TOSHIBA

TMP89FS60FG Sample Program

Rev 1.0
Dec. 2007

This is sample software to help customers understand Toshiba microcontrollers and learn how to create programs when developing new products.

You can download this document and the sample program from the following Web.
https://toshiba.semicon-storage.com/ap-en/design-support/document/application-note.html
Or,
http://toshiba.semicon-storage.com/ap-en/top.html/ --> Microcomputer --> Application Notes

- The information contained herein is subject to change without notice.
- TOSHIBA is continually working to improve the quality and reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to comply with the standards of safety in making a safe design for the entire system, and to avoid situations in which a malfunction or failure of such TOSHIBA products could cause loss of human life, bodily injury or damage to property. In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent TOSHIBA products specifications. Also, please keep in mind the precautions and conditions set forth in the “Handling Guide for Semiconductor Devices,” or “TOSHIBA Semiconductor Reliability Handbook” etc.
- The TOSHIBA products listed in this document are intended for usage in general electronics applications (computer, personal equipment, office equipment, measuring equipment, industrial robotics, domestic appliances, etc.). These TOSHIBA products are neither intended nor warranted for usage in equipment that requires extraordinarily high quality and/or reliability or a malfunction or failure of which may cause loss of human life or bodily injury ("Unintended Usage"). Unintended Usage include atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, medical instruments, all types of safety devices, etc.. Unintended Usage of TOSHIBA products listed in this document shall be made at the customer’s own risk.
- The products described in this document shall not be used or embedded to any downstream products of which manufacture, use and/or sale are prohibited under any applicable laws and regulations.
- TOSHIBA does not take any responsibility for incidental damage (including loss of business profit, business interruption, loss of business information, and other pecuniary damage) arising out of the use or disability to use the product.
- The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patents or other rights of TOSHIBA or the third parties.
- The products described in this document are subject to foreign exchange and foreign trade control laws.

Index

1. General description 3
2. Use Function 3
3. Terminal Use 4
4. Function 6
4-1. Operation mode selection 6
4-2. Watch dog timer 6
4-3. Voltage Detection Circuit 6
4-4. 8 bit PWM output 7
4-5. 8 bit PPG output 7
4-6. 16 bit PPG output 8
4-7. UART output 8
4-8. Clock synchronous SIO output 8
4-9. I2C output 8
4-10. 10 bit A/D input 9
4-11. Timer for clock 9
4-12. Function sequence 9
5. Software 10
5-1. main program : [main.c] 10
5-2. 8 bit PWM output : [timer8_pwm.c] 12
5-3. 8 bit PPG output : [timer8_ppg.c] 15
5-4. 16 bit PPG output : [timer16_ppg.c] 19
5-5. UART output : [uart.c] 23
5-6. Clock synchronous SIO output : [sio.c] 27
5-7. I2C output : [sbi.c] 30
5-8. 10 bit A/D input : [adc.c] 35
5-9. Timer for clock : [rtc.c] 38

1. General description

This sample program is created targeting at the TMP89FS60FG.

It can execute main MCU-embedded functions one by one. You can get some parts out of the sample program; use it to call the function you would like to perform.

2. Use Function

Function	Channel	Use presence, use
SLOW/STOP		Use
Watch dog timer		Use
Voltage Detection Circuit		Use
Time Base Timer		Make internal timing
8 bit timer	TC00	8 bit PWM output
	TC01	8 bit PPG output
	TC02	Unused
	TC03	Unused
16 bit timer	TCA0	16 bit PPG output
	TCA1	Unused
Timer for clock		0.5s signal output
UART	UART0	Unused
	UART1	Unused
	UART2	UART data output
SIO	SIO0	Unused
	SIO1	SIO data output
Serial bus	SBI	I2C data output
10 bit A/D converter	AIN8	A/D data entry
	AIN0-AIN7	Unused
	AIN9-AIN15	Unused
External interrupt	INT0	Slow mode trigger
	INT5	Stop mode trigger
	INT1/2/3/4	Unused

3. Terminal Use

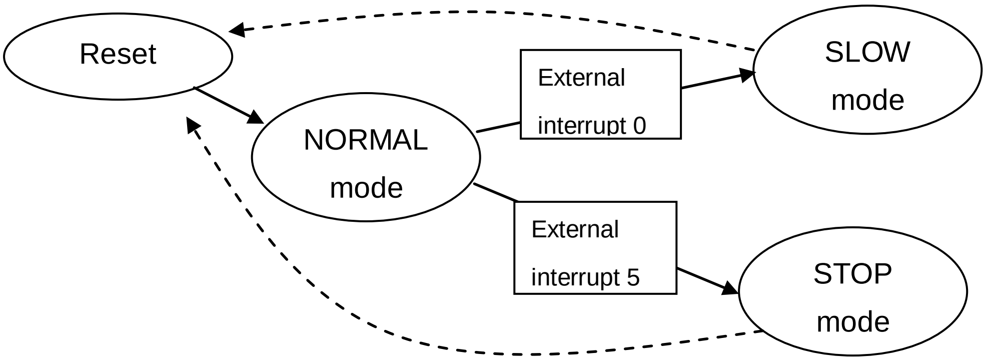
No	Name	Use
1	VSS	GND
2	P00(XIN)	High frequency oscillator connection
3	P01(XOUT)	High frequency oscillator connection
4	MODE	Test pin for out-going test(Fix to low level)
5	VDD	+5V
6	P02(XTIN)	Low frequency oscillator connection
7	P03(XTOUT)	Low frequency oscillator connection
8	P10(^RESET)	Reset signal input pin
9	P11(^INT5/^STOP)	**External interrupt 5**
10	P12(^INT0)	**External interrupt 0**
11	P13(INT1)	Unused
12	P20(TXD0/SO0/OCDCK)	Unused
13	P21(RXD0/SI0/OCKIO)	Unused
14	P22(SCLK0)	Unused
15	P23(SDA0/SO0)	**I2C bus data output**
16	P24(SCL0/SI0)	**I2C bus clock output**
17	P25(SCLK0)	Unused
18	P26	Unused
19	P27	Unused
20	AVSS	GND
21	AVDD	+5V
22	VAREF	+5V
23	P40(AIN0/KWI0)	Unused
24	P41(AIN1/KWI1)	Unused
25	P42(AIN2/KWI2)	Unused
26	P43(AIN3/KWI3)	Unused
27	P44(AIN4/KWI4)	Unused
28	P45(AIN5/KWI5)	Unused
29	P46(AIN6/KWI6)	Unused
30	P47(AIN7/KWI7)	Unused
31	P50(AIN8)	**Analog input**
32	P51(AIN9)	Unused
33	P52(AIN10)	Unused
34	P53(AIN11)	Unused
35	P54(AIN12)	Unused
36	P55(AIN13)	Unused

No	Name	Use
37	P56(AIN14)	Unused
38	P57(AIN15)	Unused
39	P70(TC00/^PPG00/^PWM00)	**8 bit PWM output**
40	P71(TC01/^PPG01/^PWM01)	**8 bit PPG output**
41	P72(TCA0/^PPGA0)	**16bit PPG output**
42	P73(TCA1/^PPGA1)	Unused
43	P74(^DVO)	Unused
44	P75(INT2)	Unused
45	P76(INT3)	Unused
46	P77(INT4)	Unused
47	P80(TC02/^PPG02/^PWM02)	Unused
48	P81(TC03/^PPG03/^PWM03)	Unused
49	P82	Unused
50	P83	Unused
51	P84	**0.5s signal output**
52	P90(TXD1/SO1)	**SIO data output**
53	P91(RXD1/SI1)	Unused
54	P92(SCLK1)	**SIO clock output**
55	P93(TXD2)	**UART data output**
56	P94(RXD2)	Unused
57	PB0	**A/D conversion data output**
58	PB1	**A/D conversion data output**
59	PB2	**A/D conversion data output**
60	PB3	**A/D conversion data output**
61	PB4	**A/D conversion data output**
62	PB5	**A/D conversion data output**
63	PB6	**A/D conversion data output**
64	PB7	**A/D conversion data output**

4. Function

4-1. Operation mode selection

The high frequency clock and the low frequency clock are made the oscillation state by a NORMAL mode. The operation mode (NORMAL/SLOW/STOP) is changed by external interrupt input in the NORMAL mode. These interrupts are detected in the NORMAL mode. When external interrupt 0 input, it changes to the SLOW mode. When external interrupt 5 input, it changes to the STOP mode. Its state is maintained until it takes Reset for a shifted operation mode.



Each operation with these modes makes as follows.

Function	NORMAL mode	SLOW mode	STOP mode
Watch dog timer	**Operation**	**Operation**	Stop
8 bit PWM output	**Operation**	Stop	Stop
8 bit PPG output	**Operation**	Stop	Stop
16 bit PPG output	**Operation**	Stop	Stop
0.5s signal output	**Operation**	**Operation**	Stop
UART data output	**Operation**	Stop	Stop
SIO output	**Operation**	Stop	Stop
I2C data output	**Operation**	Stop	Stop
A/D data entry/output	**Operation**	Stop	Stop

4-2. Watch dog timer

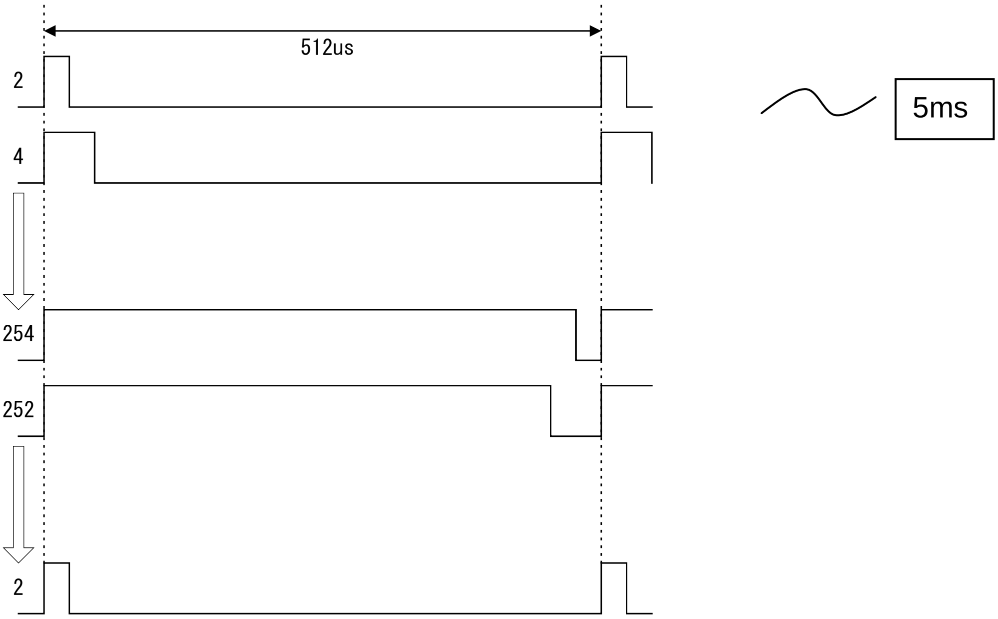
As for the watch dog timer way in NORMAL mode and in SLOW mode it operates (It's established so that a reset may occur by running out of control detection.). With NORMAL mode it to WDTT1/0=11, with SLOW mode sets to WDTT1/0=00, clears the timer within those detection times.

4-3. Voltage Detection Circuit

The voltage detection circuit detects any decrease in the supply voltage and generates voltage detection reset signals.

4-4. 8 bit PWM output

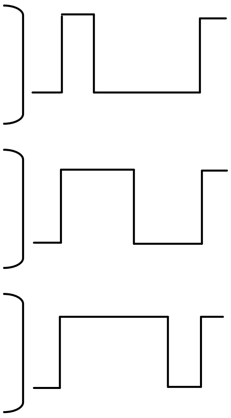
It's a signal of 508us (set value =0xFE) from "H" width 4us (set value =0x02) in cycle 512us, It changes it every 5ms and outputs from PWM00 terminal.



4-5. 8 bit PPG output

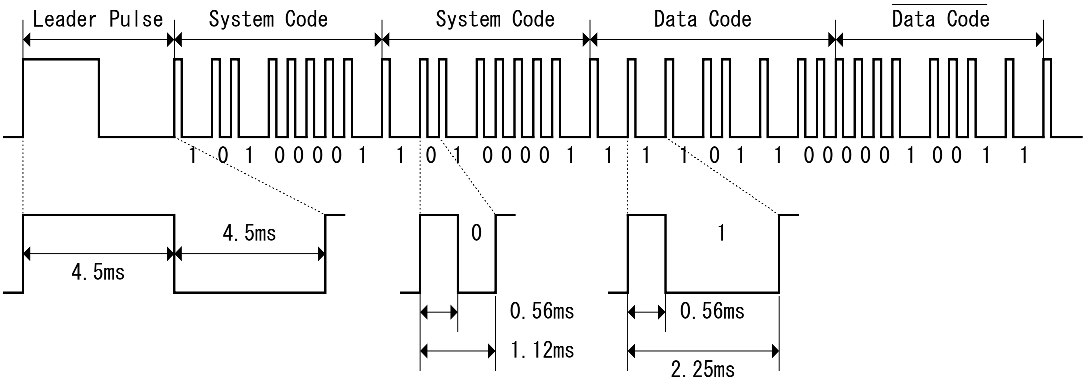
Changing the waveform which changes duty and frequency to every 1s concerning 1 waveform, it outputs from the PPG01 terminal. This is repeated. When the line commutation type buzzer is connected, it becomes buzzer sound.

Step	Frequency	Duty	Output Time
1	2441Hz	25%	1s
2	1221Hz	25%	1s
3	610Hz	25%	1s
4	2441Hz	50%	1s
5	1221Hz	50%	1s
6	610Hz	50%	1s
7	2441Hz	75%	1s
8	1221Hz	75%	1s
9	610Hz	75%	1s



4-6. 16 bit PPG output

Remote control signal (for infrared ray remote control transmission IC: TC9243AFG) the waveform is output. The identical waveform is output every 200ms. The output data is 0x85, 0x85, 0x37, 0xC8, (it outputs from the lower bit).



4-7. UART output

From UART2 the optional character string (the ASCII cord/code) it outputs with 8 bit UART mode.

Item	Value
Baud rate	9600bps
Parity	It is not
Stop bit	1 bits
Output character string (A)	“TOSHIBA.”
Character string interval	When (A) 1 times is output, opening the interval of 100ms, it repeats 10 times.

4-8. Clock synchronous SIO output

The clock synchronous SIO signal is output from SIO1. It outputs 2 bytes by the 20ms cycle. The data makes the value which the increment is 1 at a time is done from 0x0000 to 0xFFFF.

4-9. I2C output

It's the 10ms cycle and outputs 1 byte by I2C. The clock frequency is set n=5. An acknowledging clock doesn't output because ACK can't be received. The output data is 0xA3.

