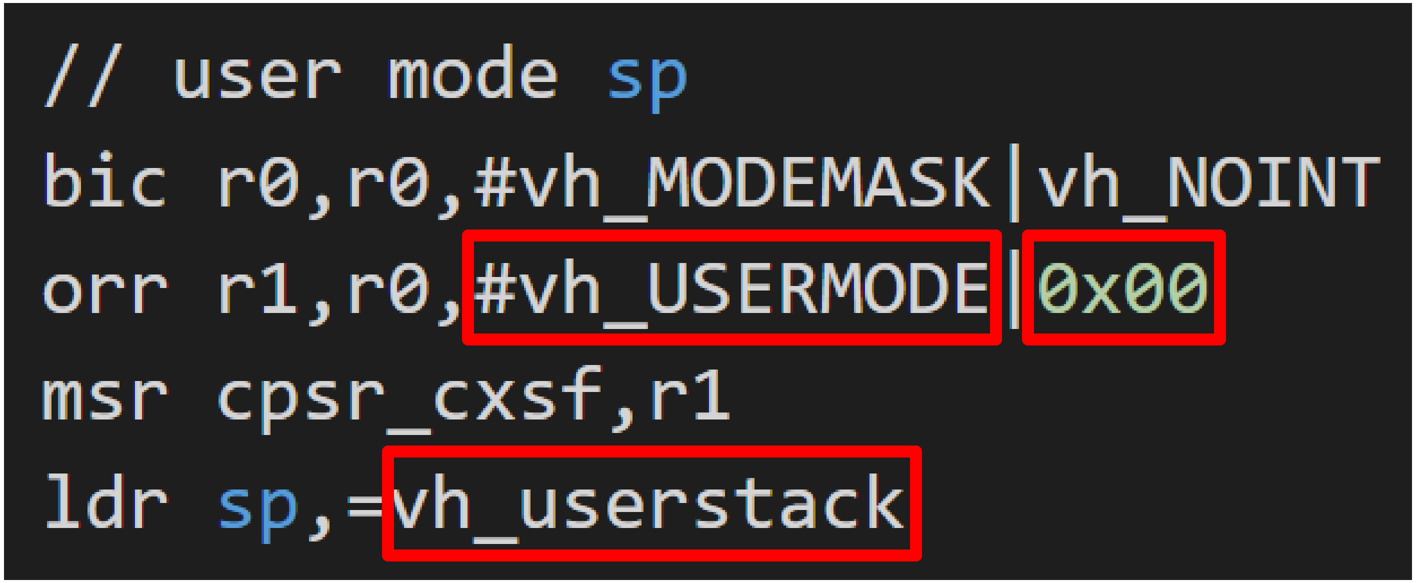
Embedded System Design
Practice 5

TaeWook Kim & SeokHyun Hong
Hanyang University

USER MODE STACK

User Mode Stack Setting

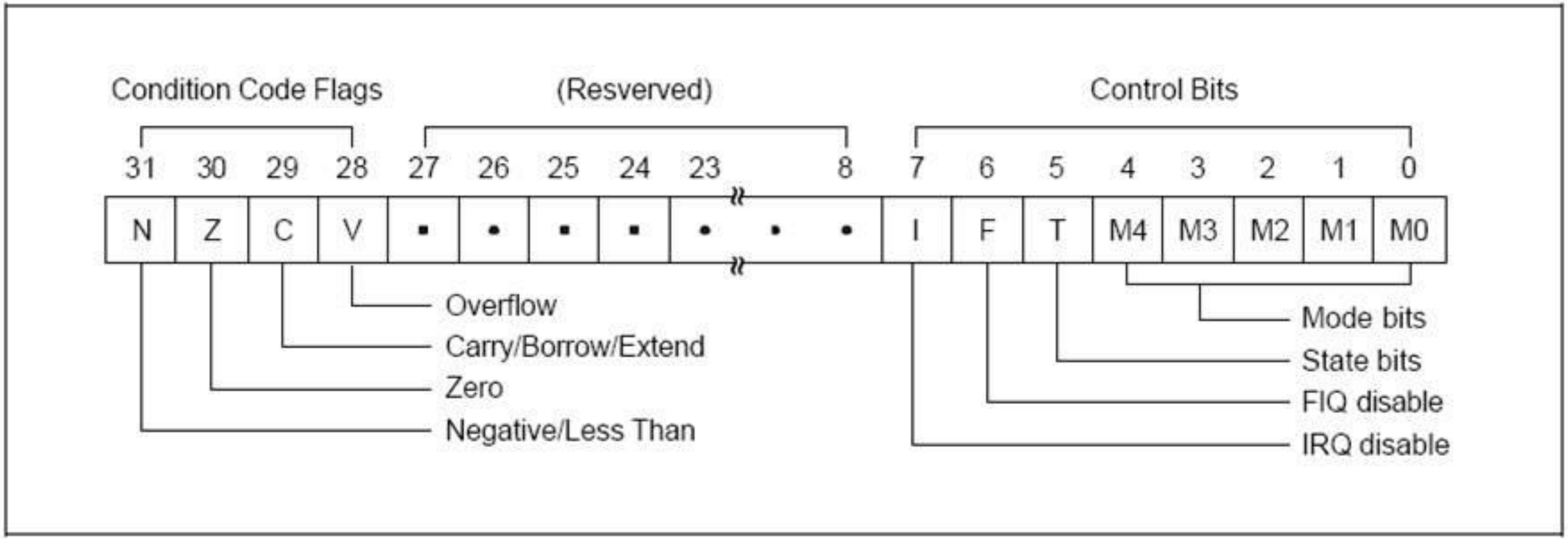
User Mode Stack Setup Code



```
// user mode sp
bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_USERMODE|0x00
msr cpsr_cxsf,r1
ldr sp,=vh_userstack
```

User Mode Stack Setting

CPSR register



User Mode Stack Setting

User Mode Setup

1) change mode bit in the CPSR register

2) Enable interrupt bit in the CPSR register

3) Setup stack pointer address

User Mode Stack Setting

Update CPSR Register

```
msr r0, cpsr
…
bic r0, r0, vh_MODEMASK | vh_NOINT
```

vh_MODEMASK | vh_NOINT = 0x0001 1111 | 1100 0000

= 0x1101 1111

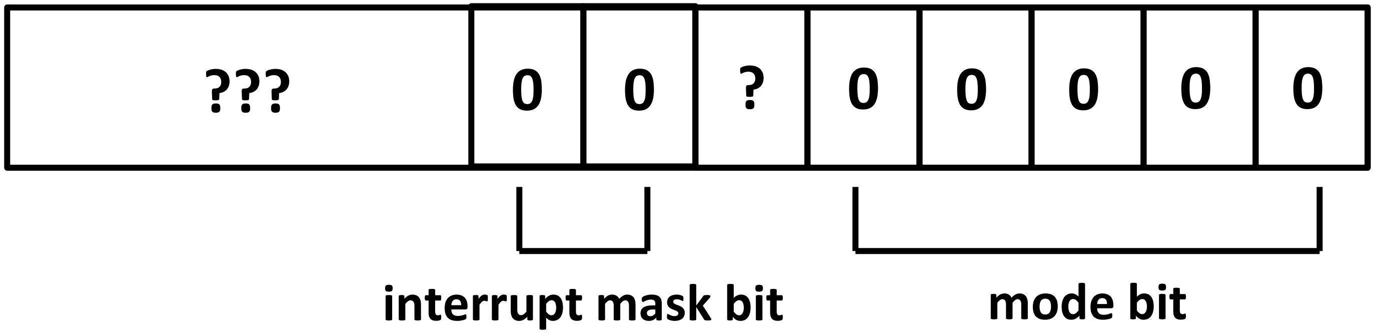
r0 = r0 & ~0x1101 1111

= r0 & 0x1111 … 0010 0000

User Mode Stack Setting

Update CPSR Register

r0 register



??? 0 0 ? 0 0 0 0 0

interrupt mask bit

mode bit

User Mode Stack Setting

Update CPSR Register

```
msr r0, cpsr
…
bic r0, r0, vh_MODEMASK | vh_NOINT
orr r1, r0, vh_USERMODE | 0x00
```

```
vh_USERMODE | 0x00 = 0x0001 0000 | 0x00
                   = 0x0001 0000
```

