8], IF1NO[7:0]

따라서 IF1STAT의 0x80은 7번째 비트이고 IF1STAT을 비교하므로 Busy flag를 확인하게 된다.

Busy bit는 메시지 램과 인터페이스 레지스터간 전송이 완료 될 경우 0이 되므로 while문을 통하여 해당 비트가 0 이 될 때 까지 기다리는 것이다.

---

canReg1->IF1MSK = 31, 30, 10, 9, 8, 7, 6, 5, 4 비트 ON

31 - MXtd flag > 확장 ID식별자를 필터를 위하여 사용한다.

30 - MDir flag > 메시지 direction 비트를 필터를 위하여 사용된다.

28~0 Msk[n] > mask설정 0x7F0으로 설정 0b 0111 1111 0000 이다.

해당 Msk를 통하여 식별자가 0b 0000 ~ 0b 1111 까지 사용 가능하다.

---

canReg1->IF1ARB = 31, 30, 29, 0번 비트 ON

ID = 1 이 되며, MsgVal = 1, Xtd = 1, Dir = 1이 된다.

해당 레지스터들은 메시지 객체의 구성을 확인해야 한다.

27.5.1 *Structure of Message Objects*

Figure 27-4 shows the structure of a message object.

The grayed fields are those parts of the message object that are represented in dedicated registers. For example, the transmit request flags of all message objects are represented in centralized transmit request registers.

Figure 27-4. Structure of a Message Object

Message Object												
UMask	Msk[28:0]	MXtd	MDir	EoB	unused	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
MsgVal	ID[28:0]	Xtd	Dir	DLC[3:0]	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

MsgVal = 1 이므로 데이터 메시지 객체가 메시지 핸들러에 의해 사용한다.

Xtd = 1이므로 확장 ID를 사용한다.

Dir = 1 이므로 송신 방향(TxRqst에 적용)이며, data frame이 송신된다.

---

canReg1->IF1MCTL = 12, 3 bit ON

Umask = 1 마스크 사용여부 결정 즉, 마스크 필터를 사용한다.

DLC[3:0] = 0b100 즉, 8을 나타내며 데이터의 길이가 8임을 나타낸다.

해당 레지스터에서 수신 송신 인터럽트, 데이터의 끝 등의 설정이 가능하다.

---

canReg1->IF1CMD = 0xF8; // 3, 4, 5, 6, 7 비트 ON

IF1CMD는 IF1 Command register(IF1CMD)의 23, 22, 21, 20, 19번 비트 온.

23 - 전송방향은 IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 향한다.

22 - IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 Mask bit 전달.

21 - IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 arb bits 전달.

20 - IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 message control bit 전달.

19 - 비트 무시 영향 없음. (1이 된 상태를 읽는 경우는 IntPnd를 초기화한다.)

---

canReg1->IF1NO = 0x1; // 0번비트 활성화

Message Number를 1로 한다. (메시지 RAM에 들어있는 내용 즉 객체 번호이다.)

```
while ((canREG1->IF1STAT & 0x80U) ==0x80U)
    {
    } /* Wait */
    canREG1->IF1CMD  = 0x87U;
```

busy 비트 대기후에,

canREG1->IF1CMD = 0x87U; // 7, 2, 1, 0 번 비트 활성화

// message command register의 23 ~ 16 비트 기준, 23, 18, 17, 16비트 온.

23 - 전송방향은 IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 향한다.

18 - 메시지 객체 안에 TxRqst/NewDat를 세트한다.

17 - 데이터 바이트 0 ~ 3을 IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 향한다.

16 - 데이터 바이트 4 ~ 7을 IF1/IF2 레지스터 세트에서 message number로 주소 지정된 message object로 향한다.

즉, IF1 IF2 에서 Message RAM으로 데이터 이동하거나 Message Ram에서 IF1 IF2 레지스터로 데이터가 이동함을 알 수 있다. 해당 IF 레지스터를 조작하여 데이터 이동 방향을 설정 할 수 있다.

---

canREG1->BTR = 0b 0 0011 1001 1100 1001 // 0, 3, 6, 7, 8, 11, 12, 13 비트 ON

Bit Timing Register (DCAN BTR)

기존에 CAN 통신에서 지연 등을 처리하기 위한 Tseg2, Tseg1, SJW, BRP 등의 값을 넣어준다.

