
임베디드시스템설계 실습 (5)

Embedded System Design

**Real-Time Computing and Communications Lab.
Hanyang University**

UART 설정

VPOS 커널을 포팅하기 위한 준비

1. 커널 컴파일 + 커널 이미지를 RAM에 적재
2. Startup code 작성
3. UART 설정
4. TIMER 설정
5. Hardware Interrupt Handler 구현
 - (1) UART Interrupt
 - (2) Timer Interrupt
6. Software Interrupt Entering/Leaving Routine 구현

목차

1. Memory-mapped I/O
2. UART
3. UART Register Setting
4. UART Data transmit & receive

MEMORY-MAPPED I/O

CPU의 I/O 장치 액세스

□ 입/출력 장치(I/O 장치)를 사용하려면

- I/O 장치의 메모리나 레지스터에 액세스해야 함
 - 레지스터나 메모리에 값을 쓰거나 읽음
 - I/O 장치에 데이터를 전달하거나 I/O 장치로부터 데이터를 받을 수 있음

□ I/O 장치로 인해 CPU는 2개의 주소 공간 필요

- I/O
- Memory

□ I/O 주소공간과 Memory 주소공간을 분리? 통합?

- 하나의 메모리 공간에 → Memory-mapped I/O
- 서로 다른 주소 공간에 → Port-mapped I/O

Port-mapped I/O

□ 정의

- **I/O-mapped I/O**
- 메모리와 **I/O**의 주소 공간을 분리하여 접근
- **I/O**의 주소 공간에 접근하기 위해 기계어 명령어가 따로 존재
 - IN, OUT 명령어(8085)
- 인텔의 마이크로프로세서(**x86** 등)에서 주로 사용

Memory-mapped I/O

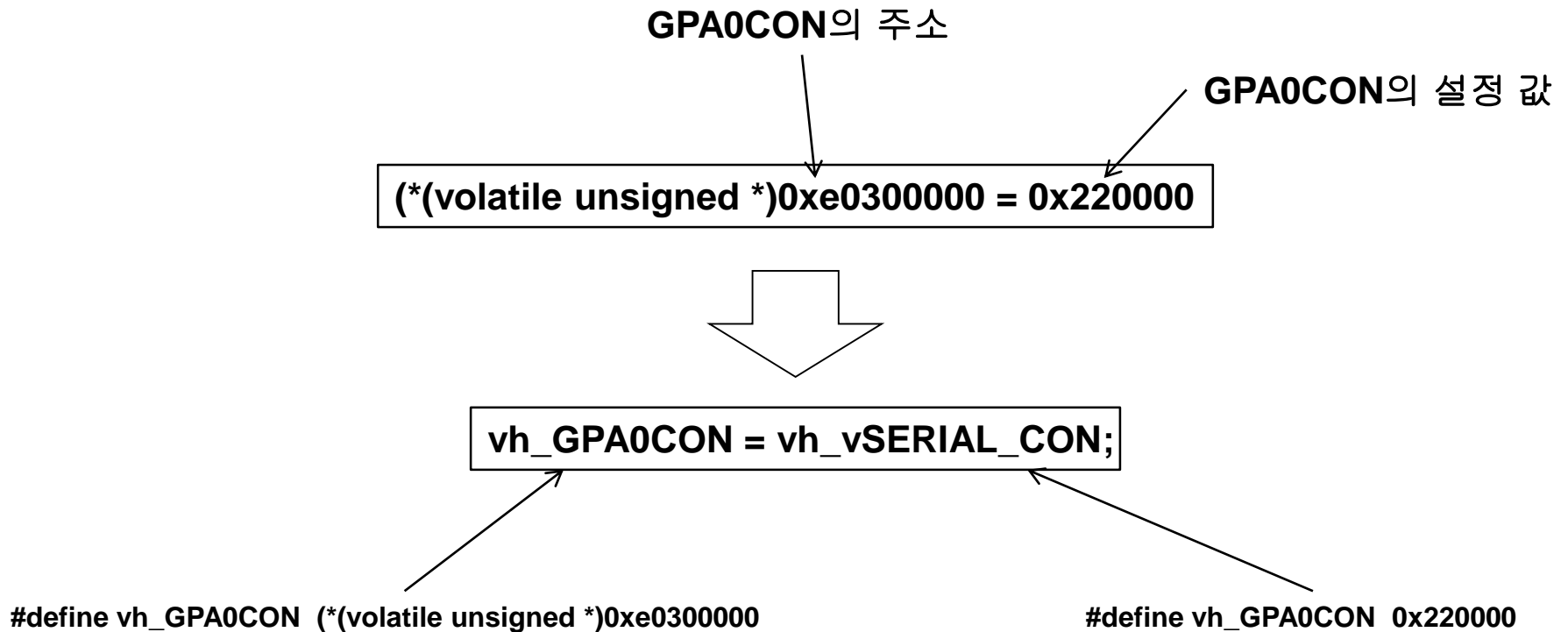
□ 정의

- 입출력과 메모리의 주소 공간을 분리하지 않고 하나의 메모리 공간에 배치
- 입출력 장치의 메모리나 레지스터를 전체 메모리의 일부로 취급
 - 전체 메모리의 특정 영역에 할당, 배치
- **CPU가 I/O의 레지스터에 접근할 때 메모리의 특정 주소로 접근**

□ 예시

- **ULCON1 : UART Line Control Register**
 - 메모리의 0xec000400 번지에 ULCON1 레지스터를 할당
 - 0xec000400에 데이터를 저장 → ULCON1에 데이터 저장
 - 0xec000400에서 데이터를 읽음 → ULCON1에서 데이터 읽음

Memory-mapped I/O : 예시



UART

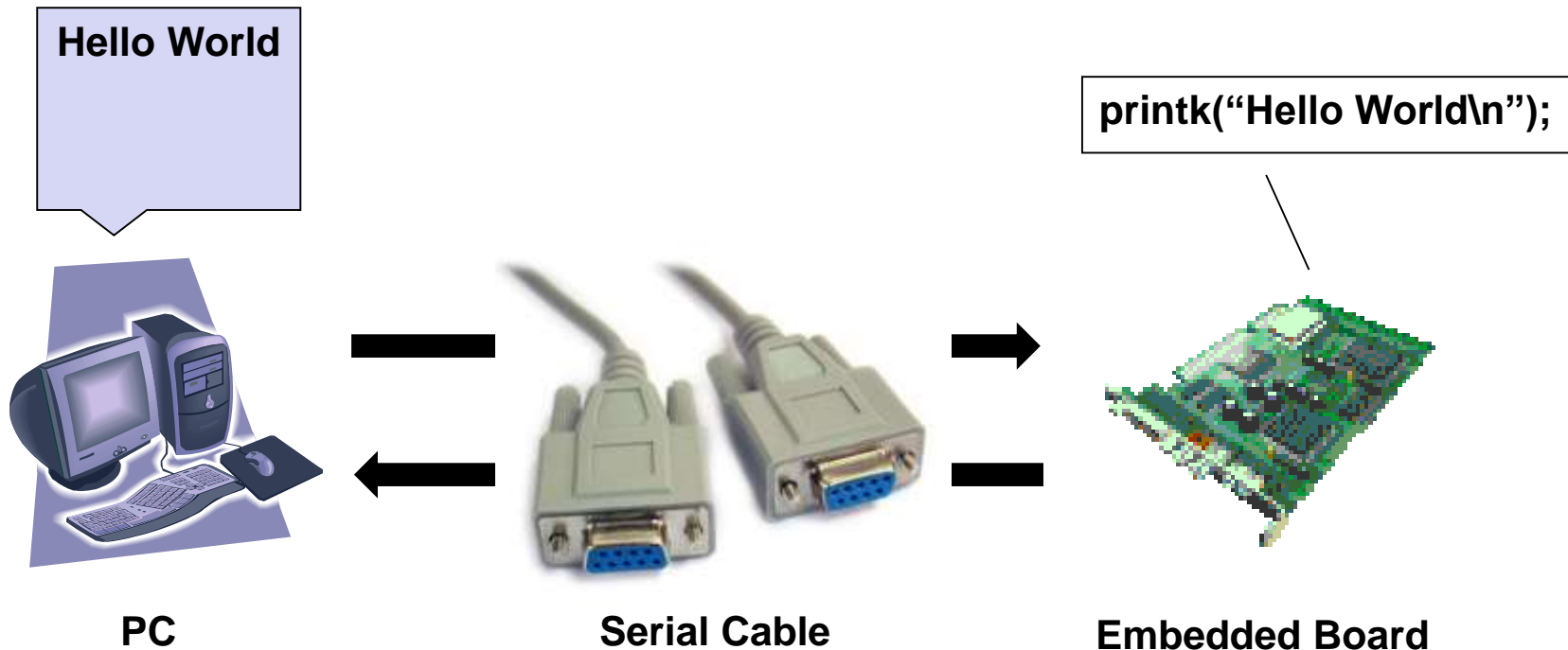
임베디드 보드와 PC의 데이터 교환

□ 보드 → PC

- 데이터를 받아 PC의 디스플레이에 출력

□ PC → 보드

- 키보드 입력을 받아 임베디드 보드에서 쉘 명령어 실행



통신 방법

□ 데이터의 전송 단위에 따른 분류

▪ 병렬 통신(**Parallel Communication**)

- 여러 개의 병렬 채널을 사용해 다수의 데이터 비트들을 동시에 송수신
- 컴퓨터 내부에서 정보 교환을 할 때 사용
- 고속의 통신속도, 한꺼번에 많은 정보를 처리
- 통신 거리의 제한, 구현의 어려움, 비용이 높음

▪ 직렬 통신(**Serial Communication**)

- 하나의 채널을 사용해 데이터 비트들을 1개의 비트단위로 외부와 송수신
- 컴퓨터가 외부의 장치와 통신할 때 사용
- 구현이 쉽고, 비용이 낮음
- 모뎀, LAN, RS-232 등

통신 방법

□ 직렬 통신의 분류

▪ 동기식(Synchronous) 통신

- 2개의 디바이스를 동기화하여 정해진 타이밍에 데이터를 송수신
- 데이터의 교환이 없어도 제어용의 신호가 흐르고 있어 상대와의 동기를 유지
- 미리 정해진 수의 데이터를 전송

