CANTUS-CAN

- *UART* -

32bits EISC Microprocessor *CANTUS*

Ver 1.1
April 24, 2013

Advanced Digital Chips Inc.

History

2013-02-19	Released	
2013-04-24	Modified	CANTUS-CAN

CANTUS-CAN Application Note : #0005B UART

Office

(Gwanyang-dong, Keumkang Penterium IT Tower) 22F, A-Tower,
282, Hagui-ro, Dongan-gu, Anyang-si, Gyeonggi-do, SEOUL 431-810 Rep. of KOREA

Tel : +82-31-463-7500
Fax : +82-31-463-7588
URL : http://www.adc.co.kr

— Table of Contents —

— List of Figures —

— List of Tables —

1 Summary

이 문서는 CANTUS의 UART에 대한 Application Note이다.

CANTUS는 독립적인 송/수신이 가능한 16Bytes의 FIFO가 포함된 8채널의 UART를 가지고 있다. FIFO Mode는 선택적으로 사용할 수 있으며, UART는 수신상태(Receiver Line Status), 수신(Received Data Available), 송신가능(Transmit Holding Empty)에 따라 Core에 Interrupt Request를 발생한다.

이 문서는 CANTUS의 UART로 Data를 송/수신하기 위한 방법을 기술 한다.

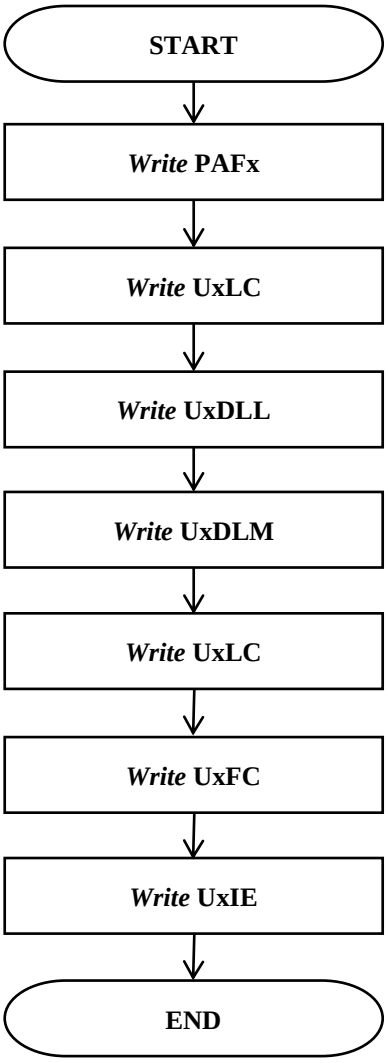
CANTUS의 UART는 CANTUS Datasheet '13 UART'를 참조 하라.

2 UART Initialize

2.1 Register Set

CANTUS의 UART를 송/수신으로 사용하기 위해 먼저 다음과 같은 순서로 Register를 설정한다.

그림 2-1 UART Initialize Register Set



PAFx

해당 Port의 Pin을 UART로 사용하기 위해 '00b'을 쓴다. Port의 Pin의 Alternation은 해당 bit(2)에 쓰는 값으로 설정 된다.

표 2-1 Port Alternate Function 4 Register

Group	Index	Pin	1st	2nd	3rd	4th(default)
			00	01	10	11
PAF4 0x80020030	0	26	TX[0]		I2S_SDO	P4.0
	1	27	RX[0]		I2S_SDI	P4.1
	2	28	TX[1]	SPI_MOSI	SDCD_DATA[3]	P4.2
	3	29	RX[1]	SPI_MISO	SDCD_DATA[2]	P4.3
	4	30	TX[2]	SPI_SCK	SDCD_CLK	P4.4
	5	31	RX[2]		SDCD_CMD	P4.5
	6	32	TX[3]		TWI_SCL	P4.6
	7	33	RX[3]		TWI_SDA	P4.7

UxLC (UART Channel Line Control Register)

Parity, Stop bit, Data length를 설정한다. 또한 UxDLL/DLM을 설정하기 위해 DLAB를 '1b'로 쓰며, 설정 후 반드시 '0b'로 써야 한다.

표 2-2 UART Channel Line Control Register (UxLC)

Bit	R/W	Description	Default Value
31 : 8	R	Reserved.	-
7	RW	DLAB : Divisor Latch Access Bit DLAB이 "1" 일 때, Divisor Latch Registers의 Read/Write와 FIFO Control Register의 Read가 가능하다.	0
6	RW	SB : Set Break SB가 "1" 일 때, Serial Data Output에 Logic "0"이 출력된다. SB는 내부 Transmitter Logic에는 영향을 미치지 않으며, 단지 Serial Output에만 영향을 미친다.	0
5	RW	SP : Stick Parity 0 : Disables Stick Parity 1 : PEN, EPS, SP가 "1"일 때, Parity Bit "0" PEN, SP가 "1"이고, EPS가 "0" 일 때, Parity Bit "1"	0
4	RW	EPS : Even Parity Select 0 : Select Odd Parity 1 : Select Even Parity	0
3	RW	PEN : Parity Enable Bit 0 : Disables Parity 1 : Enables Parity	0
2	RW	STB : Number of Stop Bit 0 : 1 Stop bit 1 : 2 Stop bits(만약, WLS Bit에서 5 Bits/Character를 선택했다면, 1.5 Stop bits 을 갖는다.)	0
1 : 0	RW	WLS : Word Length Select 00 : 5 Bits/Character 01 : 6 Bits/Character 10 : 7 Bits/Character 11 : 8 Bits/Character	00

UxDLL (UART Channel Divisor Latch LSB Register)

UART Divisor Latch Value의 하위 8bit 값으로 UART Baud Rate를 설정 한다.