---

canREG1->TIOC, canReg1->RIOC 에 3번 비트가 세트 되어 있다

해당 레지스터는 CAN 포트를 gpio핀 처럼 사용할 경우 쓰는 레지스터이다. 3번비트를 세트 하여 CAN 통신 포트로 활용한다고 선언하면 된다.

---

canReg1->CTL &= ~(uint32)(0x41U); // 0번비트 와 6번비트 만 0으로 마스킹.

6 - CCE flag > CPU가 BTR 레지스터에 접근하지 못하도록 한다. (즉 초기 통신 지연을 감안한 여러 계산 값들을 조작하지 못하도록 한다.)

0 - Init flag > 초기화 모드를 끝내고 Normal Operation 모드로 진입한다.

```
void   canInit(void);
uint32 canTransmit(canBASE_t *node, uint32 messageBox, const uint8 * data);
uint32 canGetData(canBASE_t *node, uint32 messageBox, uint8 * const data);
uint32 canGetID(canBASE_t *node, uint32 messageBox);
void canUpdateID(canBASE_t *node, uint32 messageBox, uint32 msgBoxArbitVal);
uint32 canSendRemoteFrame(canBASE_t *node, uint32 messageBox);
uint32 canFillMessageObjectData(canBASE_t *node, uint32 messageBox, const uint8 *
data);
uint32 canIsTxMessagePending(canBASE_t *node, uint32 messageBox);
uint32 canIsRxMessageArrived(canBASE_t *node, uint32 messageBox);
uint32 canIsMessageBoxValid(canBASE_t *node, uint32 messageBox);
uint32 canGetLastError(canBASE_t *node);
uint32 canGetErrorLevel(canBASE_t *node);
void   canEnableErrorNotification(canBASE_t *node);
void   canDisableErrorNotification(canBASE_t *node);
void   canEnableStatusChangeNotification(canBASE_t *node);
void   canDisableStatusChangeNotification(canBASE_t *node);
void   canEnableloopback(canBASE_t *node, canloopBackType_t Loopbacktype);
void   canDisableloopback(canBASE_t *node);
void   canIoSetDirection(canBASE_t *node,uint32 TxDir,uint32 RxDir);
void   canIoSetPort(canBASE_t *node, uint32 TxValue, uint32 RxValue);
uint32 canIoTxGetBit(canBASE_t *node);
uint32 canIoRxGetBit(canBASE_t *node);
void can1GetConfigValue(can_config_reg_t *config_reg, config_value_type_t type);
```

캔 활용 인터페이스 함수.

```
void canInit(void);
```

->

HALCOGEN에서 세팅한 초기화 값이 세팅 된다.

```
uint32 canTransmit(canBASE_t *node, uint32 messageBox, const uint8 * data);
```

->

can 데이터 전송 함수. 해당 데이터가 정상적으로 전송 되면 1 리턴, 실패 시 0이 리턴.

canBASE_t *node : canREG1 ~ 4까지 사용 CAN 모듈 번호를 나타낸다.

uint32 messageBox : 메시지 박스 번호를 나타낸다. HALCOGEN에서 설정한 메시지 박스.

const uint8 * data : 송신 할 8바이트 데이터 (CAN은 최대 8바이트 데이터)

TXRQx[] 레지스터를 읽음으로서, 해당 메시지 객체의 전송 요청이 진행 중인지 없는지 확인.

전송 요청이 없다면,

Direction = Write : IF1 레지스터에 매개변수 데이터를 집어넣는 방식을 통하여 통신한다.

busy bit가 0으로 될 때 까지 기다리는 동작도 포함한다.

```
uint32 canGetData(canBASE_t *node, uint32 messageBox, uint8 * const data);
```

->

can 데이터 수신 함수. 새로운 데이터가 있다면, 1 리턴, 실패 시 0이 리턴. 데이터 분실 발생 시 3 리턴 된다.

canBASE_t *node : canREG1 ~ 4까지 사용 CAN 모듈 번호를 나타낸다.

uint32 messageBox : 메시지 박스 번호를 나타낸다. HALCOGEN에서 설정한 메시지 박스.

const uint8 * data : 수신 할 8바이트 데이터 (CAN은 최대 8바이트 데이터)

NWDATx[] 레지스터를 읽음으로서, 해당 메시지 객체에 새로운 데이터가 존재하는지 확인.

Dir = Read : 메시지 객체로부터 IF2 레지스터에 데이터를 집어넣는 방식을 통하여 통신.

busy bit가 0으로 될 때 까지 기다리는 동작도 포함한다.