User Mode Stack Setting

Update CPSR Register

???	0	0	?	0	0	0	0	0	r0 register

or

???	0	0	0	1	0	0	0	0	constant

=

???	0	0	?	1	0	0	0	0	r1 register

User Mode Stack Setting

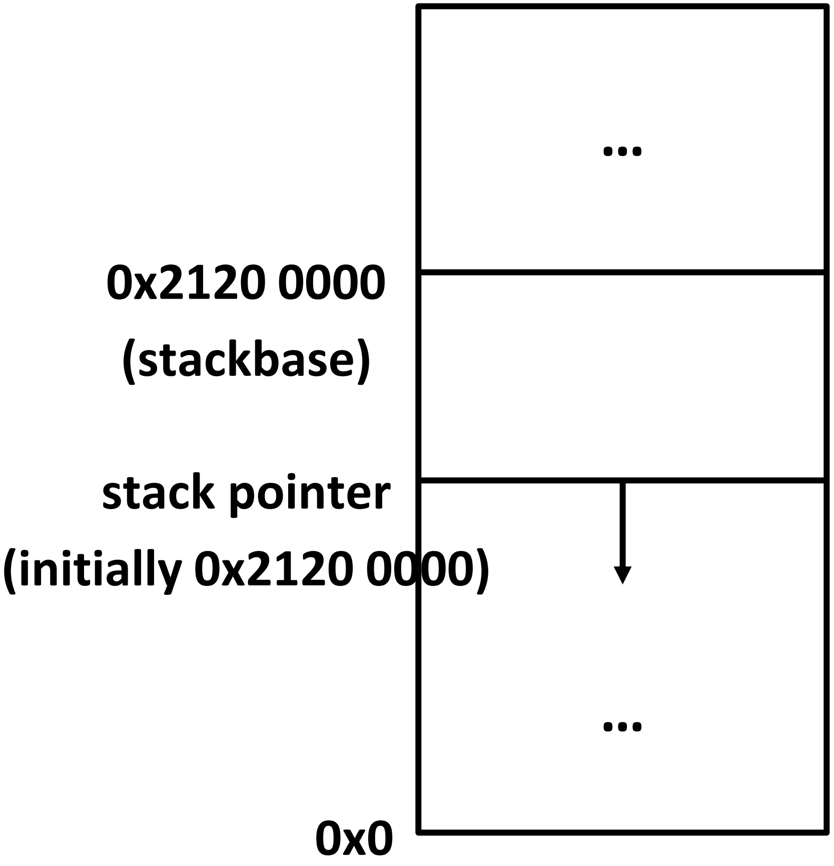
Update CPSR Register

```
msr r0, cpsr
…
bic r0, r0, vh_MODEMASK | vh_NOINT
orr r1, r0, vh_USERMODE | 0x00
msr cpsr_cxsf, r1
```

User Mode Stack Setting

Update Stack Pointer

```
ldr sp ,=vh_userstack
```



...

0x2120 0000

(stackbase)

stack pointer

(initially 0x2120 0000)

...

0x0

UART SETTING

Preparation for the VPOS kernel porting

1. Implement Startup code

2. UART Settings

3. TIMER Settings

4. Implement Hardware Interrupt Handler
   (1) UART Interrupt
   (2) Timer Interrupt

5. Implement Software Interrupt Entering/Leaving Routine

6. Kernel compile + load kernel image in RAM

Contents

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

MEMORY-MAPPED I/O

CPU I/O Device Access

- **To use input/output devices (I/O devices)**
  - Must access memory or registers in I/O device
    - Writing or reading values in registers or memory
    - Can transfer data to or receive data from I/O devices

- **CPU requires two address spaces due to I/O devices**
  - I/O
  - Memory

- **Should we separate or integrate I/O address space and memory address space?**
  - In one memory space ➔ Memory-mapped I/O
  - In different address spaces ➔ Port-mapped I/O

Port-mapped I/O

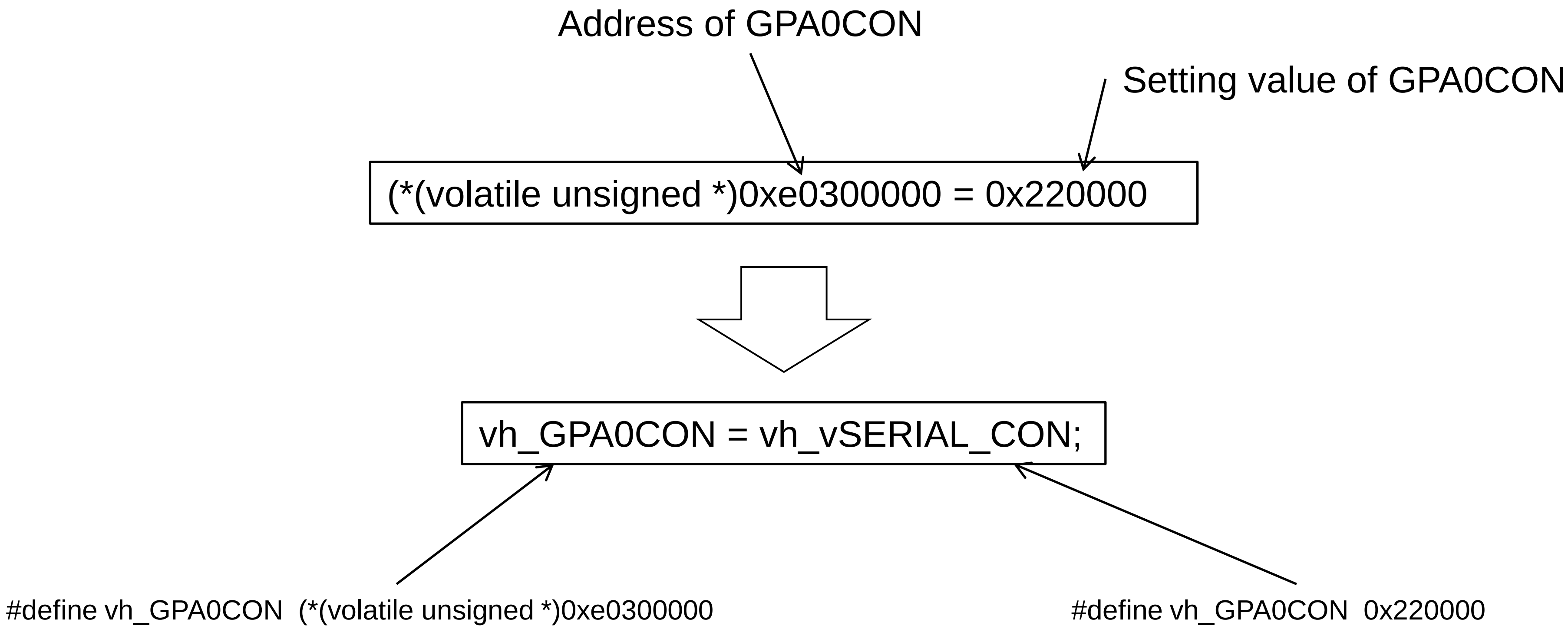
- **Definition**
  - – I/O-mapped I/O
  - – Separate memory and I/O address space
  - – There are special machine instructions to access the I/O address space.
    - IN, OUT commands (8085)
  - – Mainly used on Intel microprocessors (such as x86)

Memory-mapped I/O

- **Definition**
  - Place I/O and memory address space in a single memory space
  - Treat memory and register of I/O device as part of whole memory
    - Allocate and place a specific area of memory
  - When the CPU accesses I/O registers, it accesses a specific address in memory

- **Example**
  - ULCON1 : UART Line Control Register
    - Assign the ULCON1 register to memory address 0xec000400
    - Store data at 0xec000400 → Store data in ULCON1
    - Read data at 0xec000400 → Read data from ULCON1

Memory-mapped I/O : Example



Address of GPA0CON

Setting value of GPA0CON

```
(*(volatile unsigned *)0xe0300000 = 0x220000
```

```
vh_GPA0CON = vh_vSERIAL_CON;
```

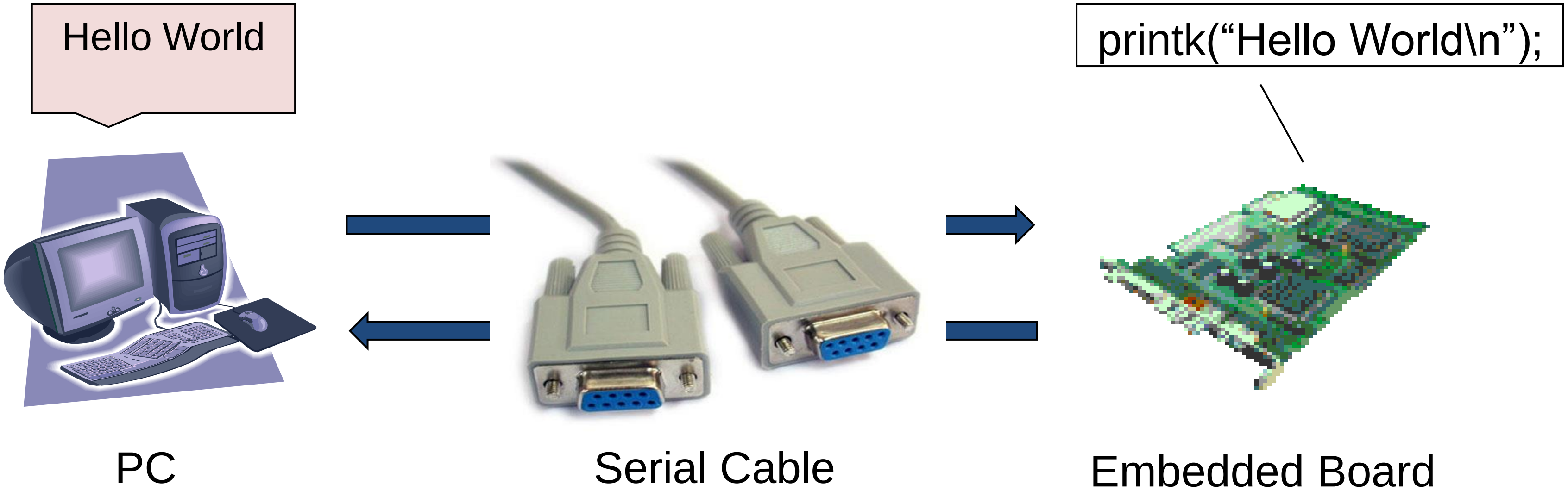
```
#define vh_GPA0CON  (*(volatile unsigned *)0xe0300000
```

```
#define vh_GPA0CON  0x220000
```