4-10. 10 bit A/D input

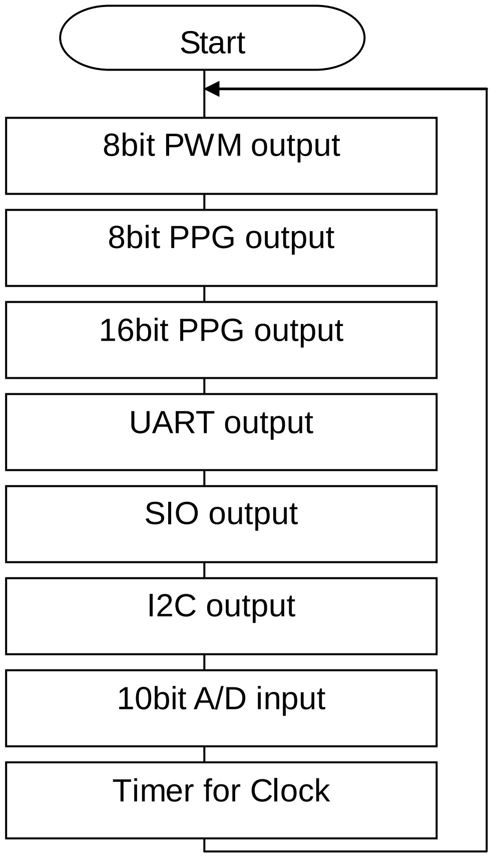
From AIN8 the data is taken in every 2ms 10 times, the upper 8 bits are output to the PB port.

4-11. Timer for clock

The signal reversed by the 0.5s cycle is output 10 times from P84 terminal. It operates in NORMAL mode and SLOW mode.

4-12. Function sequence

The function mentioned above is carried out sequentially, and it is repeated.



5. Software

5-1. main program : [main.c]

```
void main(void)
{
    WDCTR = 0x27;                               /* Start WDT */
    gMode = 0;
    test = 0x00;
    fSlowNormalChangeCheck = FALSE;             /* The Flag for Slow and Normal Mode change.
                                                   FALSE —> TRUE represent Normal changes to Slow
                                                   TURE --> FALSE represent Slow changes to Normal */

    fStopModeCheck = FALSE;                     /* The Flag for Normal/Stop Mode change */

    fSlowModeCheck = FALSE;                     /* The Flag for current Mode,
                                                   FALSE —> Normal; TRUE --> Slow */

    for (;;)
    {

        WDCDR = 0x4E;                           /* Clear WDT counter */
        if ((fSlowNormalChangeCheck == TRUE) && (fSlowModeCheck == FALSE))
        {
            fSlowModeCheck = TRUE ;
            gMode = 7;
            _asm(" CLR (_WDCTR).5");            /* Stop WDT counter */
            _asm(" SET (_SYSCR2).4 ");
            _asm(" NOP ");
            _asm(" NOP ");
            _asm(" NOP ");
            _asm(" CLR (_SYSCR2).6 ");          /* Change to Slow Mode */
            WDCTR = 0x01;                       /* Select 2^11/fs,reset output */
            WDCDR = 0xB1;                       /* CLear the 8-bit up counter */
            _asm(" SET (_WDCTR).5");            /* Start WDT counter */
        }
        if (fStopModeCheck == TRUE)
        {
            _asm(" SET (_SYSCR1).7 ");          /* Change to Stop Mode */
        }

        switch (gMode)
        {
            case 0:
                sample_tc00_pwm();
                gMode++;
                break;

            case 1:
                sample_tc01_ppg();
                gMode++;
                break;

            case 2:
                sample_ta0_ppg();
                gMode++;
                break;

            case 3:
                sample_uart();
                gMode++;
                break;

            case 4:
                sample_sio();
                gMode++;
                break;
64
65              case 5:
66                  sample_sbi();
67                  gMode++;
68                  break;
69
70              case 6:
71                  sample_adc();
72                  gMode++;
73                  break;
74
75              case 7:
76                  sample_rtc();
77                  if (fSlowModeCheck == FALSE)
78                  {
79                      gMode ++;
80                  }
81                  break;
82
83              default:
84                  gMode = 0x00;
85                  break;
86          }
87      }
88  }
```

The main program consists of the two processes: the boot process of SLOW/STOP and sort process of the embedded functions using gMode. Each embedded function shifts to the next function by incrementing gMode.

If you would like to operate it as stand alone, edit the line 17 as follows to fix the value of gMode.

gmode=n; Set n to the value from one to seven.

5-2. 8 bit PWM output : [timer8_pwm.c]

5-2-1. Control process

```
1   void sample_tc00_pwm(void)
2   {
3           fTC00Check = FALSE;
4           fTC00Cycle = FALSE;
5           gTC00Cnt = 5;
6
7           TC00Init();
8           TC00Start();
9           while (fTC00Check == FALSE)
10          {
11                  WDCDR = 0x4E;       /* Clear WDT counter */
12          }
13          TC00Stop();
14  }
```

[line 7 : TC00 initialize] ➡ 5-2-2

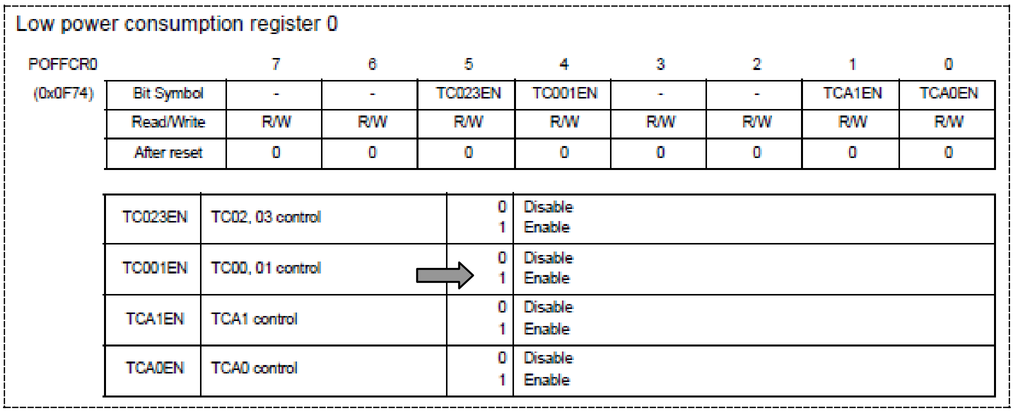
[line 8 : TC00 start] ➡ 5-2-3

[line 13 : TC00 stop] ➡ 5-2-4

5-2-2. Initialize process

```
1   void TC00Init(void)
2   {
3           _asm(" SET (_P7FC).0 ");
4           _asm(" SET (_P7CR).0 ");    /* Set P70 as the 8-bit PWM output port */
5           POFFCR0 = 0x10;              /* Set TC001EN = 1 */
6           _DI();
7           EIRH = EIRH | 0x10;         /* Enable INTTC00 */
8           _EI();
9           T00MOD = 0xA2;               /* Set 8-bit PWM mode and fcgck/2^4 */
10          T00PWM = 0x02;               /* 4us x 2/(2^4/fcgck) */
11  }
```

[line 5 : Enable the TC00 power]

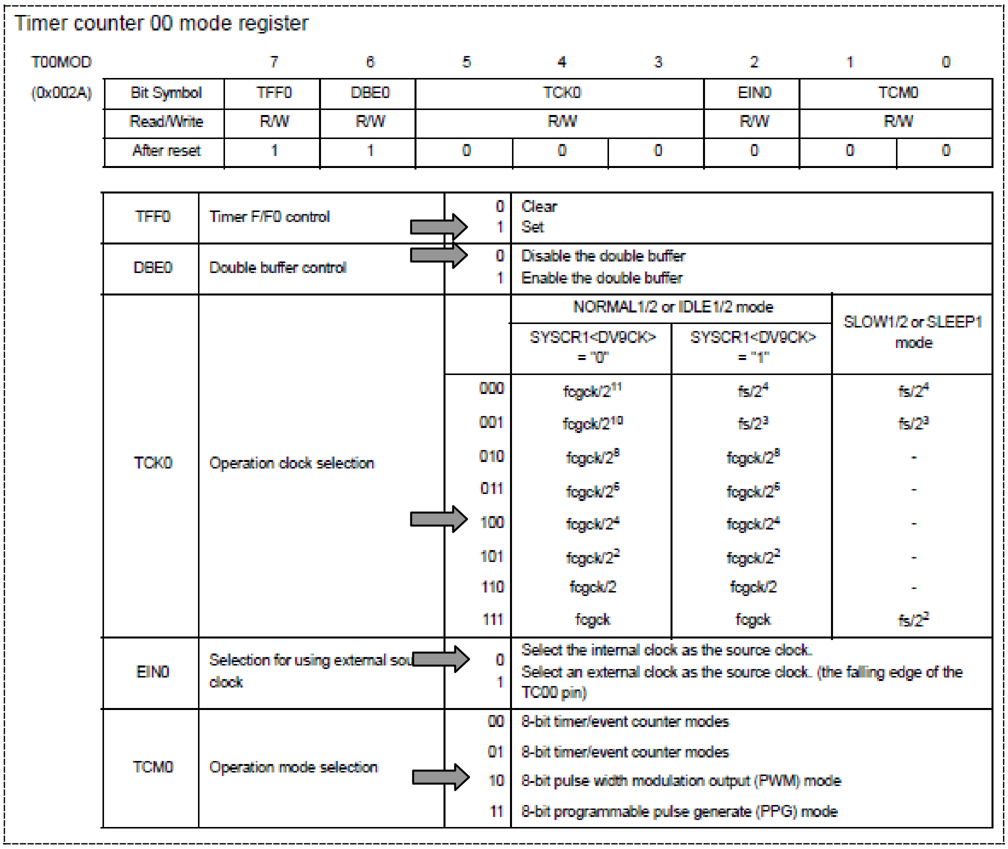


Low power consumption register 0

POFFCR0 (0x0F74)		7	6	5	4	3	2	1	0
	Bit Symbol	-	-	TC023EN	TC001EN	-	-	TCA1EN	TCA0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

TC023EN	TC02, 03 control	0 1	Disable Enable
TC001EN	TC00, 01 control	0 1	Disable Enable
TCA1EN	TCA1 control	0 1	Disable Enable
TCA0EN	TCA0 control	0 1	Disable Enable

[line 9 : Set 8bit PWM mode and clock rate]



Timer counter 00 mode register

T00MOD		7	6	5	4	3	2	1	0
(0x002A)	Bit Symbol	TFF0	DBE0	TCK0			EIN0	TCM0	
	Read/Write	R/W	R/W	R/W			R/W	R/W	
	After reset	1	1	0	0	0	0	0	0

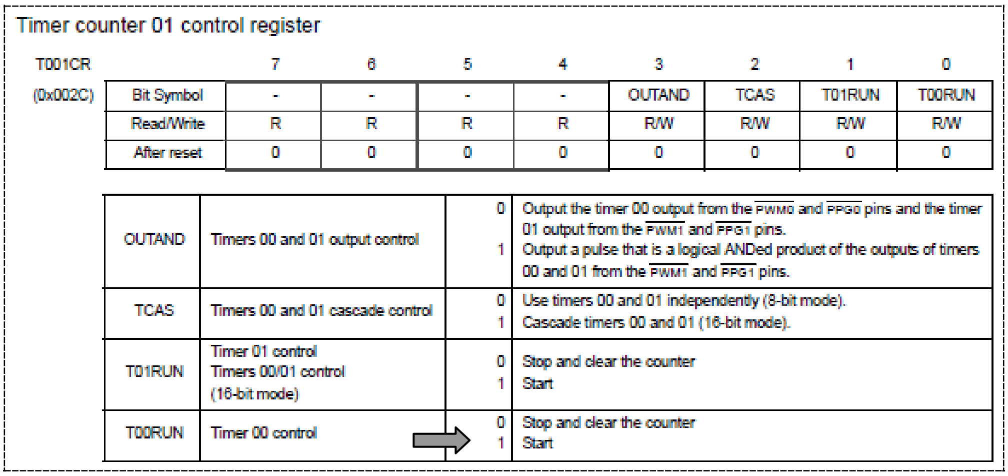
TFF0	Timer F/F0 control	0 1	Clear Set		
DBE0	Double buffer control	0 1	Disable the double buffer Enable the double buffer		
TCK0	Operation clock selection		NORMAL1/2 or IDLE1/2 mode		SLOW1/2 or SLEEP1 mode
			SYSCR1<DV9CK> = "0"	SYSCR1<DV9CK> = "1"	
		000	fcgck/2^{11}	fs/2^4	fs/2^4
		001	fcgck/2^{10}	fs/2^3	fs/2^3
		010	fcgck/2^8	fcgck/2^8	-
		011	fcgck/2^6	fcgck/2^6	-
		100	fcgck/2^4	fcgck/2^4	-
		101	fcgck/2^2	fcgck/2^2	-
		110	fcgck/2	fcgck/2	-
		111	fcgck	fcgck	fs/2^2
EIN0	Selection for using external sou clock	0 1	Select the internal clock as the source clock. Select an external clock as the source clock. (the falling edge of the TC00 pin)		
TCM0	Operation mode selection	00 01 10 11	8-bit timer/event counter modes 8-bit timer/event counter modes 8-bit pulse width modulation output (PWM) mode 8-bit programmable pulse generate (PPG) mode		

[line 10 : Set the timer value]

5-2-3. Timer start process

```
1   void TC00Start(void)
2   {
3           __asm(" SET (_T001CR).0 ");
4   }
```

[line 3 : Start the timer count]



Timer counter 01 control register

T001CR		7	6	5	4	3	2	1	0
(0x002C)	Bit Symbol	-	-	-	-	OUTAND	TCAS	T01RUN	T00RUN
	Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

OUTAND	Timers 00 and 01 output control	0 1	Output the timer 00 output from the $\overline{\text{PWM0}}$ and $\overline{\text{PPG0}}$ pins and the timer 01 output from the $\overline{\text{PWM1}}$ and $\overline{\text{PPG1}}$ pins. Output a pulse that is a logical ANDed product of the outputs of timers 00 and 01 from the $\overline{\text{PWM1}}$ and $\overline{\text{PPG1}}$ pins.
TCAS	Timers 00 and 01 cascade control	0 1	Use timers 00 and 01 independently (8-bit mode). Cascade timers 00 and 01 (16-bit mode).
T01RUN	Timer 01 control Timers 00/01 control (16-bit mode)	0 1	Stop and clear the counter Start
T00RUN	Timer 00 control	0 1	Stop and clear the counter Start

5-2-4. Timer stop process

```
 1  void TC00Stop(void)
 2  {
 3          T001CR = 0x00;
 4  }
```

[line 3 : Stop the timer]

5-2-5. Interrupt process

```
 1  void _interrupt IntTC00(void)
 2  {
 3          static UINT8_t i = 0;
 4
 5          i++;
 6          if (i >= gTC00Cnt)
 7          {
 8                  i = 0;
 9                  if (fTC00Cycle == FALSE)
10                  {
11                          T00PWM = T00PWM + 2;
12                          if (T00PWM >= 254)
13                          {
14                                  fTC00Cycle = ~fTC00Cycle;
15                          }
16                  }
17                  else
18                  {
19                          T00PWM = T00PWM - 2;
20                          if (T00PWM <= 0)
21                          {
22                                  fTC00Cycle = ~fTC00Cycle;
23                                  fTC00Check = TRUE;
24                          }
25                  }
26          }
27  }
```

[line 11,19 : Set the next timer value]

5-3. 8 bit PPG output : [timer8_ppg.c]