▪ 비동기식(Asynchronous) 통신

- 디바이스끼리 타이밍을 맞추지 않고 송신 측에서 임의의 타이밍에 데이터를 송신
- 데이터를 송신할 때 데이터의 양단에 시작 비트(start bit)와 정지 비트(stop bit)를 첨가하고 휴지 기간을 가짐
- 데이터는 한 번에 한 문자씩 송수신 (5~8비트)
- 키보드 입력 같은 짧은 데이터 전송에 주로 사용

UART

□ 컴퓨터의 데이터 전송

- 컴퓨터 내부에서는 병렬 통신 방식으로 데이터를 전송
 - 병렬 전송은 아주 짧은 거리에서만 유효
- 컴퓨터 외부로 데이터를 전송할 때는 직렬 방식을 사용
- 데이터를 송신 **or** 수신할 때 신호를 바꿔줄 하드웨어 장치가 필요
 - 송신 : 병렬 신호 → 직렬 신호
 - 수신 : 직렬 신호 → 병렬 신호

□ UART (Universal Asynchronous Receiver Transmitter)

- 범용 비동기 송수신기 (비동기식 통신 컨트롤러)
- 데이터를 송신할 때, 병렬 형태의 비트를 직렬 형태의 비트로 변환
- 데이터를 수신할 때, 직렬 형태의 비트를 병렬 형태의 비트로 변환

UART의 기능

□ 병렬 신호를 직렬 신호로, 직렬 신호를 병렬 신호로 변환

□ 데이터를 전송할 때,

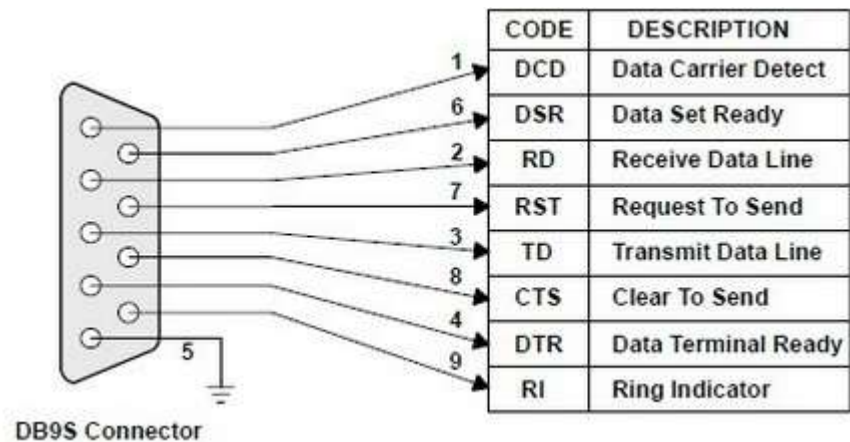
- 시작 비트와 정지 비트를 추가
 - 시작 비트를 통해 데이터의 수신을 확인
- 패리티 비트 추가
 - 수신측에서 패리티를 확인하여 데이터 오류를 감지

□ 키보드나 마우스로부터 들어오는 인터럽트를 처리

UART = RS-232?

□ RS-232

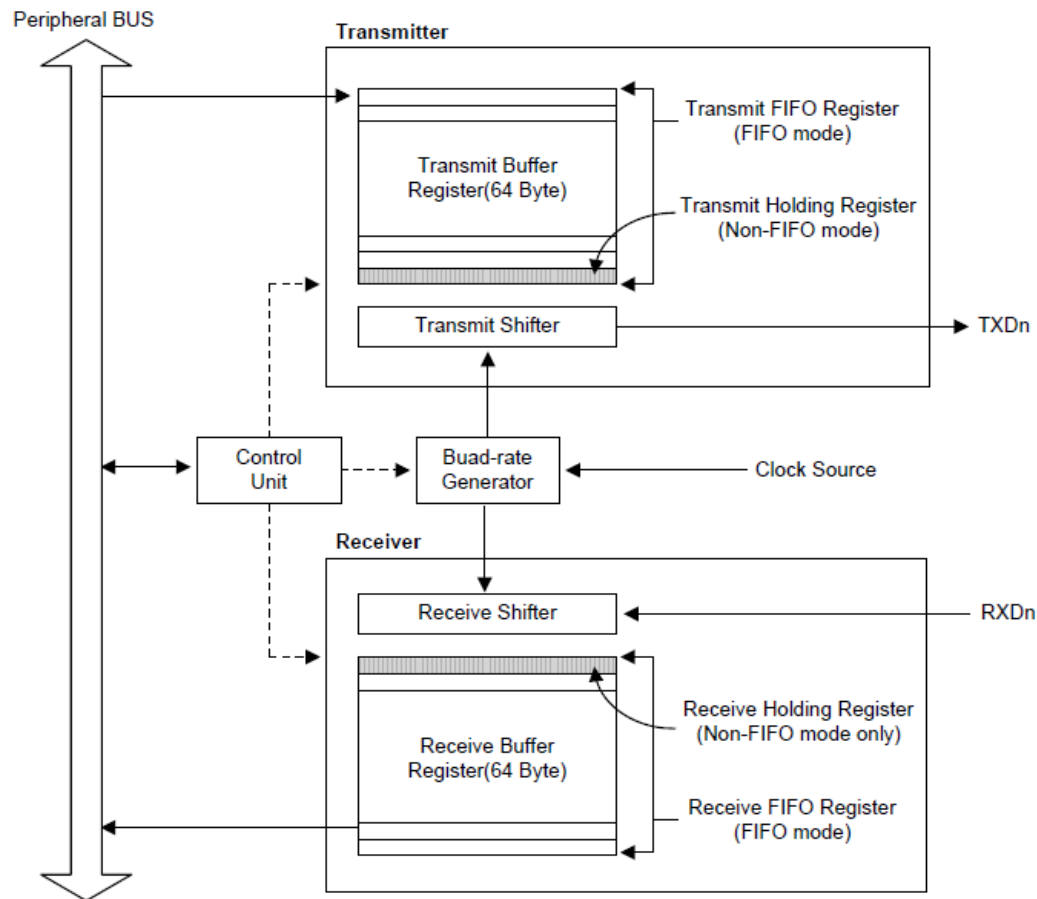
- UART에서 변환된 디지털 신호를 외부와 송수신할 때 사용
 - 디지털 신호를 외부와 인터페이스 시키는 전기적인 신호 방식
 - UART의 디지털 신호는 에너지가 너무 작아 멀리 보낼 수 없고 노이즈도 심함
 - 전압을 높여 데이터를 전송하면 더 멀리 보내 수 있음
- ➔ RS-232 (5V → 12V)



RS-232 Connector

UART Block Diagram

□ S5PC100의 UART (Datasheet Page.626)



In FIFO mode, all 64 Byte of Buffer register are used as FIFO register.
In non-FIFO mode, only 1 Byte of Buffer register is used as Holding register.

Transmitter (TX)

□ 역할

- 병렬 데이터 비트를 직렬 데이터 비트로 변환
- 시작 비트와 정지 비트, 패리티 비트를 첨가

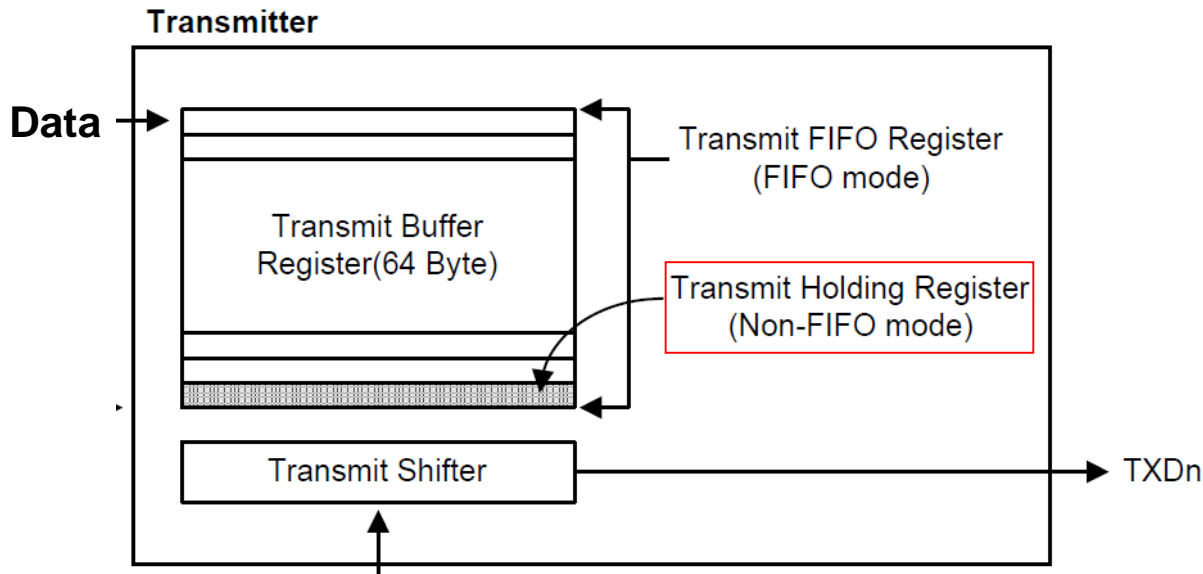
□ 모드

- **Non-FIFO Mode** : 버퍼를 사용하지 않음
- **FIFO Mode** : 버퍼 사용

Transmitter (TX)의 모드

□ Non-FIFO

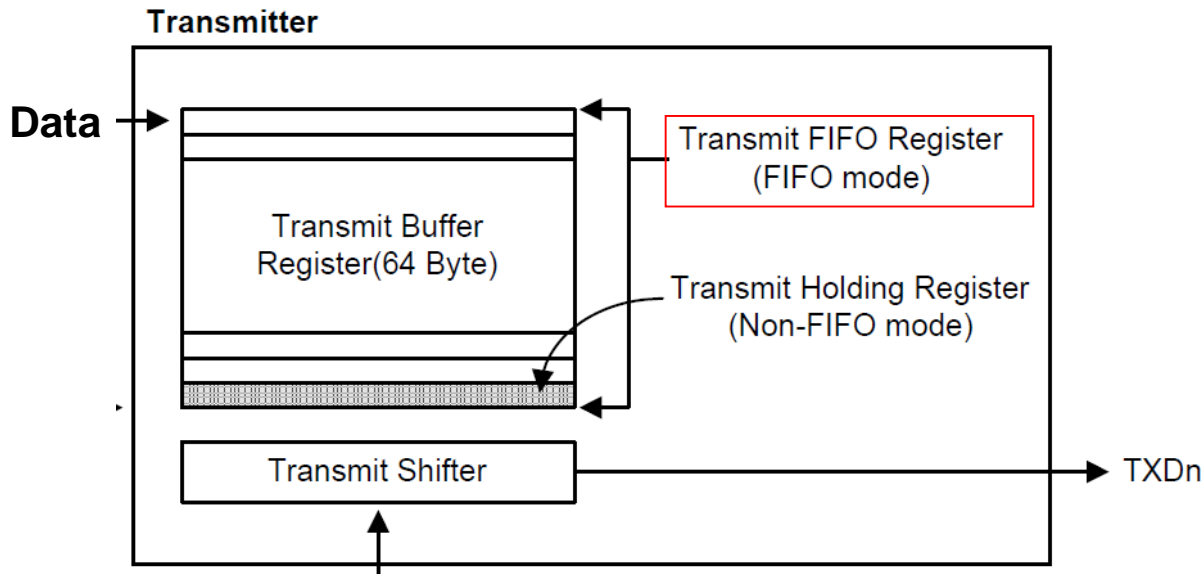
- 데이터를 **Transmit Holding Register(THR)**에 저장
- **Transmit Shift Register(TSR)**가 비어 있을 때, **THR**에 저장된 데이터는 **TSR**로 전송
- **TSR**은 TX 출력 핀으로 데이터를 시프트(shift)