UART

Data transfer between embedded board and PC

- **Board → PC**
  - Receive data and show on PC's display
- **PC → Board**
  - Get keyboard input and execute shell command on embedded board



Hello World

printk(“Hello World\n”);

PC

Serial Cable

Embedded Board

Communication method

- **Classification by data transmission unit**
  - Parallel Communication
    - Multiple parallel channels are used to simultaneously transmit and receive multiple data bits
    - Used when exchanging information inside the computer
    - High-speed communication speed, processing a lot of information at once
    - Limit of communication distance, implementation difficulty, high cost
  - Serial Communication
    - One channel is used to send and receive data bits in units of one bit
    - Used when the computer communicates with an external device
    - Easy to implement, low cost
    - Modem, LAN, RS-232, etc.

Communication method

- **Classification of serial communication**
  - Synchronous communication
    - Synchronizes two devices to transmit and receive data at fixed timing
    - Even if there is no exchange of data, the signal for control exists, so that it synchronizes with the other party.
    - Transfer a predetermined number of data
  - Asynchronous communication
    - Transmitting data at random timing on the transmitting side without timing the devices to each other
    - When sending data, it adds a start bit and a stop bit to each end of the data.
    - Send and receive one character at a time (5 to 8 bits)
    - Mainly used for short data transfer such as keyboard input

UART

- **Data transfer on a computer**
  - Inside the computer, data is transmitted in parallel
    - Parallel transmission is only valid for very short distances
  - Use serial transmission to transfer data outside the computer
  - Requires a hardware device to change the signal when transmitting or receiving data
    - Transmit : parallel signal → serial signal
    - Receive : serial signal → parallel signal

- **UART (Universal Asynchronous Receiver Transmitter)**
  - When transmitting data, convert parallel bits to serial bits
  - When receiving data, convert serial bits to parallel bits

Function of UART

- **Convert a parallel signal to a serial signal, a serial signal to a parallel signal**

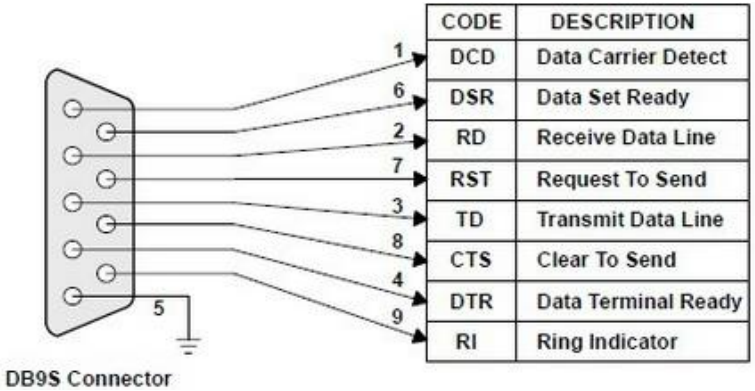
- **When transmitting data,**
  - Add start and stop bits
    - Check data reception via start bit
  - Add parity bit
    - Receiver detects the data error by checking the parity.

- **Handles incoming interrupts from the keyboard or mouse**

UART = RS-232?

- **RS-232**
  - Used when transmitting/receiving digital signal converted from UART to/from external device
  - An electrical signaling system that interfaces digital signals to the outside
  - As the UART's digital signal is small, it can not be sent far away and has high noise.
    - Data can be sent longer by increasing the voltage

    ➔ RS-232 (5V → 12V)

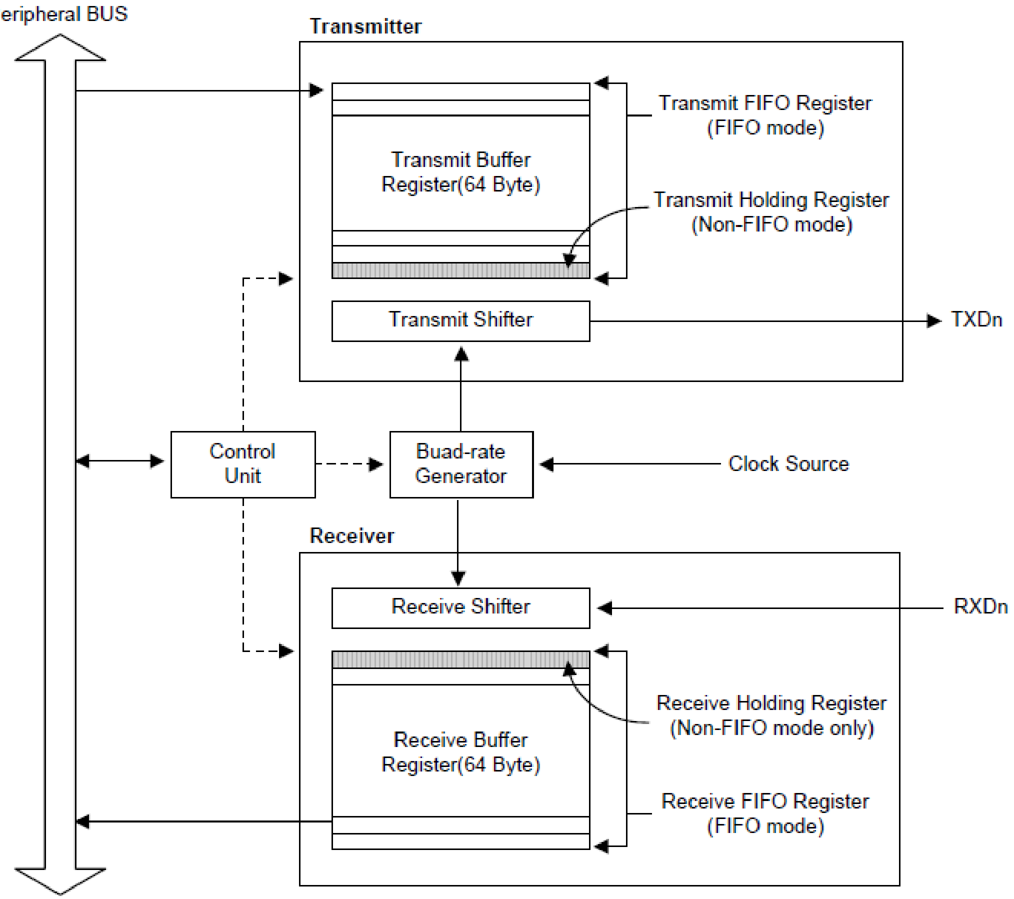


CODE	DESCRIPTION
DCD	Data Carrier Detect
DSR	Data Set Ready
RD	Receive Data Line
RST	Request To Send
TD	Transmit Data Line
CTS	Clear To Send
DTR	Data Terminal Ready
RI	Ring Indicator

RS-232 Connector

UART Block Diagram

- **UART of S5PC100 (Datasheet Page.626)**


Peripheral BUS
Transmitter
Transmit Buffer Register(64 Byte)
Transmit FIFO Register (FIFO mode)
Transmit Holding Register (Non-FIFO mode)
Transmit Shifter
TXDn
Control Unit
Buad-rate Generator
Clock Source
Receiver
Receive Shifter
RXDn
Receive Holding Register (Non-FIFO mode only)
Receive Buffer Register(64 Byte)
Receive FIFO Register (FIFO mode)

