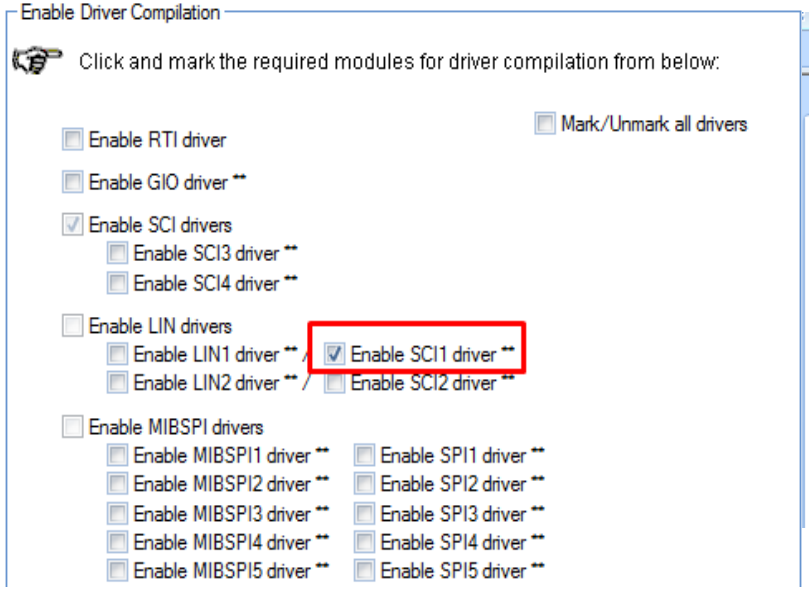
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