표 2-3 UART Channel Divisor latch LSB Register (UxDLL)

Bit	*R/W*	*Description*	*Default Value*
31: 8	R	Reserved.	-
7 : 0	RW	Divisor Latch Least Significant Byte	0x00

*** DLAB가 "1" 일 때 Access 가능하다.

UxDLM (UART Channel Divisor Latch MSB Register)

UART Divisor Latch Value의 상위 8bit 값으로 UART Baud Rate를 설정 한다.

표 2-4 UART Channel Divisor Latch MSB Register (UxDLM)

Bit	*R/W*	*Description*	*Default Value*
31: 8	R	Reserved.	-
7 : 0	RW	Divisor Latch Most Significant Byte	0x00

*** DLAB가 "1" 일 때 Access 가능하다.

UART Baud Rate

UART의 Baud Rate는 아래와 같은 식을 따른다.

$$UART\ \ Baud\ \ Rate = \frac{f_{PCLK}}{16 \times UDL}$$

UART Divisor Latch Value (UDL) = UDLM[7:0] << 8 + UDLL[7:0]

표 2-5 UART Baud Rate

f_{PCLK} *(MHz)*		*1.024*	*2.048*	*5.6448*	*11.2896*	*24.0*	*48.0*
2400 bps	UDL	27	53	147	294	625	1250
	ERR(%)	1.23	0.63	0.00	0.00	0.00	0.00
4800 bps	UDL	-	27	74	147	313	625
	ERR(%)	-	1.23	0.68	0.00	0.16	0.00
9600 bps	UDL	-	-	37	74	156	313
	ERR(%)	-	-	0.68	0.68	0.16	0.16
14400 bps	UDL	-	9	25	49	104	208
	ERR(%)	-	1.23	2.00	0.00	0.16	0.16
19200 bps	UDL	-	-	18	37	78	156
	ERR(%)	-	-	2.08	0.68	0.16	0.16
38400 bps	UDL	-	-	9	18	39	78
	ERR(%)	-	-	2.08	2.08	0.16	0.16
57600 bps	UDL	-	-	6	12	26	52
	ERR(%)	-	-	2.08	2.08	0.16	0.16
115200bps	UDL	-	-	3	6	13	26
	ERR(%)	-	-	2.08	2.08	0.16	0.16

*** ERR 이 2.2% 이상에서는 UART 동작의 안정성을 보장 받을 수 없다.

- f_{PCLK}

  APB영역에 공급되는 Clock이다. CANTUS Datasheet '4 CLOCKS AND POWER MANAGEMENT'을 참조하라.

UxFC (UART Channel FIFO Control Register)

FIFO Mode에서 Receiver FIFO Trigger Level을 설정하며, 필요에 따라 FIFO를 사용하지 않고, 16450 UART Mode로 설정한다.

표 2-6 UART Channel FIFO Control Register (UxFC)

Bit	*R/W*	*Description*	*Default Value*
31 : 8	R	Reserved.	-
7 : 6	RW	RFTL : Receiver FIFO Trigger Level 00 : 1 Byte 01 : 4 Byte 10: 8 Byte 11 : 14 Byte	00
5 : 3	R	Reserved	-
2	RW	XFR : XMIT FIFO Reset XFR가 "1" 일 때, XMIT FIFO 내의 모든 데이터는 Reset 된다. 그러나 Shift Register 내의 데이터는 Reset 되지 않는다.	0
1	RW	RFR : RCVR FIFO Reset RFR가 "1" 일 때, RCVR FIFO 내의 모든 데이터는 Reset 된다, 그러나 Shift Register 내의 데이터는 Reset 되지 않는다.	0
0	RW	FIFOEN : FIFO Enable Bit 0 : 16450 UART Mode 1 : Enables FIFO	0

*** DLAB가 "0" 일 때는 Write Mode 이고, DLAB가 "1" 일 때는 Read Mode 이다.

UxIE (UART Channel Interrupt Enable Register)

UART Interrupt를 사용하기 위해서 UxIE를 설정한다.

표 2-7 UART Channel Interrupt Enable Register (UxIE)

Bit	*R/W*	*Description*	*Default Value*
31: 3	R	Reserved.	-
2	RW	RLSIEN : Receiver Line Status Interrupt Enable bit 0 : Disable 1 : Enable	0
1	RW	THEIEN : Transmitter Holding Empty Interrupt Enable bit 0 : Disable 1 : Enable	0
0	RW	RDAIEN : Received Data Available Interrupt Enable bit 0 : Disable 1 : Enable	0

2.2 Function Set

아래는 CANTUS의 Port 4의 0, 1 Pin을 UART TX/RX로 설정하고, UART는 None Parity, 1 Stop bit, 8 bits/Character로 설정한 예이다. APB Clock 48MHz에서 115200 Bps이다.

```
// #define UART_FIFO_USE
// #define UART_INTERRUPT_ENABLE

void UART0_ISR(void)
{
}

main()
{
  …

  *R_PAF4 = F_PAF4_0_TX0 | F_PAF4_1_RX0;

  *R_U0LC = F_ULC_DLAB | F_ULC_WLS_8;
  *R_U0DLL = 0x1a;
  *R_U0DLM = 0;
  *R_U0LC &= ~(F_ULC_DLAB);

  #ifdef UART_FIFO_USE
  *R_U0FC = F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN;
  #else
  *R_U0FC = F_UFC_XFR | F_UFC_RFR;
  #endif

  #ifdef UART_INTERRUPT_ENABLE
  *R_U0IE = (F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN);
  setinterrupt(INTNUM_UART0, UART0_ISR);
  EnableInterrupt(INTNUM_UART0, TRUE);
  #else
  *R_U0IE &= ~(F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN);
  #endif

  …
}
```

***R_PAF4**

Port Alternate Function Register는 SDK/include/CANTUS/paf.h에

#define R_PAF4 ((volatile unsigned int*)0x80020030)
#define F_PAF4_0_TX0 (0 << 0)
#define F_PAF4_1_RX0 (0 << 2)

로 정의되어 있다.