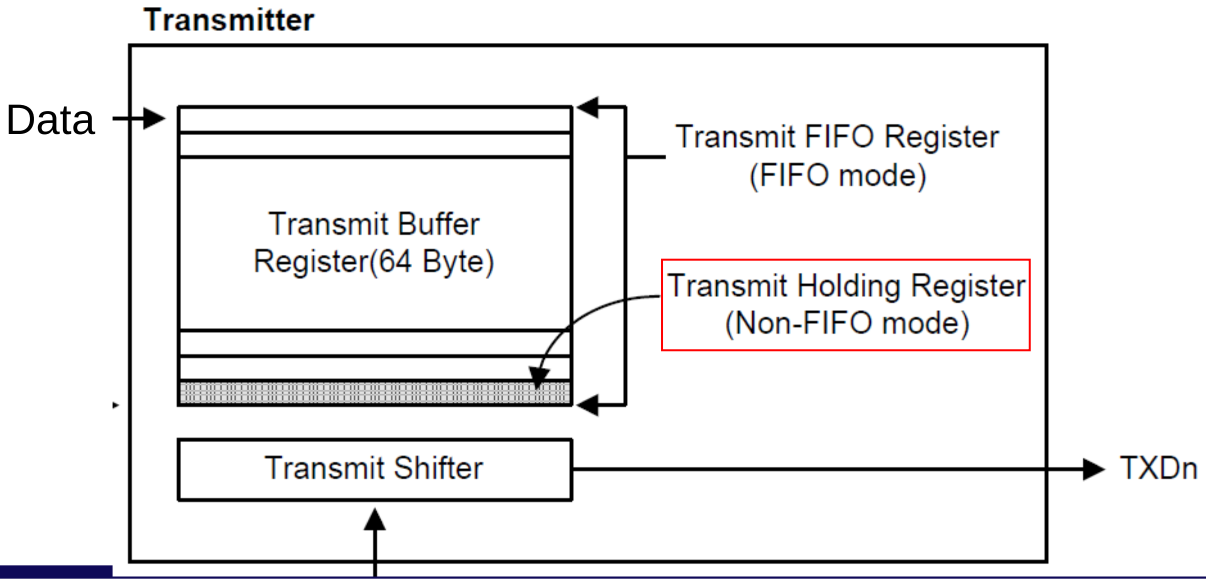
In FIFO mode, all 64 Byte of Buffer register are used as FIFO register.
In non-FIFO mode, only 1 Byte of Buffer register is used as Holding register.

Transmitter (TX)

- **Role**
  - Convert parallel data bits to serial data bits
  - Add start bit, stop bit, parity bit
- **Mode**
  - Non-FIFO Mode : Do not use buffers
  - FIFO Mode : Using buffers

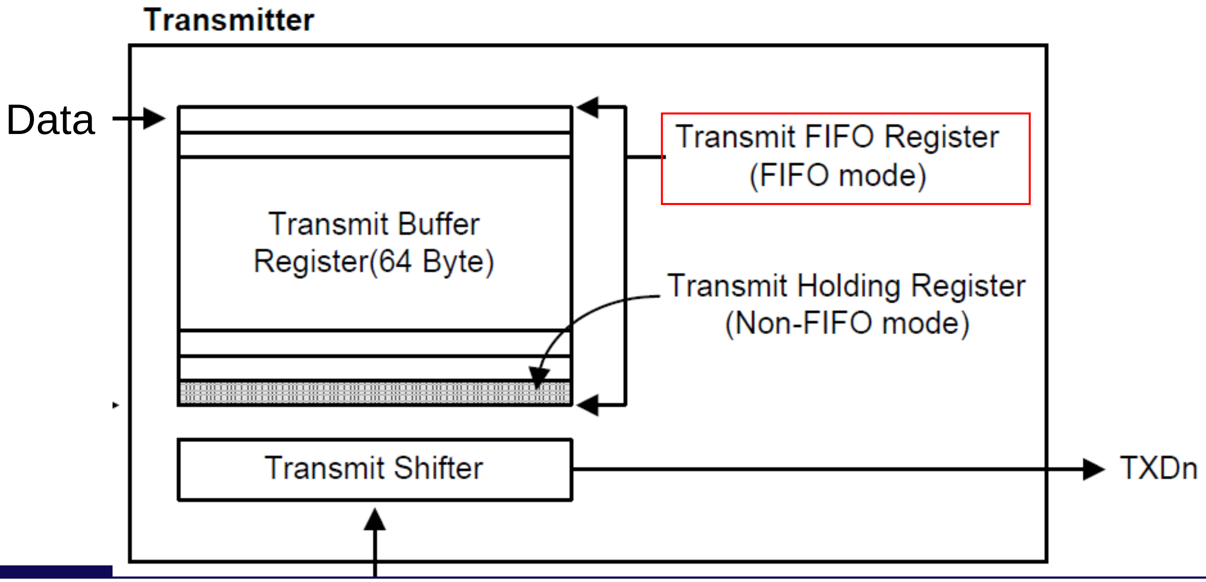
Mode of Transmitter (TX)

- **Non-FIFO**
  - Data is stored in the Transmit Holding Register(THR)
  - When the Transmit Shift Register (TSR) is empty, the data stored in the THR is transmitted to the TSR
  - The TSR shifts the data to the TX output pin



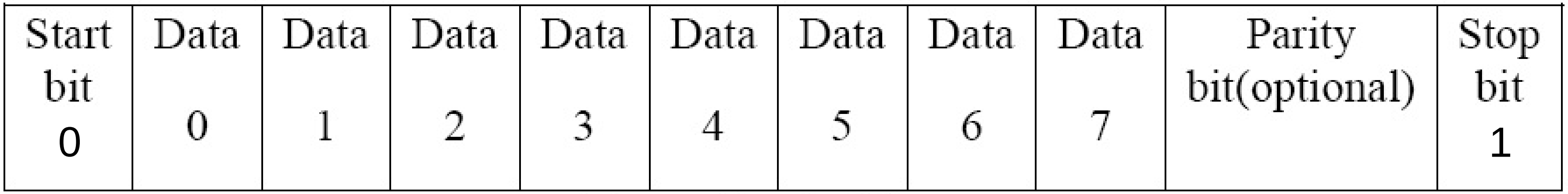
Mode of Transmitter (TX)

- **FIFO**
  - Data is stored in the Transmit Holding Register (THR)
  - The data to be transmitted is queued in the TX FIFO (queue).
  - When the TSR is empty, the TX FIFO data is transmitted to the Transmit Shift Register (TSR)
  - The TSR shifts the data to the TX output pin



Character Framing of Transmitter (TX)

- **Data Frame**
  - Start bit : 0
  - 5~8 Data bits
  - Parity bit
  - Stop bit : 1
    - Stop bit is 1 ~ 2 bits depending on hardware



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

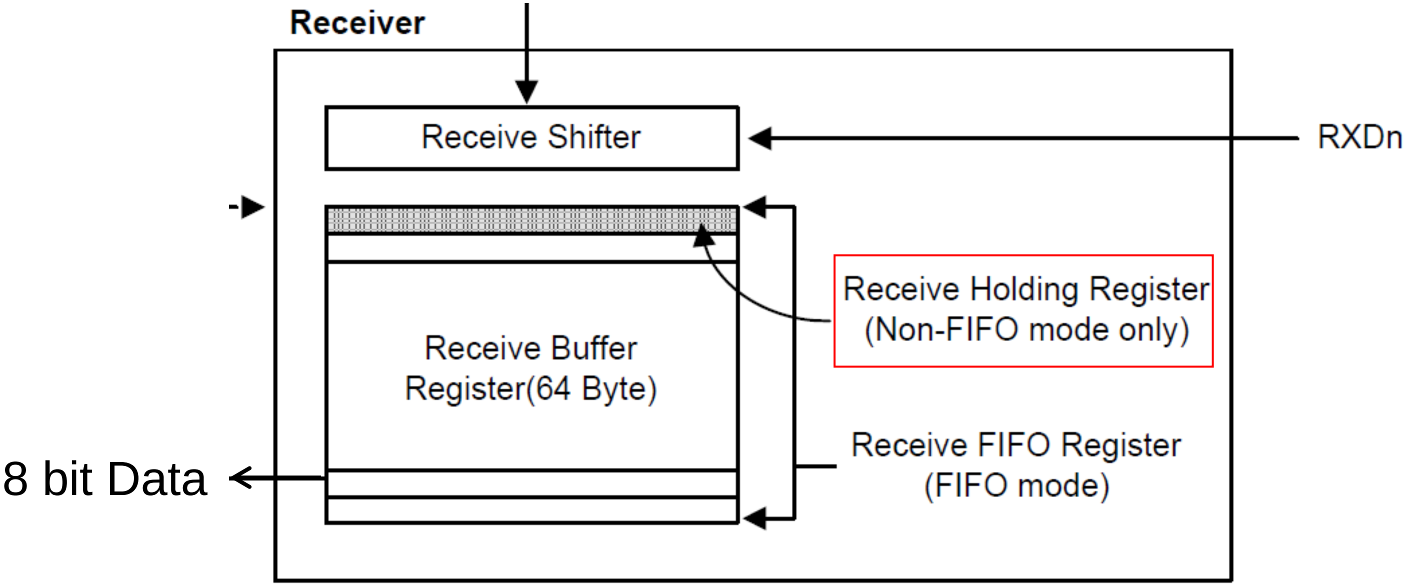
Receiver (RX)

- **Role**
  - Convert serial data bits to parallel data bits
  - Detect start bit and receive data