5-3-1. Control process

```
void sample_tc01_ppg(void)
{
    UINT8_t i = 0;

    fTC01Check = FALSE;
    fTC01Cycle = FALSE;
    sRatioMode = 0;
    sPPGCnt = 2441;

    TC01Init();
    TC01Start();
    while (fTC01Check == FALSE)
    {
         WDCDR = 0x4E;                              /* Clear WDT counter */
        if (fTC01Cycle == TRUE)
        {
            fTC01Cycle = FALSE;
            sRatioMode++;
            if (sRatioMode >= 9)
            {
                sRatioMode = 0;
                i++;
                if (i >= sRunTimeCnt)
                {
                    fTC01Check = TRUE;
                }
            }
            switch (sRatioMode)
            {
                case 0:
                    T01REG = 0x1A;
                    T01PWM = 0x06;
                    sPPGCnt = 2441;
                    break;                      /* 2441HZ,25% duty */

                case 1:
                    T01REG = 0x33;
                    T01PWM = 0x0D;
                    sPPGCnt = 1221;
                    break;                      /* 1221HZ,25% duty */

                case 2:
                    T01REG = 0x66;
                    T01PWM = 0x1A;
                    sPPGCnt = 610;              /* 610HZ,25% duty */
                    break;

                case 3:
                    T01REG = 0x1A;
                    T01PWM = 0x0C;
                    sPPGCnt = 2441;
                    break;                      /* 2441HZ,50% duty */

                case 4:
                    T01REG = 0x33;
                    T01PWM = 0x1A;
                    sPPGCnt = 1221;
                    break;                      /* 1221HZ,50% duty */

                case 5:
                    T01REG = 0x66;
                    T01PWM = 0x34;
                    sPPGCnt = 610;              /* 610HZ,50% duty */
64                      break;
65
66                  case 6:
67                      T01REG = 0x1A;
68                      T01PWM = 0x12;
69                      sPPGCnt = 2441;
70                      break;                  /* 2441HZ,75% duty */
71
72                  case 7:
73                      T01REG = 0x33;
74                      T01PWM = 0x27;
75                      sPPGCnt = 1221;
76                      break;                  /* 1221HZ,75% duty */
77
78                  case 8:
79                      T01REG = 0x66;
80                      T01PWM = 0x4E;
81                      sPPGCnt = 610;           /* 610HZ,75% duty */
82                      break;
83
84                  default:
85                      break;
86              }
87          }
88      }
89      TC01Stop();
90  }
```

[line 10 : Initialize the timer] ➡ 5-3-2

[line 11 : Start the timer] ➡ 5-3-3

[line 31,32 : Set the timer value (2441HZ,25% duty)]

[line 37,38 : Set the timer value (1221HZ,25% duty)]

[line 43,44 : Set the timer value (610HZ,25% duty)]

[line 49,50 : Set the timer value (2441HZ,50% duty)]

[line 55,56 : Set the timer value (1221HZ,50% duty)]

[line 61,62 : Set the timer value (610HZ,50% duty)]

[line 67,68 : Set the timer value (2441HZ,75% duty)]

[line 73,74 : Set the timer value (1221HZ,75% duty)]

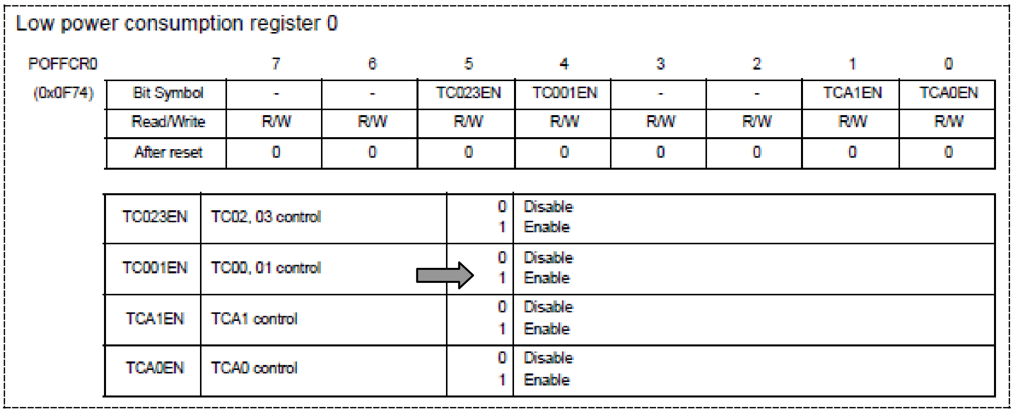
[line 79,80 : Set the timer value (610HZ,75% duty)]

[line 89 : Stop the timer] ➡ 5-3-4

5-3-2. Initialize process

```
 1  void TC01Init(void)
 2  {
 3      _asm(" SET (_P7FC).1 ");        /* Set P71 as the 8-bit PPG output port */
 4      _asm(" SET (_P7CR).1 ");
 5      POFFCR0 = 0x10;
 6      _DI();
 7      EIRH = EIRH | 0x20;
 8      _EI();
 9      T01MOD = 0x9B;                  /* Set 8-bit PPG mode and fcgck/2^6 */
10      T01REG = 0x1A;                   /* (1/2441)/(2^6/fcgck) */
11      T01PWM = 0x06;
12  }
```

[line 5 : Enable the TC01 power]

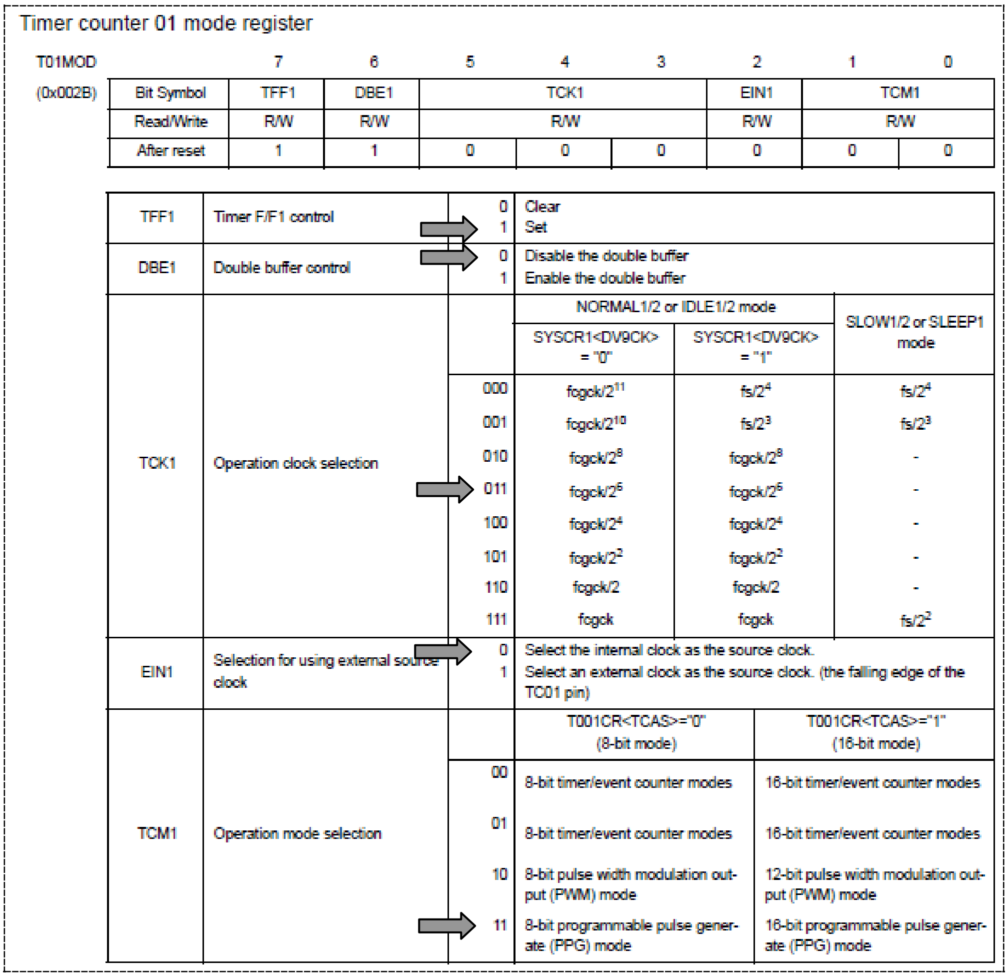


Low power consumption register 0

POFFCR0		7	6	5	4	3	2	1	0
(0x0F74)	Bit Symbol	-	-	TC023EN	TC001EN	-	-	TCA1EN	TCA0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

TC023EN	TC02, 03 control	0 1	Disable Enable
TC001EN	TC00, 01 control	0 1	Disable [✓ Enable]
TCA1EN	TCA1 control	0 1	Disable Enable
TCA0EN	TCA0 control	0 1	Disable Enable

[line 9 : Set 8bit PPG mode and clock rate]



Timer counter 01 mode register

T01MOD		7	6	5	4	3	2	1	0
(0x002B)	Bit Symbol	TFF1	DBE1	TCK1			EIN1	TCM1	
	Read/Write	R/W	R/W	R/W			R/W	R/W	
	After reset	1	1	0	0	0	0	0	0

TFF1	Timer F/F1 control	0 1	Clear [✓ Set]
DBE1	Double buffer control	0 1	[✓ Disable the double buffer] Enable the double buffer

			NORMAL1/2 or IDLE1/2 mode		SLOW1/2 or SLEEP1 mode
			SYSCR1<DV9CK> = "0"	SYSCR1<DV9CK> = "1"	
TCK1	Operation clock selection	000	$fcgck/2^{11}$	$fs/2^{4}$	$fs/2^{4}$
		001	$fcgck/2^{10}$	$fs/2^{3}$	$fs/2^{3}$
		010	$fcgck/2^{8}$	$fcgck/2^{8}$	-
		[✓ 011]	$fcgck/2^{6}$	$fcgck/2^{6}$	-
		100	$fcgck/2^{4}$	$fcgck/2^{4}$	-
		101	$fcgck/2^{2}$	$fcgck/2^{2}$	-
		110	fcgck/2	fcgck/2	-
		111	fcgck	fcgck	$fs/2^{2}$

EIN1	Selection for using external source clock	0 1	[✓ Select the internal clock as the source clock.] Select an external clock as the source clock. (the falling edge of the TC01 pin)

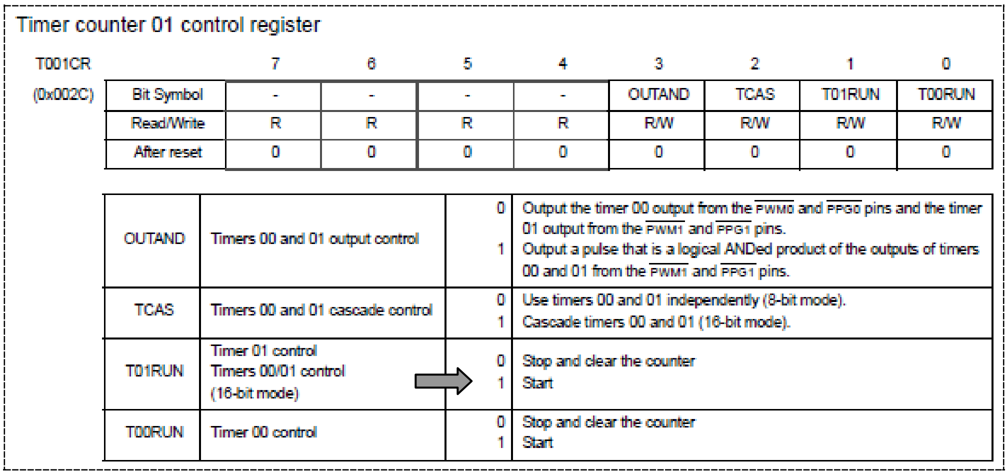
			T001CR<TCAS>="0" (8-bit mode)	T001CR<TCAS>="1" (16-bit mode)
TCM1	Operation mode selection	00	8-bit timer/event counter modes	16-bit timer/event counter modes
		01	8-bit timer/event counter modes	16-bit timer/event counter modes
		10	8-bit pulse width modulation output (PWM) mode	12-bit pulse width modulation output (PWM) mode
		[✓ 11]	8-bit programmable pulse generate (PPG) mode	16-bit programmable pulse generate (PPG) mode

[line 10,11 : Set the initial timer value]

5-3-3. Timer start process

```
1   void TC01Start(void)
2   {
3           __asm(" SET (_T001CR).1 ");
4   }
```

[line 3 : Start the timer count]



Timer counter 01 control register

T001CR		7	6	5	4	3	2	1	0
(0x002C)	Bit Symbol	-	-	-	-	OUTAND	TCAS	T01RUN	T00RUN
	Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

OUTAND	Timers 00 and 01 output control	0 1	Output the timer 00 output from the $\overline{PWM0}$ and $\overline{PPG0}$ pins and the timer 01 output from the $\overline{PWM1}$ and $\overline{PPG1}$ pins. Output a pulse that is a logical ANDed product of the outputs of timers 00 and 01 from the $\overline{PWM1}$ and $\overline{PPG1}$ pins.
TCAS	Timers 00 and 01 cascade control	0 1	Use timers 00 and 01 independently (8-bit mode). Cascade timers 00 and 01 (16-bit mode).
T01RUN	Timer 01 control Timers 00/01 control (16-bit mode)	0 1	Stop and clear the counter Start
T00RUN	Timer 00 control	0 1	Stop and clear the counter Start

5-3-4. Timer stop process

```
1  void TC01Stop(void)
2  {
3          T001CR = 0x00;
4  }
```

[line 3 : Stop the TC01]

5-3-5. Interrupt process

```
void _interrupt IntTC01(void)
{
        static UINT16_t i = 0;

        i++;
        if (i >= sPPGCnt)
        {
                i = 0;
                fTC01Cycle = TRUE;
        }
}
```

5-4. 16 bit PPG output : [timer16_ppg.c]

5-4-1. Control process

```
 1  void sample_ta0_ppg(void)
 2  {
 3      UINT8_t i = 0;
 4  
 5      f200msCheck = FALSE;
 6      fTA0Check = FALSE;
 7      gCheckCnt = 200;
 8  
 9      TCA0Init();
10      TCA0Start();
11      TBTInit();
12      TBTStart();
13      while (fTA0Check == FALSE)
14      {
15          WDCDR = 0x4E;                /* Clear WDT counter */
16          if (f200msCheck == TRUE)
17          {
18              f200msCheck = FALSE;
19              i++;
20              if (i < sRunTimeCnt)
21              {
22                  TCA0Start();
23              }
24              else
25              {
26                  fTA0Check = TRUE;
27              }
28          }
29      }
30      TCA0Stop();
31      TBTStop();
32  }
```

[line 9 : Initialize the TCA0] ➡ 5-4-2

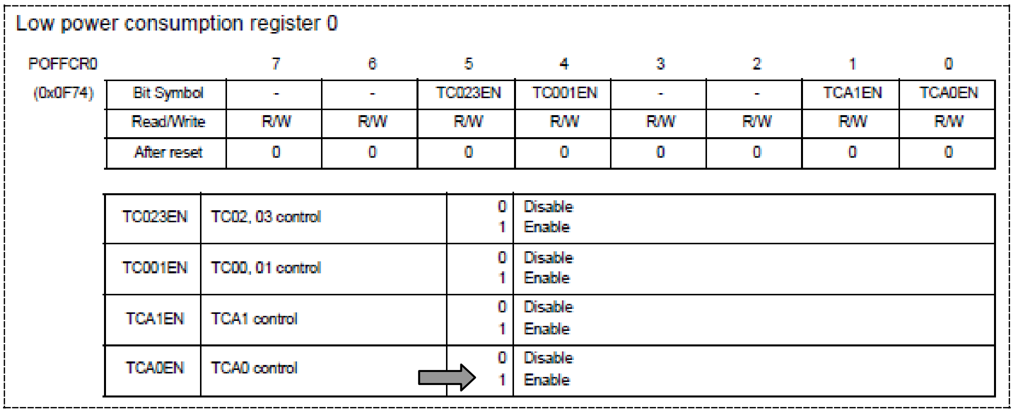
[line 10 : Start the TCA0] ➡ 5-4-3

[line 30 : Stop the TCA0] ➡ 5-4-4

5-4-2. Initialize process

```
 1  void TCA0Init(void)
 2  {
 3      _asm(" SET (_P7CR).2 ");        /* P72 as the 16-bit PPG output port */
 4      _asm(" SET (_P7FC).2 ");
 5      POFFCR0 = 0x01;                 /* Set TCA0EN = 1 */
 6      _DI();
 7      EIRH = EIRH | 0x40;
 8      _EI();
 9      TA0MOD = 0x13;                  /* fcgck/2^2 */
10      TA0CR = 0xC0;
11  }
```