▶ 여기서는 Port Alternate Function Register 4에 F_PAF4_0_TX0, F_PAF4_1_RX0를 대입하여 Port 4의 0번을 TX0로 1번을 RX0로 설정한다.

*R_U0LC

UART Channel Line Control Register는 SDK/include/CANTUS/uart.h에

```
#define R_U0LC ((volatile unsigned int*)0x8002140C)
#define F_ULC_DLAB                ( 1<< 7)
#define F_ULC_WLS_8               ( 3<< 0)
```

로 정의되어 있다.

▶ 여기서는 UART Channel Line Control Register에 (F_ULC_DLAB | F_ULC_WLS_8)을 대입하여 R_U0DLL과 R_U0DLM에 접근 가능토록 설정하고, UART를 None Parity, 1 Stop bit, 8 bits/Character로 설정한다. R_U0DLL과 R_U0DLM에 접근이 끝나면 R_U0LC의 DLAB를 0로 써야 이후 R_U0FC, R_U0IE, R_U0II, R_U0RB, R_U0TH에 접근할 수 있다.

*R_U0DLL

UART Channel Divisor Latch LSB Register는 SDK/include/CANTUS/uart.h에

```
#define R_U0DLL ((volatile unsigned int*)0x80021400)
```

로 정의되어 있다.

▶ 여기서는 UART Channel Divisor Latch LSB Register에 UART Baud Rate 설정을 위한 UDL의 하위 8 bits를 쓴다.

*R_U0DLM

UART Channel Divisor Latch MSB Register는 SDK/include/CANTUS/uart.h에

```
#define R_U0DLM ((volatile unsigned int*)0x80021404)
```

로 정의되어 있다.

▶ 여기서는 UART Channel Divisor Latch MSB Register에 UART Baud Rate 설정을 위한 UDL의 상위 8 bits를 쓴다.

*R_U0FC

UART Channel FIFO Control Register는 SDK/include/CANTUS/uart.h에

```
#define R_UART_FC7 ((volatile unsigned int*)0x80021408)
#define F_UFC_XFR                 ( 1<< 2)
#define F_UFC_RFR                 ( 1<< 1)
#define F_UFC_FIFOEN              ( 1<< 0)
```

로 정의되어 있다.

▶ 여기서는 UART Channel FIFO Control Register에 (F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN)을 대입하여 RCVR FIFO와 XMIT FIFO를 Reset시키고 FIFO Enable Mode로 설정한다.

*R_U0IE

UART Channel Interrupt Enable Register는 SDK/include/CANTUS/uart.h에

```
#define R_U0IE ((volatile unsigned int*)0x80021404)
#define F_UIE_RLSIEN                    ( 1<< 2)
#define F_UIE_THEIEN                    ( 1<< 1)
#define F_UIE_RDAIEN                    ( 1<< 0)
```

로 정의되어 있다.

▶ 여기서는 UART Channel Interrupt Enable Register에 (F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN)를 대입하여 Receiver Line Status Interrupt, Transmitter Holding Empty Interrupt, Received Data Available Interrupt를 Enable로 설정하거나 ~(F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN)를 대입하여 Disable로 설정한다. UART Interrupt가 발생하면 UxII를 읽어 Interrupt Type을 판단할 수 있다.

setinterrupt()

setinterrupt()는 SDK/include/CANTUS /interrupt.h에

```
BOOL setinterrupt(INTERRUPT_TYPE intnum, void (*fp)());
```

로 선언되어 있고,

SDK/Library/interrupt.c에

```
BOOL setinterrupt (INTERRUPT_TYPE intnum, void (*fp)())
{
        if (intnum >= INTNUM_MAX)
                return FALSE;

        UserVector_table[intnum]=fp;
        return TRUE;

}
```

로 정의되어 있다.

▶ interrupt에 관련된 내용은 AN_2004_INTERRUPT를 참조 하라.

EnableInterrupt()

EnableInterrupt()는 SDK/include/CANTUS/interrupt.h에

void EnableInterrupt(INTERRUPT_TYPE num, BOOL b);

로 선언되어 있고,

SDK/Library/interrupt.c에

```
void EnableInterrupt(INTERRUPT_TYPE num, BOOL b)
{
	CRITICAL_ENTER();
	if (!b) //disable
	{
		*R_INTEN &= (~(1 << num));
		*R_INTMASKCLR |= ( 1 << num);
	}
	else
	{
		*R_INTMASKSET = (1 << num);
		*R_INTEN |= ( 1 << num);
	}
	CRITICAL_EXIT();
}
```

로 정의되어 있다.