- **Mode**
  - Non-FIFO Mode : Do not use buffers
  - FIFO Mode : Using buffers

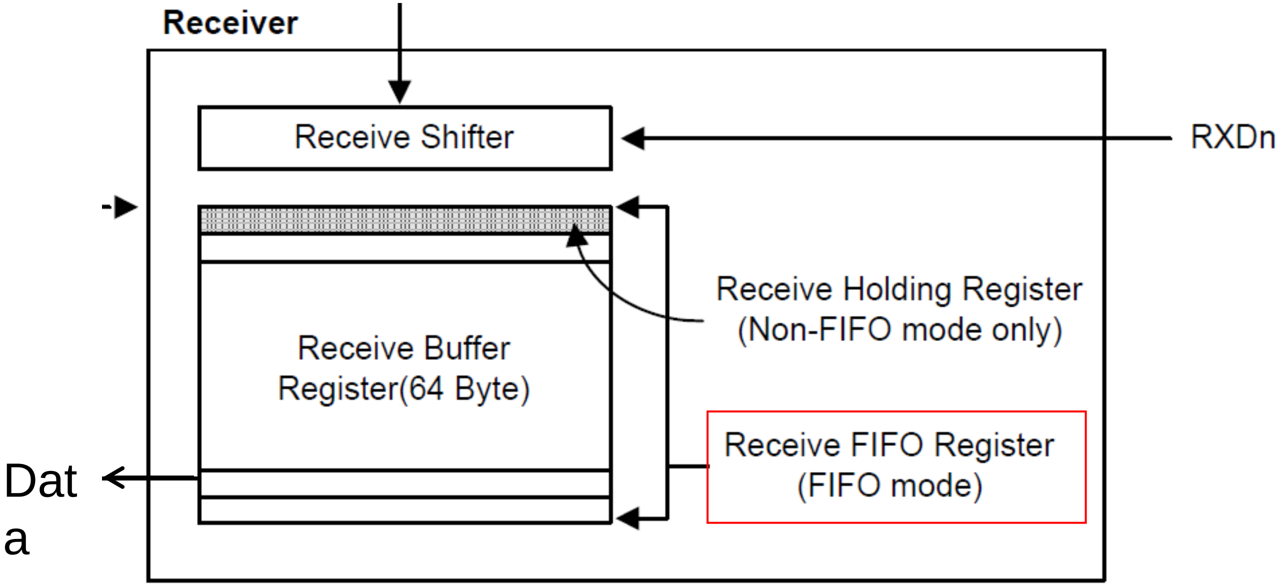
Mode of Receiver (RX)

- **Non-FIFO**
  - Receive external data from Receive Shift Register(RSR)
  - Send data to Receive Holding Register(RHR)
  - CPU reads RHR data



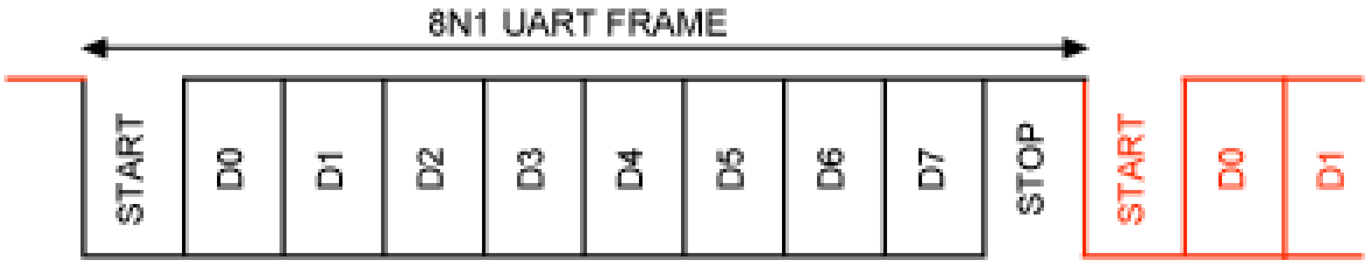
Mode of Receiver (RX)

- **FIFO**
  - Receive external data from Receive Shift Register(RSR)
  - Load data into the RX FIFO (Queue)
  - CPU reads data from Receive Holding Register (RHR)



Receiver (RX) Character Validation

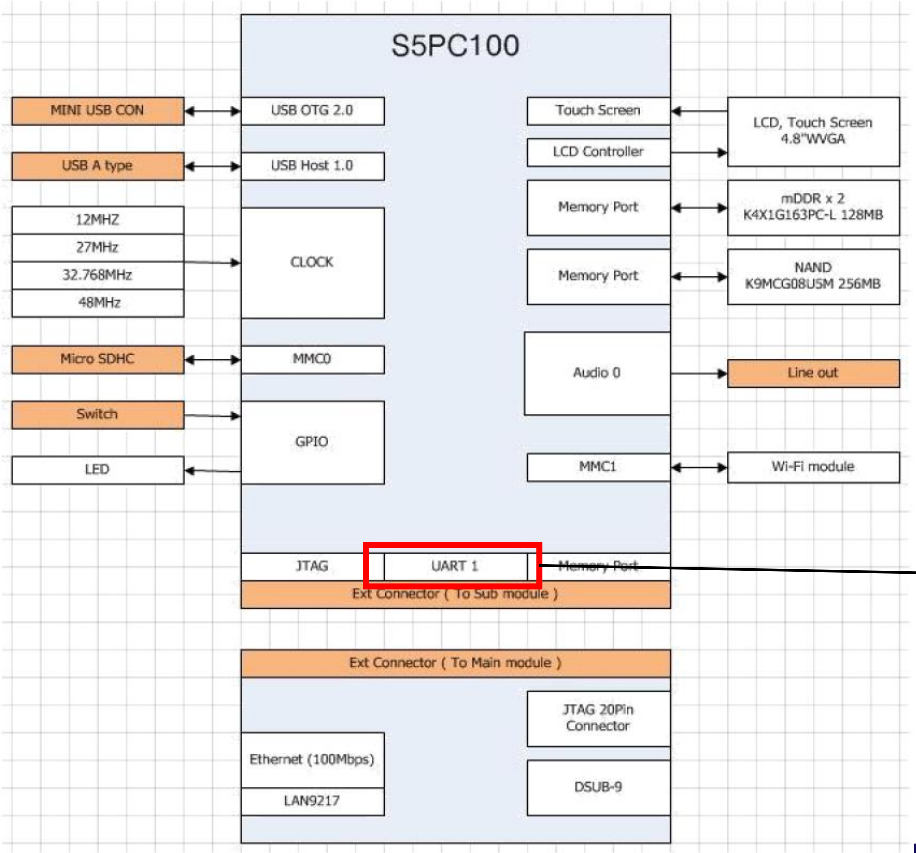
- **Detect start bit and stop bit**
  - HIGH → LOW : Start bit arrives at Receiver
  - guard time after LOW → HIGH : Stop bit arrives at Receiver
  - Data bit and parity bit exist between two bits



UART REGISTER SETTING

UART of S5PC100

- **S5PC100 Block Diagram**
  - Communication with PC using UART1 of S5PC100



Connect to D-SUB 9P

Hardware setup and control

- **How to control hardware with software?**

  ➔Use hardware registers

- **Registers to control the hardware**
  - Control register : Set up and control hardware
  - Status register : Check the current hardware status

UART control with software

- **Coding order**
  1. Enable UART1 pin 4 and 5 of GPIO A0 and set Full-Up
  2. Set control registers including ULCON
  3. Check bit 0(Receive) and bit 2(Transmit) of the UTRSTAT register.
  4. Write data to UTXH register or read data in URXH register

UART registers

- **Control registers**
  - ULCON
  - UCON
  - UFCON
  - UINTM
  - UINTP
  - UBRDIV

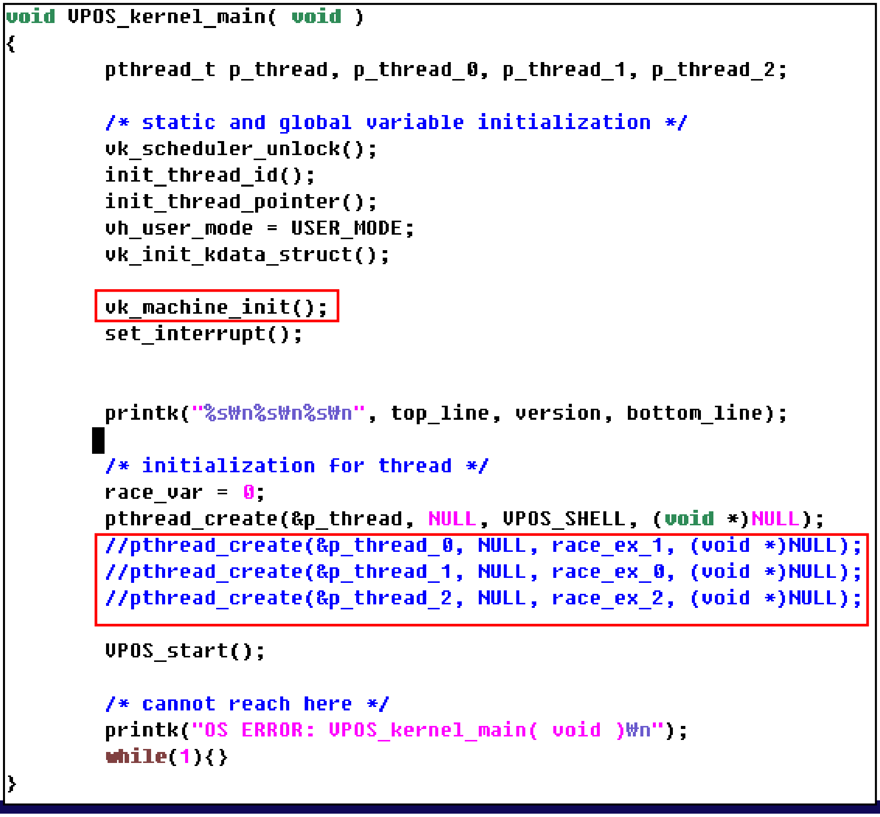
- **Status register**
  - UTRSTAT

- **Buffer registers**
  - UTXH
  - URXH

VPOS_kernel_main()