[line 5 : Enable the TCA0 power]

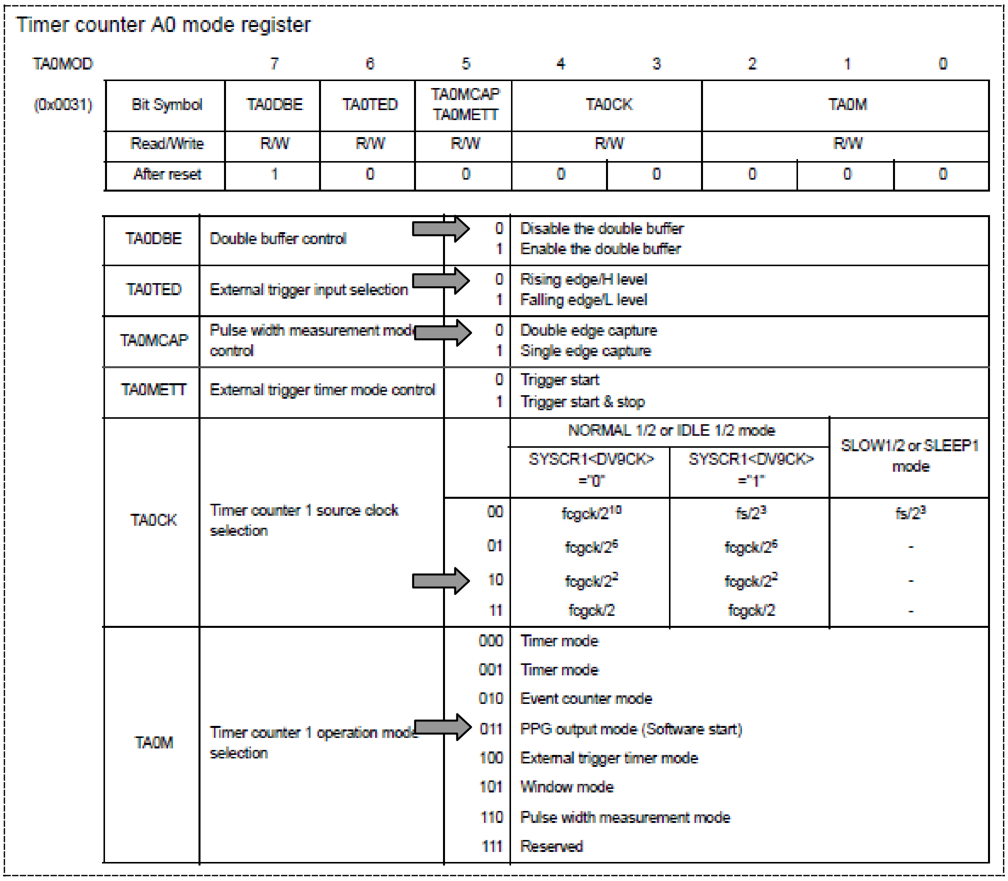


Low power consumption register 0

POFFCR0		7	6	5	4	3	2	1	0
(0x0F74)	Bit Symbol	-	-	TC023EN	TC001EN	-	-	TCA1EN	TCA0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

TC023EN	TC02, 03 control	0 1	Disable Enable
TC001EN	TC00, 01 control	0 1	Disable Enable
TCA1EN	TCA1 control	0 1	Disable Enable
TCA0EN	TCA0 control	0 ⇨ 1	Disable Enable

[line 9 : Set 16bit PPG mode and clock rate]

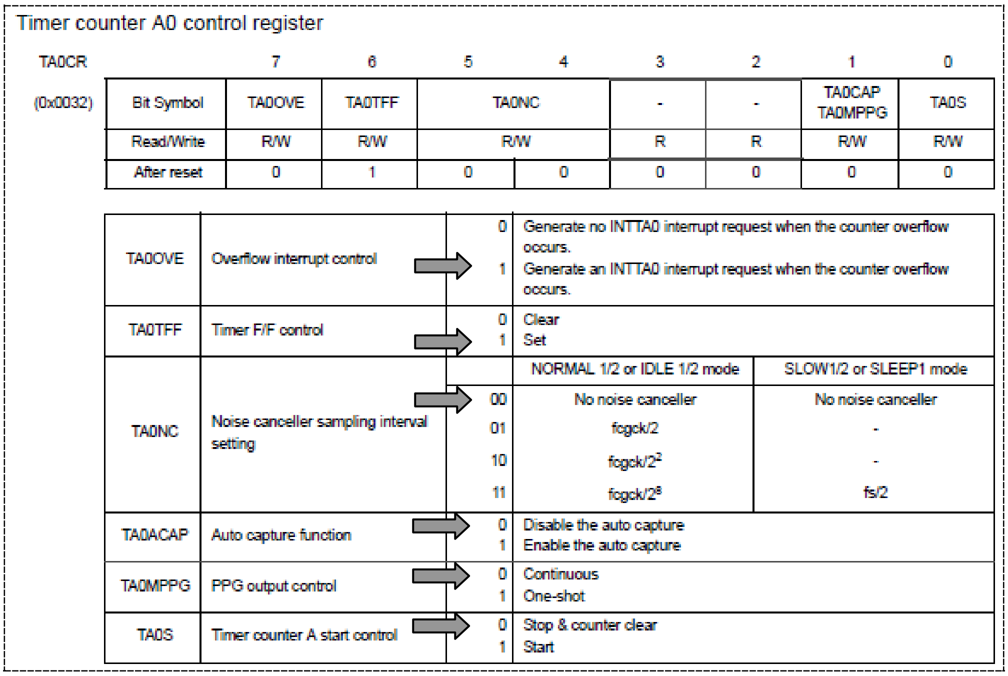


Timer counter A0 mode register

TA0MOD		7	6	5	4	3	2	1	0
(0x0031)	Bit Symbol	TA0DBE	TA0TED	TA0MCAP TA0METT	TA0CK		TA0M		
	Read/Write	R/W	R/W	R/W	R/W		R/W		
	After reset	1	0	0	0	0	0	0	0

TA0DBE	Double buffer control	⇨ 0 1	Disable the double buffer Enable the double buffer		
TA0TED	External trigger input selection	⇨ 0 1	Rising edge/H level Falling edge/L level		
TA0MCAP	Pulse width measurement mode control	⇨ 0 1	Double edge capture Single edge capture		
TA0METT	External trigger timer mode control	0 1	Trigger start Trigger start & stop		
TA0CK	Timer counter 1 source clock selection		NORMAL 1/2 or IDLE 1/2 mode		SLOW1/2 or SLEEP1 mode
			SYSCR1<DV9CK> ="0"	SYSCR1<DV9CK> ="1"	
		00	fcgck/2^{10}	fs/2^3	fs/2^3
		01	fcgck/2^6	fcgck/2^6	-
		⇨ 10	fcgck/2^2	fcgck/2^2	-
		11	fcgck/2	fcgck/2	-
TA0M	Timer counter 1 operation mode selection	000	Timer mode		
		001	Timer mode		
		010	Event counter mode		
		⇨ 011	PPG output mode (Software start)		
		100	External trigger timer mode		
		101	Window mode		
		110	Pulse width measurement mode		
		111	Reserved		

[line 10 : Set the timer state]



Timer counter A0 control register

TA0CR		7	6	5	4	3	2	1	0
(0x0032)	Bit Symbol	TA0OVE	TA0TFF	TA0NC		-	-	TA0CAP TA0MPPG	TA0S
	Read/Write	R/W	R/W	R/W		R	R	R/W	R/W
	After reset	0	1	0	0	0	0	0	0

TA0OVE	Overflow interrupt control	0	Generate no INTTA0 interrupt request when the counter overflow occurs.	
		1	Generate an INTTA0 interrupt request when the counter overflow occurs.	
TA0TFF	Timer F/F control	0	Clear	
		1	Set	
TA0NC	Noise canceller sampling interval setting		NORMAL 1/2 or IDLE 1/2 mode	SLOW1/2 or SLEEP1 mode
		00	No noise canceller	No noise canceller
		01	fcgck/2	-
		10	fcgck/2^2	-
		11	fcgck/2^8	fs/2
TA0ACAP	Auto capture function	0	Disable the auto capture	
		1	Enable the auto capture	
TA0MPPG	PPG output control	0	Continuous	
		1	One-shot	
TA0S	Timer counter A start control	0	Stop & counter clear	
		1	Start	

5-4-3. Timer start process

```
1   void TCA0Start(void)
2   {
3           TA0DRAL = 0x28;
4           TA0DRAH = 0x23;
5           TA0DRBL = 0x94;
6           TA0DRBH = 0x11;
7           _asm(" SET (_TA0CR).0 ");
8   }
```

[line 3-6 : Set the timer initial value]

[line 7 : Start the timer count]

5-4-4. Timer stop process

```
1   void TCA0Stop(void)
2   {
3           TA0CR = 0xC0;
4   }
```

[line 3 : Stop the TA0]

5-4-5. Interrupt process

```
1   void _interrupt IntTCA0(void)
2   {
3           static UINT8_t i = 0;
4   
5           if (i < 33)
6           {
7                   if (((OUTPUTDATA >> i) & 0x01) == 0)
8                   {
9                           TA0DRAL = 0x60;
10                          TA0DRAH = 0x04;
11                          TA0DRBL = 0x30;
12                          TA0DRBH = 0x02;
13                          i++;
14                  }
15                  else
16                  {
17                          i++;
18                          TA0DRAL = 0xCA;
19                          TA0DRAH = 0x08;
20                          TA0DRBL = 0x30;
21                          TA0DRBH = 0x02;
22                  }
23          }
24          else
25          {
26                  i = 0;
27                  TCA0Stop();
28          }
29  }
```

[line 9-12,18-21 : Set the next timer value]

5-5. UART output : [uart.c]

5-5-1. Control process

```
 1  void sample_uart(void)
 2  {
 3          UINT8_t i = 0;
 4
 5          fUARTTXCheck = FALSE;
 6          f100msCheck = FALSE;
 7          gCheckCnt = 100;
 8
 9          UARTTXInit();
10          TBTInit();
11          UARTTXStart();
12          TBTStart();
13          UARTTXTrans();
14          while (fUARTTXCheck == FALSE)
15          {
16                  WDCDR = 0x4E;                   /* Clear WDT counter */
17                  if (f100msCheck == TRUE)
18                  {
19                          f100msCheck = FALSE;
20                          i++;
21                          if (i < sRunTimeCnt)
22                          {
23                                  UARTTXStart();
24                                  UARTTXTrans();
25                          }
26                          else
27                          {
28                                  fUARTTXCheck = TRUE;
29                          }
30                  }
31          }
32          UARTTXStop();
33          TBTStop();
34  }
```

[line 9 : Initialize UART] ➡ 5-5-2

[line 11 : Start UART] ➡ 5-5-3

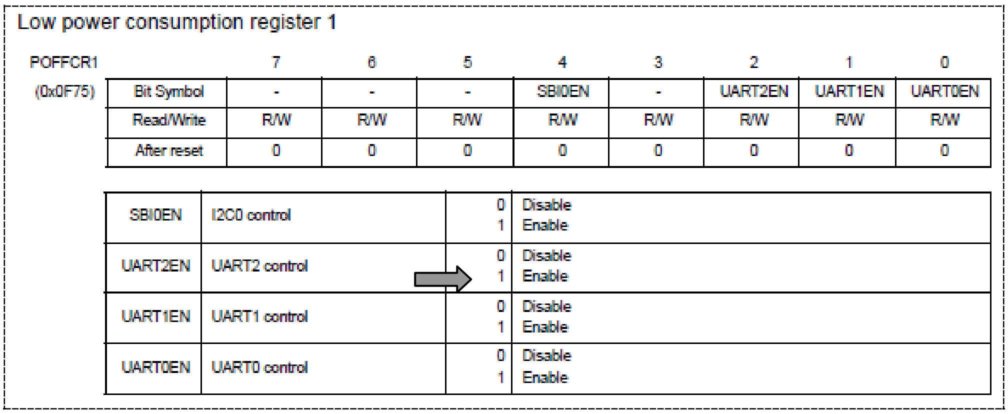
[line 13 : Set first character] ➡ 5-5-4

[line 32 : Stop UART] ➡ 5-5-5

5-5-2. Initialize process

```
 1  void UARTTXInit(void)
 2  {
 3          POFFCR1 = 0x04;                 /* Set UART2EN = 1 */
 4          _asm(″ SET (_P9CR).3 ″);
 5          _asm(″ SET (_P9FC).3 ″);        /* Set P93 as the UART2 output */
 6
 7          P9DR = 0x08;
 8          P9OUTCR = 0x08;
 9          P9PU = 0x08;
10
11          _DI();
12          EIRD = EIRD | 0x08;             /* Enable INTTXD2 */
13          _EI();
14          UART2CR1 = 0x00;                /* 1 stop bit,No parity */
15          UART2CR2 = 0x00;
16          UART2DR = 0x19;                 /* 9600bps */
17  }
```

[line 3 : Enable the UART power]

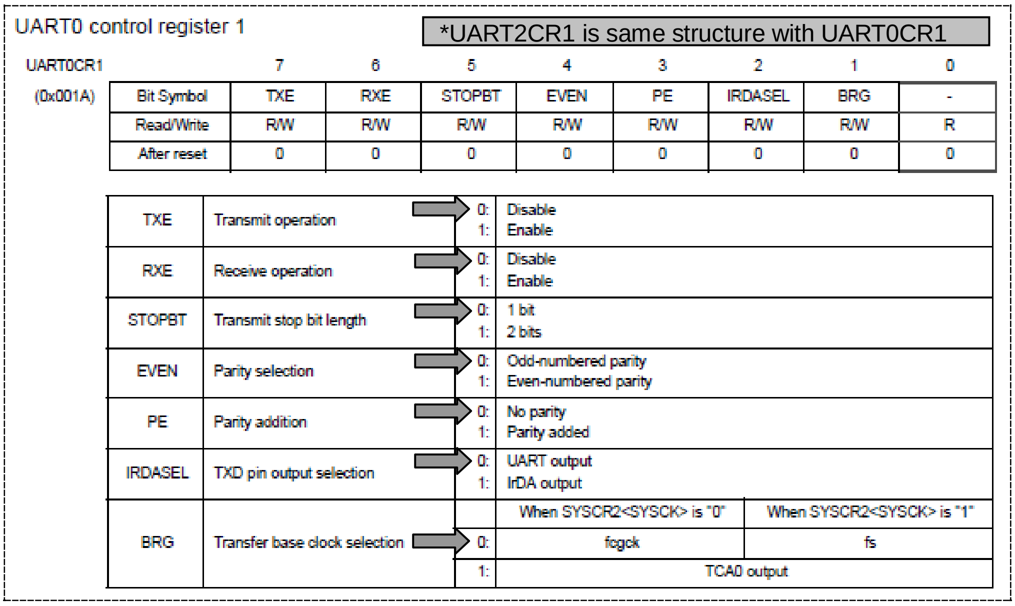


Low power consumption register 1

POFFCR1		7	6	5	4	3	2	1	0
(0x0F75)	Bit Symbol	-	-	-	SBI0EN	-	UART2EN	UART1EN	UART0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