DLC를 읽어서 8바이트 이상이면 8바이트로 고정하는 작업도 포함.

데이터 분실 여부를 확인도 한다. (***IF1MCTL의 14번 비트 확인***)

```
uint32 canGetID(canBASE_t *node, uint32 messageBox);
```

->

can데이터의 ID를 얻어오는 함수. 리턴 값으로 해당 메시지 박스의 ID가 전달된다.

IF 레지스터의 방향을 읽기로 전환.

Arbitration bit (ID + DIR + Xtd + MsgVal)을 넘버에 맞는 메시지 객체에서 IF레지스터로 전송한다.

(Message object -> IF Register)

```
msgBoxID = (node->IF2ARB & 0x1FFFFFFFU);
//0b 0001 1111 1111 1111 1111 1111 1111 1111
```

를 통하여 IF1ARB 레지스터의 ID값[28:0] 을 받아와서 리턴 한다.

```
void canUpdateID(canBASE_t *node, uint32 messageBox, uint32 msgBoxArbitVal);
```

->

canBASE_t *node : CAN 모듈의 넘버

uint32 messageBox : 메시지 박스 번호

uint32 msgBoxArbitVal : 입력하고 싶은 Arbital 정보(ID + DIR + Xtd)

ex) 0b 1110 0000 0000 0000 0000 0000 0000 0001 인 경우 xtd 기반 ID가 1이다.

can데이터의 ID를 수정하는 함수. 리턴 값은 없다.

IF 레지스터의 방향을 쓰기로 전환.

Arbitration bit (ID + DIR + Xtd + MsgVal)을 IF레지스터에서 넘버에 맞는 메시지 객체로 전송한다.

(IF Register -> Message object)

IF 레지스터의 Arbitration(ID + DIR + Xtd)을 초기화하고, 매개변수 msgBoxArbitVal를 입력한다.

```
uint32 canSendRemoteFrame(canBASE_t *node, uint32 messageBox);
```

->

메시지 객체에 전송을 요청하는 함수. 성공 시 1 리턴, 실패 시 0을 리턴 한다.

canBASE_t *node : CAN 모듈 번호

uint32 messageBox : 메시지 박스 번호

```
if ((node->TXRQx[regIndex] & bitIndex) != 0U)
    {
        success = 0U;
    }
```

해당 함수는 TXRQx[] 의 비트를 확인하여 전송요청을 받은지 안 받은지 확인한다.

전송 요청이 없을 경우에만 해당 if문을 빠져 나간다.

23	22	21	20	19	18	17	16
WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/NewDat	Data A	Data B
R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0	R/WP-0

23, 18번을 On 나머지는 OFF.

IF 레지스터에서 메시지 객체로 방향 설정. (Write)

18	TxRqst/NewDat		Access Transmission Request bit.
		0	Direction = Read: NewDat bit will not be changed. Direction = Write: TxRqst/NewDat bit will be handled according to the Control bit.
		1	Direction = Read: Clears NewDat bit in the message object. Direction = Write: Sets TxRqst/NewDat in the message object.
			Note: If a CAN transmission is requested by setting TxRqst/NewDat in this register, the TxRqst/NewDat bits in the message object will be set to 1 and independent of the values in IF1/IF2 Message Control Register. A read access to a message object can be combined with the reset of the control bits IntPnd and NewDat. The values of these bits transferred to the IF1/IF2 Message Control Register always reflect the status before resetting them.

메시지 객체의 TxRqst/NewDat 레지스터를 세트 해준다.

해당 메시지 객체에 전송요청 비트를 심는 것이다.

```
uint32 canFillMessageObjectData(canBASE_t *node, uint32 messageBox, const uint8 *
data);
```

->

해당 함수는 메시지 객체에 원하는 값을 채워 넣는 함수이다. 리턴 값은 성공 1 실패 0

canBASE_t *node : CAN 모듈 번호

uint32 messageBox : 메시지 박스 넘버

const uint8 * data : 채워 넣고 싶은 데이터

```
if ((node->TXRQx[regIndex] & bitIndex) != 0U)
    {
        success = 0U;
    }
```

전송 요청이 있는지 확인.

```
node->IF1CMD = 0x83U; // 0b 1000 0011
```

전송 방향은 IF 레지스터에서 메시지 객체로 방향 설정. (Write)

17	Data A		Access Data Bytes 0–3.
		0	Data Bytes 0–3 will not be changed.
		1	Direction = Read: The Data Bytes 0–3 will be transferred from the message object addressed by the Message Number (Bits [7:0]) to the corresponding IF1/IF2 Register set. Direction = Write: The Data Bytes 0–3 will be transferred from the IF1/IF2 Register set to the message object addressed by the Message Number (Bits [7:0]). **Note:** The duration of the message transfer is independent of the number of bytes to be transferred.