- **Functions**
  - Initialize the VPOS kernel data structure
  - Initialize hardware such as serial device and timer
  - Enable interrupt
  - Print boot message
  - Create a shell thread
  - Enter scheduler calling
    VPOS_start routine

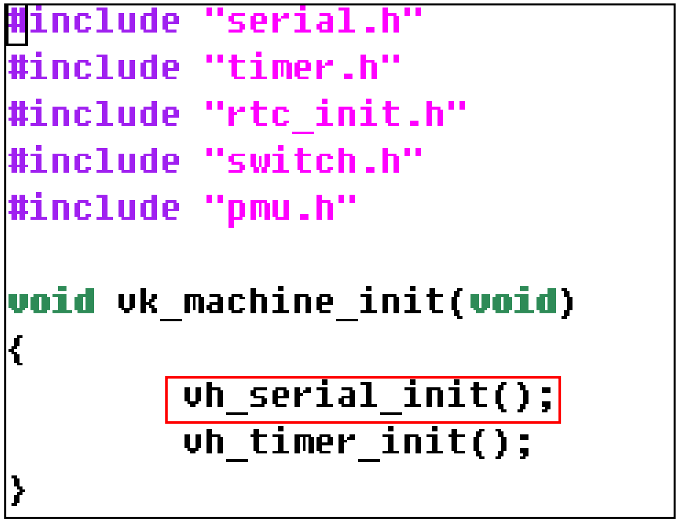
- **Source code location**
  - vpos/kernel/kernel.start.c



```
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, 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();

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

vk_machine_init()

- **Code**
  - Initialize hardware device
  - vh_serial_init() : Initialize UART
  - vh_timer_init() : Initialize Timer