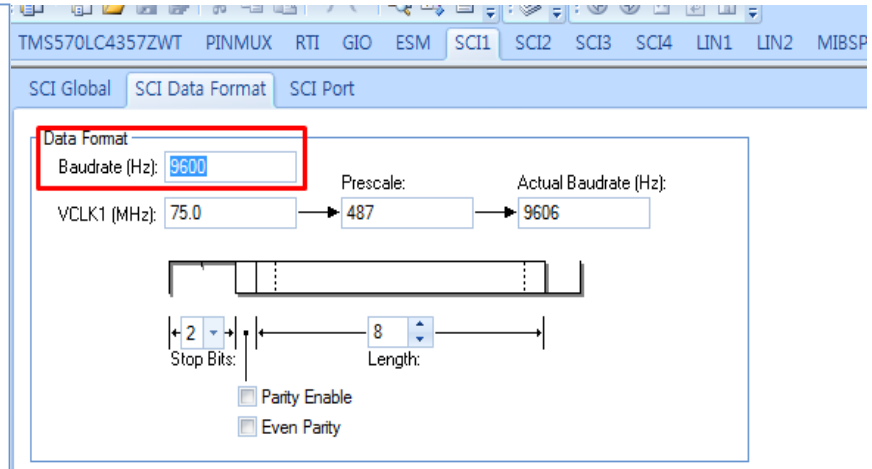
목차

4. Cortex-R5F Hercules Safety MCU - (SCI 설정)

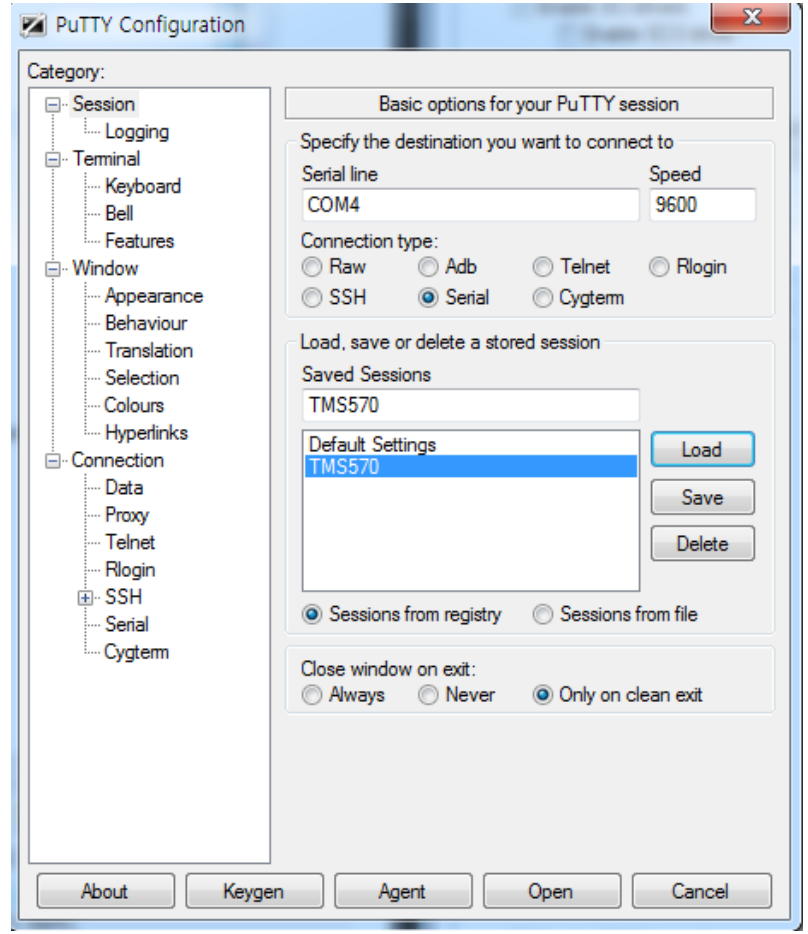
<1>



<2>



<3>



4. Cortex-R5F Hercules Safety MCU - (sci 예제 코드)

```c
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_sci.h"

#define TSIZE1    6
uint8 TEXT1[TSIZE1] = {'H', 'E', 'L', 'L', 'O', ' '};

#define TSIZE2    12
uint8 TEXT2[TSIZE2] = {'T', 'I', ' ', 'H', 'E', 'R', 'C',
                       'U', 'L', 'L', 'E', 'S'};

#define TSIZE3    12
uint8 TEXT3[TSIZE3] = {'S', 'A', 'F', 'E', 'T', 'Y', ' ',
                       'R', 'C', 'U', '\n', '\r'};

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 length);

void wait(uint32 time);
#define UART sciREG1
int main(void)
{

    sciInit();

    for(;;){
        sciDisplayText(UART, &TEXT1[0], TSIZE1);
        sciDisplayText(UART, &TEXT2[0], TSIZE2);
        sciDisplayText(UART, &TEXT3[0], TSIZE3);
        wait(20000);
    }
    return 0;
}

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 length){
    while(length--){
        while( (UART->FLR & 0x4) == 4);

        sciSendByte(UART, *text++);
    }
}

void wait(uint32 time){
    int i;
    for(i=0; i<time; i++);
}
```

```c
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_etpwm.h"
#include "HL_sci.h"

#include <string.h>
#include <stdio.h>

#define UART        sciREG1

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len);
void pwmSet(void);
void wait(uint32 delay);

uint32 rx_data = 0;
uint32 tmp = 0;
uint32 value = 0;

#define IDX    6
uint32 duty_arr[IDX] = {1000, 1200, 1400, 1600, 1800, 2000};

 void pwmSet(void)
 {
    value = duty_arr[rx_data];
    etpwmSetCmpA(etpwmREG1, value);
    wait(10000);
 }

 void wait(uint32 delay)
 {
    int i;

    for(i = 0; i < delay; i++)
        ;
 }

 void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len)
 {
    while(len--)
    {
        while((UART->FLR & 0x4) == 4)
            ;
        sciSendByte(UART, *text++);
    }
 }
```

```c
int main(void)
{
    char txt_buf[256] = {0};
    unsigned int buf_len;

    sciInit();

    sprintf(txt_buf, "SCI Configuration Success!!₩n₩r");
    buf_len = strlen(txt_buf);
    sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
    wait(1000000);

    etpwmInit();

    sprintf(txt_buf, "ETPWM Configuration Success!!₩n₩r");
    buf_len = strlen(txt_buf);
    sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);

    etpwmStartTBCLK();
    wait(1000000);

    sprintf(txt_buf, "Please Press Key(0 ~ 5)!!₩n₩r");
    buf_len = strlen(txt_buf);
    sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
    wait(1000000);

    for(;;)
    {
        tmp = sciReceiveByte(UART);
        rx_data = tmp - 48;

        sprintf(txt_buf, "rx = %d₩n₩r₩0", rx_data);
        buf_len = strlen(txt_buf);
        sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);

        pwmSet();

        sprintf(txt_buf, "PWM Duty = %d₩n₩r₩0", value);
        buf_len = strlen(txt_buf);
        sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
    }

    return 0;
}
```

4. Cortex-R5F Hercules Safety MCU - (sciInit() 분석 1)

```
/** - bring SCI1 out of reset */
sciREG1->GCR0 = 0U;
sciREG1->GCR0 = 1U;
```


30.7.1 SCI Global Control Register 0 (SCIGCR0)

The SCIGCR0 register defines the module reset. Figure 30-8 and Table 30-4 illustrate this register.