SBI0EN	I2C0 control	0 1	Disable Enable
UART2EN	UART2 control	0 1	Disable Enable
UART1EN	UART1 control	0 1	Disable Enable
UART0EN	UART0 control	0 1	Disable Enable

[line 14 : Stet stop bit & parity]



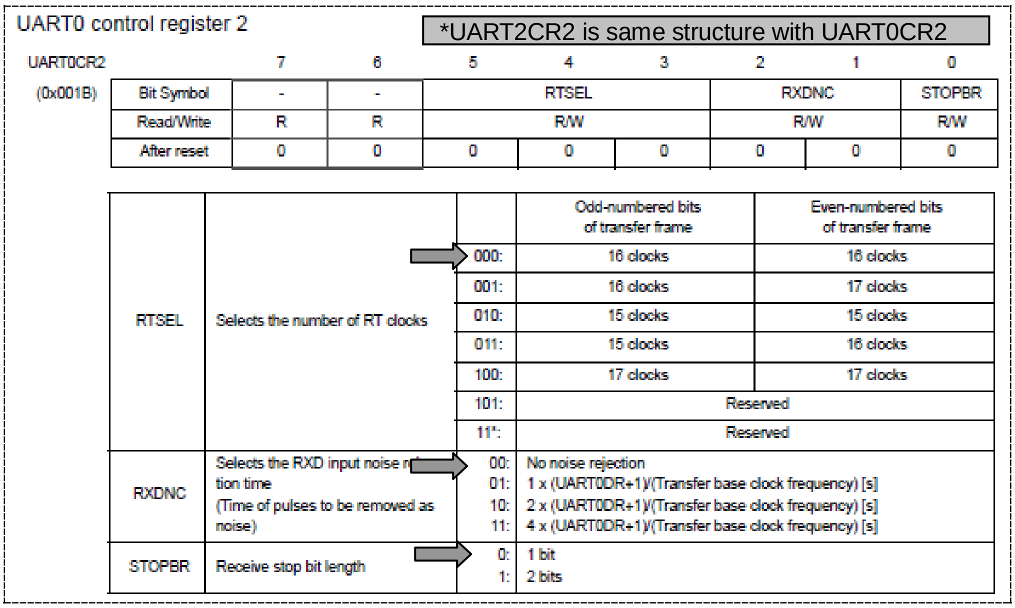
UART0 control register 1

*UART2CR1 is same structure with UART0CR1

UART0CR1		7	6	5	4	3	2	1	0
(0x001A)	Bit Symbol	TXE	RXE	STOPBT	EVEN	PE	IRDASEL	BRG	-
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
	After reset	0	0	0	0	0	0	0	0

TXE	Transmit operation	0: 1:	Disable Enable	
RXE	Receive operation	0: 1:	Disable Enable	
STOPBT	Transmit stop bit length	0: 1:	1 bit 2 bits	
EVEN	Parity selection	0: 1:	Odd-numbered parity Even-numbered parity	
PE	Parity addition	0: 1:	No parity Parity added	
IRDASEL	TXD pin output selection	0: 1:	UART output IrDA output	
BRG	Transfer base clock selection		When SYSCR2<SYSCK> is "0"	When SYSCR2<SYSCK> is "1"
		0:	fcgck	fs
		1:	TCA0 output	

[line 15 : Select “no noise rejection”]



UART0 control register 2

*UART2CR2 is same structure with UART0CR2

UART0CR2		7	6	5	4	3	2	1	0
(0x001B)	Bit Symbol	-	-	RTSEL			RXDNC		STOPBR
	Read/Write	R	R	R/W			R/W		R/W
	After reset	0	0	0	0	0	0	0	0

RTSEL	Selects the number of RT clocks		Odd-numbered bits of transfer frame	Even-numbered bits of transfer frame
		000:	16 clocks	16 clocks
		001:	16 clocks	17 clocks
		010:	15 clocks	15 clocks
		011:	15 clocks	16 clocks
		100:	17 clocks	17 clocks
		101:	Reserved	
		11*:	Reserved	
RXDNC	Selects the RXD input noise r tion time (Time of pulses to be removed as noise)	00: 01: 10: 11:	No noise rejection 1 x (UART0DR+1)/(Transfer base clock frequency) [s] 2 x (UART0DR+1)/(Transfer base clock frequency) [s] 4 x (UART0DR+1)/(Transfer base clock frequency) [s]	
STOPBR	Receive stop bit length	0: 1:	1 bit 2 bits	

[line 16 : Set baud rate]

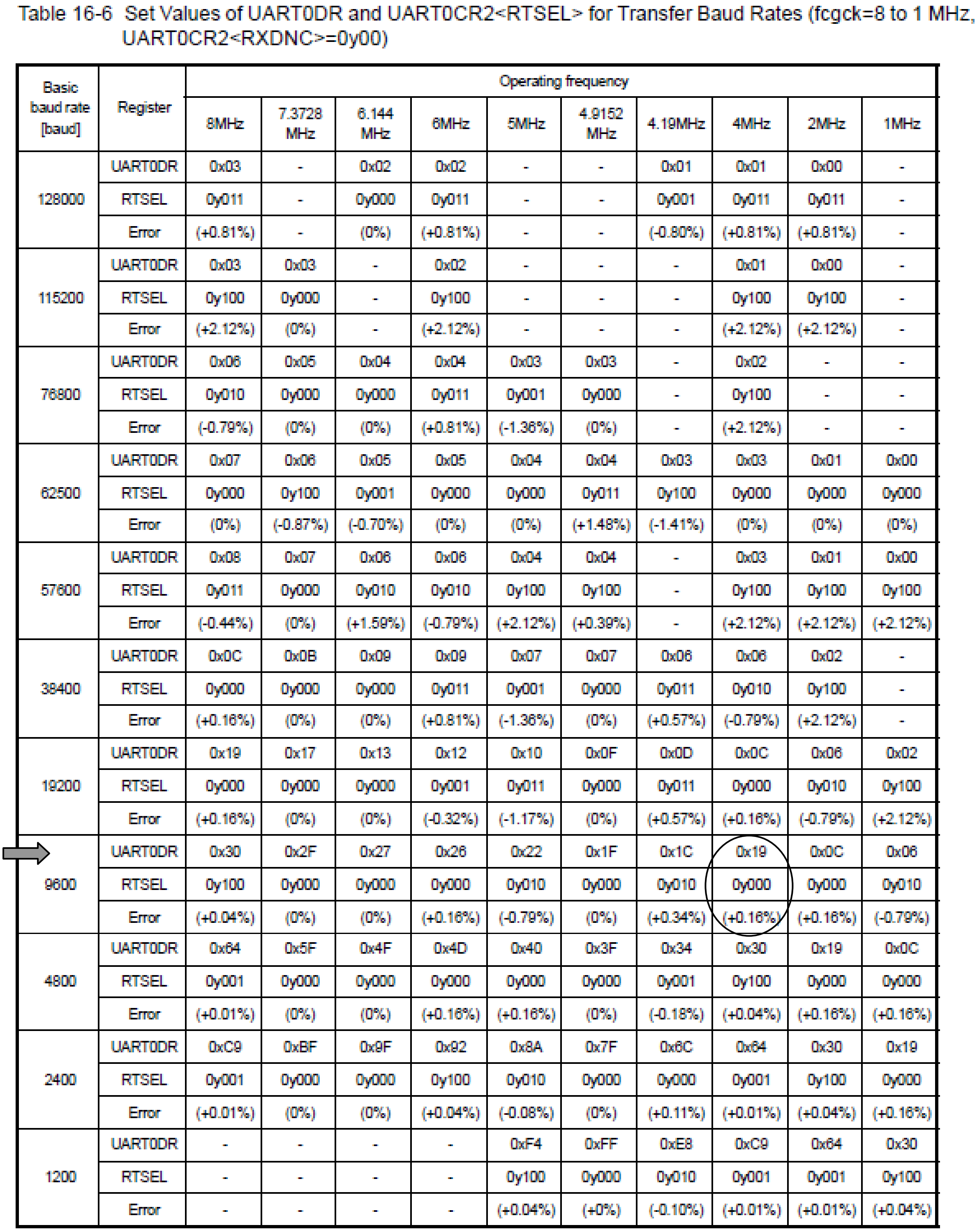


Table 16-6 Set Values of UART0DR and UART0CR2<RTSEL> for Transfer Baud Rates (fcgck=8 to 1 MHz, UART0CR2<RXDNC>=0y00)

Basic baud rate [baud]	Register	Operating frequency									
		8MHz	7.3728 MHz	6.144 MHz	6MHz	5MHz	4.9152 MHz	4.19MHz	4MHz	2MHz	1MHz
128000	UART0DR	0x03	-	0x02	0x02	-	-	0x01	0x01	0x00	-
	RTSEL	0y011	-	0y000	0y011	-	-	0y001	0y011	0y011	-
	Error	(+0.81%)	-	(0%)	(+0.81%)	-	-	(-0.80%)	(+0.81%)	(+0.81%)	-
115200	UART0DR	0x03	0x03	-	0x02	-	-	-	0x01	0x00	-
	RTSEL	0y100	0y000	-	0y100	-	-	-	0y100	0y100	-
	Error	(+2.12%)	(0%)	-	(+2.12%)	-	-	-	(+2.12%)	(+2.12%)	-
76800	UART0DR	0x06	0x05	0x04	0x04	0x03	0x03	-	0x02	-	-
	RTSEL	0y010	0y000	0y000	0y011	0y001	0y000	-	0y100	-	-
	Error	(-0.79%)	(0%)	(0%)	(+0.81%)	(-1.36%)	(0%)	-	(+2.12%)	-	-
62500	UART0DR	0x07	0x06	0x05	0x05	0x04	0x04	0x03	0x03	0x01	0x00
	RTSEL	0y000	0y100	0y001	0y000	0y000	0y011	0y100	0y000	0y000	0y000
	Error	(0%)	(-0.87%)	(-0.70%)	(0%)	(0%)	(+1.48%)	(-1.41%)	(0%)	(0%)	(0%)
57600	UART0DR	0x08	0x07	0x06	0x06	0x04	0x04	-	0x03	0x01	0x00
	RTSEL	0y011	0y000	0y010	0y010	0y100	0y100	-	0y100	0y100	0y100
	Error	(-0.44%)	(0%)	(+1.59%)	(-0.79%)	(+2.12%)	(+0.39%)	-	(+2.12%)	(+2.12%)	(+2.12%)
38400	UART0DR	0x0C	0x0B	0x09	0x09	0x07	0x07	0x06	0x06	0x02	-
	RTSEL	0y000	0y000	0y000	0y011	0y001	0y000	0y011	0y010	0y100	-
	Error	(+0.16%)	(0%)	(0%)	(+0.81%)	(-1.36%)	(0%)	(+0.57%)	(-0.79%)	(+2.12%)	-
19200	UART0DR	0x19	0x17	0x13	0x12	0x10	0x0F	0x0D	0x0C	0x06	0x02
	RTSEL	0y000	0y000	0y000	0y001	0y011	0y000	0y011	0y000	0y010	0y100
	Error	(+0.16%)	(0%)	(0%)	(-0.32%)	(-1.17%)	(0%)	(+0.57%)	(+0.16%)	(-0.79%)	(+2.12%)
9600	UART0DR	0x30	0x2F	0x27	0x26	0x22	0x1F	0x1C	0x19	0x0C	0x06
	RTSEL	0y100	0y000	0y000	0y000	0y010	0y000	0y010	0y000	0y000	0y010
	Error	(+0.04%)	(0%)	(0%)	(+0.16%)	(-0.79%)	(0%)	(+0.34%)	(+0.16%)	(+0.16%)	(-0.79%)
4800	UART0DR	0x64	0x5F	0x4F	0x4D	0x40	0x3F	0x34	0x30	0x19	0x0C
	RTSEL	0y001	0y000	0y000	0y000	0y000	0y000	0y001	0y100	0y000	0y000
	Error	(+0.01%)	(0%)	(0%)	(+0.16%)	(+0.16%)	(0%)	(-0.18%)	(+0.04%)	(+0.16%)	(+0.16%)
2400	UART0DR	0xC9	0xBF	0x9F	0x92	0x8A	0x7F	0x6C	0x64	0x30	0x19
	RTSEL	0y001	0y000	0y000	0y100	0y010	0y000	0y000	0y001	0y100	0y000
	Error	(+0.01%)	(0%)	(0%)	(+0.04%)	(-0.08%)	(0%)	(+0.11%)	(+0.01%)	(+0.04%)	(+0.16%)
1200	UART0DR	-	-	-	-	0xF4	0xFF	0xE8	0xC9	0x64	0x30
	RTSEL	-	-	-	-	0y100	0y000	0y010	0y001	0y001	0y100
	Error	-	-	-	-	(+0.04%)	(+0%)	(-0.10%)	(+0.01%)	(+0.01%)	(+0.04%)

5-5-3. UART start process

```
1   void UARTTXStart(void)
2   {
3           _asm(″ SET (_UART2CR1).7 ″);
4   }
```

[line 3 : Start UART transfer]

5-5-4. UART first data set process

```
1   void UARTTXTrans(void)
2   {
3           TD2BUF = OutputData[0];
4   }
```

[line 3 : Set first data to the transfer buffer]

5-5-5. UART stop process

```
1   void UARTTXStop(void)
2   {
3           UART2CR1 = 0x00;
4   }
```

[line 3 : Stop the UART]

5-5-6. Interrupt process

```
1   void _interrupt IntTXD2(void)
2   {
3           static UINT8_t i = 0;
4   
5           i++;
6           if (i <= 7)
7           {
8                   TD2BUF = OutputData[i];
9           }
10          else
11          {
12                  UARTTXStop();
13                  i = 0;
14          }
15  }
```

[line 8 : Set the next transfer data]

[line 12 : Stop UART]

5-6. Clock synchronous SIO output : [sio.c]

5-6-1. Control process

```
 1  void sample_sio(void)
 2  {
 3          fSIOTXCheck = FALSE;
 4          f20msCheck = FALSE;
 5          gCheckCnt = 20;
 6          cSIOTXData.word = 0x00;
 7
 8          TBTInit();
 9          TBTStart();
10          SIOTXInit();
11          SIOTXTrans();
12          SIOTXStart();
13          while (fSIOTXCheck == FALSE)
14          {
15                  WDCDR = 0x4E;                           /* Clear WDT counter */
16                  if (f20msCheck == TRUE)
17                  {
18                          f20msCheck = FALSE;
19                          cSIOTXData.word++;
20                          if (cSIOTXData.word == 0x00)
21                          {
22                                  fSIOTXCheck = TRUE;
23                          }
24                          else
25                          {
26                                  SIOTXTrans();
27                                  SIOTXStart();
28                          }
29                  }
30          }
31          SIOTXStop();
32          TBTStop();
33  }
```