```
#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();
}
```

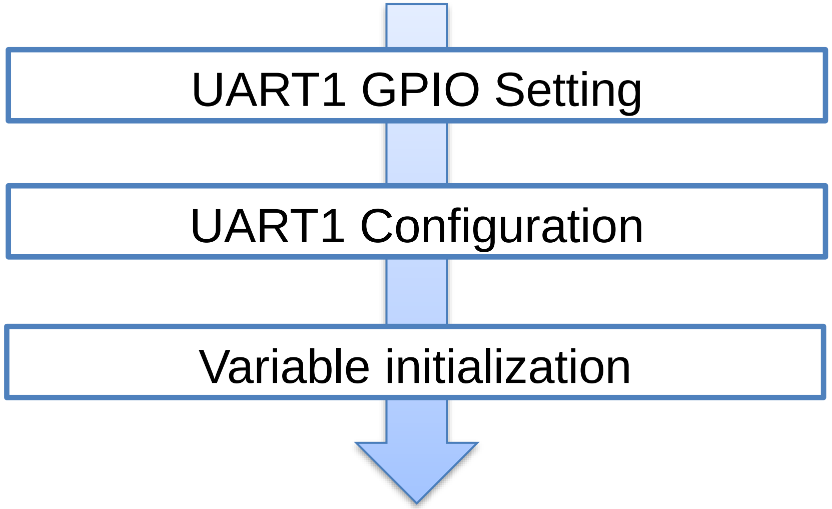
- **Source code location**
  - vpos/kernel/machine_init.c

UART-related files

- **UART Source Code**
  - vpos/hal/io/serial.c
    - Initialize UART
    - Data Write(putc), Read(getc)
    - UART Interrupt Handler
  - vpos/hal/include/vh_io_hal.h
    - Define address of UART related register
    - Define register-related setting values
    - Define TX/RX Operation

vh_serial_init()

- **Initialize UART1**
  - S5PC100 has 4 UART channels
  - S5PC100 communicates with PC through UART1



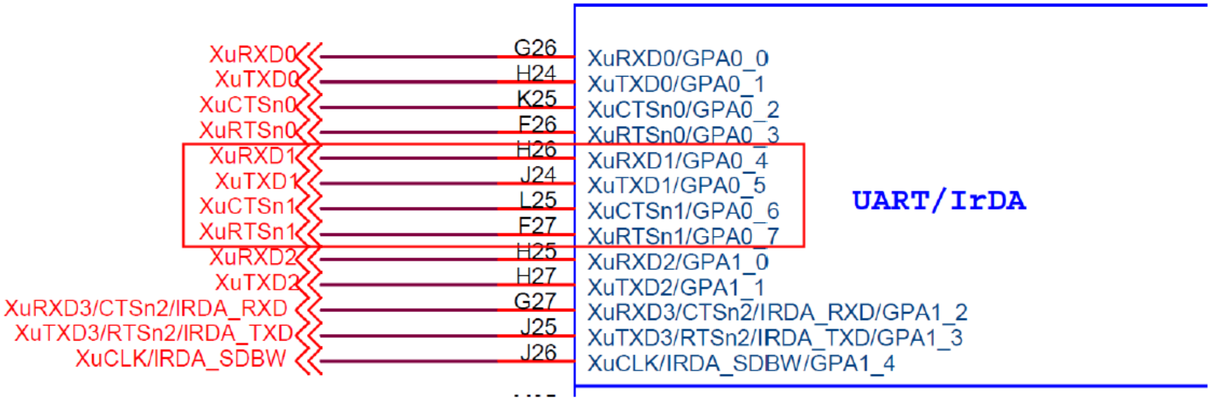
UART1 GPIO Setting

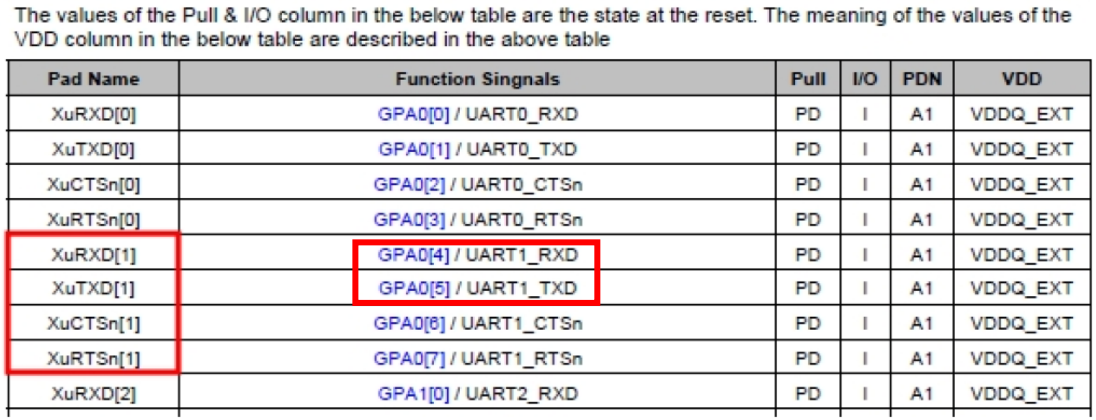
UART1 Configuration

Variable initialization

UART1 GPIO Setting

- **GPIO MAP**





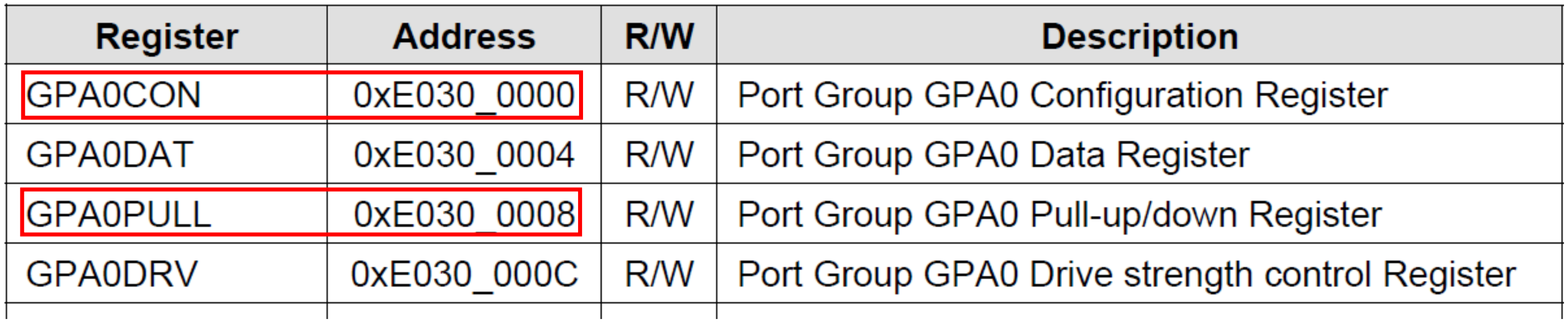
The values of the Pull & I/O column in the below table are the state at the reset. The meaning of the values of the VDD column in the below table are described in the above table

Pad Name	Function Singnals	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

- **Related registers and setting method**
  - Connect pin 4 and pin 5 to UART1 on GPA0CON
  - Pull-Up Enable Pins 4 and 5 in GPA0PULL

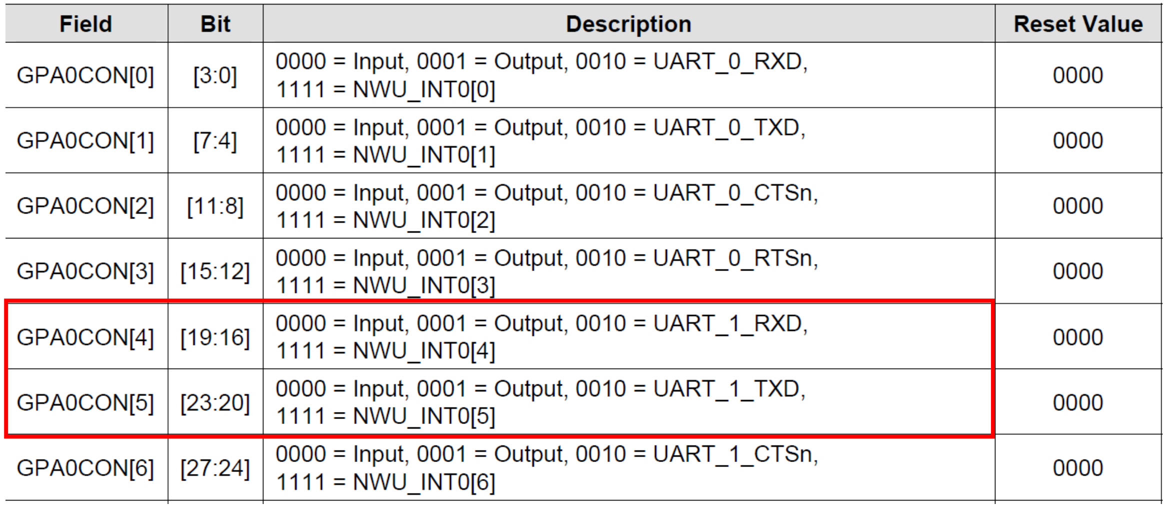


Register	Address	R/W	Description
GPA0CON	0xE030_0000	R/W	Port Group GPA0 Configuration Register
GPA0DAT	0xE030_0004	R/W	Port Group GPA0 Data Register
GPA0PULL	0xE030_0008	R/W	Port Group GPA0 Pull-up/down Register
GPA0DRV	0xE030_000C	R/W	Port Group GPA0 Drive strength control Register

Datasheet Page. 70

UART1 GPIO Setting

- **GPA0CON**
  - Connects pin 4 and 5 of GPIO A0 port to RX and TX of UART1
  - GPA0CON[4] = Config Register Field on Pin 4
  - GPA0CON[5] = Config Register Field on Pin 5
  - To connect to UART, set 0010 value in the corresponding field.
  - ➔ 0x220000

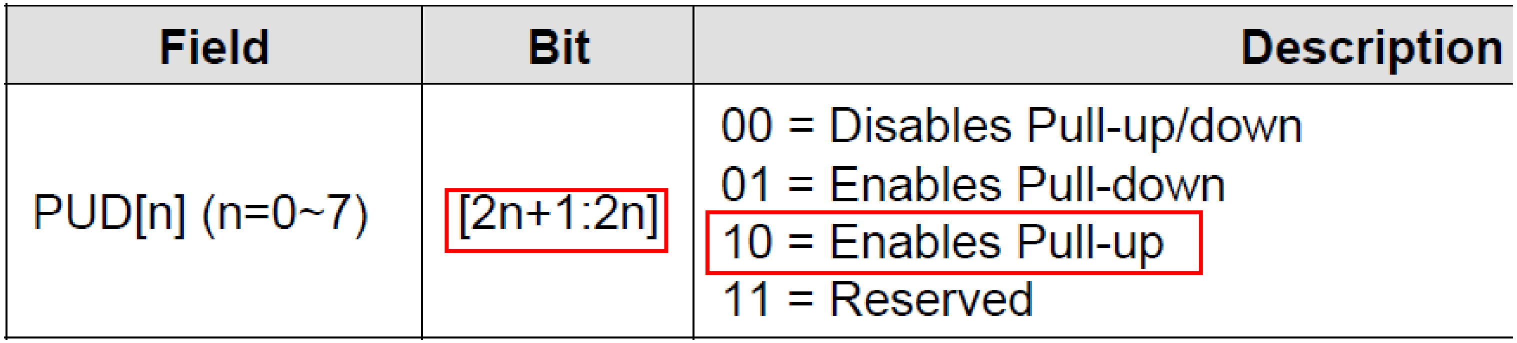


Field	Bit	Description	Reset Value
GPA0CON[0]	[3:0]	0000 = Input, 0001 = Output, 0010 = UART_0_RXD, 1111 = NWU_INT0[0]	0000
GPA0CON[1]	[7:4]	0000 = Input, 0001 = Output, 0010 = UART_0_TXD, 1111 = NWU_INT0[1]	0000
GPA0CON[2]	[11:8]	0000 = Input, 0001 = Output, 0010 = UART_0_CTSn, 1111 = NWU_INT0[2]	0000
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] = Pull up/down of pin 4
  - PUD[11:10] = Pull up/down of pin 5
  - Pull-up the pin 4 and 5 of GPIO A0 port
  - ➔ 0xa00

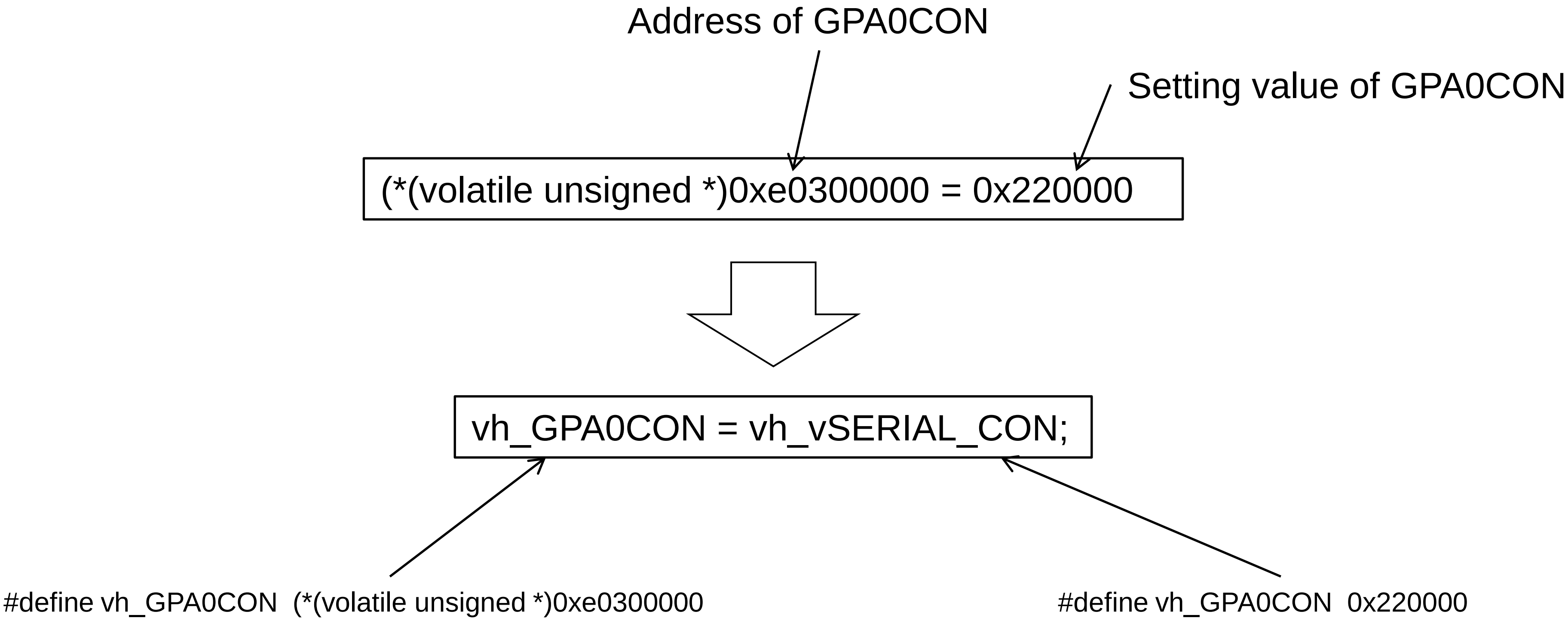


Field	Bit	Description
PUD[n] (n=0~7)	[2n+1:2n]	00 = Disables Pull-up/down 01 = Enables Pull-down 10 = Enables Pull-up 11 = Reserved

Datasheet Page. 82

Register access in C code

- **Register access method**



Address of GPA0CON

Setting value of GPA0CON

