


Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 정한별

hanbulkr@gmail.com

SCI (UART) 프로그래밍

2018. 7. 12. 21:39

https://blog.naver.com/hanbulkr/221317923720

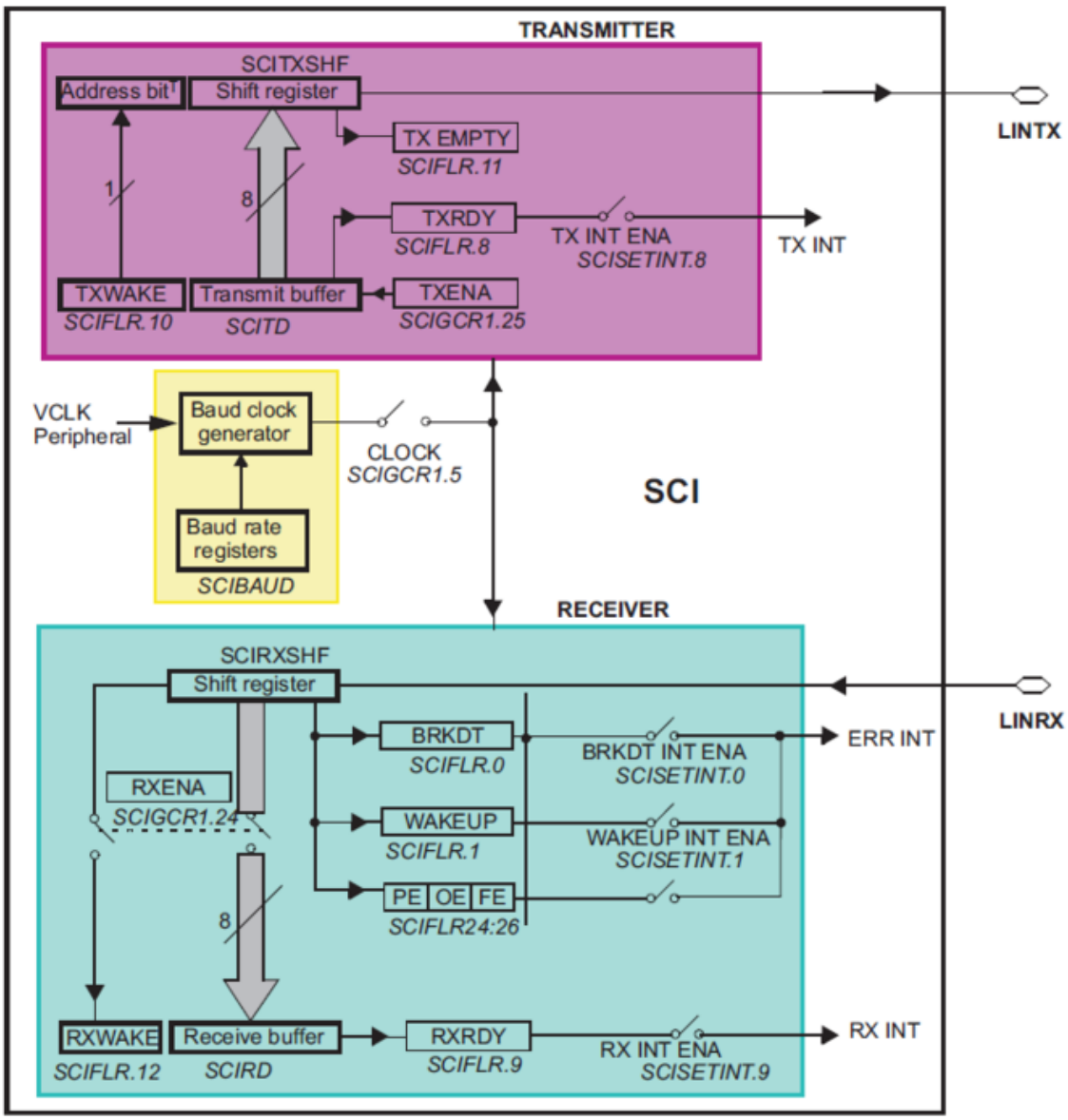
SCI(Serial Communication Interface - UART)

<기본 설명>

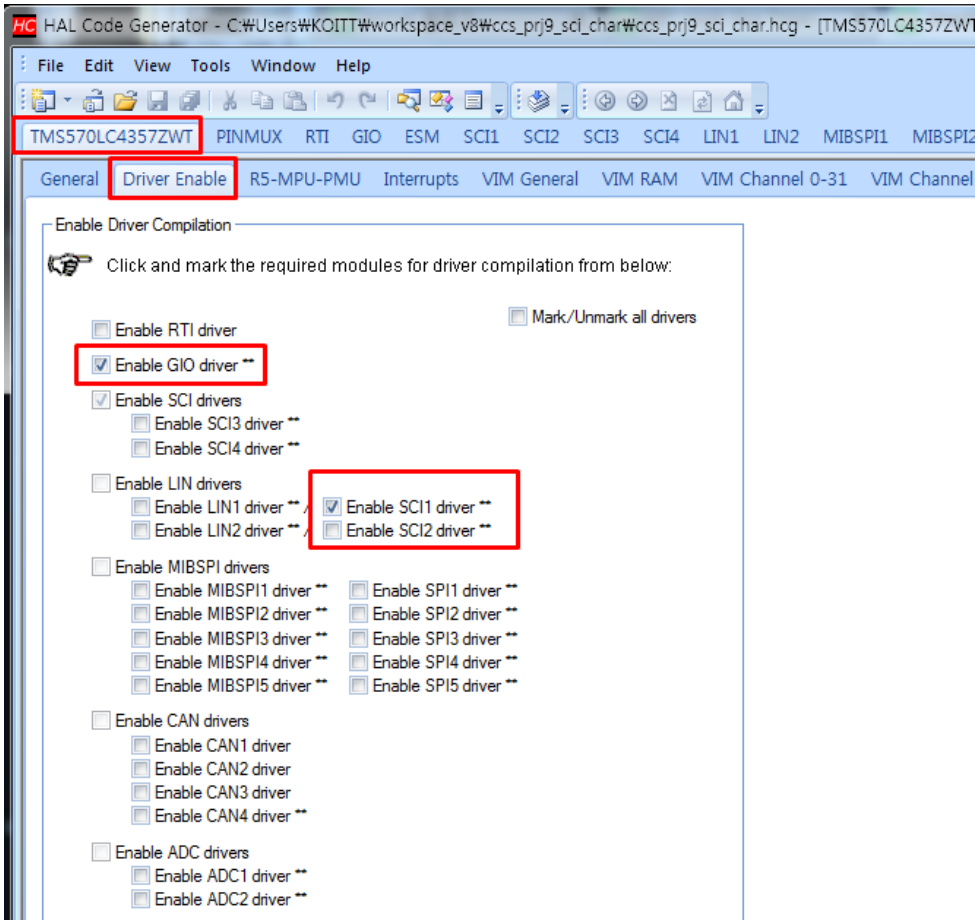
SCI / LIN 모듈은 SCI 또는 LIN으로 작동하도록 프로그래밍 할 수 있습니다. 모듈의 핵심은 다음과 같습니다.
SCI. LIN 호환성을 달성하기 위해 SCI의 하드웨어 기능이 보강되었습니다.
SCI 모듈은 범용 비동기식 수신기 / 송신기로, 표준 nonreturn 0 형식으로. SCI는 예를 들어 RS-232 포트 또는 Kline을 통해 통신하는 데 사용할 수 있습니다.
LIN 표준은 SCI (UART) 직렬 데이터 링크 형식을 기반으로합니다.
통신 개념은 임의의 네트워크 간 멀티 캐스트 전송을위한 메시지 식별 기능이있는 단일 마스터 / 다중 슬레이브 노드.

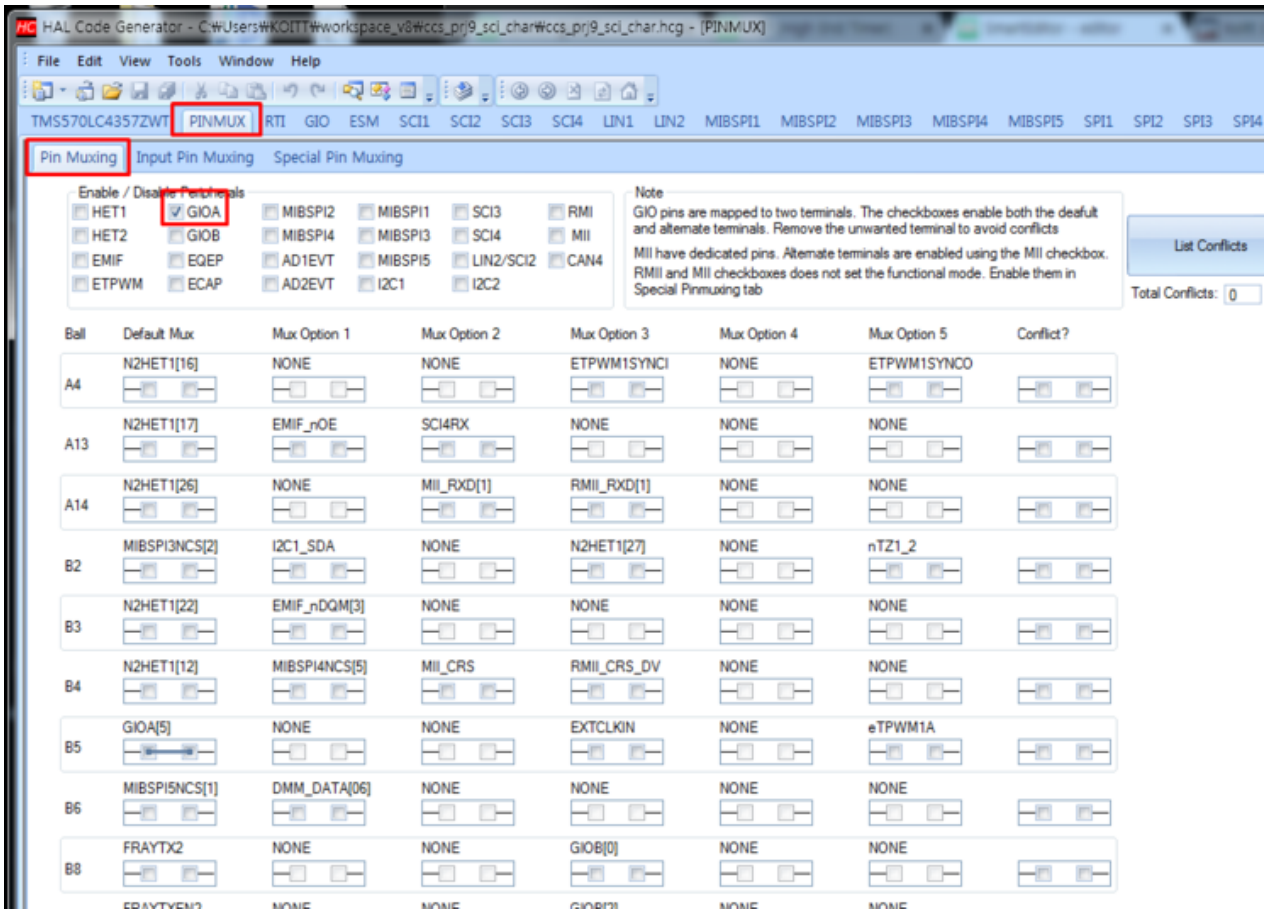
- 표준 범용 비동기 송수신기 (UART) 통신
- 전이중 또는 반이중 작동을 지원합니다.
- 표준 NRZ (nonreturn to zero) 형식
- 호환 모드에서 더블 버퍼링 된 수신 및 전송 기능
- 개별적으로 활성화 된 두 개의 인터럽트 라인을 지원합니다 (레벨 0 및 레벨 1).
- 다음을 기준으로 문자 당 3 ~ 13 비트의 구성 가능한 프레임 형식 :
  - 1에서 8 비트까지 프로그래밍 할 수있는 데이터 워드 길이
  - 주소 비트 모드의 추가 주소 비트
  - 0 또는 1 패리티 비트, 홀수 또는 짝수 패리티에 대해 패리티 프로그래밍 가능
  - 1 개 또는 2 개의 정지 비트에 대해 정지 프로그래밍 가능
- 비동기 또는 등시성 통신 모드
- 두 개의 다중 프로세서 통신 형식으로 두 개 이상의 장치 간 통신 가능
- 절전 모드는 다중 프로세서 통신 중에 CPU 리소스를 비운 다음 수신 메시지를 수신하기 위해 웨이크 업할 수 있습니다
- 24 비트 프로그래밍 가능 보드 속도는 224 가지 보드 속도를 지원하므로 고정밀 보드 속도 선택 기능을 제공합니다.
- 100MHz 주변 장치 클럭에서 3.125Mbits / s는 달성 가능한 최대 보드 율입니다
- 데이터 전송 및 수신을 위해 직접 메모리 액세스 (DMA)를 사용할 수있는 기능
- 5 개의 오류 플래그와 7 개의 상태 플래그가 SCI 이벤트에 대한 자세한 정보를 제공합니다.
- 두 개의 외부 핀 : LINRX 및 LINTX
- 다중 버퍼 수신 및 전송 장치