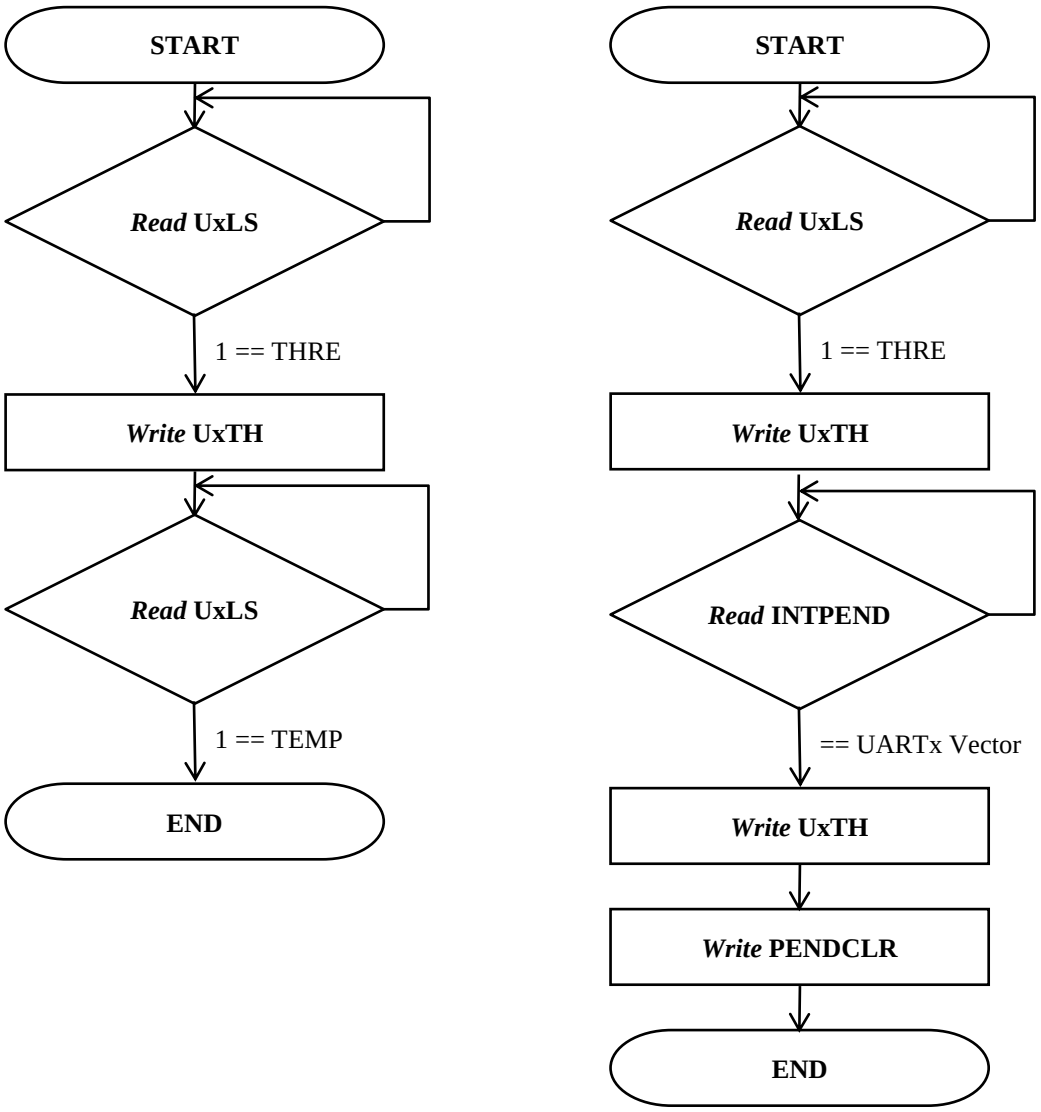
▶ interrupt에 관련된 내용은 AN_1004_INTERRUPT를 참조 한다.

3 UART Transmit

3.1 Register Set

CANTUS의 UART 전송은 UxLS의 THRE가 Set 되어 있는 상태에서, UxTH에 Data를 씀으로써 이루어 진다. UxIE.THEIEN이 설정되어 있다면 Interrupt를 사용할 수 있다. 이를 위한 Register설정은 다음과 같다.

그림 3-1 UART Data Transmission Register Set



UxLS (UART Channel Line Status Register)

UxLS를 읽어 UART의 상태를 판단한다.
THRE는 FIFO Mode가 아닌 경우 THR의 Data가 TSR로 전송되어 비어 있음을 의미 한다. THRE가 Set되어 있을 때 송신할 Data를 UxTH에 쓴다. FIFO Mode에서는 Transmitter FIFO의 마지막 Data가 TSR로 옮겨 져 비어 있음을 나타낸다.

TEMP는 FIFO Mode가 아닌 경우 THR의 Data가 TSR을 통해 상대방에 송신되어 THR과 TSR이 비어 있음을 나타낸다. FIFO Mode에서는 Transmitter FIFO의 Data가 TSR을 통해 상대방에 송신되어 Transmitter FIFO와 TSR이 비어 있음을 나타낸다.

UxTH (UART Channel Transmitter Holding Register)

UART로 전송할 Data를 쓴다. FIFO Enable이면 Transmitter FIFO에 써지며, 이 Data는 TSR을 통해 전송된다.
UART 전송에 Interrupt를 사용할 경우, Interrupt Request가 발생하면 FIFO Size만큼 UxTH에 전송할 Data를 쓴다.