Bit	Field	Value	Description
16	Data B		Access Data Bytes 4–7.
		0	Data Bytes 4–7 will not be changed.
		1	Direction = Read: The Data Bytes 4–7 will be transferred from the message object addressed by the Message Number (Bits [7:0]) to the corresponding IF1/IF2 Register set. Direction = Write: The Data Bytes 4–7 will be transferred from the IF1/IF2 Register set to the message object addressed by the Message Number (Bits [7:0]). **Note:** The duration of the message transfer is independent of the number of bytes to be transferred.

해당 DATA 바이트 (0~3, 4~7)가 IF 레지스터에서 메시지 객체로 간다고 설정함.

실제 IF가 가지고 있는 데이터

```
node->IF1DATx[s_canByteOrder[i]] = *data;
```

해당 레지스터에 함수 매개변수이 값을 넣는다.

희안한 점은 canTransmit() 함수는 TxRqst/NewDat 레지스터를 설정하여 메시지 객체에 값을 채워 넣고 전달을 요청하고, canFillMessageObjectData() 함수는 메시지 객체에 값만 채워 넣는다는 것이다.

```
uint32 canIsTxMessagePending(canBASE_t *node, uint32 messageBox)
```
->

해당 함수는 메시지 전달이 있는지 확인하는 것이다.
리턴 값으로 전송요청이 존재 한다면 1 전송요청이 없다면 0을 리턴한다.

```
    flag = node->TXRQx[regIndex] & bitIndex;
```
해당 메시지 박스의 전송 요청을 확인한다.

```
uint32 canIsRxMessageArrived(canBASE_t *node, uint32 messageBox)
```
->
해당 함수는 전달받은 메시지가 있는지 확인하는 함수이다.

리턴 값으로 전달받은 메시지가 존재 하면 1, 전달받은 메시지가 없다면 0을 리턴한다.

```
    flag = node->NWDATx[regIndex] & bitIndex;
```
flag에 NEW 데이터를 확인하여 저장. 존재하면 1 이 저장!

```
uint32 canGetLastError(canBASE_t *node);
```
->
해당 함수는 CAN모듈에 발생한 에러를 확인하는 함수이다.
리턴 값으로 에러 번호가 리턴 된다.

에러번호 0 : 에러없음.
에러번호 1 : Stuff Error. 메시지 일부에 5개 이상의 동일 비트 감지.
에러번호 2 : Form Error. 수신 프레임 형식이 잘못됨.
에러번호 3 : Ack Error. 캔 통신에서 확인 응답을 받지 못했음.
에러번호 4 : Bit1 Error. 메시지 전송 중 전송 값과 모니터링 값이 다름
에러번호 5 : Bit0 Error. 메시지 전송 중 전송 값과 모니터링 값이 다름
에러번호 6 : CRC Error. 수신 된 메시지의 CRC 합이 잘 못됨
에러번호 7 : CAN bus event 감지 안됨. CPU가 Error and Status Reg 읽을 때.

```
uint32 canGetErrorLevel(canBASE_t *node)
```

7	BOff		Bus-Off State
		0	The CAN module is not Bus-Off state.
		1	The CAN module is in Bus-Off state.
6	EWarn		Warning State
		0	Both error counters are below the error warning limit of 96.
		1	At least one of the error counters has reached the error warning limit of 96.
5	EPass		Error Passive State
		0	On CAN Bus error, the DCAN could send active error frames.
		1	The CAN Core is in the error passive state as defined in the CAN Specification.

에러 레벨을 리턴 한다.
7,6,5 비트의 값을 확인하여 현재 CAN 통신 환경을 확인 할 수 있다.