[line 10 : Initialize SIO] ➡ 5-6-2

[line 11 : Set first transfer data] ➡ 5-6-3

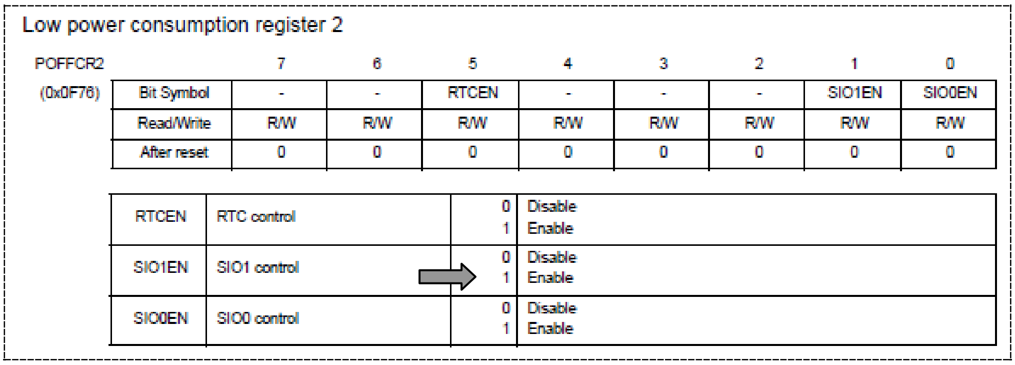
[line 12 : Start SIO] ➡ 5-6-4

[line 31 : Stop SIO] ➡ 5-6-5

5-6-2. Initialize process

```
 1  void SIOTXInit(void)
 2  {
 3          POFFCR2 = 0x02;                 /* Enable SIO1 */
 4          _asm(" SET (_P9CR).0 ");        /* Set P90 as the SIO data output port */
 5          _asm(" SET (_P9FC).0 ");
 6          _asm(" SET (_P9CR).2 ");        /* Set P92 as the SCLK1 output pin */
 7          _asm(" SET (_P9FC).2 ");
 8          P9DR = 0x05;                    /* P90 and P92 output the H level first */
 9          P9OUTCR = 0x05;                 /* P90 and P92 open drain output */
10          P9PU = 0x05;                    /* The build-in pull-up resistor connected */
11          SERSEL = 0x08;
12
13          _DI();
14          EIRE = EIRE | 0x40;             /* Enable the   INTSIO1 */
15          _EI();
16
17          SIO1CR = 0x01;                  /* LSB first, 8-bit transmit mode ,SIOCKS = fc/2^9
18                                             Data transmission at the Falling edge */
19  }
```

[line 3 : Enable the SIO power]

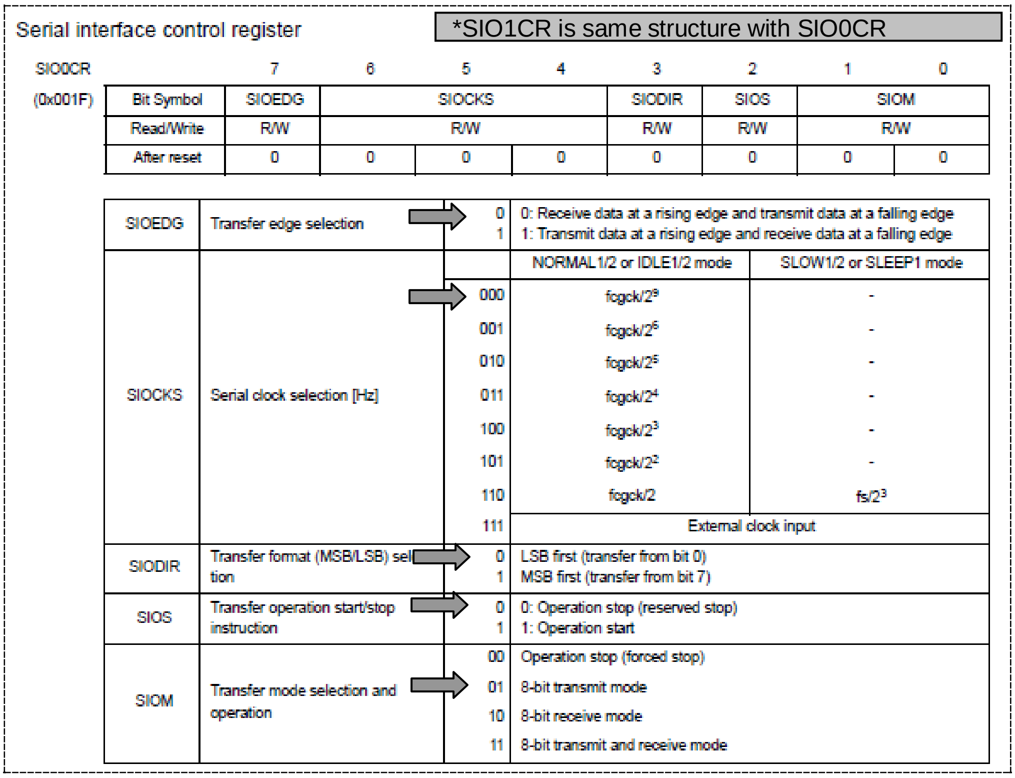


Low power consumption register 2

POFFCR2		7	6	5	4	3	2	1	0
(0x0F76)	Bit Symbol	-	-	RTCEN	-	-	-	SIO1EN	SIO0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

RTCEN	RTC control	0 1	Disable Enable
SIO1EN	SIO1 control	0 1	Disable Enable
SIO0EN	SIO0 control	0 1	Disable Enable

[line 17 : Set SIO tranfer mode]



Serial interface control register

*SIO1CR is same structure with SIO0CR

SIO0CR		7	6	5	4	3	2	1	0
(0x001F)	Bit Symbol	SIOEDG	SIOCKS			SIODIR	SIOS	SIOM	
	Read/Write	R/W	R/W			R/W	R/W	R/W	
	After reset	0	0	0	0	0	0	0	0

SIOEDG	Transfer edge selection	0 1	0: Receive data at a rising edge and transmit data at a falling edge 1: Transmit data at a rising edge and receive data at a falling edge	
SIOCKS	Serial clock selection [Hz]		NORMAL1/2 or IDLE1/2 mode	SLOW1/2 or SLEEP1 mode
		000	$fcgck/2^9$	-
		001	$fcgck/2^6$	-
		010	$fcgck/2^5$	-
		011	$fcgck/2^4$	-
		100	$fcgck/2^3$	-
		101	$fcgck/2^2$	-
		110	fcgck/2	$fs/2^3$
		111	External clock input	
SIODIR	Transfer format (MSB/LSB) selection	0 1	LSB first (transfer from bit 0) MSB first (transfer from bit 7)	
SIOS	Transfer operation start/stop instruction	0 1	0: Operation stop (reserved stop) 1: Operation start	
SIOM	Transfer mode selection and operation	00 01 10 11	Operation stop (forced stop) 8-bit transmit mode 8-bit receive mode 8-bit transmit and receive mode	

5-6-3. SIO start process

```
1   void SIOTXStart(void)
2   {
3       _asm(" SET (_SIO1CR).2 ");                   /* Start SIO function */
4   }
```

[line 3 : Start SIO transfer]

5-6-4. SIO first data set process

```
1   void SIOTXTrans(void)
2   {
3       SIO1BUF = cSIOTXDataL;                     /* Write data into the buffer */
4   }
```

[line 3 : Set first data to the transfer buffer]

5-6-5. SIO stop process

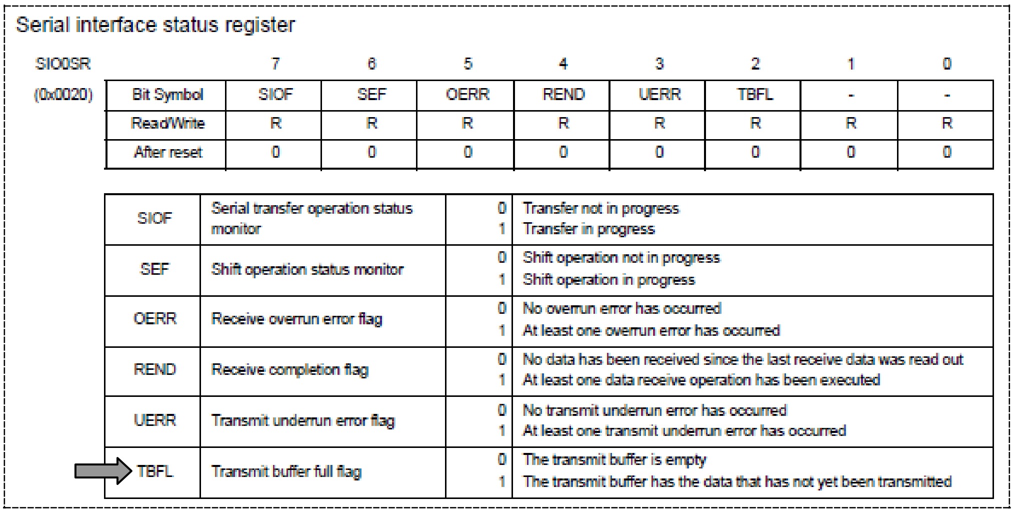
```
1   void SIOTXStop(void)
2   {
3       SIO1CR = 0x01;                             /* Stop SIO function */
4   }
```

[line 3 : Stop the SIO]

5-6-6. Interrupt process

```
 1  void _interrupt IntSIO1(void)
 2  {
 3          static UINT8_t i = 0;
 4
 5          i++;
 6          if (i == 1)
 7          {
 8                  while ((SIO1SR & 0x04) == 0x04);          /* TBFL = 0 */
 9                  SIO1BUF = cSIOTXDataH;
10          }
11          else if (i == 2)
12          {
13                  SIOTXStop();
14          }
15          else
16          {
17                  i = 0;
18          }
19  }
```

[line 8 : Check the transfer buffer full]



Serial interface status register

SIO0SR		7	6	5	4	3	2	1	0
(0x0020)	Bit Symbol	SIOF	SEF	OERR	REND	UERR	TBFL	-	-
	Read/Write	R	R	R	R	R	R	R	R
	After reset	0	0	0	0	0	0	0	0

SIOF	Serial transfer operation status monitor	0 1	Transfer not in progress Transfer in progress
SEF	Shift operation status monitor	0 1	Shift operation not in progress Shift operation in progress
OERR	Receive overrun error flag	0 1	No overrun error has occurred At least one overrun error has occurred
REND	Receive completion flag	0 1	No data has been received since the last receive data was read out At least one data receive operation has been executed
UERR	Transmit underrun error flag	0 1	No transmit underrun error has occurred At least one transmit underrun error has occurred
TBFL	Transmit buffer full flag	0 1	The transmit buffer is empty The transmit buffer has the data that has not yet been transmitted

[line 12 : Set the next data]

[line 13 : Stop SIO]

5-7. I2C output : [sbi.c]

5-7-1. Control process

```
 1  void sample_sbi(void)
 2  {
 3      UINT8_t i = 0;
 4      fSBITXCheck = FALSE;
 5      f10msCheck = FALSE;
 6      gCheckCnt = 10;
 7  
 8      SBIDeviceInit();
 9      SBITXInit();
10      SBITXStart();
11      TBTInit();
12      TBTStart();
13      while (fSBITXCheck == FALSE)
14      {
15          WDCDR = 0x4E;                           /* Clear WDT counter */
16          if (f10msCheck == TRUE)
17          {
18              f10msCheck = FALSE;
19              i++;
20              if (i < sRunTimeCnt)
21              {
22                  SBITXStart();
23              }
24              else
25              {
26                  fSBITXCheck = TRUE;
27              }
28          }
29      }
30      SBITXStop();
31      TBTStop();
32  }
```

[line 8 : Initialize the I2C(SBI)] ➡ 5-7-2

[line 9 : Initialize the I2C port & interrupt] ➡ 5-7-3

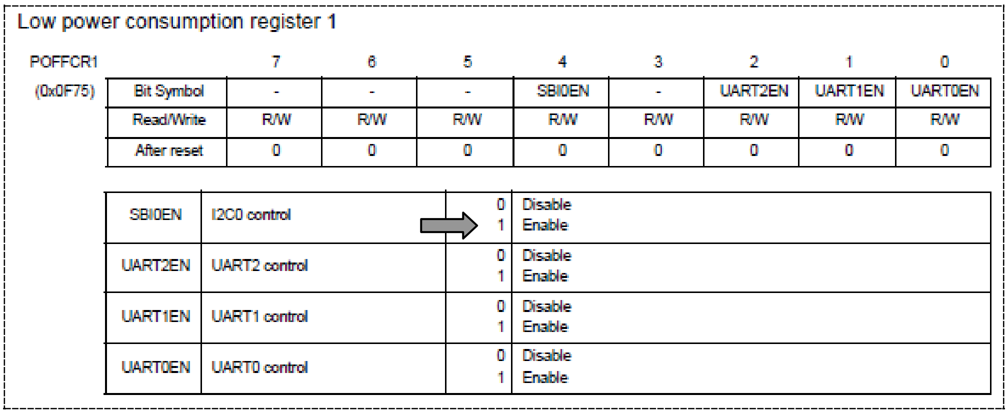
[line 10 : Start the I2C]➡ 5-7-4

[line 30 : Stop the I2C]➡ 5-7-5

5-7-2. Initialize process

```
1   void SBIDeviceInit(void)
2   {
3       POFFCR1 = 0x10;
4       while (P2PRD == 0x0C);
5       SBI0CR2 = 0x18;               /* SBI mode */
6       SBI0CR1 = 0x15;
7       I2C0AR = 0x00;
8       SBI0CR2 = 0x18;               /* Slave mode */
9   }
```

[line 3 : Enable the SBI power]

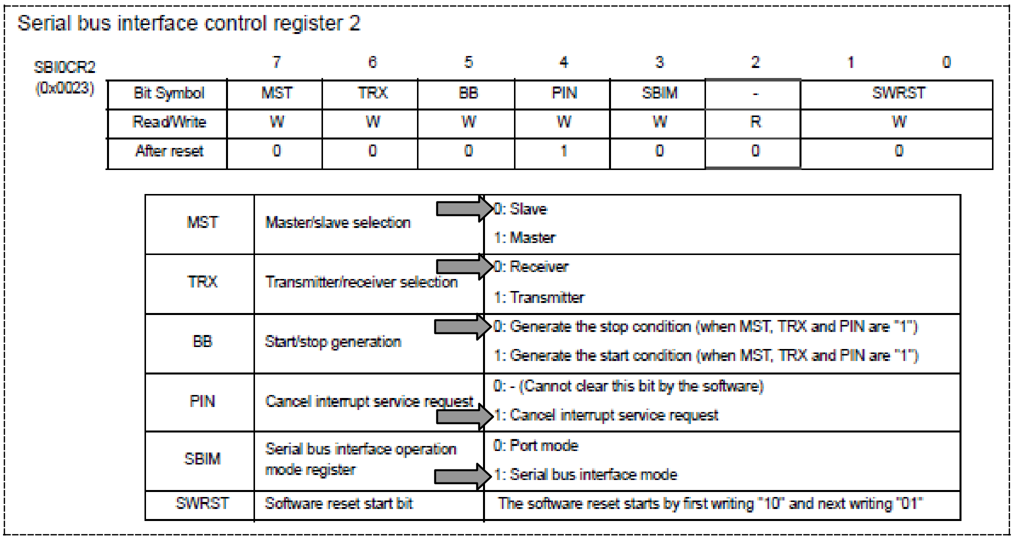


Low power consumption register 1

POFFCR1		7	6	5	4	3	2	1	0
(0x0F75)	Bit Symbol	-	-	-	SBI0EN	-	UART2EN	UART1EN	UART0EN
	Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	After reset	0	0	0	0	0	0	0	0