Transmitter (TX)의 모드

□ FIFO

- 데이터를 Transmit Holding Register(THR)에 저장
- 전송할 데이터를 TX FIFO에 적재 (Queue)
- TSR이 비어있을 때, TX FIFO의 데이터를 Transmit Shift Register(TSR)로 전송
- TSR은 TX 출력 핀으로 데이터를 시프트(shift)



Transmitter (TX)의 Character Framing

□ Data Frame

- Start bit : 0
- 5~8 Data bits
- Parity bit
- Stop bit : 1
 - Stop bit는 하드웨어에 따라 1~2 bits

Start bit 0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Parity bit(optional)	Stop bit 1
-------------------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-------------------------	------------------

Receiver (RX)

□ 역할

- 직렬 데이터 비트를 병렬 데이터 비트로 변환
- **Start bit**를 감지, 데이터를 수신

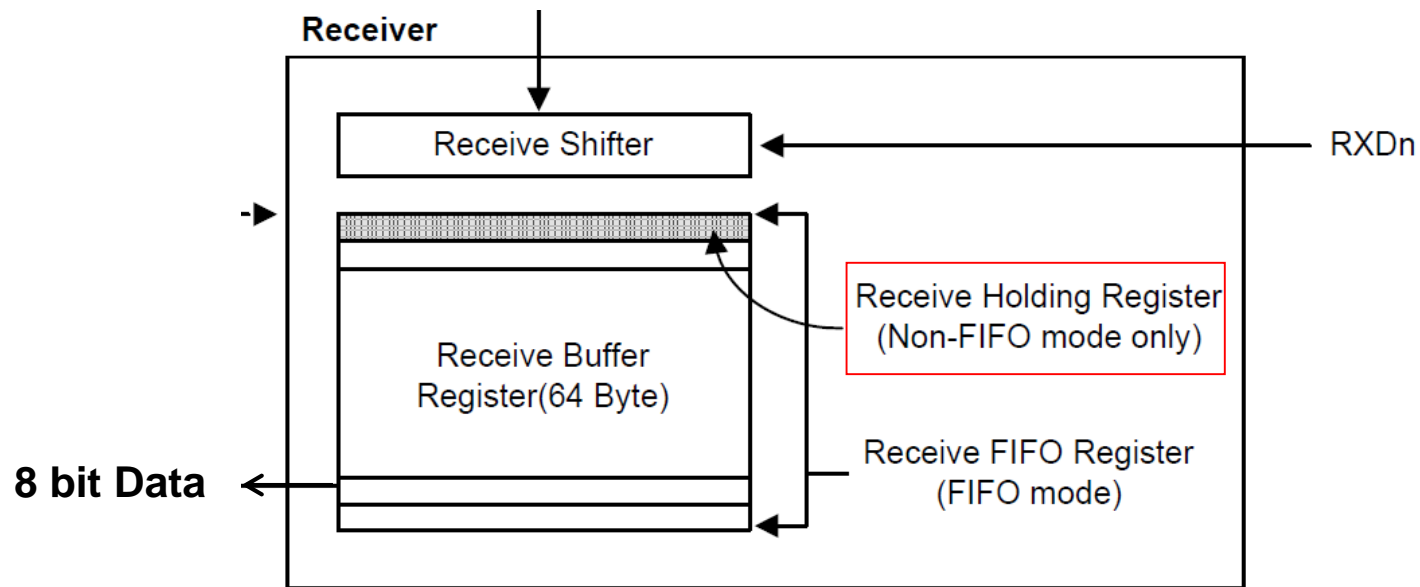
□ 모드

- **Non-FIFO Mode** : 버퍼를 사용하지 않음
- **FIFO Mode** : 버퍼 사용

Receiver (RX)의 모드

□ Non-FIFO

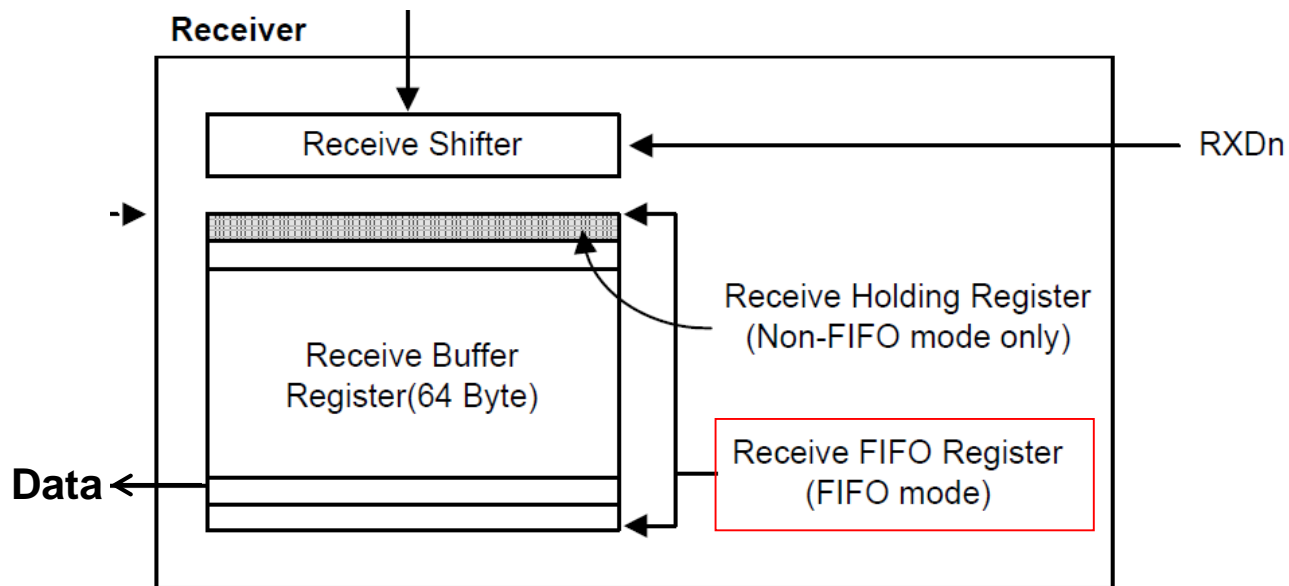
- 외부 데이터를 **Receive Shift Register (RSR)**에서 받음
- 데이터를 **Receive Holding Register (RHR)**로 전송
- **CPU가 RHR의 데이터를 읽음**



Receiver (RX)의 모드

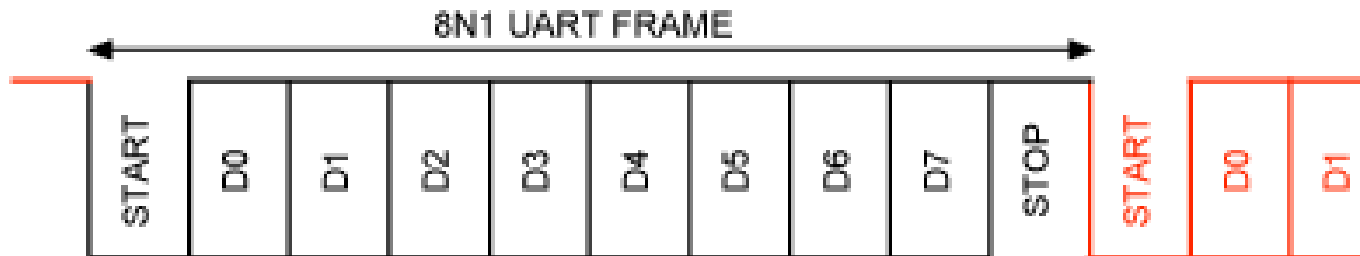
□ FIFO

- 외부 데이터를 **Receive Shift Register (RSR)**에서 받음
- 데이터를 **RX FIFO**에 적재 (Queue)
- CPU가 **Receive Holding Register (RHR)**의 데이터를 읽음



Receiver (RX) Character Validation

- 시작 비트(start bit)와 정지 비트(stop bit) 감지
 - HIGH → LOW : Start bit가 Receiver에 도착
 - LOW → HIGH 후 guard time : Stop bit가 Receiver에 도착
 - 두 bit 사이에 data bit와 parity bit 존재