Figure 30-8. SCI Global Control Register 0 (SCIGCR0) [offset = 00]

31	16
Reserved	
R-0	

15	1	0
Reserved		RESET
R-0		R/WP-0

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

Table 30-4. SCI Global Control Register 0 (SCIGCR0) Fied Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reads return 0. Writes have no effect.
0	RESET		This bit resets the SCI module.
		0	SCI module is in reset.
		1	SCI module is out of reset.
			Note: Read/Write in privileged mode only.

	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~0
GCR0 :	0000	0000	0000	0000	0000	0000	0000	0001

4. Cortex-R5F Hercules Safety MCU – (sciInit() 분석 2)

```
/** - Disable all interrupts */
sciREG1->CLEARINT    = 0xFFFFFFFFU;
sciREG1->CLEARINTLVL = 0xFFFFFFFFU;
```

CLEARINT – 모든 인터럽트 disabled

CLEARINTLVL – 해당 인터럽트 line을 각각의 interrupt level로 mapping

30.7.4 SCI Clear Interrupt Register (SCICLEARINT)

Figure 30-11 and Table 30-7 illustrate this register. SCICLEARINT register is used to clear the selected enabled interrupts with out accessing SCISETINT register.

Figure 30-11. SCI Clear Interrupt Register (SCICLEARINT) [offset = 10h]

31 … 27	26	25	24
Reserved	CLR FE INT	CLR OE INT	CLR PE INT
R-0	R/W-0	R/W-0	R/W-0

23 … 19	18	17	16
Reserved	CLR RX DMA ALL	CLR RX DMA	CLR TX DMA
R-0	R/W-0	R/W-0	R/W-0

15 … 10	9	8
Reserved	CLR RX INT	CLR TX INT
R-0	R/W-0	R/W-0

7 … 2	1	0
Reserved	CLR WAKEUP INT	CLR BRKDT INT
R-0	R/W-0	R/W-0

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

Table 30-7. SCI Clear Interrupt Register (SCICLEARINT) Field Descriptions

Bit	Field	Value	Description
31-27	Reserved	0	Reads return 0. Writes have no effect.
26	CLR FE INT	0	Clear framing-error interrupt. This bit disables the framing-error interrupt when set. *Read:* The interrupt is disabled. *Write:* No effect.
		1	*Read:* The interrupt is enabled. *Write:* The interrupt is disabled.
25	CLR CE INT	0	Clear overrun-error interrupt. This bit disables the SCI overrun error interrupt when set. *Read:* The interrupt is disabled. *Write:* No effect.
		1	*Read:* The interrupt is enabled. *Write:* The interrupt is disabled.
24	CLR PE INT	0	Clear parity interrupt. This bit disables the parity error interrupt when set. *Read:* The interrupt is disabled. *Write:* No effect.
		1	*Read:* The interrupt is enabled. *Write:* The interrupt is disabled.
23-19	Reserved	0	Reads return 0. Writes have no effect.
18	CLR RX DMA ALL	0	Clear receive DMA all. This bit clears the receive DMA request for address frames when set. Only receive data frames generate a DMA request. *Read:* Receive DMA request for address frames is disabled; Instead, RX interrupt requests are enabled for address frames. Receive DMA requests are still enabled for data frames. *Write:* No effect.
		1	*Read:* The receive DMA request for address and data frames is enabled. *Write:* The receive DMA request for address and data frames is disabled.

30.7.6 SCI Clear Interrupt Level Register (SCICLEARINTLVL)

Figure 30-13 and Table 30-9 illustrate this register.