SBI0EN	I2C0 control	0 1	Disable Enable
UART2EN	UART2 control	0 1	Disable Enable
UART1EN	UART1 control	0 1	Disable Enable
UART0EN	UART0 control	0 1	Disable Enable

[line 5 : Set the SBI mode]

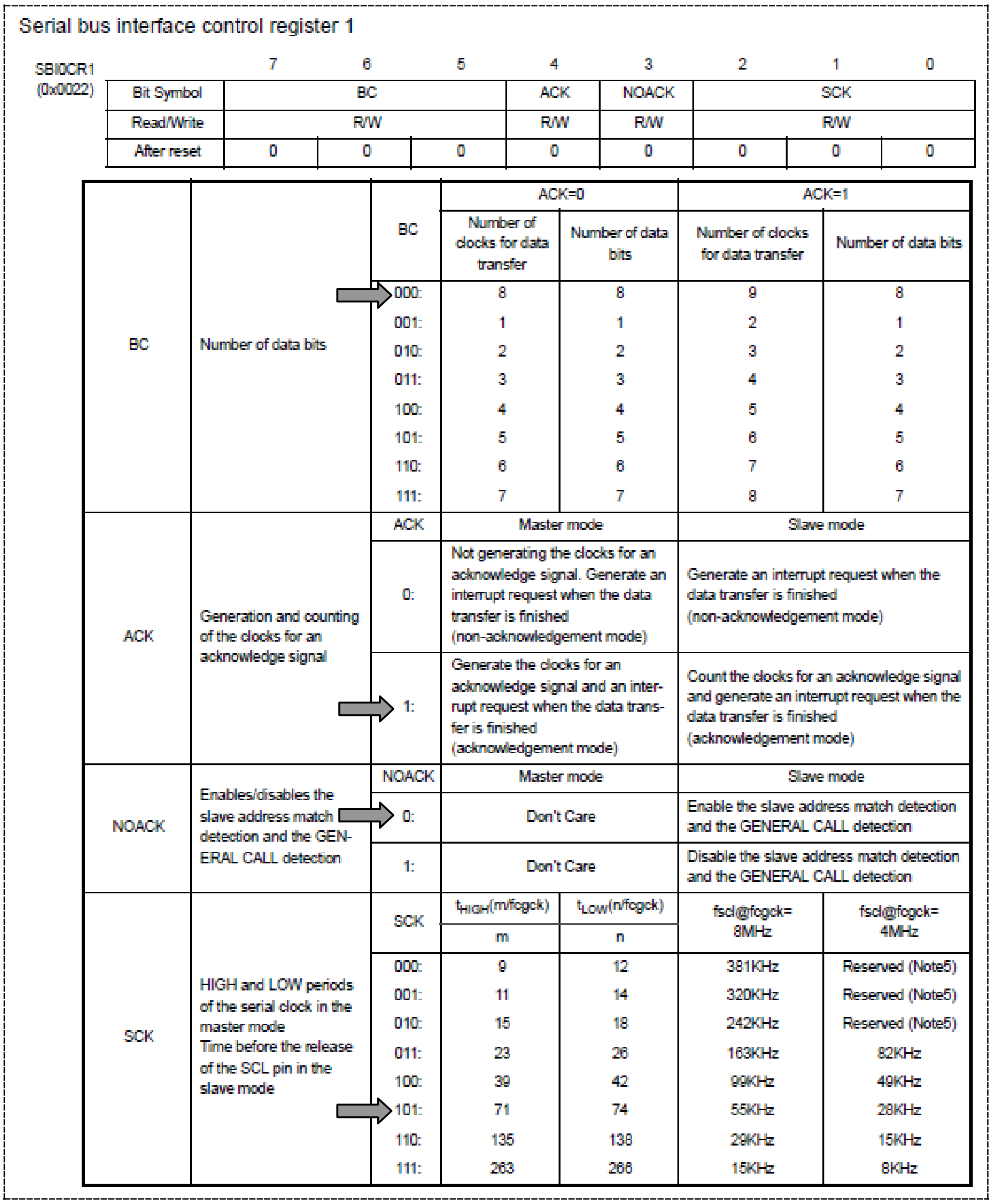


Serial bus interface control register 2

SBI0CR2		7	6	5	4	3	2	1	0
(0x0023)	Bit Symbol	MST	TRX	BB	PIN	SBIM	-	SWRST	
	Read/Write	W	W	W	W	W	R	W	
	After reset	0	0	0	1	0	0	0	

MST	Master/slave selection	0: Slave 1: Master
TRX	Transmitter/receiver selection	0: Receiver 1: Transmitter
BB	Start/stop generation	0: Generate the stop condition (when MST, TRX and PIN are "1") 1: Generate the start condition (when MST, TRX and PIN are "1")
PIN	Cancel interrupt service request	0: - (Cannot clear this bit by the software) 1: Cancel interrupt service request
SBIM	Serial bus interface operation mode register	0: Port mode 1: Serial bus interface mode
SWRST	Software reset start bit	The software reset starts by first writing "10" and next writing "01"

[line 6 : Set the clock]



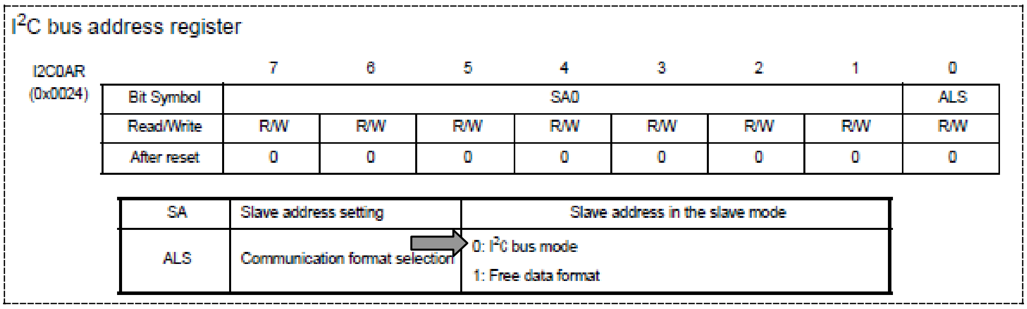
Serial bus interface control register 1

SBI0CR1 (0x0022)

	7	6	5	4	3	2	1	0
Bit Symbol	BC			ACK	NOACK	SCK		
Read/Write	R/W			R/W	R/W	R/W		
After reset	0	0	0	0	0	0	0	0

			ACK=0		ACK=1	
		BC	Number of clocks for data transfer	Number of data bits	Number of clocks for data transfer	Number of data bits
BC	Number of data bits	000:	8	8	9	8
		001:	1	1	2	1
		010:	2	2	3	2
		011:	3	3	4	3
		100:	4	4	5	4
		101:	5	5	6	5
		110:	6	6	7	6
		111:	7	7	8	7
ACK	Generation and counting of the clocks for an acknowledge signal	ACK	Master mode		Slave mode	
		0:	Not generating the clocks for an acknowledge signal. Generate an interrupt request when the data transfer is finished (non-acknowledgement mode)		Generate an interrupt request when the data transfer is finished (non-acknowledgement mode)	
		1:	Generate the clocks for an acknowledge signal and an interrupt request when the data transfer is finished (acknowledgement mode)		Count the clocks for an acknowledge signal and generate an interrupt request when the data transfer is finished (acknowledgement mode)	
NOACK	Enables/disables the slave address match detection and the GENERAL CALL detection	NOACK	Master mode		Slave mode	
		0:	Don't Care		Enable the slave address match detection and the GENERAL CALL detection	
		1:	Don't Care		Disable the slave address match detection and the GENERAL CALL detection	
SCK	HIGH and LOW periods of the serial clock in the master mode Time before the release of the SCL pin in the slave mode	SCK	t_{HIGH}(m/fcgck)	t_{LOW}(n/fcgck)	fscl@fcgck= 8MHz	fscl@fcgck= 4MHz
			m	n		
		000:	9	12	381KHz	Reserved (Note5)
		001:	11	14	320KHz	Reserved (Note5)
		010:	15	18	242KHz	Reserved (Note5)
		011:	23	26	163KHz	82KHz
		100:	39	42	99KHz	49KHz
		101:	71	74	55KHz	28KHz
		110:	135	138	29KHz	15KHz
		111:	263	266	15KHz	8KHz

[line 7 : Set the I2C address]



I^{2}C bus address register

I2C0AR (0x0024)

	7	6	5	4	3	2	1	0
Bit Symbol	SA0							ALS
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

SA	Slave address setting	Slave address in the slave mode
ALS	Communication format selection	0: I^{2}C bus mode 1: Free data format

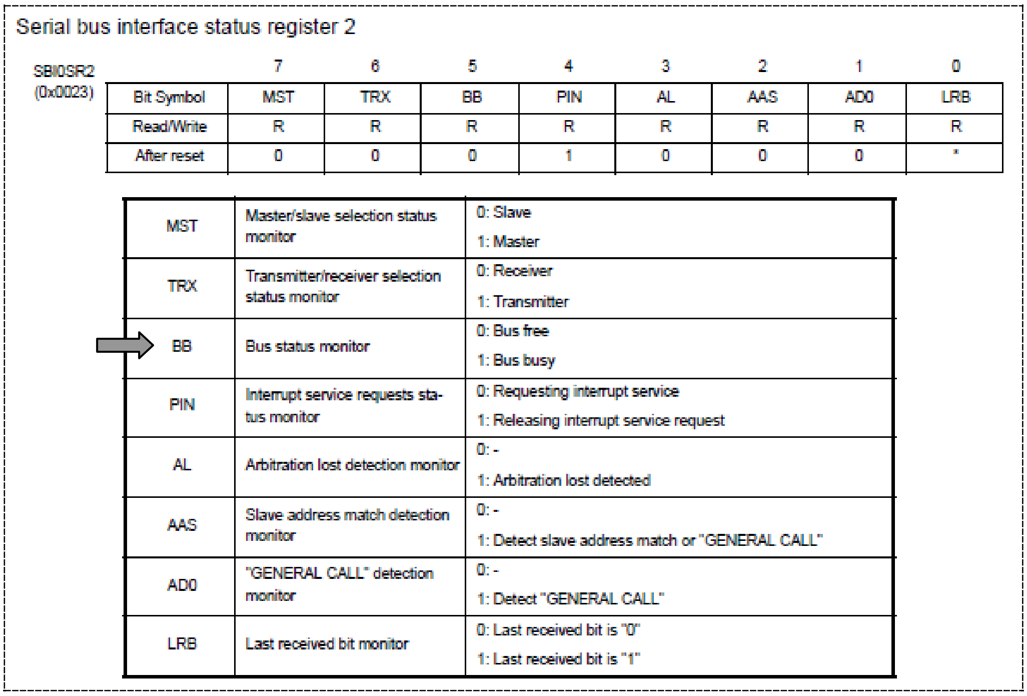
5-7-3. Initialize process 2

```
void SBITXInit(void)
{
        _DI();
        EIRH = EIRH | 0x80;
        _EI();
        _asm(" SET (_P2CR).3 ");
        _asm(" SET (_P2FC).3 ");          /* Set P23 as the SBI data output port */
        _asm(" SET (_P2CR).4 ");
        _asm(" SET (_P2FC).4 ");          /* Set P24 as the SBI Clock output port */
        P2DR = 0x18;
}
```

5-7-4. I2C(SBI) start process

```
1    void SBITXStart(void)
2    {
3            UINT8_t temp;
4
5            temp = cSlaveAddr + cSBI_WRITE;
6            SBI0CR1 = 0x15;
7            while ((SBI0SR2 & 0x20) == 0x20);
8            SBI0DBR = temp;
9            SBI0CR2 = 0xF8;
10   }
```

[line 7 : Check th bus free]



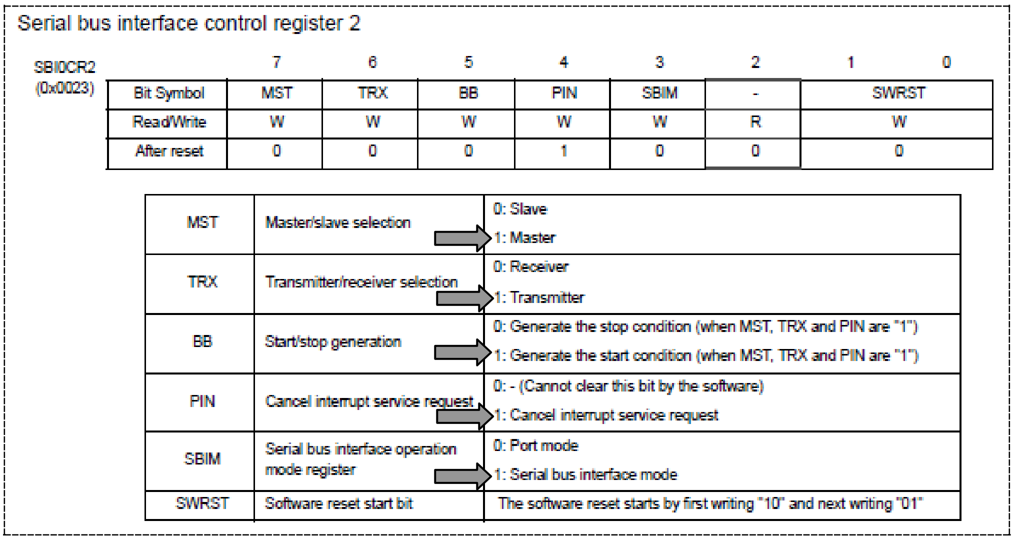
Serial bus interface status register 2

SBI0SR2 (0x0023)	7	6	5	4	3	2	1	0
Bit Symbol	MST	TRX	BB	PIN	AL	AAS	AD0	LRB
Read/Write	R	R	R	R	R	R	R	R
After reset	0	0	0	1	0	0	0	*

MST	Master/slave selection status monitor	0: Slave 1: Master
TRX	Transmitter/receiver selection status monitor	0: Receiver 1: Transmitter
BB	Bus status monitor	0: Bus free 1: Bus busy
PIN	Interrupt service requests status monitor	0: Requesting interrupt service 1: Releasing interrupt service request
AL	Arbitration lost detection monitor	0: - 1: Arbitration lost detected
AAS	Slave address match detection monitor	0: - 1: Detect slave address match or "GENERAL CALL"
AD0	"GENERAL CALL" detection monitor	0: - 1: Detect "GENERAL CALL"
LRB	Last received bit monitor	0: Last received bit is "0" 1: Last received bit is "1"

[line 8 : Set the first data(address)]

[line 9 : Start I2C transfer]



Serial bus interface control register 2

SBI0CR2 (0x0023)

	7	6	5	4	3	2	1	0
Bit Symbol	MST	TRX	BB	PIN	SBIM	-	SWRST	
Read/Write	W	W	W	W	W	R	W	
After reset	0	0	0	1	0	0	0	

MST	Master/slave selection	0: Slave 1: Master
TRX	Transmitter/receiver selection	0: Receiver 1: Transmitter
BB	Start/stop generation	0: Generate the stop condition (when MST, TRX and PIN are "1") 1: Generate the start condition (when MST, TRX and PIN are "1")
PIN	Cancel interrupt service request	0: - (Cannot clear this bit by the software) 1: Cancel interrupt service request
SBIM	Serial bus interface operation mode register	0: Port mode 1: Serial bus interface mode
SWRST	Software reset start bit	The software reset starts by first writing "10" and next writing "01"