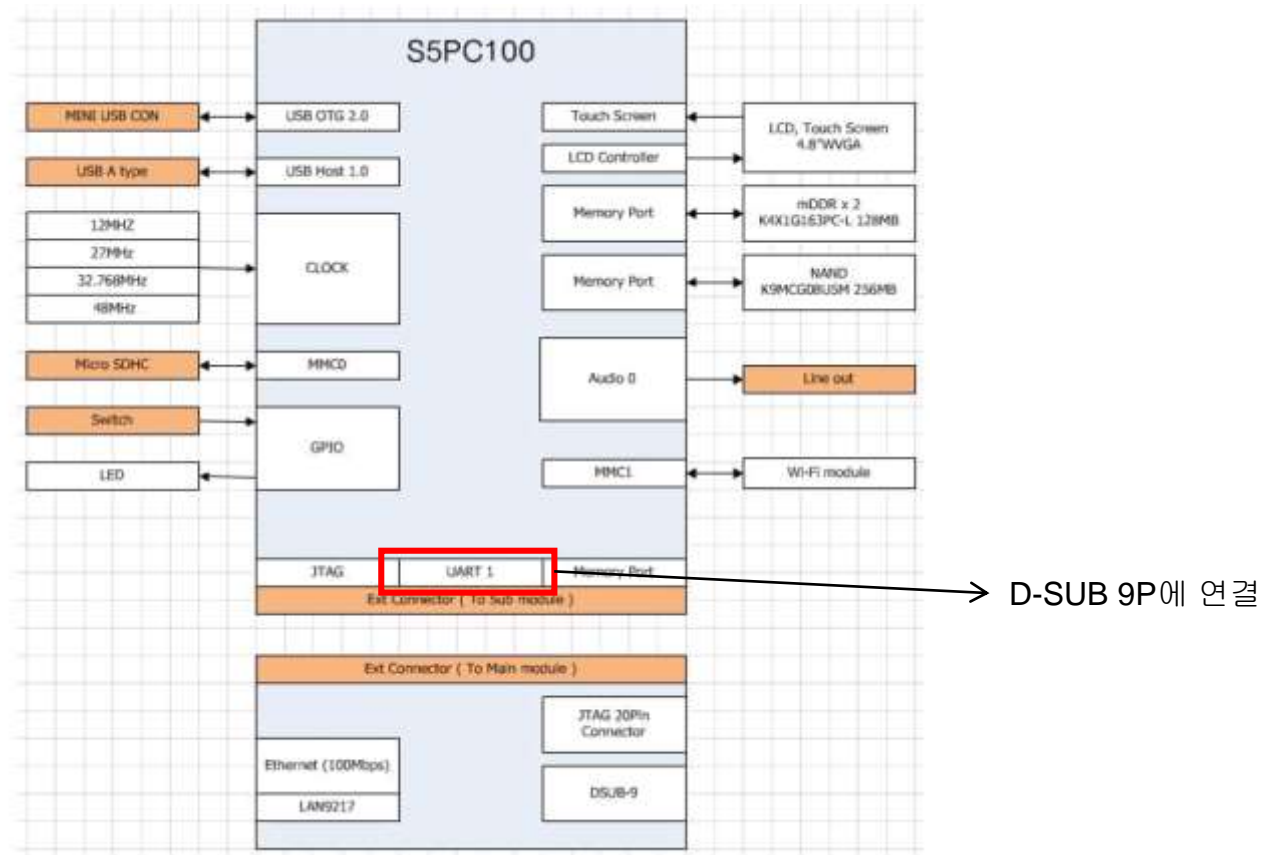


UART REGISTER SETTING

S5PC100의 UART

□ S5PC100 Block Diagram

- S5PC100의 UART1을 사용하여 PC와 통신



하드웨어 설정 및 제어

□ 소프트웨어로 어떻게 하드웨어를 제어?

→ 하드웨어의 레지스터를 사용

□ 하드웨어를 제어하기 위해 사용할 레지스터

- 제어(**control**) 레지스터 : 하드웨어를 설정하고 제어
- 상태(**status**) 레지스터 : 현재 하드웨어의 상태를 확인

소프트웨어로 하드웨어 제어

□ 코딩 순서

1. 하드웨어가 연결된 **GPIO** 핀을 **enable**
2. 제어 레지스터의 특정 비트를 **write**하여 원하는 기능을 **On/Off**하고 추가 설정을 함
3. 상태 레지스터로 현재 하드웨어의 상태를 확인
 - 데이터 수신 완료
 - 데이터 송신 완료
 - 인터럽트 수신
 - 데이터를 저장하는 레지스터가 비었음
 - 데이터를 저장하는 레지스터에 값이 있음
 - ...
4. 레지스터에 데이터를 **write**하거나 **read**
 - 버퍼 or 데이터 레지스터

소프트웨어로 UART 제어

□ 코딩 순서

1. **UART1**이 연결된 **GPIO A0**의 4번 핀과 5번 핀을 **Enable**시키고 **Full-Up**
2. **ULCON**을 비롯한 제어 레지스터를 설정
3. **UTRSTAT** 레지스터의 0번 비트(**Receive**)와 2번 비트(**Transmit**)를 확인
4. **UTXH**나 **URXH** 레지스터에 데이터를 **write**하거나 **read**

UART의 레지스터

□ 제어 레지스터

- ULCON
- UCON
- UFCON
- UINTM
- UINTP
- UBRDIV

□ 상태 레지스터

- UTRSTAT

□ 버퍼 레지스터

- UTXH
- URXH

VPOS_kernel_main()

□ 소개

- VPOS 커널 데이터 구조체를 초기화
- 시리얼 장치와 타이머 등 하드웨어를 초기화
- 인터럽트 **enable**
- 부팅 메시지 출력
- 쉘 스레드 생성
- 스케줄러 호출하는 **VPOS_start** 루틴으로 진입

□ 소스 코드 위치

- vpos/kernel/kernel.start.c

```
void VPOS_kernel_main( void )
{
    pthread_t p_thread, p_thread_0, p_thread_1, p_thread_2;

    /* static and global variable initialization */
    vk_scheduler_unlock();
    init_thread_id();
    init_thread_pointer();
    vh_user_mode = USER_MODE;
    vk_init_kdata_struct();

    vk_machine_init();
    set_interrupt();

    printk("%s\n%s\n%s\n", top_line, version, bottom_line);

    /* initialization for thread */
    race_var = 0;
    pthread_create(&p_thread, NULL, UPOS_SHELL, (void *)NULL);
    //pthread_create(&p_thread_0, NULL, race_ex_1, (void *)NULL);
    //pthread_create(&p_thread_1, NULL, race_ex_0, (void *)NULL);
    //pthread_create(&p_thread_2, NULL, race_ex_2, (void *)NULL);

    UPOS_start();

    /* cannot reach here */
    printk("OS ERROR: UPOS_kernel_main( void )\n");
    while(1){}
}
```


vk_machine_init()

□ Code

- 하드웨어 장치 초기화
- **vh_serial_init() : UART 초기화**
- **vh_timer_init() : Timer 초기화**

□ 소스 코드 위치

- **vpos/kernel/kernel_start.c**

```
#include "serial.h"
#include "timer.h"
#include "rtc_init.h"
#include "switch.h"
#include "pmu.h"

void vk_machine_init(void)
{
    vh_serial_init();
    vh_timer_init();
}
```

UART 관련 파일

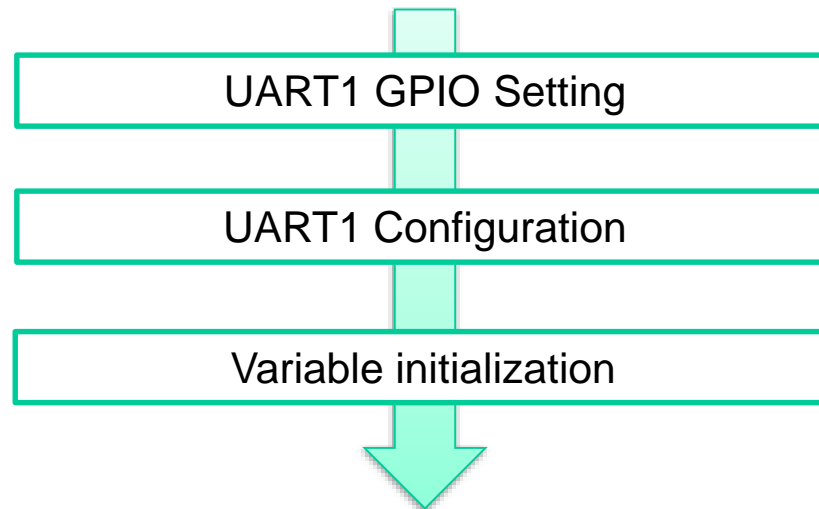
□ UART Source Code

- **vpos/hal/io/serial.c**
 - UART 초기화
 - 데이터 Write(putc), Read(getc)
 - UART Interrupt Handler
- **vpos/hal/include/vh_io_hal.h**
 - UART 관련 레지스터의 주소 정의
 - 레지스터 관련 설정 값 정의
 - TX/RX Operation 정의

vh_serial_init()

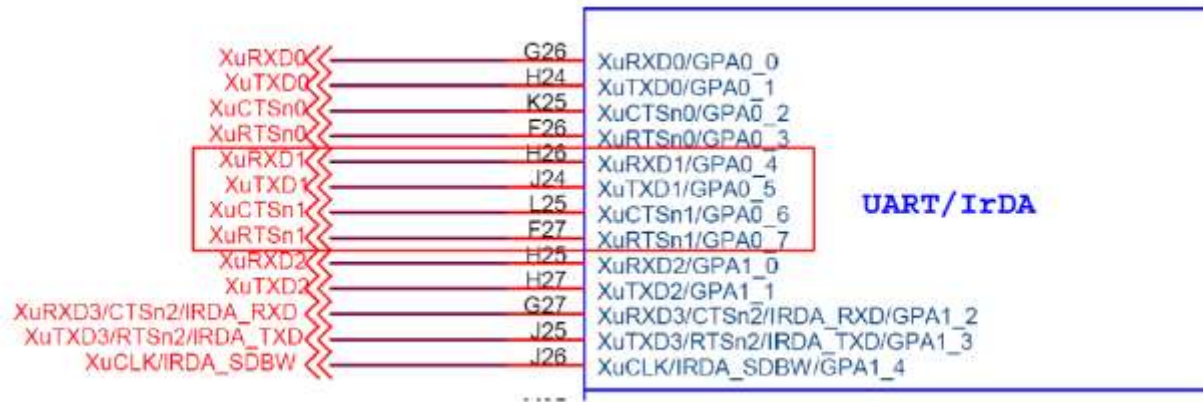
□ UART1 초기화

- S5PC100은 4개의 UART 채널을 가짐
- S5PC100은 UART1을 통해 PC와 통신



UART1 GPIO Setting

□ GPIO MAP



The values of the Pull & I/O column in the below table are the state at the reset. The meaning of the values of the VDD column in the below table are described in the above table