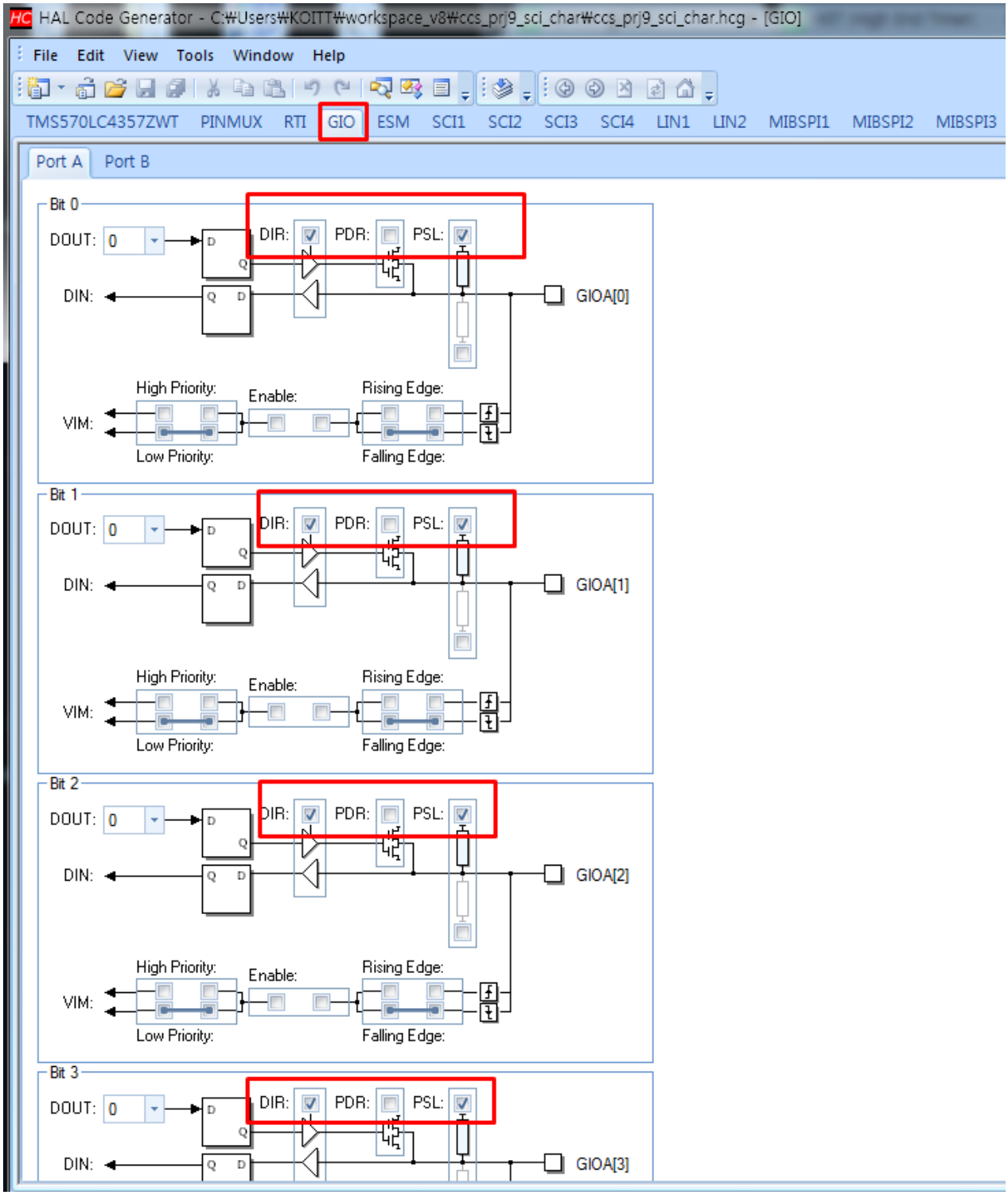
Figure 29-1. SCI Block Diagram



1) HALCoGen


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj9_sci_char#ccs_prj9_sci_char.hcg - [TMS570LC4357ZWT
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2
General Driver Enable R5-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel
Enable Driver Compilation
Click and mark the required modules for driver compilation from below:
Mark/Unmark all drivers
Enable RTI driver
Enable GIO driver **
Enable SCI drivers
Enable SCI3 driver **
Enable SCI4 driver **
Enable LIN drivers
Enable LIN1 driver **
Enable LIN2 driver **
Enable SCI1 driver **
Enable SCI2 driver **
Enable MIBSPI drivers
Enable MIBSPI1 driver **
Enable MIBSPI2 driver **
Enable MIBSPI3 driver **
Enable MIBSPI4 driver **
Enable MIBSPI5 driver **
Enable SPI1 driver **
Enable SPI2 driver **
Enable SPI3 driver **
Enable SPI4 driver **
Enable SPI5 driver **
Enable CAN drivers
Enable CAN1 driver
Enable CAN2 driver
Enable CAN3 driver
Enable CAN4 driver **
Enable ADC drivers
Enable ADC1 driver **
Enable ADC2 driver **


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj9_sci_char#ccs_prj9_sci_char.hcg - [PINMUX]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4
Pin Muxing Input Pin Muxing Special Pin Muxing
Enable / Disable Peripherals
HET1 HET2 EMIF ETPWM GIOA GIOB EQEP ECAP MIBSPI2 MIBSPI4 AD1EVT AD2EVT MIBSPI1 MIBSPI3 MIBSPI5 I2C1 SCI3 SCI4 LIN2/SCI2 I2C2 RMII MII CAN4
Note
GIO pins are mapped to two terminals. The checkboxes enable both the deafult and alternate terminals. Remove the unwanted terminal to avoid conflicts
MII have dedicated pins. Alternate terminals are enabled using the MII checkbox.
RMII and MII checkboxes does not set the functional mode. Enable them in Special Pinmuxing tab
List Conflicts
Total Conflicts: 0
Ball Default Mux Mux Option 1 Mux Option 2 Mux Option 3 Mux Option 4 Mux Option 5 Conflict?
A4 N2HET1[16] NONE NONE ETPWM1SYNCI NONE ETPWM1SYNCO
A13 N2HET1[17] EMIF_nOE SCI4RX NONE NONE NONE
A14 N2HET1[26] NONE MII_RXD[1] RMII_RXD[1] NONE NONE
B2 MIBSPI3NCS[2] I2C1_SDA NONE N2HET1[27] NONE nTZ1_2
B3 N2HET1[22] EMIF_nDQM[3] NONE NONE NONE NONE
B4 N2HET1[12] MIBSPI4NCS[5] MII_CRS RMII_CRS_DV NONE NONE
B5 GIOA[5] NONE NONE EXTCLKIN NONE eTPWM1A
B6 MIBSPI5NCS[1] DMM_DATA[06] NONE NONE NONE NONE
B8 FRAYTX2 NONE NONE GIOB[0] NONE NONE



2) CCS

- Hellow 보내기.(Tx)

```
 ANY DIRECT, INDIRECT, INCIDENTAL,


/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */
```

```
#include <HL_gio.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* USER CODE BEGIN (1) */
/* USER CODE END */


/* USER CODE BEGIN (2) */
/* USER CODE END */
void delay(uint32 time)
{
    while(time--)
        ;
}


void sci_Display(sciBASE_t *sci, uint32 length, uint8 * data)
{
    while(length--){

        while((sci->FLR & 0x4) == 4 );
        ;

        sciSendByte(sci, *data++);
    }
}


//void sciSendByte(sciBASE_t *sci, uint8 byte)


int main(void)
{
/* USER CODE BEGIN (3) */
    sciInit();
    gioInit();


    char test_msg[7] = {'H','E','L','L','O','\r','\n'};
    int x = 0;

    while(1){
        for(x = 0; x<7 ;x++)
            sciSendByte(sciREG1 ,test_msg[x]);

        sci_Display(sciREG1, strlen(test_msg), (uint8 *)test_msg);

        delay(10000000);
    }
```

```
/* USER CODE END */

    return 0;
}


/* USER CODE BEGIN (4) */
/* USER CODE END */
```

- Tx → Rx , Rx → Tx 형식으로 echo 통신.(debug용도 처럼)

1) 한글자 보내고 받기. (숫자 입력에 따른 글자를 ECHO 통신 하기_)

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include <HL_gio.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* USER CODE BEGIN (1) */
/* USER CODE END */


/* USER CODE BEGIN (2) */
/* USER CODE END */
void delay(uint32 time)
{
    while(time--)
        ;
}


void sci_Display(sciBASE_t *sci, uint32 length, uint8 * data)
{
    while(length--){

        while((sci->FLR & 0x4) == 4 );
        ;

        sciSendByte(sci, *data++);
    }
}
void sci_scanf(sciBASE_t *sci, uint8 rev_data)
{
    while(!sciIsRxReady(sci))
        ;
    rev_data =(uint8) sciReceiveByte(sci);
```

```
        sciSendByte(sci, rev_data);
        if(rev_data == '\n' || rev_data == '\r')
            if(rev_data == '\n' || rev_data == '\r')
                sciSend(sciREG1, 2, "\r\n");
    }

    //void sciSendByte(sciBASE_t *sci, uint8 byte)


    int main(void)
    {
    /* USER CODE BEGIN (3) */
        sciInit();
        gioInit();

        uint8 recieve_data[128];

        while(1){
            sci_scanf(sciREG1 , *recieve_data);

    //        memset(recieve_data, 0, sizeof(recieve_data));
        }

    /* USER CODE END */

        return 0;
    }


    /* USER CODE BEGIN (4) */
    /* USER CODE END */
```

- 수신 값에 따라 led 출력하기.

2) 문장 보내고 받기. (문장을 보내면 그에 따른 답변이 오도록 통신 하기_) -> 명령어에 따른 동작 구문의 예로 사용.

- 글자를

"wow hello guy!"

"ok!"

"thank you"

"you wellcome"

"goodbye!!"

를 치고 엔터를 누르면 그 값에 따른 대답이 오도록 하였다.

나중에 프로젝트중 구현해야 하는 명령어를 받았을 때 동작을 이런 형식으로 해야 하기 때문에 구현을 한 것이다.