Figure 30-13. SCI Clear Interrupt Level Register (SCICLEARINTLVL) [offset = 18h]

31 … 27	26	25	24
Reserved	CLR FE INT LVL	CLR OE INT LVL	CLR PE INT LVL
R-0	R/W-0	R/W-0	R/W-0

23 … 19	18	17 … 16
Reserved	CLR RX DMA ALL INT LVL	Reserved
R-0	R/W-0	R-0

15 … 10	9	8
Reserved	CLR RX INT LVL	CLR TX INT LVL
R-0	R/W-0	R/W-0

7 … 2	1	0
Reserved	CLR WAKEUP INT LVL	CLR BRKDT INT LVL
R-0	R/W-0	R/W-0

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

Table 30-9. SCI Clear Interrupt Level Register (SCICLEARINTLVL) Field Descriptions

Bit	Field	Value	Description
31-27	Reserved	0	Reads return 0. Writes have no effect.
26	CLR FE INT LVL	0	Clear framing-error interrupt. *Read:* The interrupt level is mapped to the INT0 line. *Write:* No effect.
		1	*Read:* The interrupt level is mapped to the INT1 line. *Write:* The interrupt level is mapped to the INT0 line.

	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~0
CLEARTINT :	1111	1111	1111	1111	1111	1111	1111	1111

	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~0
CLEARINTLVL :	1111	1111	1111	1111	1111	1111	1111	1111

4. Cortex-R5F Hercules Safety MCU - (sciInit() 분석 3)

```
/** - global control 1 */
sciREG1->GCR1 =  (uint32)((uint32)1U << 25U)  /* enable transmit */
               | (uint32)((uint32)1U << 24U)  /* enable receive */
               | (uint32)((uint32)1U << 5U)   /* internal clock (device has no clock pin) */
               | (uint32)((uint32)(2U-1U) << 4U)  /* number of stop bits */
               | (uint32)((uint32)0U << 3U)  /* even parity, otherwise odd */
               | (uint32)((uint32)0U << 2U)  /* enable parity */
               | (uint32)((uint32)1U << 1U);  /* asynchronous timing mode */
```

29.7.2 SCI Global Control Register 1 (SCIGCR1)

The SCIGCR1 register defines the frame format, protocol, and communication mode used by the SCI. Figure 29-29 and Table 29-12 illustrate this register.

Figure 29-29. SCI Global Control Register 1 (SCIGCR1) (offset = 04h)

31	...	26	25	24
Reserved			TXENA	RXENA
R-0			R/W-0	R/W-0

23	...	18	17	16
Reserved			CONT	LOOP BACK
R-0			R/W-0	R/W-0

15	14	13	12	11	10	9	8
Reserved		STOP EXT FRAME	HGEN CTRL	CTYPE	MBUF MODE	ADAPT	SLEEP
R-0		R/WL-0	R/WL-0	R/WL-0	R/W-0	R/WL-0	R/W-0

7	6	5	4	3	2	1	0
SWnRST	LIN MODE	CLOCK	STOP	PARITY	PARITY ENA	TIMING MODE	COMM MODE
R/W-0	R/W-0	R/W-0	R/WC-0	R/WC-0	R/W-0	R/WC-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; WL = Write in LIN mode only; WC = Write in SCI-compatible mode only; -*n* = value after reset