Pad Name	Function Signals	Pull	I/O	PDN	VDD
XuRXD[0]	GPA0[0] / UART0_RXD	PD	I	A1	VDDQ_EXT
XuTXD[0]	GPA0[1] / UART0_TXD	PD	I	A1	VDDQ_EXT
XuCTSn[0]	GPA0[2] / UART0_CTSn	PD	I	A1	VDDQ_EXT
XuRTSn[0]	GPA0[3] / UART0_RTSn	PD	I	A1	VDDQ_EXT
XuRXD[1]	GPA0[4] / UART1_RXD	PD	I	A1	VDDQ_EXT
XuTXD[1]	GPA0[5] / UART1_TXD	PD	I	A1	VDDQ_EXT
XuCTSn[1]	GPA0[6] / UART1_CTSn	PD	I	A1	VDDQ_EXT
XuRTSn[1]	GPA0[7] / UART1_RTSn	PD	I	A1	VDDQ_EXT
XuRXD[2]	GPA1[0] / UART2_RXD	PD	I	A1	VDDQ_EXT

Datasheet Page. 58

UART1 GPIO Setting

□ 관련 레지스터와 설정 방법

- **GPA0CON**에서 4번 핀과 5번 핀을 **UART1**에 연결
- **GPA0PULL**에서 4번 핀과 5번 핀을 **Pull-Up Enable** 시킴

Register	Address	R/W	Description
GPA0CON	0xE030_0000	R/W	Port Group GPA0 Configuration Register
GPA0DAT	0xE030_0004	R/W	Port Group GPA0 Data Register
GPA0PULL	0xE030_0008	R/W	Port Group GPA0 Pull-up/down Register
GPA0DRV	0xE030_000C	R/W	Port Group GPA0 Drive strength control Register

Datasheet Page. 70

UART1 GPIO Setting

□ GPA0CON

- GPIO A0포트의 4번 핀과 5번 핀을 UART1의 RX와 TX에 연결
- GPA0CON[4] = 4번 핀의 Config Register Field
- GPA0CON[5] = 5번 핀의 Config Register Field
- UART와 연결하기 위해서는 해당 field에 0010 값을 입력
- ➔ 0x220000

Field	Bit	Description	Reset Value
GPA0CON[0]	[3:0]	0000 = Input, 0001 = Output, 0010 = UART_0_RXD, 1111 = NWU_INT0[0]	0000
GPA0CON[1]	[7:4]	0000 = Input, 0001 = Output, 0010 = UART_0_TXD, 1111 = NWU_INT0[1]	0000
GPA0CON[2]	[11:8]	0000 = Input, 0001 = Output, 0010 = UART_0_CTSn, 1111 = NWU_INT0[2]	0000
GPA0CON[3]	[15:12]	0000 = Input, 0001 = Output, 0010 = UART_0_RTSn, 1111 = NWU_INT0[3]	0000
GPA0CON[4]	[19:16]	0000 = Input, 0001 = Output, 0010 = UART_1_RXD, 1111 = NWU_INT0[4]	0000
GPA0CON[5]	[23:20]	0000 = Input, 0001 = Output, 0010 = UART_1_TXD, 1111 = NWU_INT0[5]	0000
GPA0CON[6]	[27:24]	0000 = Input, 0001 = Output, 0010 = UART_1_CTSn, 1111 = NWU_INT0[6]	0000

Datasheet Page. 82

UART1 GPIO Setting

□ GPA0PULL

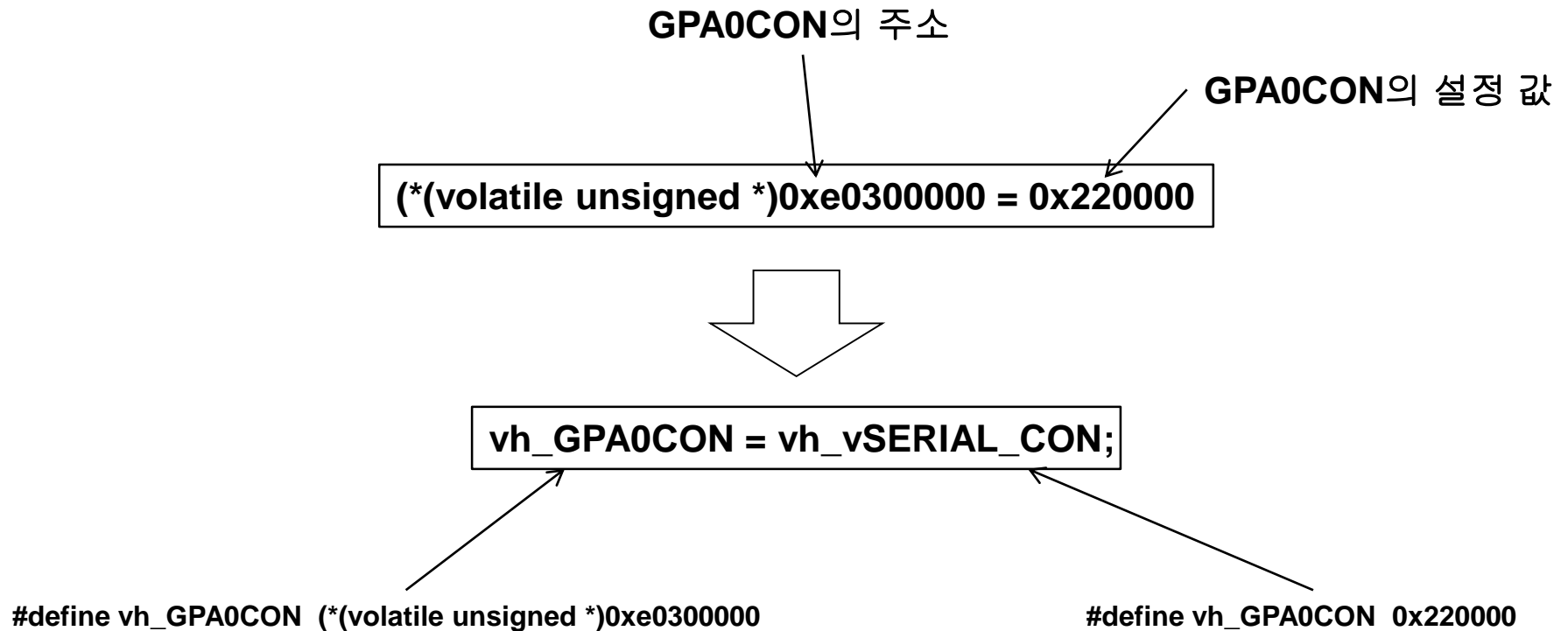
- PUD[9:8] = 4번 핀의 Pull up/down 결정
- PUD[11:10] = 5번 핀의 Pull up/down 결정
- GPIO A0포트의 4번 핀과 5번 핀을 Pull-Up
- ➔ 0xa00

Field	Bit	Description
PUD[n] (n=0~7)	[2n+1:2n]	00 = Disables Pull-up/down 01 = Enables Pull-down 10 = Enables Pull-up 11 = Reserved

Datasheet Page. 82

C코드에서 레지스터 접근

□ 레지스터 접근 방법



UART1 GPIO Setting

□ Code

▪ vpos/hal/include/vh io hal.h (GPIO, UART)

```
/*  
    GPIO  
    */  
#define vh_GPA0CON      (*(volatile unsigned *)0xe0300000)  
#define vh_GPA0PUD      (*(volatile unsigned *)0xe0300008)
```

```
/*  
    UART address  
    */  
#define vh_vSERIAL_CON      0x220000  
#define vh_vSERIAL_PUD      0xa00
```

▪ vpos/hal/io/serial.c의 vh_serial_init()

```
// UART 1 Setting  
// UART 1 GPIO setting  
vh_GPA0CON = vh_vSERIAL_CON;  
vh_GPA0PUD = vh_vSERIAL_PUD;
```

UART1 Configuration

□ Setting sequence

4.1 SETTING SEQUENCE OF SPECIAL FUNCTION REGISTER

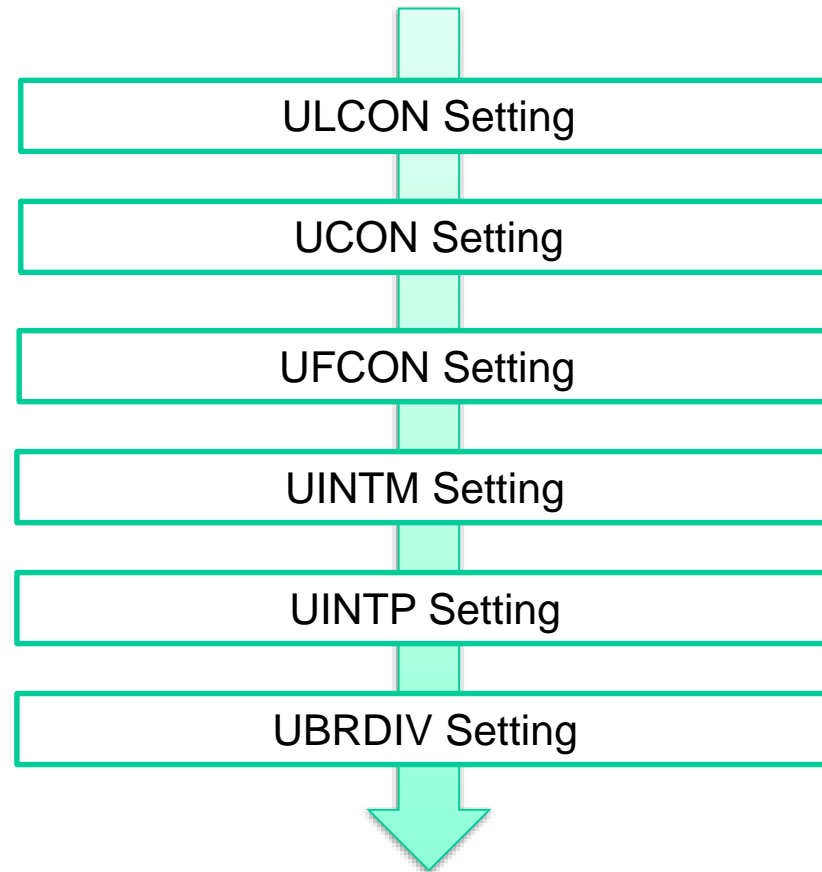
Special Function Register should be set as the following sequence.