```
    /* USER CODE BEGIN (0) */
    /* USER CODE END */

    /* Include Files */
```

```
#include <HL_gio.h>
#include <HL_reg_sci.h>
#include <HL_hal_stdtypes.h>
#include <HL_sci.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* USER CODE BEGIN (1) */
/* USER CODE END */

/* USER CODE BEGIN (2) */
/* USER CODE END */

uint8 data_buffer[]= {0};
uint8 check_msg1[]="wow hello guy!";
uint8 check_msg2[]="ok!";
uint8 check_msg3[]="thank you";
uint8 check_msg4[]="you wellcome";
uint8 check_msg5[]="goodbye!!";
uint8 check_msg6[]="non";

int idx = 0;

void delay(uint32 time)
{
    while (time--)
        ;
}

void sci_Display(sciBASE_t *sci, uint32 length, uint8 * data)
{
    while (length--)
    {
        while ((sci->FLR & 0x4) == 4)
            ;;
        sciSendByte(sci, *data++);
    }
}

void check_msg(uint8 * data)
{
    if(!strncmp((const char *)data, "hello",5)){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, strlen((const char *)check_msg1),check_msg1);
        sciSend(sciREG1, 2, "\r\n");
    }
    else if(!strncmp((const char *)data, "start",5)){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, strlen((const char *)check_msg2),check_msg2);
        sciSend(sciREG1, 2, "\r\n");
    }
    else if(!strncmp((const char *)data, "nice",4)){
```

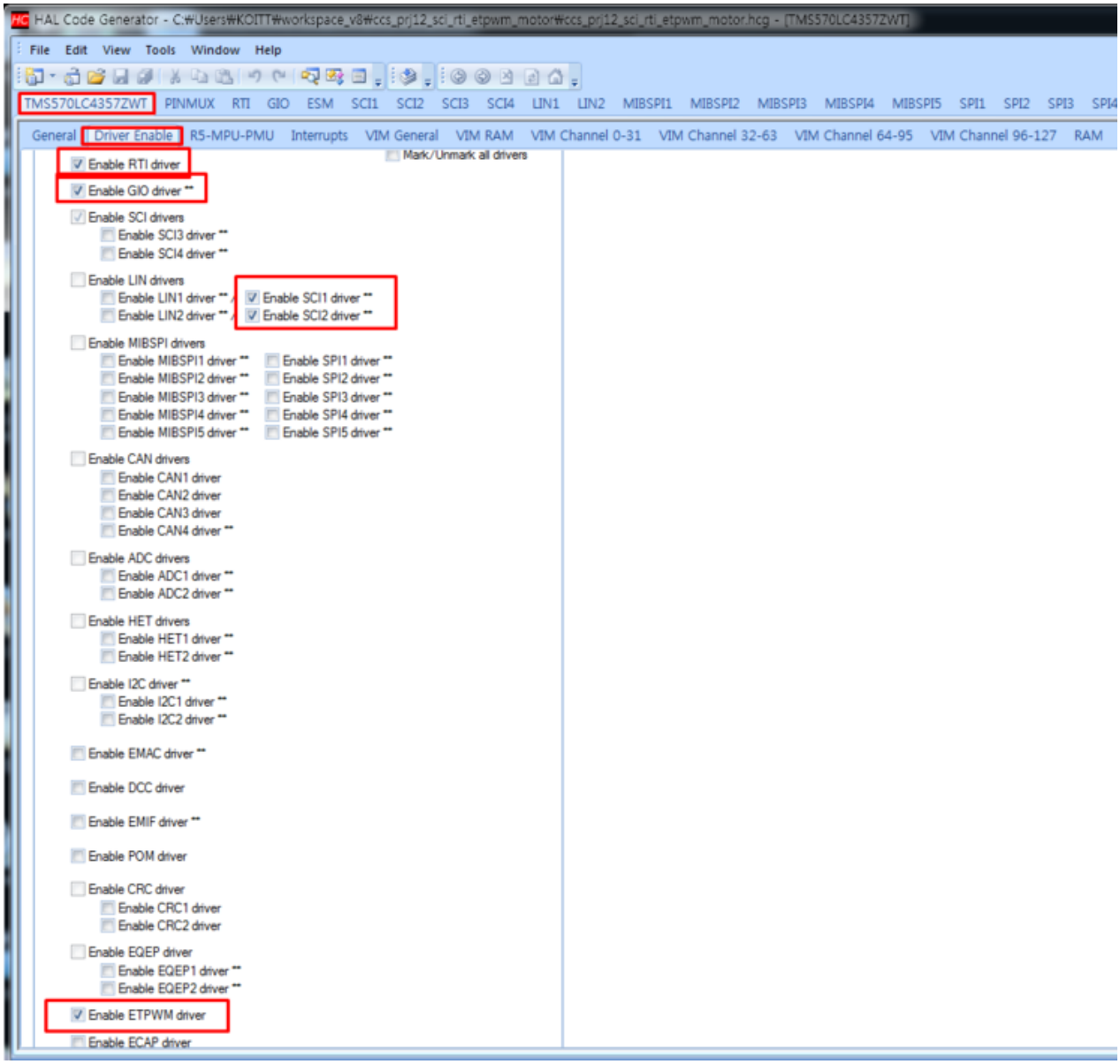
```
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, strlen((const char *)check_msg3),check_msg3);
        sciSend(sciREG1, 2, "\r\n");
    }
    else if(!strncmp((const char *)data, "thank you",9)){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, strlen((const char *)check_msg4),check_msg4);
        sciSend(sciREG1, 2, "\r\n");
    }
    else if(!strncmp((const char *)data, "bye",3)){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, strlen((const char *)check_msg5),check_msg5);
        sciSend(sciREG1, 2, "\r\n");
    }
    else{
        sciSend(sciREG1, 2, "\r\n");
        //sciSend(sciREG1, 12,"its not mine");
        sciSend(sciREG1, strlen((const char *)check_msg6),check_msg6);
        sciSend(sciREG1, 2, "\r\n");
    }
}

void sci_scanf(sciBASE_t *sci, uint8 rev_data)
{
    while (!sciIsRxReady(sci))
        ;
    rev_data = (uint8) sciReceiveByte(sci);
    data_buffer[idx++]=rev_data;
    sciSendByte(sci, rev_data);
    if (rev_data == '\n' || rev_data == '\r')
    {
        if (rev_data == '\n' || rev_data == '\r')
        {
            sciSend(sciREG1, 2, "\r\n");
            sciSend(sciREG1, 15, "receive msg: " );
            sciSend(sciREG1, strlen((const char *)data_buffer), data_buffer);
            sciSend(sciREG1, 2, "\r\n");

            check_msg(data_buffer);

            idx = 0;
        }
    }
}

//void sciSendByte(sciBASE_t *sci, uint8 byte)

int main(void)
{
/* USER CODE BEGIN (3) */
sciInit();
gioInit();


uint8 recieve_data[128];
```

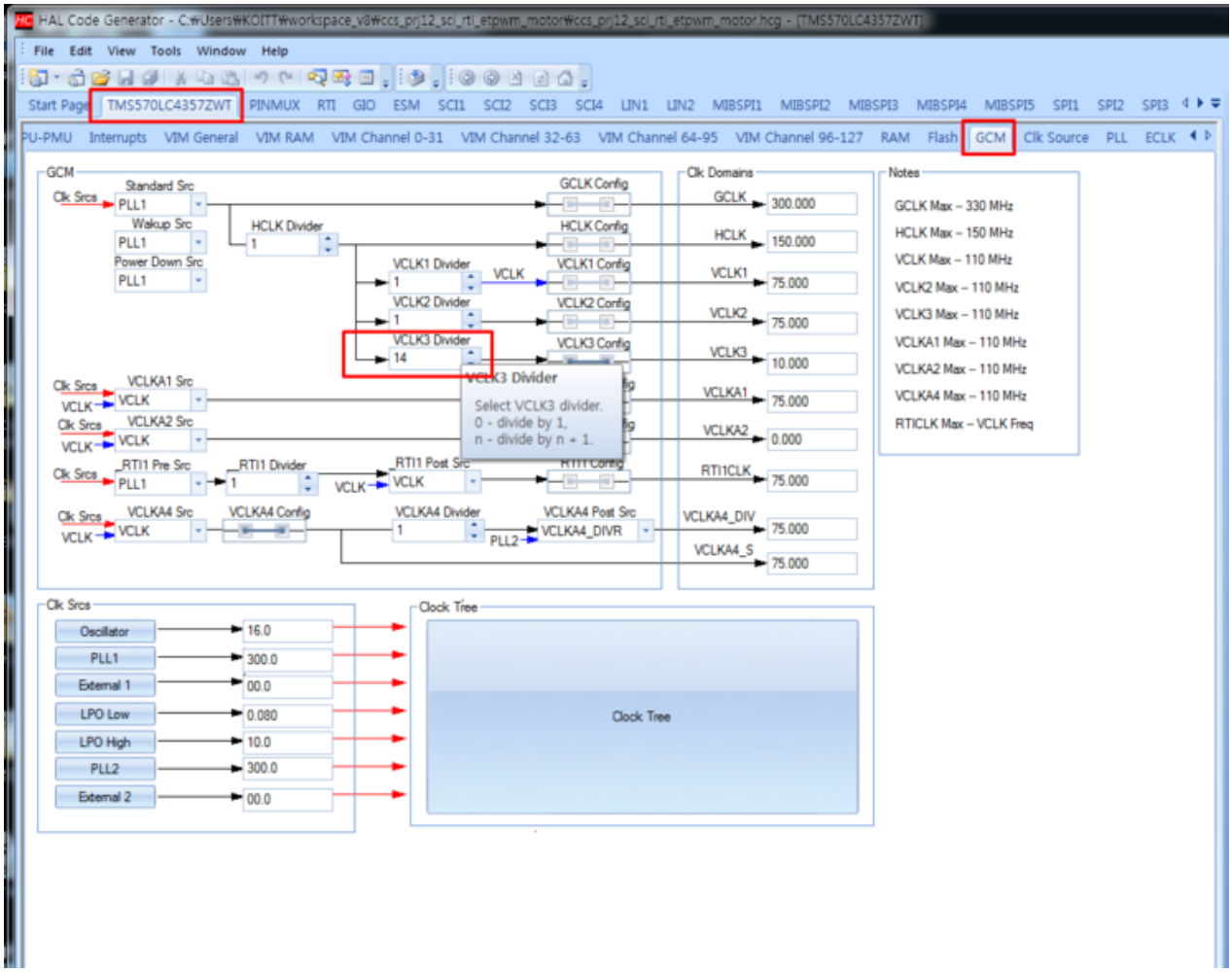
```
while (1)
{
    // 스켄에프처럼 입력을 받고 화면에 띄우고 끝냄.
    // 내가 만들것은 문자을 받아서 넘기는 함수를 구현해야 한다.

    sci_scanf(sciREG1, *recieve_data);