Table 29-12. SCI Global Control Register 1 (SCIGCR1) Field Descriptions

Bit	Field	Value	Description
31-26	Reserved	0	Reads return 0. Writes have no effect.
25	TXENA		Transmit enable. This bit is effective in LIN and SCI modes. Data is transferred from SCITD, or the TDy (with y=0, 1,...7) buffers in LIN mode to the SCITXSHF shift out register only when the TXENA bit is set.
		0	Transfers from SCITD or TDy to SCITXSHF are disabled.
		1	Transfers from SCITD or TDy to SCITXSHF are enabled.
			Note: Data written to SCITD or the transmit multi-buffer before TXENA is set is not transmitted. If TXENA is cleared while transmission is ongoing, the data previously written to SCITD is sent (including the checksum byte in LIN mode).
24	RXENA		Receive enable. This bit is effective in LIN and SCI modes. RXENA allows or prevents the transfer of data from SCIRXSHF to SCIRD or the receive multi-buffers.
		0	The receiver will not transfer data from the shift buffer to the receive buffer or multi-buffers.
		1	The receiver will transfer data from the shift buffer to the receive buffer or multi-buffers.
			Note: Clearing RXENA stops received characters from being transferred into the receive buffer or multi-buffers, prevents the RX status flags from being updated by receive data, and inhibits both receive and error interrupts. However, the shift register continues to assemble data regardless of the state of RXENA.
			Note: If RXENA is cleared before a frame is completely received, the data from the frame is not transferred into the receive buffer.
			Note: If RXENA is set before a frame is completely received, the data from the frame is transferred into the receive buffer. If RXENA is set while SCIRXSHF is in the process of assembling a frame, the status flags are not guaranteed to be accurate for that frame. To ensure that the status flags correctly reflect what was detected on the bus during a particular frame, RXENA should be set before the detection of that frame.
23-18	Reserved	0	Reads return 0. Writes have no effect.
17	CONT		Continue on suspend. This bit is effective in LIN and SCI modes. This bit has an effect only when a program is being debugged with an emulator, and it determines how the SCI/LIN operates when the program is suspended. The SCI/LIN counters are affected by this bit: when the bit is set the counters are not stopped, when the bit is cleared the counters are stopped during debug mode.
		0	When debug mode is entered, the SCI/LIN state machine is frozen. Transmissions and LIN counters are halted and resume when debug mode is exited.
		1	When debug mode is entered, the SCI/LIN continues to operate until the current transmit and receive functions are complete.
16	LOOP BACK		Loopback bit. This bit is effective in LIN and SCI modes. The self-checking option for the SCI/LIN can be selected with this bit. If the LINITX and LINRX pins are configured with SCI/LIN functionality, then the LINTX pin is internally connected to the LINRX pin. Externally, during loop back operation, the LINTX pin outputs a high value and the LINRX pin is in a high-impedance state. If this bit value is changed while the SCI/LIN is transmitting or receiving data, errors may result.
		0	Loop back mode is disabled.
		1	Loop back mode is enabled.
15-14	Reserved	0	Reads return 0. Writes have no effect.
13	STOP EXT FRAME		Stop extended frame communication. This bit is effective in LIN mode only. This bit can be written only during extended frame communication. When the extended frame communication is stopped, this bit is cleared automatically.
		0	This bit has no effect.
		1	Extended frame communication will be stopped when current frame transmission/reception is completed.
12	HGEN CTRL		HGEN control. This bit is effective in LIN mode only. This bit controls the type of mask filtering comparison.
		0	ID filtering using the ID-BYTE field in LIN Identification Register (LINID) occurs. Mask of FFh in LIN Mask Register (LINMASK) register will result in no match.
		1	ID filtering uses ID-SlaveTask BYTE (recommended). Mask of FFh in LIN Mask Register (LINMASK) register will result in ALWAYS match.
			Note: For software compatibility with future LIN modules the HGEN CTRL bit must be set to 1, the RX ID MASK must be set to FFh and the TX ID MASK must be set to FFh.
11	CTYPE		Checksum type. This bit is effective in LIN mode only. This bit controls the type of checksum to be used: classic or enhanced.
		0	Classic checksum is used.
		1	Enhanced checksum is used.
10	MBUF MODE		Multi-buffer mode. This bit is effective in LIN and SCI modes. This bit controls receive/transmit buffer usage, that is, whether the RX/TX multi-buffers are used or a single register, RD0/TD0, is used.
		0	The multi-buffer mode is disabled.
		1	The multi-buffer mode is enabled.
9	ADAPT		Adapt. This mode is effective in LIN mode only. This bit has an effect during the detection of the synch field. Two LIN protocol bit rate modes could be enabled with this bit according to the node capability file definition: automatic or select. The software and network configuration will decide which of these two modes are enabled. When this bit is cleared, the LIN 2.0 protocol fixed bit rate should be used. If the ADAPT bit is set, a SCI/LIN slave node detecting the baud rate will compare it to the prescalers in BRS register and update it if they are different. The BRS register will be updated with the new value. If this bit is not set there will be no adjustment to the BRS register.
		0	Automatic baud rate adjustment is disabled.
		1	Automatic baud rate adjustment is enabled.
8	SLEEP		SCI sleep. This bit is effective in SCI mode only. In a multiprocessor configuration, this bit controls the receive sleep function. Clearing this bit brings the SCI/LIN out of sleep mode.
		0	Sleep mode is disabled.
		1	Sleep mode is enabled.
			Note: The receiver still operates when the SLEEP bit is set; however, RXRDY is updated and SCIRD is loaded with new data only when an address frame is detected. The remaining receiver status flags are updated and an error interrupt is requested if the corresponding interrupt enable bit is set, regardless of the value of the SLEEP bit. In this way, if an error is detected on the receive data line while the SCI is asleep, software can promptly deal with the error condition.
			Note: The SLEEP bit is *not* automatically cleared when an address byte is detected.
			See Section 29.2.5.1 for more information on using the SLEEP bit for multiprocessor communication.