1. Set Line control register(ULCON#) to set a frame format.
2. Set Control register(UCON#) without Transmit mode bits and Receive mode bits.
3. Set 1'b1 on TX FIFO Reset bit and RX FIFO Reset bit of FIFO control register(UFCON) to reset TX FIFO and RX FIFO.
4. Set FIFO control register(UFCON#) to set Trigger Levels and Enable TX FIFO and RX FIFO
5. Set Modem control register(UMCON#).
6. Set Baud rate divisor register(UBRDIV#) and Dividing slot register(UDIVSLOT#) to set BAUD rate.

Datasheet Page. 633

UART1 Configuration

□ Setting sequence



UART 설정하기에 앞서

□ UART Register Base Address 정의

- `vh_io_hal.h`에서 `vh_UART_CTL_BASE` 를 정의
- 이후 UART의 레지스터 주소는 **[Base address + Offset]**으로 정의
- UART 관련 레지스터는 **0xec000000**부터 차례대로 위치
 - `vh_UART_CTL_BASE`는 0xec000000

Register	Address	R/W	Description	Reset Value
ULCON0	0xEC00_0000	R/W	UART Channel 0 Line Control Register	0x00
UCON0	0xEC00_0004	R/W	UART Channel 0 Control Register	0x00
UFCON0	0xEC00_0008	R/W	UART Channel 0 FIFO Control Register	0x0
UMCON0	0xEC00_000C	R/W	UART Channel 0 Modem Control Register	0x0
UTRSTAT0	0xEC00_0010	R	UART Channel 0 Tx/Rx Status Register	0x6
UERSTAT0	0xEC00_0014	R	UART Channel 0 Rx Error Status Register	0x0
UFSTAT0	0xEC00_0018	R	UART Channel 0 FIFO Status Register	0x00

Datasheet Page. 82

```
#define vh_UART_CTL_BASE 0xec000000 // UART 1 register base address
#define vh_ULCON (*(volatile unsigned*)(vh_UART_CTL_BASE+0x0)) //
#define vh_UCON (*(volatile unsigned*)(vh_UART_CTL_BASE+0x0)) //
#define vh_UFCON (*(volatile unsigned*)(vh_UART_CTL_BASE+0x0)) //
#define vh_UMCON (*(volatile unsigned*)(vh_UART_CTL_BASE+0x040c)) //
trol
#define vh_UBRDIV (*(volatile unsigned*)(vh_UART_CTL_BASE+0x0428))
```

ULCON Setting

□ 레지스터 : ULCON1 (0xec000400)

▪ UART Line Control Register

- UART가 송수신할 Data frame 설정

□ 설정 방법

▪ Word Length[1:0] : 8bit → 0x3

ULCONn	Bit	Description	Reset Value
Reserved	[7]		0
Infrared Mode	[6]	Determines whether to use the Infrared mode. 0 = Normal mode operation 1 = Infrared Tx/Rx mode	0
Parity Mode	[5:3]	Specifies the type of parity generation to be performed and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/ checked as 1 111 = Parity forced/ checked as 0	000
Number of Stop Bit	[2]	Specifies how many stop bits are used to signal end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicates the number of data bits to be transmitted or received per frame. 00 = 5-bit 01 = 6-bit 10 = 7-bit 11 = 8-bit	00

UCON Setting

❑ 레지스터 : UCON1 (0xec000404)

- UART Control Register

❑ 설정 방법 (Page.638)

- **Transmit Mode[3:2] & Receive Mode[1:0]**
 - Interrupt request or polling mode로 설정 (01)
- **Rx Error Status Interrupt Enable[6]**
 - Error status interrupt를 enable (1)
- **Rx Interrupt Type[8]**
 - Pulse로 설정 (0)
 - ✓ S5PC100은 Rx Interrupt Type은 무조건 Pulse로
- **Tx Interrupt Type[9]**
 - Level로 설정 (1)
- **Clock Selection[11:10]**
 - PCLK로 사용 (00)
 - ✓ 66MHz

UFCON Setting

❑ 레지스터 : **UFCON1 (0xec000408)**

- **UART FIFO control register**
 - FIFO Mode를 설정

❑ 설정 방법

- **FIFO Enable[0]**
 - Enable (1)
- **Tx FIFO Reset[2] & Rx FIFO Reset[1]**
 - FIFO Reset (1)
- **Rx FIFO Trigger Level[5:4]**
 - 1-byte (00)
- **Tx FIFO Trigger Level[7:6]**
 - 48-byte(11)

❑ **Trigger Level**은 **Interrupt** 발생 시점을 결정

- “FIFO에 **X byte** 이상/이하의 데이터가 있으면 **Interrupt** 발생”
 - X byte → Trigger Level

UINTM Setting

□ 레지스터 : UINTM1 (0xec000438)

- **UART Interrupt Mask Register**

- UART에서 발생시키는 인터럽트를 마스크

□ 설정 방법

- **Receive Interrupt**만 받고 나머진 마스크

- 0번 비트만 0으로 하고 [3:1]은 1로 설정

UINTP Setting

□ 레지스터 : UINTP1 (0xec000430)

- **UART Interrupt Pending Register**

- UART Interrupt 정보를 가지고 있음

□ 설정 방법

- 모든 **Interrupt**를 발생시킴

- [4:0] 비트 모두 1로 설정

UBRDIV Setting

❑ 레지스터 : UBRDIV1 (0xec000428)

- **UART Baud Rate Divisor Register**
 - Baud rate division 값을 저장

❑ Baud rate

- **Baud rate** : 데이터 전송 속도. 초당 비트 수
- 송수신 측의 **baud rate**가 같아야 함
- **Baud rate divisor** 식

$$Divisor = \frac{PCLK}{Baud\ Rate * 16} - 1$$

- **Clock Frequency : PCLK : 66MHz**
- **Sampling Rate : 16**
- **Baud Rate : 115200**

❑ 설정 방법

- **Divisor**를 **UBRDIV**에 저장

UART Register Setting

□ Code

- vpos/hal/include/vh_io_hal.h

```
#define vh_UART_CTL_BASE 0xec000000 // UART 1 register base address

#define vh_ULCON (*(volatile unsigned *)(vh_UART_CTL_BASE+0x400))
// line control

#define vh_UCON (*(volatile unsigned *)(vh_UART_CTL_BASE+0x404))
// control

#define vh_UFCON (*(volatile unsigned *)(vh_UART_CTL_BASE+0x408))
// FIFO control

#define vh_UMCON (*(volatile unsigned *)(vh_UART_CTL_BASE+0x40c))
// modem control

#define vh_UBRDIV (*(volatile unsigned *)(vh_UART_CTL_BASE+0x428))
// baud rate divisor

#define vh_UINTP1 (*(volatile unsigned *)(vh_UART_CTL_BASE+0x430))
#define vh_UINTSP1 (*(volatile unsigned *)(vh_UART_CTL_BASE+0x434))
#define vh_UINTM1 (*(volatile unsigned *)(vh_UART_CTL_BASE+0x438))
```

UART Register Setting

□ Code

▪ vpos/hal/io/serial.c

```
// UART 1 Configuration
vh_ULCON = 
vh_UCON = 
vh_UFCON = 
vh_UINTM1 = 
vh_UINTP1 = 
vh_UBRDIV = ((66000000 / (115200 * 16)) - 1);
```

□ 과제

- 각 레지스터의 설정 값 채우기
 - 예제 : `vh_XXXXX = 0x00;`
- 컴파일 후 **minicom**으로 확인



```
Starting kernel ...
*****
* QURIX version 3.0   xx/10/2012
*****
Race condition value = 0
Shell>abcd good pir
```

UART DATA TRANSMIT & RECEIVE

Data Transmit

□ Data : S5PC100 → Host PC

- S5PC100의 문자 데이터를 Host PC의 minicom 화면에 출력
→ `printk()`

□ `printk()`

- 화면에 문자, 숫자 등을 출력시키는 함수
- `putc(c)` 호출
 - `serial.c`에 위치
- `putc(c)`의 코드는 `vh_SERIAL_PUTC(c)`로 정의
 - `vh_io_hal.h`에 위치

vh_SERIAL_PUTC(c)

□ Code

```
while (!vh_SERIAL_WRITE_READY());  
  
vh_SERIAL_WRITE_CHAR(c);
```

□ vh_SERIAL_WRITE_READY()

- UTRSTAT1 레지스터를 확인하여 현재 Transmitter의 buffer와 shift register가 비어있는지 확인
- UTRSTAT1 : UART TX/RX Status Register

□ vh_SERIAL_WRITE_CHAR(c)

- UTXH1 레지스터에 데이터를 입력
- UTXH1 : UART Transmit Buffer Register

vh_SERIAL_PUTC(c)

□ 상태 레지스터 확인 → 레지스터에 데이터를 입력(write)

□ Code

```
while (!((vh_UTRSTAT1) & vh_UTRSTAT_TX_EMPTY));  
vh_UTXH1 = c;
```

(1 << 2) : 2번 비트

Transmitter empty	[2]	This bit is automatically set to 1 if the transmit buffer register has no valid data to transmit, and the transmit shift register is empty. 0 = Not empty 1 = Transmitter (transmit buffer & shifter register) empty
-------------------	-----	--

□ 위 형태의 코드는 리눅스의 디바이스 드라이버에서도 볼 수 있음

Data Receive

□ Data : Host PC → S5PC100

- PC의 키보드 입력이 S5PC100으로 전달
- 데이터를 수신할 때는 **Polling**과 **Interrupt** 중 선택해야 함
 - 이번 주는 Polling으로

□ getc()

- Shell에서 **getc()**를 통해 키보드 문자를 받음
- **getc()** 코드는 **serial.c**에 위치

Data Receive

□ Code

```
while (!vh_SERIAL_CHAR_READY());  
  
c = vh_SERIAL_READ_CHAR();
```

```
char getc(void)  
{  
    char c;  
    unsigned long rxstat;  
  
    while(!vh_SERIAL_CHAR_READY());  
  
    c = vh_SERIAL_READ_CHAR();  
    rxstat = vh_SERIAL_READ_STATUS();  
  
    /*  
    while(pop_idx == push_idx){  
    }  
    c = serial_buff[pop_idx++];  
    */  
    return c;  
}
```