}

/* USER CODE END */

return 0;
}

/* USER CODE BEGIN (4) */
/* USER CODE END */
```

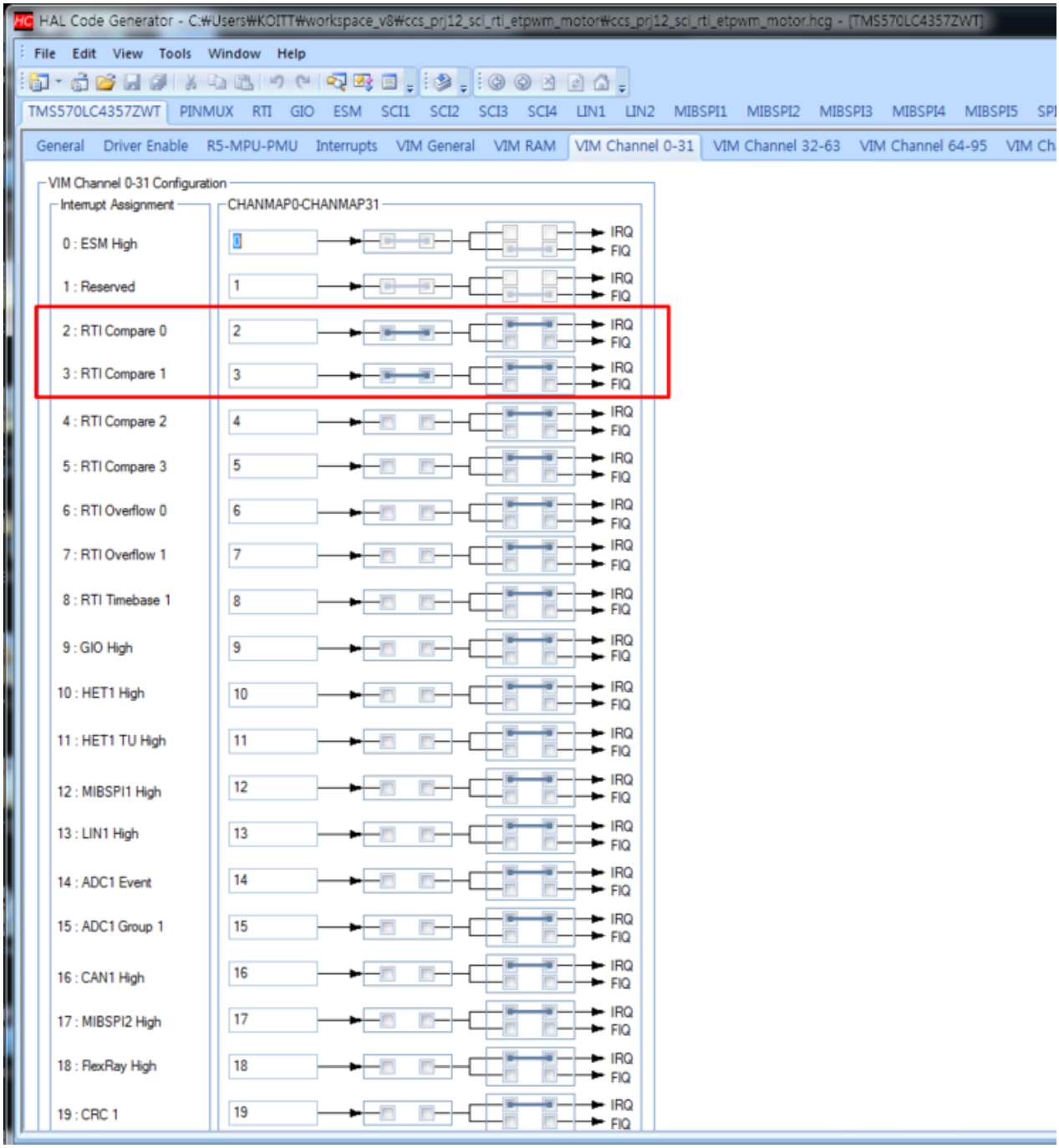
3) 인터럽트로 처리해서 받기. (언제라도 글자가 들어오면 받고 응답이 가능하게 함_)

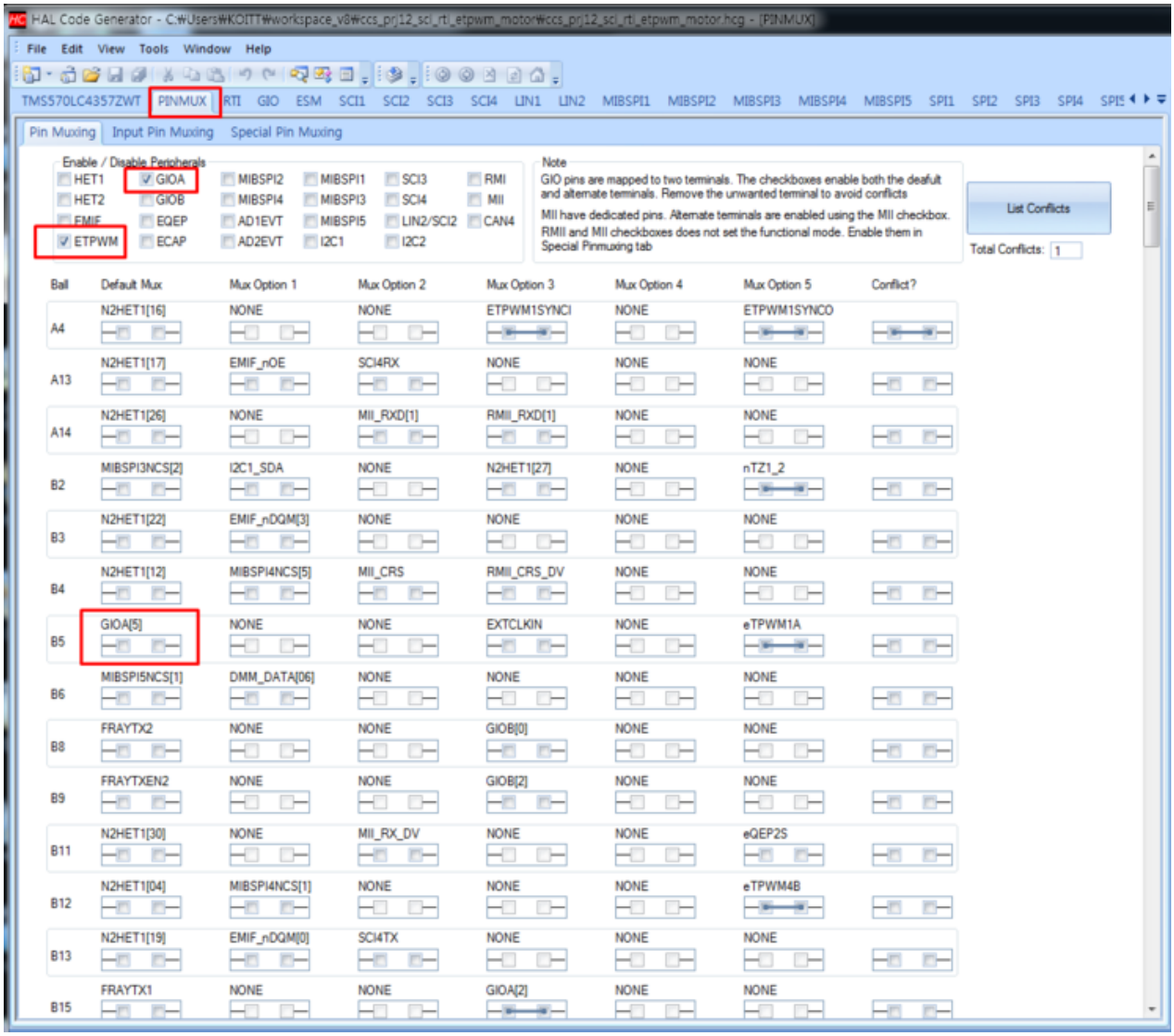
- sci_rti_etPWM - Tx 로 오는 값에 따라서 모터 제어.

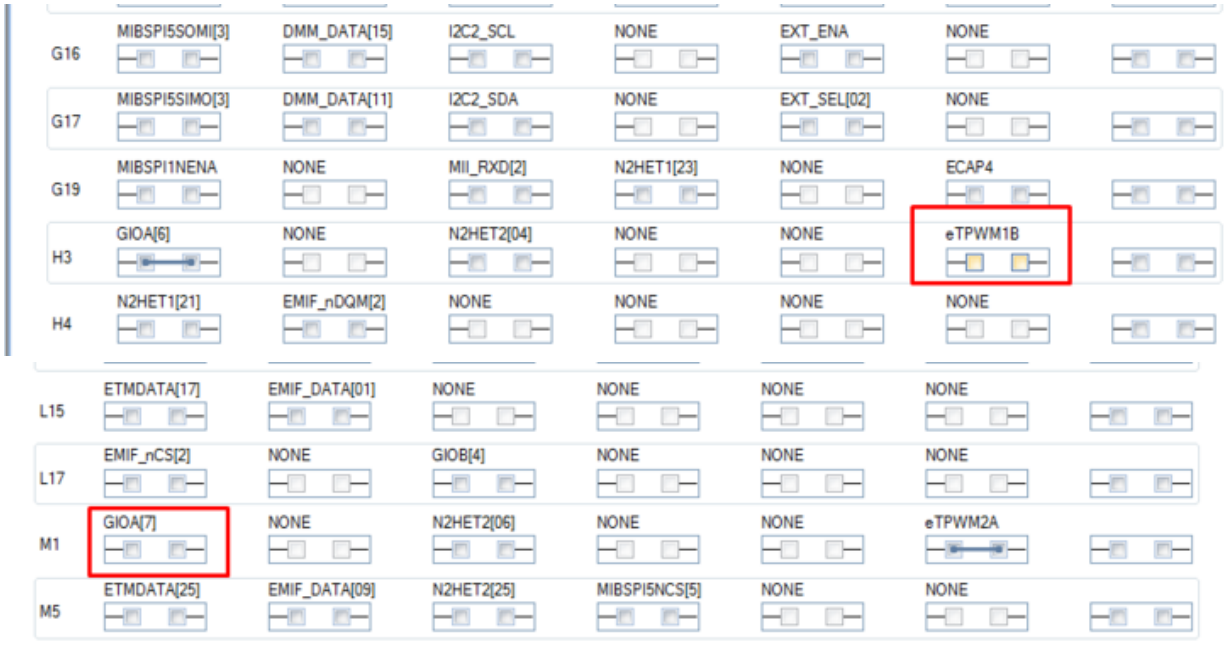
1. 자동으로 최대 최소로 왔다갔다하는 모드
2. 주어진 문자 값에 따라서 동작하는 모드


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj12_sci_rti_etpwm_motor#ccs_prj12_sci_rti_etpwm_motor.hcg - [TMS570LC4357ZWT]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4
General Driver Enable R5-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel 32-63 VIM Channel 64-95 VIM Channel 96-127 RAM
Mark/Unmark all drivers
Enable RTI driver
Enable GIO driver **
Enable SCI drivers
Enable SCI3 driver **
Enable SCI4 driver **
Enable LIN drivers
Enable LIN1 driver **
Enable LIN2 driver **
Enable SCI1 driver **
Enable SCI2 driver **
Enable MIBSPI drivers
Enable MIBSPI1 driver **
Enable MIBSPI2 driver **
Enable MIBSPI3 driver **
Enable MIBSPI4 driver **
Enable MIBSPI5 driver **
Enable SPI1 driver **
Enable SPI2 driver **
Enable SPI3 driver **
Enable SPI4 driver **
Enable SPI5 driver **
Enable CAN drivers
Enable CAN1 driver
Enable CAN2 driver
Enable CAN3 driver
Enable CAN4 driver **
Enable ADC drivers
Enable ADC1 driver **
Enable ADC2 driver **
Enable HET drivers
Enable HET1 driver **
Enable HET2 driver **
Enable I2C driver **
Enable I2C1 driver **
Enable I2C2 driver **
Enable EMAC driver **
Enable DCC driver
Enable EMIF driver **
Enable POM driver
Enable CRC driver
Enable CRC1 driver
Enable CRC2 driver
Enable EQEP driver
Enable EQEP1 driver **
Enable EQEP2 driver **
Enable ETPWM driver
Enable ECAP driver

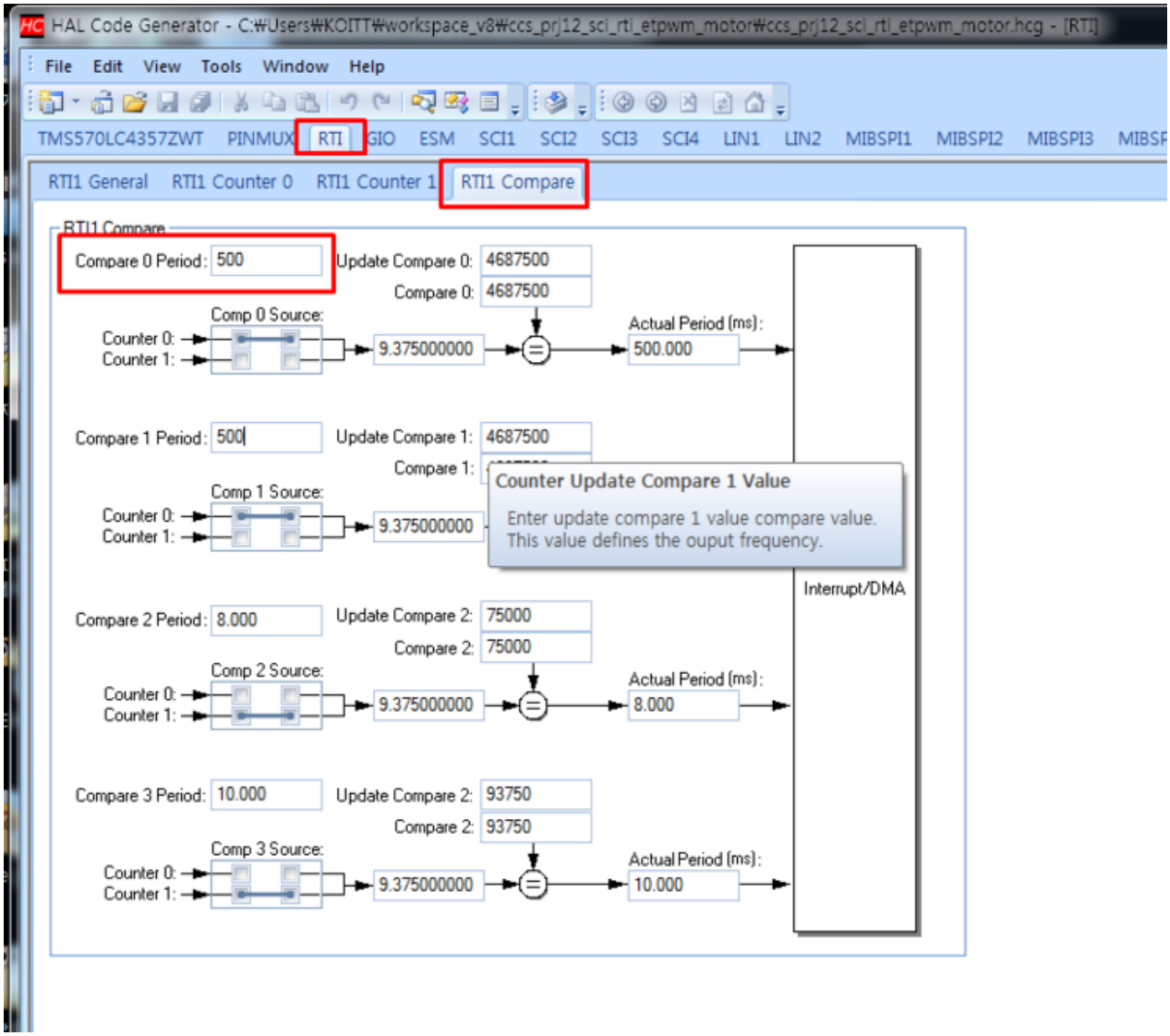

HAL Code Generator - C:\Users\KOITT\workspace_v8\ccs_prj12_sci_rti_etpwm_motor\ccs_prj12_sci_rti_etpwm_motor.hcg - (TMS570LC4357ZWT)
File Edit View Tools Window Help
Start Page
TMS570LC4357ZWT
PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3
PU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel 32-63 VIM Channel 64-95 VIM Channel 96-127 RAM Flash GCM Clk Source PLL ECLK
GCM
Clk Srcs
Standard Src
PLL1
Wakup Src
PLL1
Power Down Src
PLL1
HCLK Divider
1
GCLK Config
HCLK Config
VCLK1 Divider
1
VCLK
VCLK1 Config
VCLK2 Divider
1
VCLK2 Config
VCLK3 Divider
14