	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~0
GCR1 :	0000	0011	0000	0000	0000	0000	0001	0010

4. Cortex-R5F Hercules Safety MCU - (sciInit() 분석 4)

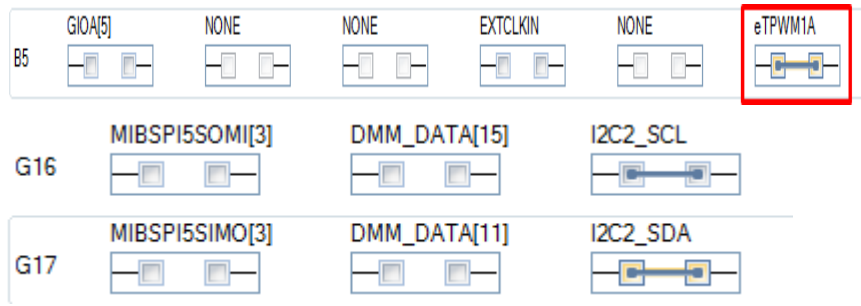
Bit	Field	Value	Description
7	SWnRST		Software reset (active low). This bit is effective in LIN and SCI modes.
		0	The SCI/LIN is in its reset state; no data will be transmitted or received. Writing a 0 to this bit initializes the SCI/LIN state machines and operating flags as defined in Table 29-13 and Table 29-14. All affected logic is held in the reset state until a 1 is written to this bit.
		1	The SCI/LIN is in its ready state; transmission and reception can be done. After this bit is set to 1, the configuration of the module should not change.
			Note: The SCI/LIN should only be configured while SWnRST = 0.
6	LIN MODE		LIN mode. This bit is effective in LIN and SCI mode. This bit controls the module mode of operation.
		0	LIN mode is disabled; SCI mode is enabled.
		1	LIN mode is enabled; SCI mode is disabled.
5	CLOCK		SCI internal clock enable. The CLOCK bit determines the source of the module clock on the SCICLK pin. It also determines whether a LIN node is a slave or master.
			SCI mode:
		0	The external SCICLK is the clock source.
			Note: If an external clock is selected, then the internal baud rate generator and baud rate registers are bypassed. The maximum frequency allowed for an externally sourced SCI clock is VCLK/16.
		1	The internal SCICLK is the clock source.
			LIN mode:
		0	The node is in slave mode.
		1	The node is in master mode.
4	STOP		SCI number of stop bits per frame. This bit is effective in SCI mode only.
		0	One stop bit is used.
		1	Two stop bits are used.
			Note: The receiver checks for only one stop bit. However in idle-line mode, the receiver waits until the end of the second stop bit (if STOP = 1) to begin checking for an idle period.
3	PARITY		SCI parity odd/even selection. This bit is effective in SCI mode only. If the PARITY ENA bit is set, PARITY designates odd or even parity.
		0	Odd parity is used.
		1	Even parity is used.
			The parity bit is calculated based on the data bits in each frame and the address bit (in address-bit mode). The start and stop fields in the frame are not included in the parity calculation.
			For odd parity, the SCI transmits and expects to receive a value in the parity bit that makes odd the total number of bits in the frame with the value of 1.
			For even parity, the SCI transmits and expects to receive a value in the parity bit that makes even the total number of bits in the frame with the value of 1.
2	PARITY ENA		Parity enable. This bit enables or disables the parity function.
			SCI or buffered SCI mode:
		0	Parity is disabled. No parity bit is generated during transmission or is expected during reception.
		1	Parity is enabled. A parity bit is generated during transmission and is expected during reception.
			LIN mode:
		0	ID field parity verification is disabled.
		1	ID field parity verification is enabled.
1	TIMING MODE		SCI timing mode bit. This bit is effective in SCI mode only. it selects the SCI timing mode.
		0	Synchronous timing is used.
		1	Asynchronous timing is used.