Bus-Off State, Warning State, Error Passive State 세가지 항목을 확인한다.

```
void   canEnableErrorNotification(canBASE_t *node);
```
DCAN CTL의 EIE flag를 세트하여 Error Interrupt 활성화.
(PER, BOff, and EWarn bits가 인터럽트 발생시킴.)

```
void   canDisableErrorNotification(canBASE_t *node);
```
DCAN CTL의 EIE flag를 클리어하여 Error Interrupt 비 활성화.

```
void   canEnableStatusChangeNotification(canBASE_t *node);
```
DCAN CTL의 SIE flag를 세트하여 Status change Interrupt 활성화.
(WakeUpPnd, RxOk, TxOk, and LEC가 인터럽트 발생시킴.)

```
void   canDisableStatusChangeNotification(canBASE_t *node);
```
DCAN CTL의 SIE flag를 클리어하여 Status change Interrupt 비 활성화.

```
void   canEnableloopback(canBASE_t *node, canloopBackType_t Loopbacktype);
```
DCAN CTL의 Test flag를 세트하여 Test Mode 진입.
매개변수 Loopbacktype을 이용하여 테스트 모드 세팅(DCAN TEST Reg)을 한다.

```
void   canDisableloopback(canBASE_t *node);
```
DCAN CTL의 Test flag를 클리어하여 Normal Mode 진입.

```
void   canIoSetDirection(canBASE_t *node,uint32 TxDir,uint32 RxDir);
```
DCAN_TX 핀을 input이나 output 설정.
DCAN_RX 핀을 input이나 output 설정. (TxDir or RxDir 0은 입력 1 출력)

```
void   canIoSetPort(canBASE_t *node, uint32 TxValue, uint32 RxValue);
```
DCAN_TX, RX핀을 High, low 설정한다.

Polling 방식 CAN 통신 예제

```
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_can.h"
#include "HL_esm.h"
#include "HL_sys_core.h"
#include "stdio.h"

/* USER CODE BEGIN (2) */

#define D_COUNT 8
#define D_SIZE 8
/* USER CODE END */

uint32 cnt = 0;
uint32 error = 0;
uint32 tx_done = 0;

uint8 tx_data[D_COUNT] = { 1, 2, 3, 4, 4, 3, 2, 1 };
uint8 rx_data[D_COUNT] = { 0 };

uint32_t checkPackets(uint8_t *src_packet, uint8_t *dst_packet, uint32_t psize);

void delay(int time)
{
    int i;
    for (i = 0; i < time; i++)
        ;
}

int main(void)
{
    int i;
    canInit();

    printf("start\n");
    canEnableErrorNotification(canREG1);

    while (1)
    {
        printf("transmit CAN message\n");
        delay(7500000);
        canTransmit(canREG1, canMESSAGE_BOX1, (const uint8 *)&tx_data[0]);
        printf("MCU ID is %d\n", canGetID(canREG1,canMESSAGE_BOX1));

        if (canIsRxMessageArrived(canREG1, canMESSAGE_BOX2)){
            canGetData(canREG1, canMESSAGE_BOX2, rx_data);
            printf("rx_data : %s\n", rx_data);
            printf("COM ID is %d\n", canGetID(canREG1,canMESSAGE_BOX2));

            for(i=0; i<sizeof(rx_data); i++){
                rx_data[i] = '\0';
            }
        }
    }
}
```

Interrupt 방식 CAN 통신 예제.

(HALCOGEN Polling방식 설정에서 CAN통신 관련 VIM 설정을 FIQ로 해주어야 한다.)

```
/* Include Files */
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_can.h"
#include "HL_esm.h"
#include "HL_sys_core.h"
#include "stdio.h"

/* USER CODE BEGIN (2) */

#define D_COUNT 8
#define D_SIZE 8
/* USER CODE END */

uint32 cnt = 0;
uint32 error = 0;
uint32 tx_done = 0;

uint8 tx_data[D_COUNT] = { 1, 2, 3, 4, 4, 3, 2, 1 };
uint8 rx_data[D_COUNT] = { 0 };

uint32_t checkPackets(uint8_t *src_packet, uint8_t *dst_packet, uint32_t psize);

void delay(int time)
{
    int i;
    for (i = 0; i < time; i++)
        ;
}

int main(void)
{
    canInit();
    _enable_interrupt_();

    printf("start\n");
    //canEnableErrorNotification(canREG1);
    //canIoSetDirection(canREG1, canMESSAGE_BOX1, canMESSAGE_BOX2);

    while (1)
    {
        printf("transmit CAN message\n");
        delay(6500000);
        canTransmit(canREG1, canMESSAGE_BOX1, tx_data);
    }
}

void canMessageNotification(canBASE_t *node, uint32_t messageBox)
{
        printf("Received CAN message\n");
        if(canIsRxMessageArrived(canREG1, canMESSAGE_BOX2)){
            canGetData(node, messageBox, rx_data);
            printf("rx_data : %s\n", rx_data);
        }
}
```