표 3-1 UART Channel Transmitter Holding Register (UxTH)

Bit	*R/W*	*Description*	*Default Value*
31: 8	R	Reserved.	-
7 : 0	R	Transmit Holding Data	-

*** DLAB가 “0” 일 때 Access 가능하다.

표 3-2 UART Channel Line Status Register (UxLS)

Bit	*R/W*	*Description*	*Default Value*
31 : 8	R	Reserved.	-
7	R	EIRF : Error in RCVR FIFO FIFO 모드가 아닌 경우 EIRF는 항상 "0"이다. FIFO 모드에서 EIRF는 RCVR FIFO 내에서 OE, PE, FE, BI 중 어느 하나라도 "1"이 설정되면, "1"이 된다. EIRF는 만약 FIFO 내에 연속적인 에러가 없다면, LSR 레지스터를 읽었을 때 Clear("0")된다.	0
6	R	TEMP : Transmitter Empty FIFO 모드가 아닌 경우 TEMP는 Transmitter Holding Register (THR)와 Transmitter Shift Register(TSR)이 모두 Empty일 때 "1"이 된다. THR 또는 TSR에 데이터가 있으면 Clear된다. FIFO 모드 에서는, TEMP는 Transmitter FIFO와 TSR이 모두 Empty일 때 "1"이 된다.	1
5	R	THRE : Transmitter Holding Register Empty FIFO 모드가 아닌 경우 THRE는 THR의 데이터가 TSR로 전송 되어 Empty가 되었을 때 "1"이 되며, THR에 전송을 위한 새로운 데이터를 쓸 수 있다. FIFO 모드에서는 Transmit FIFO가 Empty일 때 THRE가 "1"이 되며, 적어도 하나의 Byte라도 Transmit FIFO에 써지면 Clear된다. 만약 THRE interrupt(ETHREI) 가 "1"이고 THRE가 "1"이라면 Interrupt가 발생한다.	1
4	R	BINT : Break Interrupt : 수신되는 입력 데이터가 Full-word 전송 시간 동안 "0"일 때 BI는 "1"이 된다. Full-word 전송 시간은 Start, Data, Parity 그리고 Stop 비트 전송을 위한 전체 시간을 의미한다. FIFO 모드에서 이 에러는 FIFO 내의 각각의 Byte에 적용되며, BI가 발생했을 때 FIFO에는 "0"이 써진다. CPU가 LSR을 읽어 을 때 Clear 된다.	0
3	R	FERR : Framing Error FE는 수신되는 입력 데이터가 유효한 Stop 비트를 가지지 않았을 때 "1"이 된다. FIFO 모드에서 이 에러는 FIFO 내의 각각의 Byte에 적용된다. CPU가 LSR을 읽어 을 때 Clear 된다.	0
2	R	PERR : Parity Error PE는 수신되는 입력 데이터가 LCR 레지스터에 의해 선택된 Parity 비트와 같지 않을 때 "1"이 된다. FIFO 모드에서 이 에러는 FIFO 내의 각각의 Byte에 적용된다. CPU가 LSR을 읽어 을 때 Clear 된다.	0
1	R	OERR : Overrun Error OE는, FIFO 모드가 아닌 경우, RBR 내의 데이터를 읽어가기 전에 새로운 데이터가 써진 경우 "1"이 된다. FIFO 모드에서는 FIFO가 Full 상태에서 Receiver Shift Register(RSR)에 새로운 Full-word가 들어왔을 때 "1"이 된다. 이 경우 RSR은 새로운 데이터로 계속 갱신이 되지만, FIFO로 전송은 되지 않는다. CPU가 LSR을 읽어 을 때 Clear 된다.	0
0	R	DRDY : Data Ready DR은 수신된 데이터가 RBR 또는 FIFO에 써졌을 때 "1"이 된다. RBR 또는 FIFO 내의 모든 데이터가 CPU에 의해 읽혀졌을 때 Clear된다.	0

INTPEND (Interrupt Pending Register)

INTEN에 설정된 Interrupt Vector의 Interrupt Event를 기록한다.

표 3-3 Interrupt Pending Register (INTPEND)

Bit	*R/W*	*Description*	*Default Value*
31	R	Vector No. 0x3F Interrupt Pending bit	-
30	R	Vector No. 0x3E Interrupt Pending bit	-
29	R	Vector No. 0x3D Interrupt Pending bit	-
28	R	Vector No. 0x3C Interrupt Pending bit	-
27	R	Vector No. 0x3B Interrupt Pending bit	-
26	R	Vector No. 0x3A Interrupt Pending bit	-
25	R	Vector No. 0x39 Interrupt Pending bit	-
24	R	Vector No. 0x38 Interrupt Pending bit	-
23	R	Vector No. 0x37 Interrupt Pending bit	-
22	R	Vector No. 0x36 Interrupt Pending bit	-
21	R	Vector No. 0x35 Interrupt Pending bit	-
20	R	Vector No. 0x34 Interrupt Pending bit	-
19	R	Vector No. 0x33 Interrupt Pending bit	-
18	R	Vector No. 0x32 Interrupt Pending bit	-
17	R	Vector No. 0x31 Interrupt Pending bit	-
16	R	Vector No. 0x30 Interrupt Pending bit	-
15	R	Vector No. 0x2F Interrupt Pending bit	-
14	R	Vector No. 0x2E Interrupt Pending bit	-
13	R	Vector No. 0x2D Interrupt Pending bit	-
12	R	Vector No. 0x2C Interrupt Pending bit	-
11	R	Vector No. 0x2B Interrupt Pending bit	-
10	R	Vector No. 0x2A Interrupt Pending bit	-
9	R	Vector No. 0x29 Interrupt Pending bit	-
8	R	Vector No. 0x28 Interrupt Pending bit	-
7	R	Vector No. 0x27 Interrupt Pending bit	-
6	R	Vector No. 0x26 Interrupt Pending bit	-
5	R	Vector No. 0x25 Interrupt Pending bit	-
4	R	Vector No. 0x24 Interrupt Pending bit	-
3	R	Vector No. 0x23 Interrupt Pending bit	-
2	R	Vector No. 0x22 Interrupt Pending bit	-
1	R	Vector No. 0x21 Interrupt Pending bit	-
0	R	Vector No. 0x20 Interrupt Pending bit	-

*** Interrupt Pending Register의 각 비트의 값은 해당 인터럽트가 발생하였음을 나타낸다.

Interrupt Pending Register의 값은 Interrupt Pending Clear 레지스터에 의해 Clear된다.