□ vh_SERIAL_CHAR_READY()

- UTRSTAT1 레지스터를 확인하여 현재 Receiver의 buffer 레지스터에 데이터가 있는지 확인
- UTRSTAT1 : UART TX/RX Status Register

□ vh_SERIAL_READ_CHAR()


- URXH1 레지스터를 가리킴
- URXH1 : UART Receive Buffer Register

Data Receive

□ 상태 레지스터 확인 → 레지스터의 데이터를 읽음(read)

□ Code

```
while (!(vh_UTRSTAT1 & vh_UTRSTAT_RX_READY));  
  
c = vh_URXH1;
```



(1 << 0) : 0번 비트

Receive buffer data ready	[0]	<p>This bit is automatically set to 1 if receive buffer register contains valid data, received over the RXDn port.</p> <p>0 = Buffer register is empty 1 = Buffer register has a received data (In Non-FIFO mode, Interrupt or DMA is requested)</p> <p>If UART uses the FIFO, check Rx FIFO Count bits and Rx FIFO Full bit in UFSTAT register instead of this bit.</p>
---------------------------	-----	--

TIMER 설정

VPOS 커널 포팅

1. 커널 컴파일 + 커널 이미지를 RAM에 적재
2. Startup code 작성
3. UART 설정
4. **TIMER** 설정
5. Hardware Interrupt Handler 구현
 - (1) UART Interrupt
 - (2) Timer Interrupt
6. Software Interrupt Entering/Leaving Routine 구현

목차

1. Timer
2. Timer Register Setting
3. Timer Test

TIMER

Timer

□ Timer란?

- 시간이 지남에 따라 일정한 주기로 카운트 업 혹은 다운
- 정해진 카운트에 도달하면 인터럽트를 발생시키는 하드웨어 장치
- 원하는 주기로 인터럽트를 발생시키는 용도로 사용

□ Timer의 예

- Watchdog Timer
- PWM Timer
- System Timer
- ...

Timer의 기본 요소

□ Clock Source

- Timer는 **Clock Source**가 필요
- **Clock Source**를 **prescaler, divider** 값으로 원하는 **count** 주기 만들
 - S5PC100에서는 PCLK(66MHz) 사용

□ Counter

- **Clock**이 한 주기씩 지날 때마다 1씩 증가, 감소
- **Counter**가 증가하는 타입이면 0부터 **Reload value**까지 카운트 업
- **Counter**가 감소하는 타입이면 **Reload value**부터 0까지 카운트 다운

□ Reload value & Refresh

- **Counter**가 1씩 증가하거나 감소하여 정해진 값에 도달하면 **Timer**의 **refresh**가 일어남
 - Counter가 감소하는 타입이면 0이 될 때 refresh
- **Refresh**가 발생하면 인터럽트 발생시킴

Timer의 기본 요소

□ Prescaler

- **Clock Source**의 주파수를 줄이는 첫 번째 인자
- **Clock Source**의 주파수를 (인자 값+1)로 나눔
- 1 ~ 255 (S5PC100)
- ex) 66MHz, pre-scaler 값 : 1 → 33MHz

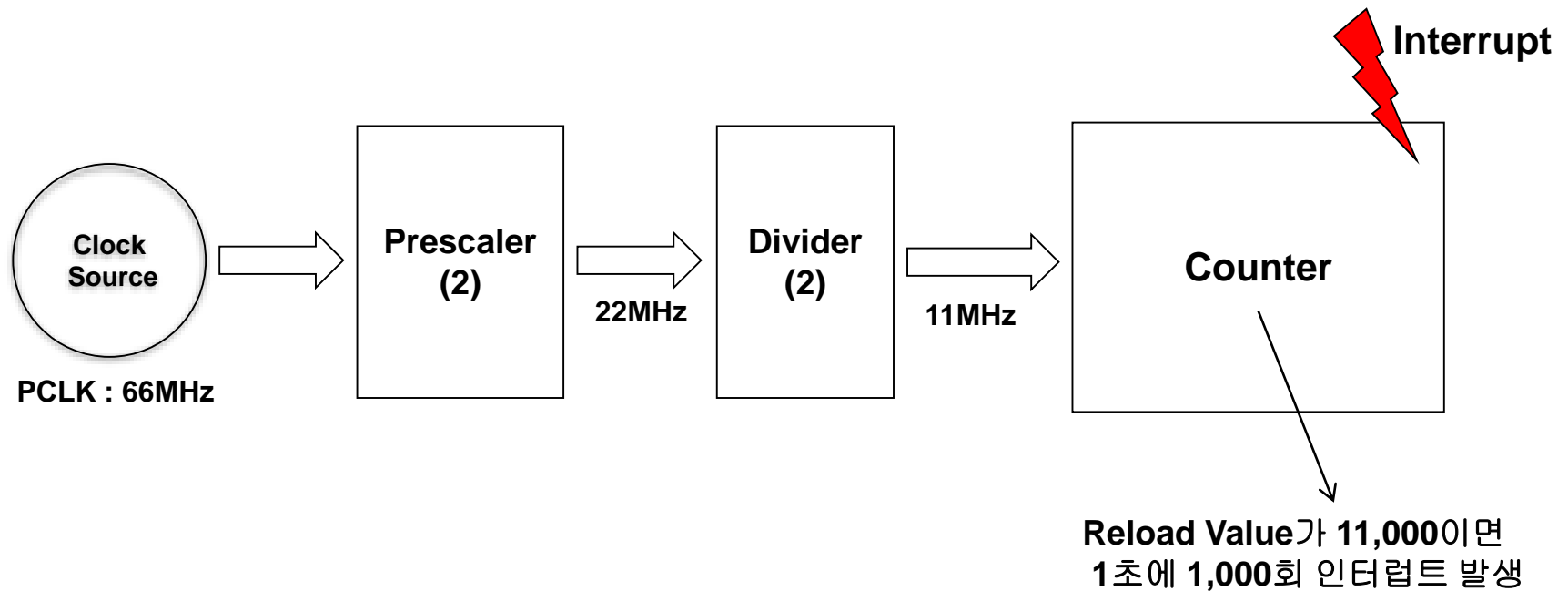
□ Divider

- **Clock Source**의 주파수를 줄이는 두 번째 인자
- **Clock Source**의 주파수를 인자 값으로 나눔
- 1, 2, 4, 8, 16, TCLK(외부 클럭)
- ex) 66MHz, divider 값 : 2 → 33MHz

□ Timer Input Clock Frequency

- 타이머(카운터)에 입력되는 실제 **clock** 주파수
- **CLK Source / ({prescaler value + 1}) / {divider value}** 식으로 계산

Timer



TIMER REGISTER SETTING

S5PC100의 Timer

□ Timer의 종류

- **PWM Timer**
- **System Timer**

□ PWM Timer

- **5개의 32bit Timer를 가지고 있음**
 - Timer 0, 1, 2
 - ✓ 외부로 신호를 보낼 수 있는 Timer
 - Timer 3, 4
 - ✓ 출력 핀이 없어 내부에서만 쓰이는 Timer
- **Clock Source는 PCLK(66MHz)**
- **Down-Counter**
 - Timer Count Buffer Register(TCNTBn)에 저장된 값(Reload value)부터 시작하여 0으로 카운트 다운
 - 0이 되면 인터럽트를 발생시켜 CPU에게 알림

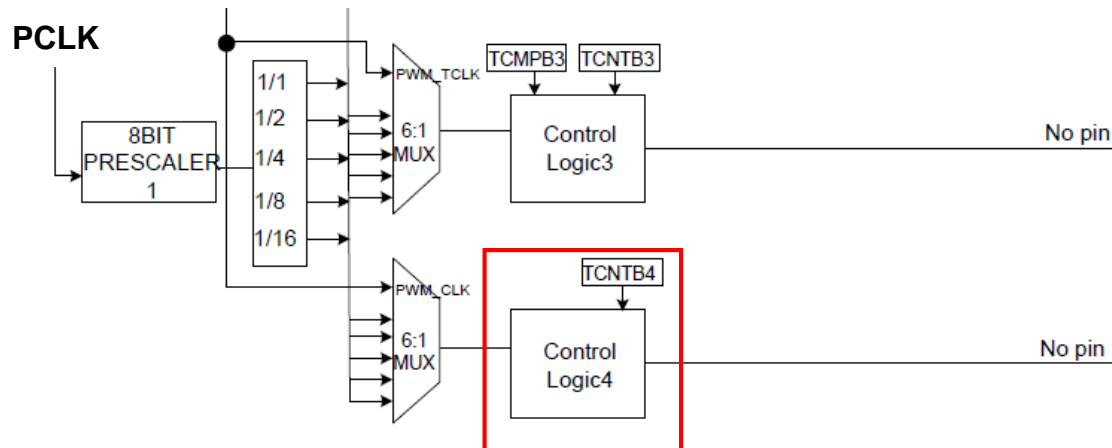
본 실습에서의 Timer

□ 용도

- 일정 주기(1초)마다 스케줄러 호출

□ Timer 4 사용

- 출력 핀을 통해 외부로 신호를 보낼 필요 없음
 - 내부 타이머 사용
 - ✓ Timer 3 또는 4
- Duty cycle을 조절할 필요 없는 Timer 4로 사용



Timer의 레지스터

□ 제어 레지스터

- TCFG0
- TCFG1
- TCON
- TINT_CSTAT (인터럽트 제어 & 상태 레지스터)

□ 버퍼 레지스터

- TCNTB4

□ 그 외 레지스터

- TCNT04

TCFG0

□ 레지스터 : TCFG0 (0xea000000)

▪ Timer Configuration Register

- Prescaler 값 설정 (1~255)
- PCLK(66MHz)에 prescaler 값을 나누어 원하는 타이머 주기(주파수)를 설정할 수 있음
 - ✓ Prescaler 값 나눌 때 1을 먼저 더함 ($PCLK / (prescaler + 1)$)

▪ Datasheet

- Page. 591

TCFG0	Bit	Description	Reset Value
Reserved	[31:24]	Reserved Bits	0x00
Dead zone length	[23:16]	Dead zone length	0x00
Prescaler 1	[15:8]	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	Prescaler 0 value for timer 0 & 1	0x01

TCFG1

□ 레지스터 : TCFG1 (0xea000004)