VCLK3 Config
VCLK3 Divider
Select VCLK3 divider.
0 - divide by 1,
n - divide by n + 1.
VCLKA1 Src
VCLK
VCLKA2 Src
VCLK
RTI1 Pre Src
PLL1
RTI1 Divider
1
RTI1 Post Src
VCLK
VCLKA4 Src
VCLK
VCLKA4 Config
VCLKA4 Divider
1
VCLKA4 Post Src
VCLKA4_DIVR
PLL2
Clk Domains
GCLK 300.000
HCLK 150.000
VCLK1 75.000
VCLK2 75.000
VCLK3 10.000
VCLKA1 75.000
VCLKA2 0.000
RTI1CLK 75.000
VCLKA4_DIV 75.000
VCLKA4_S 75.000
Notes
GCLK Max -- 330 MHz
HCLK Max -- 150 MHz
VCLK Max -- 110 MHz
VCLK2 Max -- 110 MHz
VCLK3 Max -- 110 MHz
VCLKA1 Max -- 110 MHz
VCLKA2 Max -- 110 MHz
VCLKA4 Max -- 110 MHz
RTICLK Max -- VCLK Freq
Clk Srcs
Oscillator 16.0
PLL1 300.0
External 1 00.0
LPO Low 0.080
LPO High 10.0
PLL2 300.0
External 2 00.0
Clock Tree
Clock Tree


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj12_sci_rti_etpwm_motor#ccs_prj12_sci_rti_etpwm_motor.hcg - [TMS570LC4357ZWT]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5
General Driver Enable R5-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel 32-63 VIM Channel 64-95
VIM Channel 0-31 Configuration
Interrupt Assignment
CHANMAP0-CHANMAP31
0 : ESM High
1 : Reserved
2 : RTI Compare 0
3 : RTI Compare 1
4 : RTI Compare 2
5 : RTI Compare 3
6 : RTI Overflow 0
7 : RTI Overflow 1
8 : RTI Timebase 1
9 : GIO High
10 : HET1 High
11 : HET1 TU High
12 : MIBSPI1 High
13 : LIN1 High
14 : ADC1 Event
15 : ADC1 Group 1
16 : CAN1 High
17 : MIBSPI2 High
18 : FlexRay High
19 : CRC 1
IRQ
FIQ

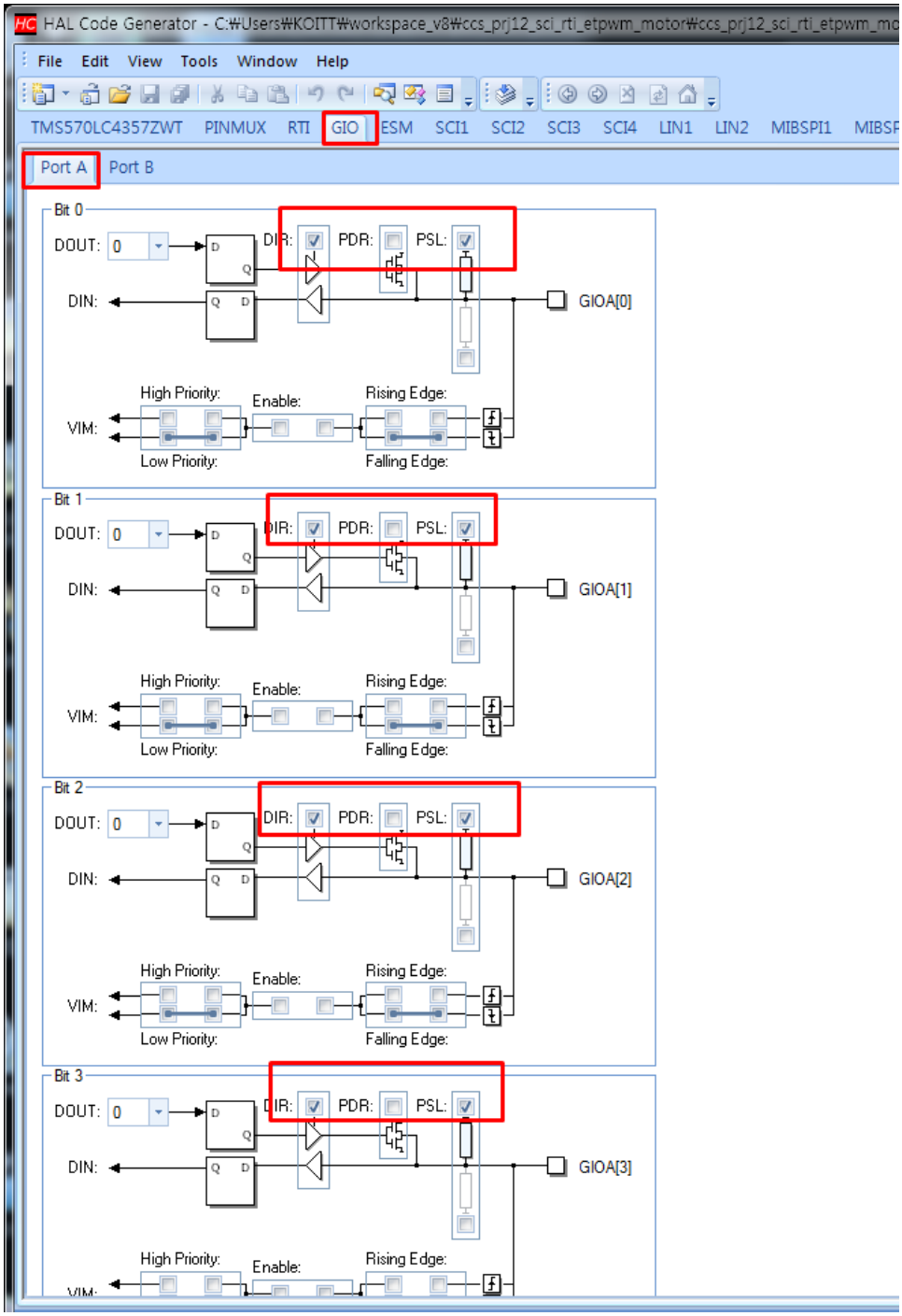


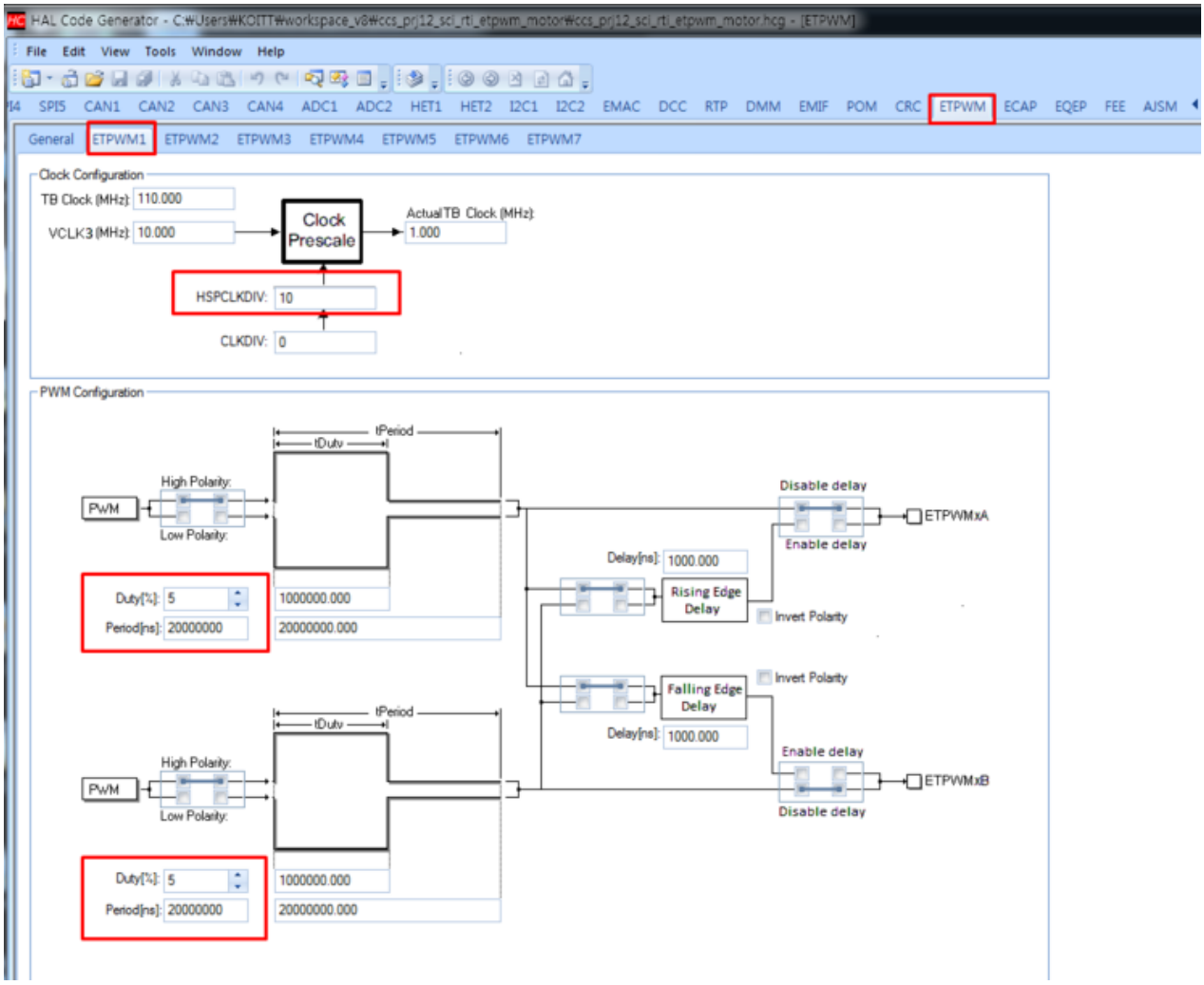


* 위의 핀을 해제 하는 이유는 같이 쓰는 것 보다 한가지로 사용하여 겹치는 신호를 방지 하기 위해서 이다.
* 하지만 내가 할 예제에서는 핀을 많이 사용 하는 것이 아니고 1가지(etpwm1)만 쓸 것이다. 그래서 사실상 상관은 없다.



혹시 모를 gio를 예제로 사용 할 수 있으니 Aport에서 0~3까지만 열어 둔다.


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj12_sci_rti_etpwm_motor#ccs_prj12_sci_rti_etpwm_mo
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSP
Port A Port B
Bit 0
DOUT: 0
DIR: PDR: PSL:
DIN:
GIOA[0]
High Priority: Enable: Rising Edge:
VIM:
Low Priority: Falling Edge:
Bit 1
DOUT: 0
DIR: PDR: PSL:
DIN:
GIOA[1]
High Priority: Enable: Rising Edge:
VIM:
Low Priority: Falling Edge:
Bit 2
DOUT: 0
DIR: PDR: PSL:
DIN:
GIOA[2]
High Priority: Enable: Rising Edge:
VIM:
Low Priority: Falling Edge:
Bit 3
DOUT: 0
DIR: PDR: PSL:
DIN:
GIOA[3]
High Priority: Enable: Rising Edge:
VIM:




```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include <HL_etpwm.h>
#include <HL_gio.h>
#include <HL_hal_stdtypes.h>if(!strncmp((const char *) data, "angle2", 6))
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "angle2\r\n");
        sciSend(sciREG1, 2, "\r\n");

        etpwmStartTBCLK();
        delay(10);
        etpwmSetCmpA(etpwmREG1,900);
        delay(10);

    }
    else if(!strncmp((const char *) data, "angle3", 6))
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "angle3\r\n");
        sciSend(sciREG1, 2, "\r\n");
```

```
        etpwmStartTBCLK();
        delay(10);
        etpwmSetCmpA(etpwmREG1,1500);
        delay(10);

    }
    else if(!strncmp((const char *) data, "angle4", 6))
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "angle4\r\n");
        sciSend(sciREG1, 2, "\r\n");

        etpwmStartTBCLK();
        delay(10);
        etpwmSetCmpA(etpwmREG1,2300);
        delay(10);

    }
    else if(!strncmp((const char *) data, "angle5", 6))
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "angle5\r\n");
        sciSend(sciREG1, 2, "\r\n");

        etpwmStartTBCLK();
        delay(10);
        etpwmSetCmpA(etpwmREG1,2600);
        delay(10);

    }
    else if(!strncmp((const char *) data, "angle6", 6))
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "angle6\r\n");
        sciSend(sciREG1, 2, "\r\n");

        etpwmStartTBCLK();
        delay(10);
        etpwmSetCmpA(etpwmREG1,3000);
        delay(10);
    }
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////
    else
    {
        sciSend(sciREG1, 2, "\r\n");
        //sciSend(sciREG1, 12,"its not mine");
        sciSend(sciREG1, strlen((const char *) check_msg6), check_msg6);
        sciSend(sciREG1, 2, "\r\n");
    }
}

void sci_scanf(sciBASE_t *sci, uint8 rev_data)
{
```

```
    while (!sciIsRxReady(sci))
        ;
    rev_data = (uint8) sciReceiveByte(sci);
    data_buffer[idx++] = rev_data;
    sciSendByte(sci, rev_data);
    if (rev_data == '\n' || rev_data == '\r')
    {
        if (rev_data == '\n' || rev_data == '\r')
        {
            sciSend(sciREG1, 2, "\r\n");
            sciSend(sciREG1, 15, "receive msg: ");
            sciSend(sciREG1, strlen((const char *) data_buffer), data_buffer);
            sciSend(sciREG1, 2, "\r\n");

            check_msg(data_buffer);

            idx = 0;
        }
    }
}

int main(void)
{
    /* USER CODE BEGIN (3) */
    sciInit();
    rtiInit();
    gioInit();
    etpwmInit();
    flag = 1U;
//    gioSetDirection(gioPORTA,0xffffU);

    rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);

//    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
//    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK1);

    _enable_IRQ_interrupt_();
    uint8 recieve_data[128];

    while (1)
    {
        sci_scanf(sciREG1, *recieve_data);
        //etpwmSetCmpA(etpwmREG1,plus_count);
    }

    /* USER CODE END */

    return 0;
}

/* USER CODE BEGIN (4) */
/* USER CODE END */
```

* 알게된 점:

servo 모터는 0.5 m/s ~ 2.5 m/s까지 해야 0~180도가 돌아가는 것을 알 수 있었다.

3) 코드 분석

<SCI 가 가지고 있는 기본 함수들>

```
/* SCI Interface Functions */
void sciInit(void);
void sciSetFunctional(sciBASE_t *sci, uint32 port);
void sciSetBaudrate(sciBASE_t *sci, uint32 baud);
uint32  sciIsTxReady(sciBASE_t *sci);
void sciSendByte(sciBASE_t *sci, uint8 byte);
void sciSend(sciBASE_t *sci, uint32 length, uint8 * data);
uint32  sciIsRxReady(sciBASE_t *sci);
uint32  sciIsIdleDetected(sciBASE_t *sci);
uint32  sciRxError(sciBASE_t *sci);
uint32  sciReceiveByte(sciBASE_t *sci);
void sciReceive(sciBASE_t *sci, uint32 length, uint8 * data);
void sciEnableNotification(sciBASE_t *sci, uint32 flags);
void sciDisableNotification(sciBASE_t *sci, uint32 flags);
void sciEnableLoopback(sciBASE_t *sci, loopBackType_t Loopbacktype);
void sciDisableLoopback(sciBASE_t *sci);
void sciEnterResetState(sciBASE_t *sci);
void sciExitResetState(sciBASE_t *sci);
void sci1GetConfigValue(sci_config_reg_t *config_reg, config_value_type_t type);
/** @fn void sciNotification(sciBASE_t *sci, uint32 flags)
*   @brief Interrupt callback
*   @param[in] sci   - sci module base address
*   @param[in] flags - copy of error interrupt flags
*
* This is a callback that is provided by the application and is called upon
* an interrupt.  The parameter passed to the callback is a copy of the
* interrupt flag register.
*/
void sciNotification(sciBASE_t *sci, uint32 flags);
```

```
< void sciSetFunctional(sciBASE_t *sci, uint32 port) >
```

* 런타임시 PCPIO0 레지스터의 값을 변경하면 기능과 GIO 모드 사이에서 SCI 핀의 기능을 동적으로 변경할 수 있습니다.

```
/* SourceId : SCI_SourceId_002 */
/* DesignId : SCI_DesignId_002 */
/* Requirements : HL_CONQ_SCI_SR6 */
/** @fn void sciSetFunctional(sciBASE_t *sci, uint32 port)
*   @brief Change functional behavior of pins at runtime.
*   @param[in] sci   - sci module base address
*   @param[in] port  - Value to write to PIO0 register
*
*   Change the value of the PCPIO0 register at runtime, this allows to
*   dynamically change the functionality of the SCI pins between functional
*   and GIO mode.
*/
void sciSetFunctional(sciBASE_t *sci, uint32 port)
{
/* USER CODE BEGIN (4) */
/* USER CODE END */

    sci->PIO0 = port;

/* USER CODE BEGIN (5) */
/* USER CODE END */
}
```

```
void sciSetBaudrate(sciBASE_t *sci, uint32 baud)
```

* baudrate runtime 값을 바꿀 수 있습니다.

```
/* SourceId : SCI_SourceId_003 */
/* DesignId : SCI_DesignId_003 */
/* Requirements : HL_CONQ_SCI_SR7 */
/** @fn void sciSetBaudrate(sciBASE_t *sci, uint32 baud)
*   @brief Change baudrate at runtime.
*   @param[in] sci  - sci module base address
*   @param[in] baud - baudrate in Hz
*
*   Change the SCI baudrate at runtime.
*/
void sciSetBaudrate(sciBASE_t *sci, uint32 baud)
{
    float64   vclk = 75.000 * 1000000.0;
    uint32 f    = ((sci->GCR1 & 2U) == 2U) ? 16U : 1U;
    uint32 temp;
    float64 temp2;
/* USER CODE BEGIN (6) */
/* USER CODE END */

    /*SAFETYMCUSW 96 S MR:6.1 <APPROVED> "Calculations including int and float cannot be avoided" */
    temp = (f*(baud));
    temp2 = ((vclk)/((float64)temp))-1U;
    temp2 = floor(temp2 + 0.5);        /* Rounding-off to the closest integer */
    sci->BRS = (uint32)((uint32)temp2 & 0x00FFFFFFU);

/* USER CODE BEGIN (7) */
/* USER CODE END */
}
```

```
uint32 sciIsTxReady(sciBASE_t *sci)
```

* Tx 버퍼 준비 플래그가 설정되어 있는지를 확인하고, 0이 아닌 플래그를 반환합니다.
  그렇지 않으면 Tx 플래그 자체가 반환됩니다.