일반적으로 해당 Interrupt가 끝날 때 Clear한다.

PENDCLR (Interrupt Pending Clear Register)

Interrupt Vector Number값으로 Write하여 INTPEND의 Event를 Clear한다. INTPEND를 Clear하지 않으면 더 이상 Core에 Interrupt Request를 발생하지 않는다.

표 3-4 Interrupt Pending Register (PENDCLR)

Bit	*R/W*	*Description*	*Default Value*
31 : 8	R	Reserved	-
7 : 0	W	Interrupt Pending Register Clear Value (0x20 ~ 0x3F)	0xFF

*** Interrupt Pending Register를 Clear 하기 위해서는 Interrupt Vector No. 값으로 Write 한다.

3.2 Function Set 1

아래는 UART Data 전송의 예이다. U0LS.THRE가 Set되면 U0TH에 전송할 Data를 쓴다. Data 전송이 완료 되면 U0LS.TEMP가 Set된다.

```
// #define UART_FIFO_USE

main()
{
  …

  *R_PAF4 = F_PAF4_0_TX0 | F_PAF4_1_RX0;

  *R_U0LC = F_ULC_DLAB | F_ULC_WLS_8;
  *R_U0DLL = 0x1a;
  *R_U0DLM = 0;
  *R_U0LC &= ~(F_ULC_DLAB);

  #ifdef UART_FIFO_USE
  *R_U0FC = F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN;
  #else
  *R_U0FC = F_UFC_XFR | F_UFC_RFR;
  #endif

  *R_U0IE &= ~(F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN);

  U8 uart_data_tx[] = {"ABCDEFGHIJKLMNOPQRSTUVWXYZ\r\n"};
  U32 i=0,j;
  U32 data_tx_length;

  while(1)
  {
    if(*R_U0LS & F_ULS_THRE)
    {
      #ifdef UART_FIFO_USE
      if(sizeof(uart_data_tx) < UART_FIFO_DEPTH)
      {
        for(i=0;i<sizeof(uart_data_tx);i++)
        {
          *R_U0TH = uart_data_tx[i];
        }
        while(!(*R_U0LS & F_ULS_TEMP));
      }
      else
      {
        data_tx_length = sizeof(uart_data_tx);
        i=0;
        do
        {
          if(data_tx_length > UART_FIFO_DEPTH)
          {
            for(j=0;j<UART_FIFO_DEPTH;j++)
            {
              *R_U0TH = uart_data_tx[i++];
            }
            data_tx_length -= UART_FIFO_DEPTH;
          }
          else
```

```
          {
            for(j=0;j<data_tx_length;j++)
            {
              *R_U0TH = uart_data_tx[i++];
            }
            data_tx_length = 0;
          }
          while(!(*R_U0LS & F_ULS_TEMP));
        }while(data_tx_length!=0);
      }
      #else
      // 16450 UART Mode
      i=0;
      do
      {
        *R_U0TH = uart_data_tx[i++];
        while(!(*R_U0LS & F_ULS_TEMP));
      }while(i<sizeof(uart_data_tx));
      #endif
      delayms(500);
    }
  }

  …
}
```

*R_U0LS

UART Channel Line Status Register는 SDK/include/CANTUS/uart.h에

#define R_U0LS ((volatile unsigned int*)0x80021414)
#define F_ULS_THRE (1<< 5)
#define F_ULS_TEMP (1<< 6)

로 정의되어 있다.

▶ 여기서는 R_U0LS를 F_ULS_THRE와 비교하여 Transmitter Holding Register Empty 일 때 전송할 Data를 쓰고, F_ULS_TEMP와 비교하여 Transmitter Empty일 때까지 기다린다.

*R_U0TH

UART Channel Transmitter Holding Register는 SDK/include/CANTUS/uart.h에

#define R_U0TH ((volatile unsigned int*)0x80021400)

로 정의되어 있다.

▶ 여기서는 R_U0TH에 전송할 Data를 쓴다. FIFO Enable Mode에서는 FIFO Size만큰 Data를 쓴다.