▪ Timer Configuration Register

- Divider 값 설정 (1,2,4,8,16 or PWM_TCLK)
- Prescaler 값으로 나눈 주파수를 다시 한 번 divider 값으로 나눔
✓ $PCLK / (\{prescaler\ value + 1\}) / \{divider\ value\} = \text{Timer Freq.}$

▪ Datasheet

- Page. 592

TCFG1	Bit	Description	Reset Value
Reserved	[31:24]	Reserved Bits	0x00
Divider MUX4	[19:16]	Selects Mux input for PWM Timer 4 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = PWM_TCLK	0x00

TCON (1/2)

□ 레지스터 : TCON (0xea000008)

▪ Timer Control Register

- Timer 시작 & 정지
- TCNTB4 업데이트
- Auto Reload On/Off

▪ Datasheet

- Page. 593

TCON	Bit	Description	Reset Value
Reserved	[31:23]	Reserved Bits	0x000
Timer 4 Auto Reload on/off	[22]	0 = One-Shot 1 = Interval Mode(Auto-Reload)	0x0
Timer 4 Manual Update	[21]	0 = No Operation 1 = Update TCNTB4	0x0
Timer 4 Start/Stop	[20]	0 = Stop 1 = Start Timer 4	0x0
Timer 3 Auto Reload on/off	[19]	0 = One-Shot 1 = Interval Mode(Auto-Reload)	0x0

TCON (2/2)

□ 레지스터 : TCON (0xea000008)

- **Bit [20] Timer 4 Start/Stop**
 - 1 : Timer 4 시작
 - 0 : Timer 4 정지
- **Bit [21] Timer 4 Manual Update**
 - 1 : TCNTB4를 update
 - 0 : No operation
- **Bit [22] Timer 4 Auto Reload On/Off**
 - 1 : Timer 4의 counter가 0에 도달하면 자동으로 reload
 - 0 : Timer 4의 counter를 한 번만 작동시킴

TCNTB4

□ 레지스터 : TCNTB4 (0xea00003c)

▪ Timer4 Counter Register

- Reload Value를 설정
- Timer는 TCNTB4에 설정된 값부터 카운트 다운을 진행하다 0이 되면 인터럽트를 발생시킴
- 인터럽트를 발생시키고 counter는 다시 TCNTB4값으로 초기화

▪ 예시

- Timer 4의 Clock을 TCFG0, TCFG1을 통해 1000Hz로 설정
- TCNTB4 레지스터에 1000(10진수)이나 0x3e8을 입력
- 1초에 한 번씩 인터럽트 발생

▪ Datasheet

- Page. 596

TCNTB4	Bit	Description	Reset Value
Timer 4 Count Buffer	[31:0]	Timer 4 Count Buffer Register	0x0000_0000

TCNTO4

□ 레지스터 : TCNTO4 (0xea000040)

- **Timer4 Observation Register**
 - Counter의 현재 값을 읽을 수 있음
- **Datasheet**
 - Page. 596

TCNTO4	Bit	Description	Reset Value
Timer 4 Count Observation	[31:0]	Timer 4 Count Observation Register	0x0000_0000

VPOS_kernel_main()

□ 소개

- VPOS 커널 데이터 구조체를 초기화
- 시리얼 장치와 타이머 등 하드웨어를 초기화
- 인터럽트 **enable**
- 부팅 메시지 출력
- 쉘 스레드 생성
- 스케줄러 호출하는 **VPOS_start** 루틴으로 진입

□ 소스 코드 위치

- vpos/kernel/kernel_start.c

```
void VPOS_kernel_main( void )
{
    pthread_t p_thread, p_thread_0, p_thread_1, p_thread_2;

    /* static and global variable initialization */
    vk_scheduler_unlock();
    init_thread_id();
    init_thread_pointer();
    vh_user_mode = USER_MODE;
    vk_init_kdata_struct();

    vk_machine_init();
    set_interrupt();

    printk("%s\n%s\n%s\n", top_line, version, bottom_line);

    /* initialization for thread */
    race_var = 0;
    pthread_create(&p_thread, NULL, UPOS_SHELL, (void *)NULL);
    //pthread_create(&p_thread_0, NULL, race_ex_1, (void *)NULL);
    //pthread_create(&p_thread_1, NULL, race_ex_0, (void *)NULL);
    //pthread_create(&p_thread_2, NULL, race_ex_2, (void *)NULL);

    UPOS_start();

    /* cannot reach here */
    printk("OS ERROR: UPOS_kernel_main( void )\n");
    while(1){}
}
```

vk_machine_init()

□ Code

- 하드웨어 장치 초기화
- **vh_serial_init() : UART 초기화**
- **vh_timer_init() : Timer 초기화**

□ 소스 코드 위치

- **vpos/kernel/kernel_start.c**

```
#include "serial.h"
#include "timer.h"
#include "rtc_init.h"
#include "switch.h"
#include "pmu.h"

void vk_machine_init(void)
{
    vh_serial_init();
    vh_timer_init();
}
```

Timer 관련 파일

□ Timer Source Code

- **vpos/hal/io/timer.c**
 - Timer 초기화
 - Timer Interrupt enable/disable
 - Timer Interrupt Handler
- **vpos/hal/include/vh_io_hal.h**
 - Timer 관련 레지스터의 주소 정의

과제

□ Timer 4 관련 레지스터 주소 정의

- `vpos/hal/include/vh_io_hal.h`

□ Timer 4 초기화

- `vpos/hal/io/timer.c`
 - `vh_timer_init()` 함수에서 Timer 4 초기화

과제 - Timer 4 관련 레지스터 주소 정의

□ Timer 관련 레지스터 주소 정의

- `vpos/hal/include/vh_io_hal.h`에서 정의 (**define** 문 사용)
- 레지스터 목록
 - TCFG0
 - TCFG1
 - TCON
 - TINT_CSTAT
 - TCNTB4
 - TCNT04
- 상수 이름은 **vh_[Regiser Name]** 식으로.
ex) `vh_TCFG0`,

과제 - Timer 4 초기화

□ Timer 4 초기화

- `vpos/hal/io/timer.c`
 - `vh_timer_init()` 함수에서 Timer 4 초기화
- **Clock Source(PCLK) : 66MHz**
- **인터럽트 발생 주기 : 1초**

□ 레지스터 설정 순서

1. TCFG0 설정

- Timer 4의 prescaler 값을 지정 : 255

2. TCFG1 설정

- Timer 4의 divider 값을 지정 : 1/16

3. TCNTB4 설정

- 인터럽트 발생 주기가 1초가 되도록 계산해서 설정

TIMER TEST

TCNT00를 사용하여 Timer 테스트

□ Timer 테스트 함수 추가

- vpos/kernel/kernel_start.c

□ Timer4 Enable code (Timer4 시작)

1. TCON의 Timer4 필드를 모두 0으로 clear (20~22번 비트)
2. Auto Reload on 및 Manual Update (21번 비트와 22번 비트를 1로 set)
3. TCON의 Timer4 부분을 모두 0으로 clear
4. Auto Reload on 및 Timer4 Start (20번 비트와 22번 비트를 1로 set)

□ 루프문 안에서 TCNT04 레지스터의 값을 여러 번 확인

- 타이머가 제대로 동작하면 TCNT04의 값이 계속 바뀜

TCNT00를 사용하여 Timer 테스트

□ Timer 테스트 함수 정의

- vpos/kernel/kernel_start.c
- VPOS_kernel_main() 함수 위에 정의

□ Code

```
void TIMER_test(void)
{
    int timbuffer[10];
    int i;
    //Timer4 Start
    vh_TCON = (vh_TCON & ~0x700000) | 0x600000;
    vh_TCON = (vh_TCON & ~0x700000) | 0x500000;

    for (i=0; i<10; i++) {
        timbuffer[i] = vh_TCNT04;
        printk("timbuffer[%d] = %d\n", i, timbuffer[i]);
    }
}
```

TCNT00를 사용하여 Timer 테스트

□ Timer 테스트 함수 호출

- VPOS_kernel_main() 함수에서 테스트 함수 호출

□ Code

```
void VPOS_kernel_main( void )
{
    pthread_t p_thread, p_thread_0, p_thread_1, p_thread_2;

    /* static and global variable initialization */
    vk_scheduler_unlock();
    init_thread_id();
    init_thread_pointer();
    vh_user_mode = USER_MODE;
    vk_init_kdata_struct();

    vk_machine_init();
    set_interrupt();

    printk("%s\n%s\n%s\n", top_line, version, bottom_line);

    //TIMER4 Test.....
    TIMER_test();

    /* initialization for thread */
    race_var = 0;
    pthread_create(&p_thread, NULL, VPOS_SHELL, (void *)NULL);
    //pthread_create(&p_thread_0, NULL, race_ex_1, (void *)NULL);
    //pthread_create(&p_thread_1, NULL, race_ex_0, (void *)NULL);
    //pthread_create(&p_thread_2, NULL, race_ex_2, (void *)NULL);

    VPOS_start();
}
```

보고서 제출

□ 보고서

- 학과, 학번, 이름
- 과제 첨부
 - timer.c의 `vh_timer_init()` 소스 코드
 - ✓ TCNTB4 레지스터의 값에 대해 설명
 - `vh_io_hal.h`의 Timer 관련 레지스터 주소 정의 부분
- **Timer** 테스트 화면 첨부

Timer 테스트 화면 첨부

```
Starting kernel ...

*****
*   QURIX version 3.0      xx/10/2012      *
*****
timbuffer[0] = 16000
timbuffer[1] = 15969
timbuffer[2] = 15938
timbuffer[3] = 15906
timbuffer[4] = 15875
timbuffer[5] = 15844
timbuffer[6] = 15812
timbuffer[7] = 15781
timbuffer[8] = 15749
timbuffer[9] = 15718

Race condition value = 0

Shell>ls

*****Command_List*****
- help
- ls
- debug
- thread
- temp
*****

Shell>abc
abc: command not found
Shell>
```

제출 방법

□ 제출 방법

- 워드나 한글로 작성하여 메일에 첨부
- 문서 제목에 학번과 이름을 적을 것

□ E-mail (반드시 아래 2개의 메일 계정으로 모두 전송)

- jypark@rtcc.hanyang.ac.kr

□ 메일 제목

- [임베디드 시스템 실습 과제4]학번_이름

□ 마감일

- 다음 실습 수업시간 전까지

수고하셨습니다.