```
/* SourceId : SCI_SourceId_004 */
/* DesignId : SCI_DesignId_004 */
/* Requirements : HL_CONQ_SCI_SR8 */
/** @fn uint32 sciIsTxReady(sciBASE_t *sci)
*   @brief Check if Tx buffer empty
*   @param[in] sci - sci module base address
*
*   @return The TX ready flag
*
*   Checks to see if the Tx buffer ready flag is set, returns
*   0 is flags not set otherwise will return the Tx flag itself.
*/
uint32 sciIsTxReady(sciBASE_t *sci)
{
/* USER CODE BEGIN (8) */
/* USER CODE END */

    return sci->FLR & (uint32)SCI_TX_INT;
}
```

```
void sciSendByte(sciBASE_t *sci, uint8 byte)
```

* 폴링 모드에서 단일 바이트를 보내고 바이트를 보내기 전에 전송 버퍼가 비어있을 때까지 루틴에서 대기합니다. 대기를 피하기 위해 sciSendByte를 호출하기 전에 sciIsTxReady를 사용하여 Tx 버퍼가 비어 있는지 확인하십시오.

```
/* SourceId : SCI_SourceId_005 */
/* DesignId : SCI_DesignId_005 */
/* Requirements : HL_CONQ_SCI_SR9 */
/** @fn void sciSendByte(sciBASE_t *sci, uint8 byte)
*   @brief Send Byte
*   @param[in] sci  - sci module base address
*   @param[in] byte - byte to transfer
*
*   Sends a single byte in polling mode, will wait in the
*   routine until the transmit buffer is empty before sending
*   the byte.  Use sciIsTxReady to check for Tx buffer empty
*   before calling sciSendByte to avoid waiting.
*/
void sciSendByte(sciBASE_t *sci, uint8 byte)
{
/* USER CODE BEGIN (9) */
/* USER CODE END */

    /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Potentially infinite loop found - Hardware Status check for execution sequence" */
    while ((sci->FLR & (uint32)SCI_TX_INT) == 0U)
    {
    } /* Wait */
    sci->TD = byte;

/* USER CODE BEGIN (10) */
/* USER CODE END */
}
```

```
void sciSend(sciBASE_t *sci, uint32 length, uint8 * data)
```

* 'data'및 'length'바이트가 가리키는 데이터 블록을 보내십시오. 인터럽트가 활성화 된 경우 인터럽트 모드를 사용하여 데이터가 전송되고, 그렇지 않으면 폴링 모드가 사용됩니다. 인터럽트 모드에서 첫 번째 바이트의 전송이 시작되고 루틴이 즉시 반환됩니다. sciNotification 콜백이 호출 될 때까지 전송이 완료 될 때까지 sciSend를 다시 호출하면 안됩니다. 폴링 모드에서는 전송이 완료 될 때까지 sciSend가 반환되지 않습니다.

```
/* SourceId : SCI_SourceId_006 */
/* DesignId : SCI_DesignId_006 */
/* Requirements : HL_CONQ_SCI_SR10 */
/** @fn void sciSend(sciBASE_t *sci, uint32 length, uint8 * data)
*   @brief Send Data
*   @param[in] sci    - sci module base address
*   @param[in] length - number of data words to transfer
*   @param[in] data   - pointer to data to send
*
*   Send a block of data pointed to by 'data' and 'length' bytes
*   long.  If interrupts have been enabled the data is sent using
*   interrupt mode, otherwise polling mode is used.  In interrupt
*   mode transmission of the first byte is started and the routine
*   returns immediately, sciSend must not be called again until the
*   transfer is complete, when the sciNotification callback will
*   be called.  In polling mode, sciSend will not return  until
*   the transfer is complete.
*
*   @note if data word is less than 8 bits, then the data must be left
*         aligned in the data byte.
*/
void sciSend(sciBASE_t *sci, uint32 length, uint8 * data)
{
    uint32 index = (sci  == sciREG1) ? 0U :
                        ((sci == sciREG2) ? 1U :
                        ((sci == sciREG3) ? 2U : 3U));
    uint8 txdata;

/* USER CODE BEGIN (11) */
/* USER CODE END */
/*SAFETYMCUSW 139 S MR:13.7 <APPROVED> "Mode variable is configured in sciEnableNotification()" */
    if ((g_sciTransfer_t[index].mode & (uint32)SCI_TX_INT) != 0U)
    {
        /* we are in interrupt mode */

        g_sciTransfer_t[index].tx_length = length;
        /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
        g_sciTransfer_t[index].tx_data   = data;

        /* start transmit by sending first byte */
        /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
        txdata = *g_sciTransfer_t[index].tx_data;
        sci->TD     = (uint32)(txdata);
        /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
        /*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" */
        g_sciTransfer_t[index].tx_data++;
        sci->SETINT = (uint32)SCI_TX_INT;
    }
```

```
    else
    {
        /* send the data */
        /*SAFETYMCUSW 30 S MR:12.2,12.3 <APPROVED> "Used for data count in Transmit/Receive polling and Interrupt mode" */
        while (length > 0U)
        {
            /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Potentially infinite loop found - Hardware Status check for execution sequence" */
            while ((sci->FLR & (uint32)SCI_TX_INT) == 0U)
            {
            } /* Wait */
            /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
            txdata = *data;
            sci->TD = (uint32)(txdata);
            /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
            /*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" */
            data++;
            length--;
        }
    }

/* USER CODE BEGIN (12) */
/* USER CODE END */
}
```

uint32 sciIsRxReady(sciBASE_t *sci)

* Rx 버퍼 가득 참 플래그가 설정되었는지 확인하고 0을 반환합니다. 그렇지 않으면 플래그가 설정되지 않습니다. 그렇지 않으면 Rx 플래그가 반환됩니다.

```
/* SourceId : SCI_SourceId_007 */
/* DesignId : SCI_DesignId_007 */
/* Requirements : HL_CONQ_SCI_SR11 */
/** @fn uint32 sciIsRxReady(sciBASE_t *sci)
*   @brief Check if Rx buffer full
*   @param[in] sci - sci module base address
*
*   @return The Rx ready flag
*
*   Checks to see if the Rx buffer full flag is set, returns
*   0 is flags not set otherwise will return the Rx flag itself.
*/
uint32 sciIsRxReady(sciBASE_t *sci)
{
/* USER CODE BEGIN (13) */
/* USER CODE END */

    return sci->FLR & (uint32)SCI_RX_INT;
}
```

uint32 sciIsIdleDetected(sciBASE_t *sci)

* SCI Idle 플래그가 설정되어 있는지 확인하고 0을 반환하면 플래그가 설정되지 않습니다. 그렇지 않으면 Ilde 플래그 자체가 반환됩니다.

```
/* SourceId : SCI_SourceId_008 */
/* DesignId : SCI_DesignId_008 */
/* Requirements : HL_CONQ_SCI_SR12 */
/** @fn uint32 sciIsIdleDetected(sciBASE_t *sci)
*   @brief Check if Idle Period is Detected
*   @param[in] sci - sci module base address
*
*   @return The Idle flag
*
*   Checks to see if the SCI Idle flag is set, returns 0 is flags
*   not set otherwise will return the Ilde flag itself.
*/
uint32 sciIsIdleDetected(sciBASE_t *sci)
{
/* USER CODE BEGIN (14) */
/* USER CODE END */

    return sci->FLR & (uint32)SCI_IDLE;
}
```

uint32 sciRxError(sciBASE_t *sci)

* Rx 프레임, 오버런 및 패리티 오류 플래그를 반환하고 반환하기 전에 오류 플래그를 지웁니다.

```
/* SourceId : SCI_SourceId_009 */
/* DesignId : SCI_DesignId_009 */
/* Requirements : HL_CONQ_SCI_SR13 */
/** @fn uint32 sciRxError(sciBASE_t *sci)
*   @brief Return Rx Error flags
*   @param[in] sci - sci module base address
*
*   @return The Rx error flags
*
*   Returns the Rx framing, overrun and parity errors flags,
*   also clears the error flags before returning.
*/
uint32 sciRxError(sciBASE_t *sci)
{
    uint32 status = (sci->FLR & ((uint32)SCI_FE_INT | (uint32)SCI_OE_INT |(uint32)SCI_PE_INT));

/* USER CODE BEGIN (15) */
/* USER CODE END */

    sci->FLR = ((uint32)SCI_FE_INT | (uint32)SCI_OE_INT | (uint32)SCI_PE_INT);
    return status;
}
```

```
uint32 sciReceiveByte(sciBASE_t *sci)
```

* 폴링 모드에서 단일 바이트를받습니다. 수신 버퍼에 바이트가 없으면 루틴은 수신 될 때까지 대기합니다. 대기를 피하기 위해 sciIsRxReady를 사용하여 버퍼가 꽉 찼는 지 확인하십시오.

```
/* SourceId : SCI_SourceId_010 */
/* DesignId : SCI_DesignId_010 */
/* Requirements : HL_CONQ_SCI_SR14 */
/** @fn uint32 sciReceiveByte(sciBASE_t *sci)
*   @brief Receive Byte
*   @param[in] sci - sci module base address
*
*   @return Received byte
*
*    Receives a single byte in polling mode.  If there is
*    not a byte in the receive buffer the routine will wait
*    until one is received.   Use sciIsRxReady to check to
*    see if the buffer is full to avoid waiting.
*/
uint32 sciReceiveByte(sciBASE_t *sci)
{
/* USER CODE BEGIN (16) */
/* USER CODE END */

    /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Potentially infinite loop found - Hardware Status check for execution sequence" */
    while ((sci->FLR & (uint32)SCI_RX_INT) == 0U)
    {
    } /* Wait */

    return (sci->RD & (uint32)0x000000FFU);
}
```

```
void sciReceive(sciBASE_t *sci, uint32 length, uint8 * data)
```

* 'length'블록의 블록을 수신하여 'data'가 가리키는 데이터 버퍼에 넣습니다. 인터럽트가 활성화 된 경우 인터럽트 모드를 사용하여 데이터를 수신합니다. 그렇지 않으면 폴링 모드가 사용됩니다. 인터럽트 모드에서는 수신이 설정되고 루틴이 즉시 반환됩니다. sciReceive는 sciNotification 콜백이 호출 될 때까지 전송이 완료 될 때까지 다시 호출되어서는 안됩니다. 폴링 모드에서는 전송이 완료 될 때까지 sciReceive가 반환되지 않습니다.