3.3 Function Set 2

아래는 Interrupt를 사용한 UART Data 전송의 예이다. U0LS.THRE가 Set되면 UART Interrupt를 Enable하여 Interrupt Request가 발생하면 U0TH에 전송할 Data를 쓴다.

```
// #define UART_FIFO_USE
volatile U8 uart_data_tx[] = {"ABCDEFGHIJKLMNOPQRSTUVWXYZ\r\n"};
volatile U32 i=0,j;
volatile U32 data_tx_length;

void UART0_ISR()
{
  U32 status;
  status = (*R_U0II & F_UII_INTID);

  if((F_UII_INTID_THR & status) == F_UII_INTID_THR )
  {
    #ifdef UART_FIFO_USE
    if(sizeof(uart_data_tx) < UART_FIFO_DEPTH)
    {
      for(i=0;i<sizeof(uart_data_tx);i++)
      {
        *R_U0TH = uart_data_tx[i];
      }
    }
    else
    {
      if(data_tx_length > UART_FIFO_DEPTH)
      {
        for(j=0;j<UART_FIFO_DEPTH;j++)
        {
          *R_U0TH = uart_data_tx[i++];
        }
        data_tx_length -= UART_FIFO_DEPTH;
      }
      else
      {
        for(j=0;j<data_tx_length;j++)
        {
          *R_U0TH = uart_data_tx[i++];
        }
        data_tx_length = 0;
        EnableInterrupt(INTNUM_UART0,FALSE);
      }
    }
    #else
    *R_U0TH = uart_data_tx[i++];
    if(i==sizeof(uart_data_tx))
    {
      i=0;
      EnableInterrupt(INTNUM_UART0,FALSE);
    }
    #endif
  }
}
```

```
main()
{
  …
  *R_PAF4 = F_PAF4_0_TX0 | F_PAF4_1_RX0;

  *R_U0LC = F_ULC_DLAB | F_ULC_WLS_8;
  *R_U0DLL = 0x1a;
  *R_U0DLM = 0;
  *R_U0LC &= ~(F_ULC_DLAB);

  #ifdef UART_FIFO_USE
  *R_U0FC = F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN;
  #else
  *R_U0FC = F_UFC_XFR | F_UFC_RFR;
  #endif

  *R_U0IE = (F_UIE_RLSIEN | F_UIE_THEIEN);
  setinterrupt(INTNUM_UART0, UART0_ISR);

  while(1)
  {
    if(*R_U0LS & F_ULS_THRE)
    {
      #ifdef UART_FIFO_USE
      i=0;
      data_tx_length = sizeof(uart_data_tx);
      EnableInterrupt(INTNUM_UART0,TRUE);
      #else
      // 16450 UART Mode
      i=0;
      EnableInterrupt(INTNUM_UART0,TRUE);
      #endif
    }
  }
  …
}
```

*R_U0II

UART Channel Interrupt Identification Register는 SDK/include/CANTUS/uart.h에

#define R_U0II ((volatile unsigned int*)0x80021408)
#define F_UII_INTID (15<< 0)
#define F_UII_INTID_THR (2<< 0)

로 정의되어 있다.

▶ 여기서는 R_U0II를 F_UII_INTID_THR과 비교하여 Transmitter Holding Register Empty 일 때 전송할 Data를 쓴다.

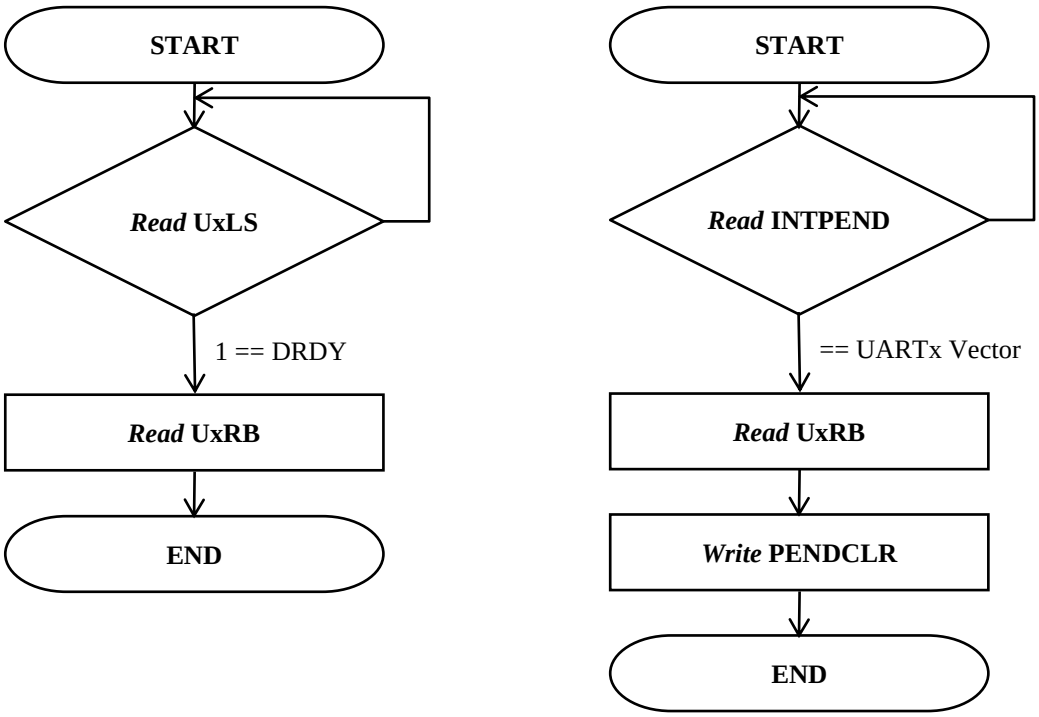
FIFO Enable이면 Transmitter FIFO에 써지며, FIFO Size만큼 UxTH에 전송할 Data를 쓴다

4 UART Receive

4.1 Register Set

CANTUS의 UART Data 수신은 UxLS의 DRDY가 Set 되어 있는 상태에서, UxRB를 읽음으로써 이루어 진다. UxIE.RDAIEN이 설정되어 있다면 Interrupt를 사용할 수 있다. 이를 위한 Register설정은 다음과 같다.

그림 4-1 UART Data Receiving Register Set



UxLS (UART Channel Line Status Register)

UxLS를 읽어 UART의 상태를 판단한다.

DRDY는 수신된 데이터가 UxRB에 있음을 의미한다. FIFO mode에서는 Receiver FIFO의 모든 Data를 읽으면 Clear 된다.

UxRB (UART Channel Receiver Buffer Register)

UART로 수신한 Data를 읽는다. FIFO Enable이면 Receiver FIFO에서 읽게 되며, UxLS.DRDY가 Clear될 때까지 읽는다.