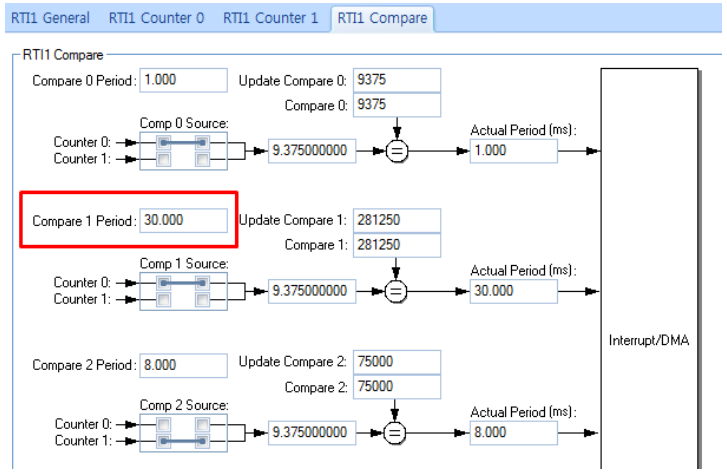
Bit	Field	Value	Description
0	COMM MODE		SCI/LIN communication mode bit. In compatibility mode it selects the SCI communication mode. In LIN mode it selects length control option for ID-field bits ID4 and ID5.
			SCI mode:
		0	Idle-line mode is used.
		1	Address-bit mode is used.
			LIN mode:
		0	ID4 and ID5 are not used for length control.
		1	ID4 and ID5 are used for length control.