5-7-5. SBI stop process

```
 1  void SBITXStop(void)
 2  {
 3          SBI0CR2 = 0xD8;
 4          while ((SBI0SR2 & 0x20 ) == 0x20);
 5  }
```

[line 3 : Stop the SBI]

[line 4 : Check the bus free]

5-7-6. Interrupt process

```
 1  void _interrupt IntSBI0(void)
 2  {
 3          static UINT8_t i = 0;
 4  
 5          i++;
 6          if (i == 1)
 7          {
 8                  if ((SBI0SR2 & 0x01) == 0x00)
 9                  {
10                          SBI0DBR = cSBITXData;
11                  }
12          }
13          else
14          {
15                  i = 0;
16                  SBITXStop();
17                  SBI0CR2 = 0x1A;
18                  SBI0CR2 = 0x19;
19          }
20  }
```

[line 8 : Check the last bit]

[line 10 : Set the next data]

[line 16 : Stop SBI]

[line 17-18 : Reset SBI (1st : SWRST=10,2nd : SWRST=01)]

5-8. 10 bit A/D input : [adc.c]

5-8-1. Control process

```
 1  void sample_adc(void)
 2  {
 3      UINT8_t i = 0;
 4  
 5      fADCCheck = FALSE;
 6      f2msCheck = FALSE;
 7      fOnceCheck = FALSE;
 8      gCheckCnt = 2;
 9  
10      ADCInit();
11      ADCStart();
12      TBTInit();
13      TBTStart();
14      while (fADCCheck == FALSE)
15      {
16          WDCDR = 0x4E;                          /* Clear WDT counter */
17          if (f2msCheck == FALSE)
18          {
19              if (fOnceCheck == FALSE)
20              {
21                  while ((ADCCR2 & 0x80) == 0x00);
22                  ADCGetData();
23                  fOnceCheck = TRUE;
24              }
25          }
26          else
27          {
28              f2msCheck = FALSE;
29              i++;
30              if (i > sRunTimeCnt)
31              {
32                  fADCCheck    = TRUE;
33              }
34              else
35              {
36                  ADCStart();
37                  fOnceCheck = FALSE;
38              }
39          }
40      }
41      ADCStop();
42      TBTStop();
43  }
```

[line 10 : Initialize the ADC] ➡ 5-8-2

[line 11 : Start the AD conversion] ➡ 5-8-3

[line 21 : Check the conversion finish]

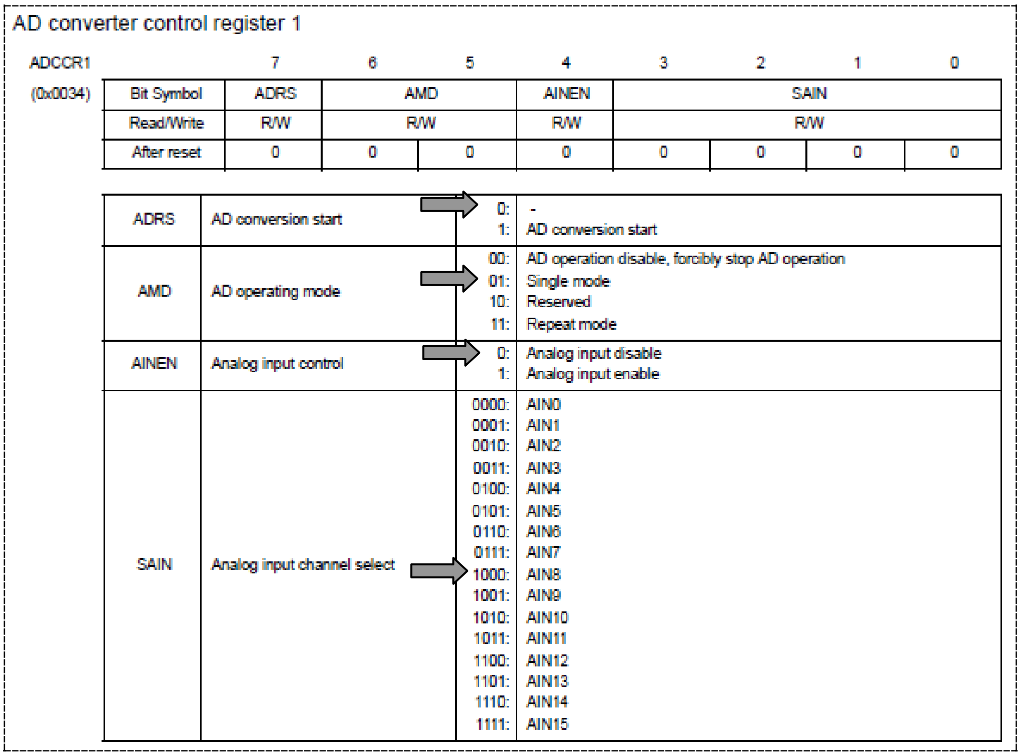
[line 22 : Get the conversion data] ➡ 5-8-4

[line 41 : Stop the ADC] ➡ 5-8-5

5-8-2. Initialize process

```
1    void ADCInit(void)
2    {
3        P5CR = 0x00;
4        _asm(" SET (_P5FC).0 ");            /* P50 as the AIN input port */
5
6        PBCR = 0xff;
7        PBDR = 0xff;
8
9        ADCCR1 = 0x38;                       /* Single mode and AIN8 */
10       ADCCR2 = 0x01;                       /* 78/fcgck */
11   }
```

[line 9 : Select conversion mode & channel]

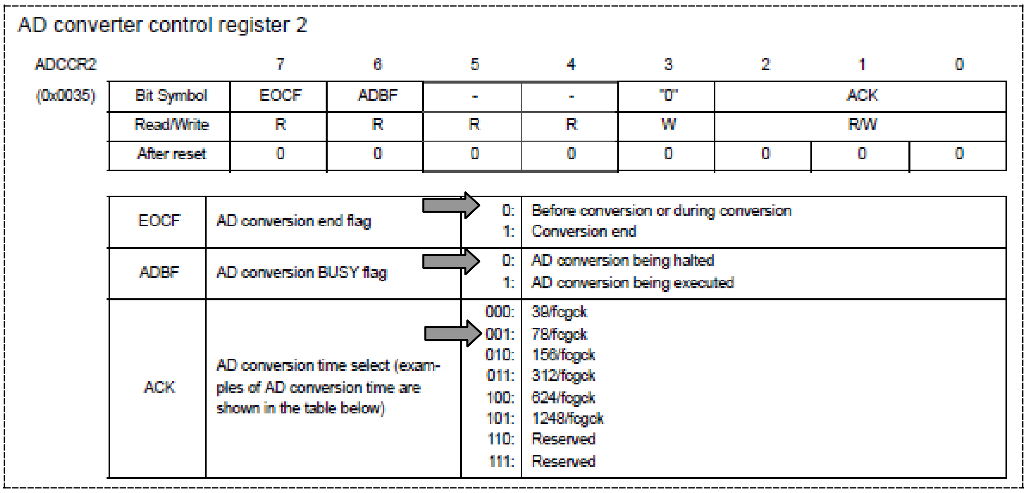


AD converter control register 1

ADCCR1		7	6	5	4	3	2	1	0
(0x0034)	Bit Symbol	ADRS	AMD		AINEN	SAIN			
	Read/Write	R/W	R/W		R/W	R/W			
	After reset	0	0	0	0	0	0	0	0

ADRS	AD conversion start	0: 1:	- AD conversion start
AMD	AD operating mode	00: 01: 10: 11:	AD operation disable, forcibly stop AD operation Single mode Reserved Repeat mode
AINEN	Analog input control	0: 1:	Analog input disable Analog input enable
SAIN	Analog input channel select	0000: 0001: 0010: 0011: 0100: 0101: 0110: 0111: 1000: 1001: 1010: 1011: 1100: 1101: 1110: 1111:	AIN0 AIN1 AIN2 AIN3 AIN4 AIN5 AIN6 AIN7 AIN8 AIN9 AIN10 AIN11 AIN12 AIN13 AIN14 AIN15

[line 10 : Set the conversion time]



AD converter control register 2

ADCCR2		7	6	5	4	3	2	1	0
(0x0035)	Bit Symbol	EOCF	ADBF	-	-	"0"	ACK		
	Read/Write	R	R	R	R	W	R/W		
	After reset	0	0	0	0	0	0	0	0

EOCF	AD conversion end flag	0: 1:	Before conversion or during conversion Conversion end
ADBF	AD conversion BUSY flag	0: 1:	AD conversion being halted AD conversion being executed
ACK	AD conversion time select (examples of AD conversion time are shown in the table below)	000: 001: 010: 011: 100: 101: 110: 111:	39/fcgck 78/fcgck 156/fcgck 312/fcgck 624/fcgck 1248/fcgck Reserved Reserved

5-8-3. ADC start process

```
1   void ADCStart(void)
2   {
3           _asm(" SET (_ADCCR1).7 ");
4   }
```

[line 3 : Start the AD conversion]

5-8-4. ADC data get process

```
1   void ADCGetData(void)
2   {
3           UINT8_t result1;
4           UINT8_t result2;
5
6           result1 = ADCDRL;
7           result2 = ADCDRH;
8           result2 = result2 << 6;
9           result1 = result1 >> 2;
10          result2 = result2 + result1;          /* output the top 8 bits in 10 bits */
11          PBDR = result2;
12  }
```

[line 6-7 : Get the conversion data]

5-8-5. ADC stop process

```
1   void ADCStop(void)
2   {
3           ADCCR1 = 0x00;
4   }
```

[line 3 : Stop the AD conversion]

5-9. Timer for clock : [rtc.c]

5-9-1. Control process

```
 1  void sample_rtc(void)
 2  {
 3          fRTCCheck = FALSE;
 4          sRTCCnt = 10;
 5  
 6          RTCInit();
 7          RTCStart();
 8          while (fRTCCheck == FALSE)
 9          {
10                  WDCDR = 0x4E;                   /* Clear WDT counter */
11          }
12          RTCStop();
13  }
```

[line 6 : Initialize the RTC] ➡ 5-9-2

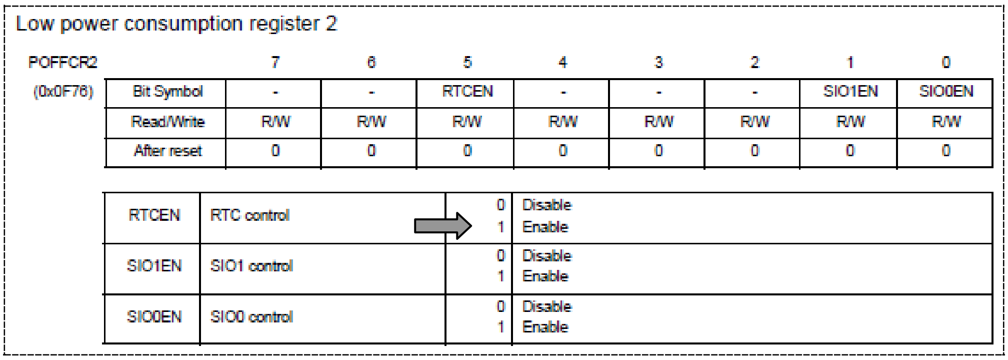
[line 7 : Start the RTC]➡ 5-9-3

[line 12 : Stop the RTC]➡ 5-9-4

5-9-2. Initialize process

```
 1  void RTCInit(void)
 2  {
 3          POFFCR2 = 0x20;
 4          _DI();
 5          EIRH = EIRH | 0x08;
 6          _EI();
 7  
 8          P8CR = 0x10;
 9          P8DR = 0x10;                    /* P84 as the output */
10  
11          RTCCR = 0x02;                   /* 0.5s cycle */
12  }
```

[line 3 : Enable the RTC power]

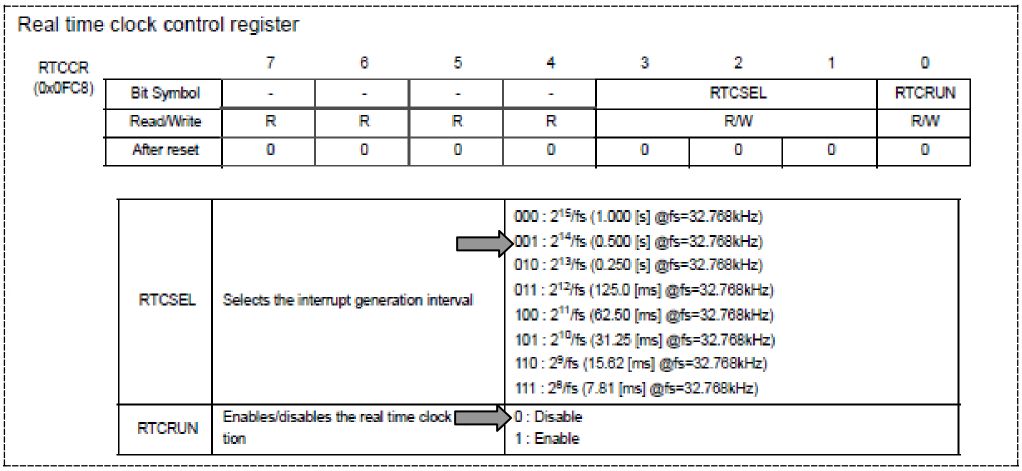


Low power consumption register 2

POFFCR2 (0x0F76)	7	6	5	4	3	2	1	0
Bit Symbol	-	-	RTCEN	-	-	-	SIO1EN	SIO0EN
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
After reset	0	0	0	0	0	0	0	0

RTCEN	RTC control	0 1	Disable Enable
SIO1EN	SIO1 control	0 1	Disable Enable
SIO0EN	SIO0 control	0 1	Disable Enable

[line 11 : Set the RTC period (0.5s)]



Real time clock control register

RTCCR (0x0FC8)

	7	6	5	4	3	2	1	0
Bit Symbol	-	-	-	-	RTCSEL			RTCRUN
Read/Write	R	R	R	R	R/W			R/W
After reset	0	0	0	0	0	0	0	0

RTCSEL	Selects the interrupt generation interval	000 : 2^{15}/fs (1.000 [s] @fs=32.768kHz) 001 : 2^{14}/fs (0.500 [s] @fs=32.768kHz) 010 : 2^{13}/fs (0.250 [s] @fs=32.768kHz) 011 : 2^{12}/fs (125.0 [ms] @fs=32.768kHz) 100 : 2^{11}/fs (62.50 [ms] @fs=32.768kHz) 101 : 2^{10}/fs (31.25 [ms] @fs=32.768kHz) 110 : 2^{9}/fs (15.62 [ms] @fs=32.768kHz) 111 : 2^{8}/fs (7.81 [ms] @fs=32.768kHz)
RTCRUN	Enables/disables the real time clock tion	0 : Disable 1 : Enable

5-9-3. Timer start process

```
1   void RTCStart(void)
2   {
3           RTCCR = 0x03;
4   }
```

[line 3 : Start the timer]

5-9-4. Timer stop process

```
1   void RTCStop(void)
2   {
3           RTCCR = 0x02;
4   }
```

[line 3 : Stop the timer]

5-9-5. Interrupt process

```
void _interrupt IntRTC(void)
{
        static UINT8_t i = 0;

        i++;
        if (i >= sRTCCnt)
        {
                fRTCCheck = TRUE;
                i = 0;
        }
        else
        {
                P84 = ~P84;
        }
}
```