```
/* SourceId : SCI_SourceId_011 */
/* DesignId : SCI_DesignId_011 */
/* Requirements : HL_CONQ_SCI_SR15 */
/** @fn void sciReceive(sciBASE_t *sci, uint32 length, uint8 * data)
*   @brief Receive Data
*   @param[in] sci    - sci module base address
*   @param[in] length - number of data words to transfer
*   @param[in] data   - pointer to data buffer to receive data
*
*   Receive a block of 'length' bytes long and place it into the
*   data buffer pointed to by 'data'.  If interrupts have been
*   enabled the data is received using interrupt mode, otherwise
*   polling mode is used.  In interrupt mode receive is setup and
*   the routine returns immediately, sciReceive must not be called
*   again until the transfer is complete, when the sciNotification
*   callback will be called.  In polling mode, sciReceive will not
*   return  until the transfer is complete.
*/
void sciReceive(sciBASE_t *sci, uint32 length, uint8 * data)
{
/* USER CODE BEGIN (17) */
/* USER CODE END */

    if ((sci->SETINT & (uint32)SCI_RX_INT) == (uint32)SCI_RX_INT)
    {
        /* we are in interrupt mode */
        uint32 index = (sci  == sciREG1) ? 0U :
                       ((sci == sciREG2) ? 1U :
                       ((sci == sciREG3) ? 2U : 3U));

        /* clear error flags */
        sci->FLR = ((uint32) SCI_FE_INT | (uint32) SCI_OE_INT | (uint32) SCI_PE_INT);

        g_sciTransfer_t[index].rx_length = length;
        /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
        g_sciTransfer_t[index].rx_data   = data;
    }
    else
    {
        while (length > 0U)
        {
            /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Potentially infinite loop found - Hardware Status check for execution sequence" */
            while ((sci->FLR & (uint32)SCI_RX_INT) == 0U)
            {
            } /* Wait */
            /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
            *data = (uint8)(sci->RD & 0x000000FFU);
            /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters are only allowed in this driver" */
            /*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" */
            data++;
            length--;
        }
    }
/* USER CODE BEGIN (18) */
/* USER CODE END */
}
```

```
void sciEnableLoopback(sciBASE_t *sci, loopBackType_t Loopbacktype)
```

* 이 기능은 자체 테스트를 위해 루프백 모드를 활성화합니다.

```
/* USER CODE BEGIN (18) */
/* USER CODE END */
}

/* SourceId : SCI_SourceId_012 */
/* DesignId : SCI_DesignId_014 */
/* Requirements : HL_CONQ_SCI_SR18 */
/** @fn void sciEnableLoopback(sciBASE_t *sci, loopBackType_t Loopbacktype)
*   @brief Enable Loopback mode for self test
*   @param[in] sci        - sci module base address
*   @param[in] Loopbacktype  - Digital or Analog
*
*   This function enables the Loopback mode for self test.
*/
void sciEnableLoopback(sciBASE_t *sci, loopBackType_t Loopbacktype)
{
/* USER CODE BEGIN (19) */
/* USER CODE END */

    /* Clear Loopback incase enabled already */
    sci->IODFTCTRL = 0U;

    /* Enable Loopback either in Analog or Digital Mode */
    sci->IODFTCTRL = (uint32)0x00000A00U
                   | (uint32)((uint32)Loopbacktype << 1U);

/* USER CODE BEGIN (20) */
/* USER CODE END */
}
```

```
void sciDisableLoopback(sciBASE_t *sci)
```

* 이 기능은 루프백 모드를 비활성화합니다.

```
/* SourceId : SCI_SourceId_013 */
/* DesignId : SCI_DesignId_015 */
/* Requirements : HL_CONQ_SCI_SR19 */
/** @fn void sciDisableLoopback(sciBASE_t *sci)
*   @brief Enable Loopback mode for self test
*   @param[in] sci        - sci module base address
*
*   This function disable the Loopback mode.
*/
void sciDisableLoopback(sciBASE_t *sci)
{
/* USER CODE BEGIN (21) */
/* USER CODE END */

    /* Disable Loopback Mode */
    sci->IODFTCTRL = 0x00000500U;

/* USER CODE BEGIN (22) */
/* USER CODE END */
}
```

```
void sciEnableNotification(sciBASE_t *sci, uint32 flags)
```

* Enable interrupts 를 하여 sci에서 인터럽트가 가능하게 해줍니다.

@param[in] flags - Interrupts to be enabled, can be ored value of:

- SCI_FE_INT - framing error,
- SCI_OE_INT - overrun error,
- SCI_PE_INT - parity error,
- SCI_RX_INT - receive buffer ready,
- SCI_TX_INT - transmit buffer ready,
- SCI_WAKE_INT - wakeup,
- SCI_BREAK_INT - break detect

```
/* SourceId : SCI_SourceId_014 */
/* DesignId : SCI_DesignId_012 */
/* Requirements : HL_CONQ_SCI_SR16 */
/** @fn sciEnableNotification(sciBASE_t *sci, uint32 flags)
*   @brief Enable interrupts
*   @param[in] sci   - sci module base address
*   @param[in] flags - Interrupts to be enabled, can be ored value of:
*                      SCI_FE_INT    - framing error,
*                      SCI_OE_INT    - overrun error,
*                      SCI_PE_INT    - parity error,
*                      SCI_RX_INT    - receive buffer ready,
*                      SCI_TX_INT    - transmit buffer ready,
*                      SCI_WAKE_INT  - wakeup,
*                      SCI_BREAK_INT - break detect
*/
void sciEnableNotification(sciBASE_t *sci, uint32 flags)
{
    uint32 index = (sci  == sciREG1) ? 0U :
                   ((sci == sciREG2) ? 1U :
                   ((sci == sciREG3) ? 2U : 3U));

/* USER CODE BEGIN (23) */
/* USER CODE END */

    g_sciTransfer_t[index].mode |= (flags & (uint32)SCI_TX_INT);
    sci->SETINT                 = (flags & (uint32)(~(uint32)(SCI_TX_INT)));

/* USER CODE BEGIN (24) */
/* USER CODE END */
}
```

```
void sciDisableNotification(sciBASE_t *sci, uint32 flags)
```

* Disable interrupts 인터럽트 사용을 하지 못하게 한다.

@param[in] flags - Interrupts to be disabled, can be ored value of:

SCI_FE_INT - framing error,

SCI_OE_INT - overrun error,

SCI_PE_INT - parity error,

SCI_RX_INT - receive buffer ready,

SCI_TX_INT - transmit buffer ready,

SCI_WAKE_INT - wakeup,

SCI_BREAK_INT - break detect

```
/* SourceId : SCI_SourceId_015 */
/* DesignId : SCI_DesignId_013 */
/* Requirements : HL_CONQ_SCI_SR17 */
/** @fn sciDisableNotification(sciBASE_t *sci, uint32 flags)
*   @brief Disable interrupts
*   @param[in] sci   - sci module base address
*   @param[in] flags - Interrupts to be disabled, can be ored value of:
*                      SCI_FE_INT    - framing error,
*                      SCI_OE_INT    - overrun error,
*                      SCI_PE_INT    - parity error,
*                      SCI_RX_INT    - receive buffer ready,
*                      SCI_TX_INT    - transmit buffer ready,
*                      SCI_WAKE_INT  - wakeup,
*                      SCI_BREAK_INT - break detect
*/
void sciDisableNotification(sciBASE_t *sci, uint32 flags)
{
    uint32 index = (sci  == sciREG1) ? 0U :
                   ((sci == sciREG2) ? 1U :
                   ((sci == sciREG3) ? 2U : 3U));

/* USER CODE BEGIN (25) */
/* USER CODE END */

    g_sciTransfer_t[index].mode &= (uint32)(~(flags & (uint32)SCI_TX_INT));
    sci->CLEARINT               = (flags & (uint32)(~(uint32)(SCI_TX_INT)));

/* USER CODE BEGIN (26) */
/* USER CODE END */
}
```

```
void sciEnterResetState(sciBASE_t *sci)
```

* 재설정 상태로 들어감 (SCI는 리셋 상태에서만 구성해야합니다.)

```
/* SourceId : SCI_SourceId_016 */
/* DesignId : SCI_DesignId_001 */
/* Requirements :   */
/** @fn sciEnterResetState(sciBASE_t *sci)
*   @brief Enter reset state
*   @param[in] sci   - sci module base address
*   @note The SCI should only be configured while in reset state
*/
void sciEnterResetState(sciBASE_t *sci)
{
    sci->GCR1 &= 0xFFFFFF7FU;
}
```

```
void sciExitResetState(sciBASE_t *sci)
```

* 재설정 상태에서 빠져 나옴 (SCI는 리셋 상태에서만 구성해야합니다.)

```
/* SourceId : SCI_SourceId_017 */
/* DesignId : SCI_DesignId_001 */
/* Requirements :   */
/** @fn scixitResetState(sciBASE_t *sci)
*   @brief Exit reset state
*   @param[in] sci   - sci module base address
*   @note The SCI should only be configured while in reset state
*/
void sciExitResetState(sciBASE_t *sci)
{
    sci->GCR1 |= 0x00000080U;
}
```

```
void sci1GetConfigValue(sci_config_reg_t *config_reg, config_value_type_t type)
```

* 이 함수는 구성 레지스터의 초기 값 또는 현재 값 ('type'매개 변수에 따라 다름)을 config_reg가 가리키는 구조체에 복사합니다

```
/* SourceId : SCI_SourceId_018 */
/* DesignId : SCI_DesignId_016 */
/* Requirements : HL_CONQ_SCI_SR23 */
/** @fn void sci1GetConfigValue(sci_config_reg_t *config_reg, config_value_type_t type)
*   @brief Get the initial or current values of the SCI1 configuration registers
*
*   @param[in] *config_reg: pointer to the struct to which the initial or current
*                           value of the configuration registers need to be stored
*   @param[in] type:    whether initial or current value of the configuration registers need to be stored
*                       - InitialValue: initial value of the configuration registers will be stored
*                                       in the struct pointed by config_reg
*                       - CurrentValue: initial value of the configuration registers will be stored
*                                       in the struct pointed by config_reg
*
*   This function will copy the initial or current value (depending on the parameter 'type')
*   of the configuration registers to the struct pointed by config_reg
*
*/

void sci1GetConfigValue(sci_config_reg_t *config_reg, config_value_type_t type)
{
    if (type == InitialValue)
    {
        config_reg->CONFIG_GCR0      = SCI1_GCR0_CONFIGVALUE;
        config_reg->CONFIG_GCR1      = SCI1_GCR1_CONFIGVALUE;
        config_reg->CONFIG_SETINT    = SCI1_SETINT_CONFIGVALUE;
        config_reg->CONFIG_SETINTLVL = SCI1_SETINTLVL_CONFIGVALUE;
        config_reg->CONFIG_FORMAT    = SCI1_FORMAT_CONFIGVALUE;
        config_reg->CONFIG_BRS       = SCI1_BRS_CONFIGVALUE;
        config_reg->CONFIG_PIO0      = SCI1_PIO0_CONFIGVALUE;
        config_reg->CONFIG_PIO1      = SCI1_PIO1_CONFIGVALUE;
        config_reg->CONFIG_PIO6      = SCI1_PIO6_CONFIGVALUE;
        config_reg->CONFIG_PIO7      = SCI1_PIO7_CONFIGVALUE;
        config_reg->CONFIG_PIO8      = SCI1_PIO8_CONFIGVALUE;
    }
    else
    {
    /*SAFETYMCUSW 134 S MR:12.2 <APPROVED> "Register read back support" */
        config_reg->CONFIG_GCR0      = sciREG1->GCR0;
        config_reg->CONFIG_GCR1      = sciREG1->GCR1;
        config_reg->CONFIG_SETINT    = sciREG1->SETINT;
        config_reg->CONFIG_SETINTLVL = sciREG1->SETINTLVL;
        config_reg->CONFIG_FORMAT    = sciREG1->FORMAT;
        config_reg->CONFIG_BRS       = sciREG1->BRS;
        config_reg->CONFIG_PIO0      = sciREG1->PIO0;
        config_reg->CONFIG_PIO1      = sciREG1->PIO1;
        config_reg->CONFIG_PIO6      = sciREG1->PIO6;
        config_reg->CONFIG_PIO7        = sciREG1->PIO7;
        config_reg->CONFIG_PIO8      = sciREG1->PIO8;
    }
}
```