4. Cortex-R5F Hercules Safety MCU - (I2C 설정)

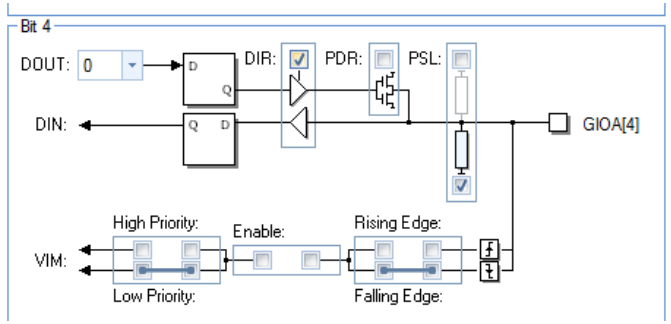
<1>

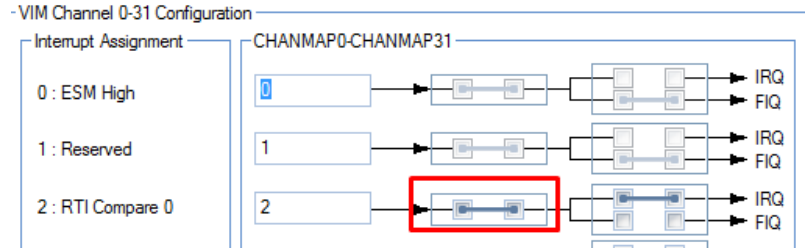
<2>

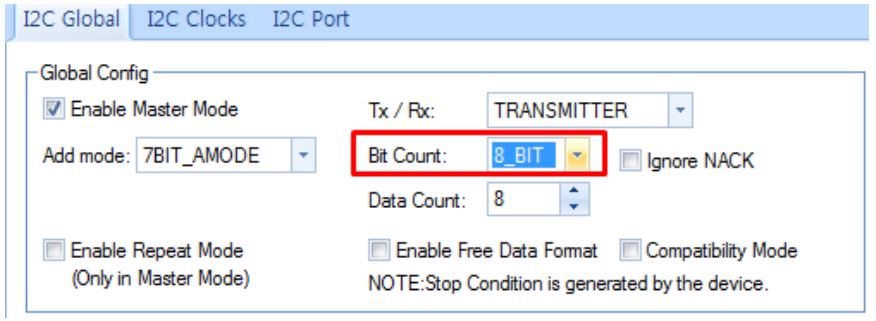




<3>







4. Cortex-R5F Hercules Safety MCU - (i2cInit() 코드 1)

```
#include <string.h>
#include <stdio.h>

#include "HL_sys_common.h"
#include "HL_sys_core.h"
#include "HL_sci.h"
#include "HL_gio.h"
#include "HL_i2c.h"
#include "HL_rti.h"

#define UART            sciREG1
#define MPU6050_ADDR    0x68

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len);
void mpu6050_enable(void);
void mpu6050_acc_config(void);

volatile char g_acc_xyz[6];
volatile int g_acc_flag;

void wait(uint32 t)
{
   t--;
}
```

```
int main(void)
{
   volatile int i;
   char txt_buf[256];
   unsigned int buf_len;
   signed short acc_x, acc_y, acc_z;
   double real_acc_x, real_acc_y, real_acc_z;

   gioInit();
   sciInit();

   wait(10000000);

   i2cInit();

   wait(10000000);

   mpu6050_enable();
   sprintf(txt_buf, "MPU6050 Enabled₩n₩r₩0");
   buf_len = strlen(txt_buf);
   sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
   wait(200);

   mpu6050_acc_config();
   sprintf(txt_buf, "MPU6050 Accelerometer Configured₩n₩r₩0");
   buf_len = strlen(txt_buf);
   sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
   wait(200);

   rtiInit();
   rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE2);
   _enable_IRQ_interrupt_();
   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK1);

   sprintf(txt_buf, "RTI Enabled₩n₩r₩0");
   buf_len = strlen(txt_buf);
   sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
```

```
   for(;;)
   {
      if(g_acc_flag)
      {
         acc_x = acc_y = acc_z = 0;
         real_acc_x = real_acc_y = real_acc_z = 0.0;

         acc_x = g_acc_xyz[0];
         acc_x = acc_x << 8;
         acc_x |= g_acc_xyz[1];
         real_acc_x = ((double)acc_x) / 2048.0;

         acc_y = g_acc_xyz[2];
         acc_y = acc_y << 8;
         acc_y |= g_acc_xyz[3];
         real_acc_y = ((double)acc_y) / 2048.0;

         acc_z = g_acc_xyz[4];
         acc_z = acc_z << 8;
         acc_z |= g_acc_xyz[5];
         real_acc_z = ((double)acc_z) / 2048.0;

         sprintf(txt_buf, "acc_x = %2.5lf₩tacc_y=%2.5lf₩tacc_z=%2.5lf₩n₩r₩0", real_acc_x,
real_acc_y, real_acc_z);

         buf_len = strlen(txt_buf);
         sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);

         g_acc_flag = 0;
      }
   }

   return 0;
}
```

4. Cortex-R5F Hercules Safety MCU - (i2cInit() 코드 2)

```
void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len)
{
    while(len--)
    {
        while((UART->FLR & 0x4) == 4)
            ;
        sciSendByte(UART, *text++);
    }
}

void mpu6050_enable(void)
{
    volatile unsigned int cnt = 2;
    unsigned char data[2] = {0x00U, 0x00U};
    unsigned char slave_word_address = 0x6bU;

    i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR);
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER);
    i2cSetStop(i2cREG2);
    i2cSetStart(i2cREG2);
    i2cSendByte(i2cREG2, slave_word_address);
    i2cSend(i2cREG2, cnt, data);

    while(i2cIsBusBusy(i2cREG2) == true)
        ;

    while(i2cIsStopDetected(i2cREG2) == 0)
        ;

    i2cClearSCD(i2cREG2);

    wait(1000000);
}
```

```
void mpu6050_acc_config(void)
{
    volatile unsigned int cnt = 1;
    unsigned char data[1] = {0x18U};
    unsigned char slave_word_address = 0x1cU;

    i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR);
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
    i2cSetCount(i2cREG2, cnt + 1);
    i2cSetMode(i2cREG2, I2C_MASTER);
    i2cSetStop(i2cREG2);
    i2cSetStart(i2cREG2);
    i2cSendByte(i2cREG2, slave_word_address);
    i2cSend(i2cREG2, cnt, data);

    while(i2cIsBusBusy(i2cREG2) == true)
        ;

    while(i2cIsStopDetected(i2cREG2) == 0)
        ;

    i2cClearSCD(i2cREG2);

    wait(1000000);
}
```

```
void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
    unsigned char slave_word_address = 0x3B;

    i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR);
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
    i2cSetCount(i2cREG2, 1);
    i2cSetMode(i2cREG2, I2C_MASTER);
    i2cSetStop(i2cREG2);
    i2cSetStart(i2cREG2);
    i2cSendByte(i2cREG2, slave_word_address);

    while(i2cIsBusBusy(i2cREG2) == true)
        ;

    while(i2cIsStopDetected(i2cREG2) == 0)
        ;

    i2cClearSCD(i2cREG2);

    i2cSetDirection(i2cREG2, I2C_RECEIVER);
    i2cSetCount(i2cREG2, 6);
    i2cSetMode(i2cREG2, I2C_MASTER);
    i2cSetStart(i2cREG2);

    i2cReceive(i2cREG2, 6, (unsigned char *)g_acc_xyz);
    i2cSetStop(i2cREG2);

    while(i2cIsBusBusy(i2cREG2) == true)
        ;

    while(i2cIsStopDetected(i2cREG2) == 0)
        ;

    i2cClearSCD(i2cREG2);

    g_acc_flag = 1;
}
```