표 4-1 UART Channel Receiver Buffer Register (UxRB)

Bit	*R/W*	*Description*	*Default Value*
31: 8	R	Reserved.	-
7 : 0	R	Receive Buffer Data	-

*** DLAB가 “0” 일 때 Access 가능하다.

4.2 Function Set 1

아래는 UART Data 수신의 예이다. UxLS.DRDY가 Set되면 UxRB를 읽는다.

```
// #define UART_FIFO_USE

main()
{
  …

  *R_PAF4 = F_PAF4_0_TX0 | F_PAF4_1_RX0;

  *R_U0LC = F_ULC_DLAB | F_ULC_WLS_8;
  *R_U0DLL = 0x1a;
  *R_U0DLM = 0;
  *R_U0LC &= ~(F_ULC_DLAB);

  #ifdef UART_FIFO_USE
  *R_U0FC = F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN;
  #else
  *R_U0FC = F_UFC_XFR | F_UFC_RFR;
  #endif

  *R_U0IE &= ~(F_UIE_RLSIEN | F_UIE_THEIEN | F_UIE_RDAIEN);

  U8 uart_data_rx[255];
  U8 uart_data_rx_Wptr;
  U8 uart_data_rx_Rptr;

  while(1)
  {
    if(*R_U0LS & F_ULS_DRDY)
    {
      #ifdef UART_FIFO_USE
      uart_data_rx_Wptr=0;
      uart_data_rx_Rptr=0;
      do
      {
        uart_data_rx[uart_data_rx_Wptr++] = *R_U0RB;
      }while(*R_U0LS & F_ULS_DRDY);

      if(*R_U0LS & F_ULS_THRE)
      {
        do
        {
          *R_U0TH = uart_data_rx[uart_data_rx_Rptr++];
          while(!(*R_U0LS & F_ULS_TEMP));
        }while(uart_data_rx_Rptr<uart_data_rx_Wptr);
      }

      #else
      // 16450 UART Mode
      uart_data_rx[0] = *R_U0RB;

      if(*R_U0LS & F_ULS_THRE)
      {
        *R_U0TH = uart_data_rx[0];
        while(!(*R_U0LS & F_ULS_TEMP));
      }
```

```
      #endif
    }
  }
  …
}
```

***R_U0LS**

UART Channel Line Status Register는 SDK/include/CANTUS/uart.h에

#define R_U0LS ((volatile unsigned int*)0x80021414)
#define F_ULS_DRDY (1<< 0)

로 정의되어 있다.

▶ 여기서는 UART Channel Line Status Register를 F_ULS_DRDY와 비교하여 Data Ready 인지 판단한다.

***R_U0RB**

UART Channel Receiver Buffer Register는 SDK/include/CANTUS/uart.h에

#define R_U0RB ((volatile unsigned int*)0x80021400)

로 정의되어 있다.

▶ 여기서는 UART Channel Receiver Buffer Register에 수신된 Data를 읽는다. FIFO Enable이면 UxLS.DRDY가 0(b)가 될 때까지 Data를 읽는다.

4.3 Function Set 2

아래는 Interrupt를 사용한 UART Data 수신의 예이다. Interrupt Request가 발생하면 U0RB를 읽는다.

```
// #define UART_FIFO_USE
volatile U8 uart_data_rx[255];
volatile U8 uart_data_rx_Wptr=0;
volatile U8 uart_data_rx_Rptr=0;

void UART0_ISR()
{
  U32 status;
  status = (*R_U0II & F_UII_INTID);

  if((F_UII_INTID_RDA & status) == F_UII_INTID_RDA )
  {
    #ifdef UART_FIFO_USE
    do
    {
      uart_data_rx[uart_data_rx_Wptr++] = *R_U0RB;
    }while(*R_U0LS & F_ULS_DRDY);
    #else
    // 16450 UART Mode
    uart_data_rx[uart_data_rx_Wptr++] = *R_U0RB;
    #endif
  }
}
main()
{
  …
  *R_PAF4 = F_PAF4_0_TX0 | F_PAF4_1_RX0;

  *R_U0LC = F_ULC_DLAB | F_ULC_WLS_8;
  *R_U0DLL = 0x1a;
  *R_U0DLM = 0;
  *R_U0LC &= ~(F_ULC_DLAB);

  #ifdef UART_FIFO_USE
  *R_U0FC = F_UFC_XFR | F_UFC_RFR | F_UFC_FIFOEN;
  #else
  *R_U0FC = F_UFC_XFR | F_UFC_RFR;
  #endif

  *R_U0IE = (F_UIE_RLSIEN | F_UIE_RDAIEN);
  setinterrupt(INTNUM_UART0, UART0_ISR);
  EnableInterrupt(INTNUM_UART0, TRUE);

  while(1)
  {
    if(*R_U0LS & F_ULS_THRE)
    {
      while(uart_data_rx_Rptr!=uart_data_rx_Wptr)
      {
        *R_U0TH = uart_data_rx[uart_data_rx_Rptr++];
        while(!(*R_U0LS & F_ULS_TEMP));
      }
    }
  }
```

```
  …
}
```

*R_U0II

UART Channel Interrupt Identification Register는 SDK/include/CANTUS/uart.h에

```
#define R_U0II ((volatile unsigned int*)0x80021408)
#define F_UII_INTID                    (15<< 0)
#define F_UII_INTID_RDA                ( 4<< 0)
```

로 정의되어 있다.

▶ 여기서는 R_U0II를 F_UII_INTID_RDA와 비교하여 Receiver Data Available일 때 수신된 Data를 읽는다.

*R_U0RB

▶ 여기서는 UART Interrupt Request가 발생하면 UART Channel Receiver Buffer Register에 수신된 Data를 읽는다. FIFO Enable이면 UxLS.DRDY가 0(b)가 될 때까지 Data를 읽는다.