```
(*(volatile unsigned *)0xe0300000 = 0x220000
```

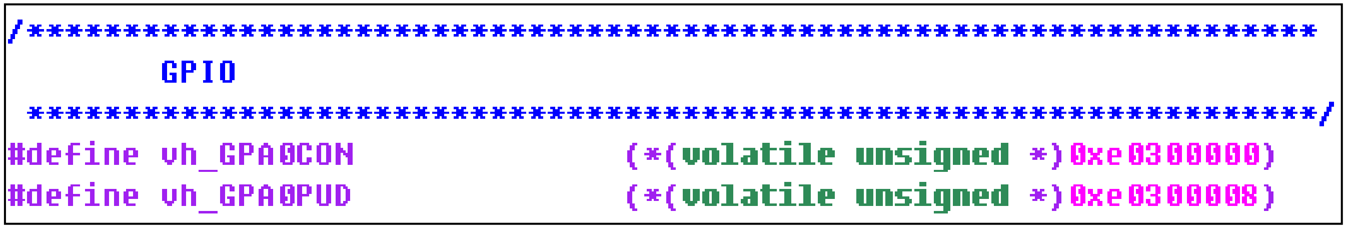
```
vh_GPA0CON = vh_vSERIAL_CON;
```

```
#define vh_GPA0CON  (*(volatile unsigned *)0xe0300000
```

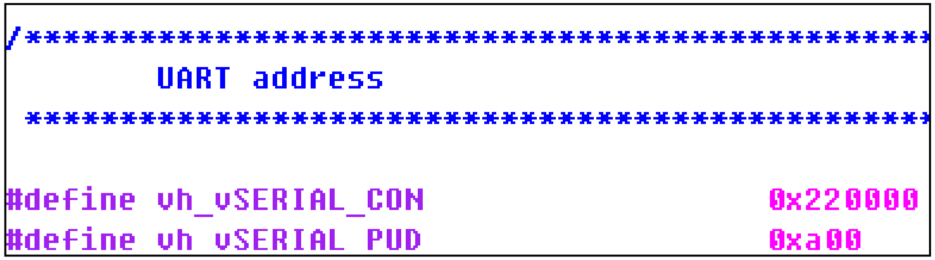
```
#define vh_GPA0CON  0x220000
```

UART1 GPIO Setting

- **Code**
  - vpos/hal/include/vh_io_hal.h (GPIO, UART)

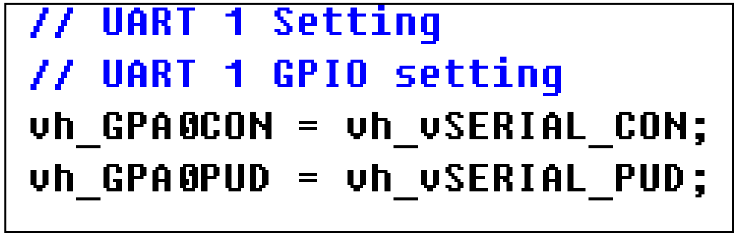


```
/********************************************************************
        GPIO
 ********************************************************************/
#define vh_GPA0CON              (*(volatile unsigned *)0xe0300000)
#define vh_GPA0PUD              (*(volatile unsigned *)0xe0300008)
```



```
/***********************************************
        UART address
 ***********************************************

#define vh_vSERIAL_CON                  0x220000
#define vh_vSERIAL_PUD                  0xa00
```

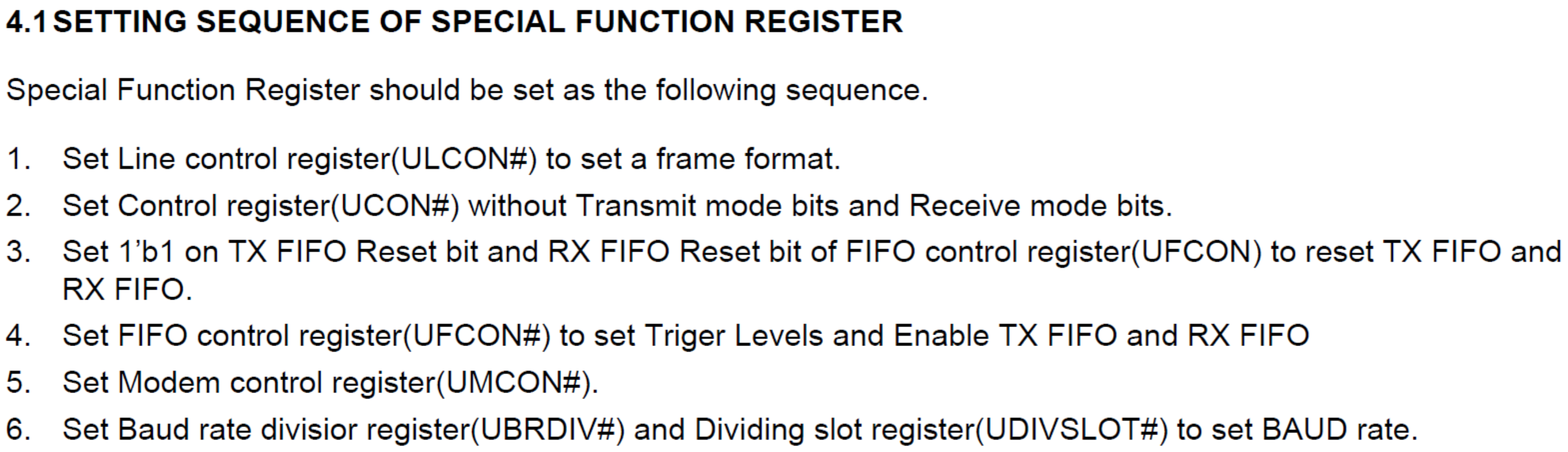
  - vpos/hal/io/serial.c → vh_serial_init()



```
// 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

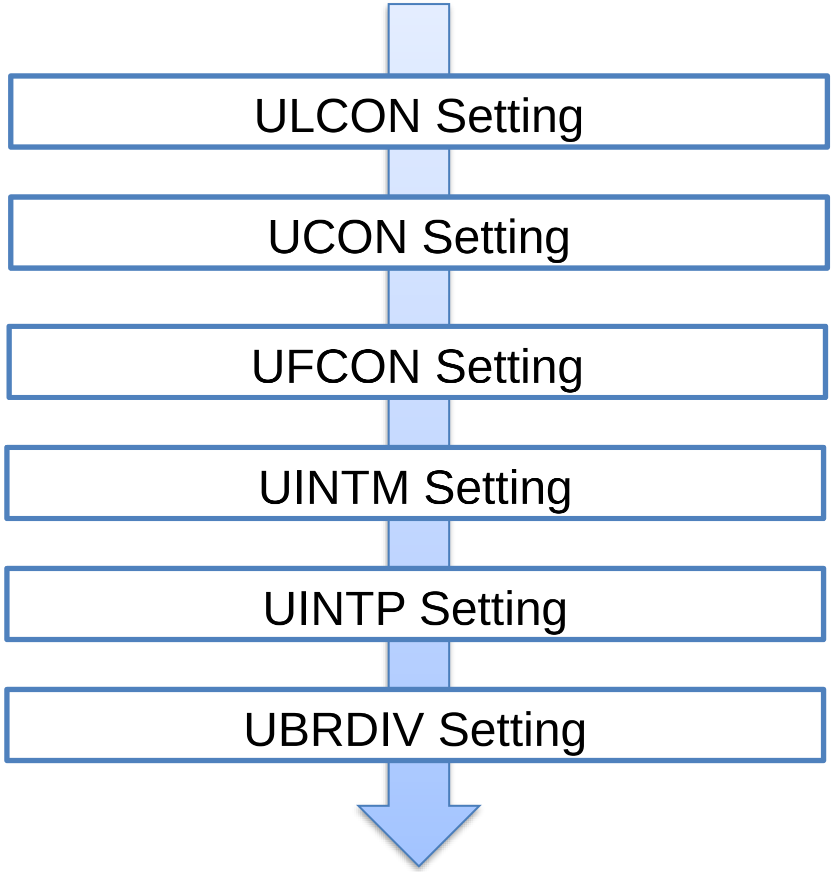
Special Function Register should be set as the following sequence.

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 Triger Levels and Enable TX FIFO and RX FIFO
5. Set Modem control register(UMCON#).
6. Set Baud rate divisior register(UBRDIV#) and Dividing slot register(UDIVSLOT#) to set BAUD rate.

Datasheet Page. 633

UART1 Configuration

- **Setting sequence**



ULCON Setting

UCON Setting

UFCON Setting

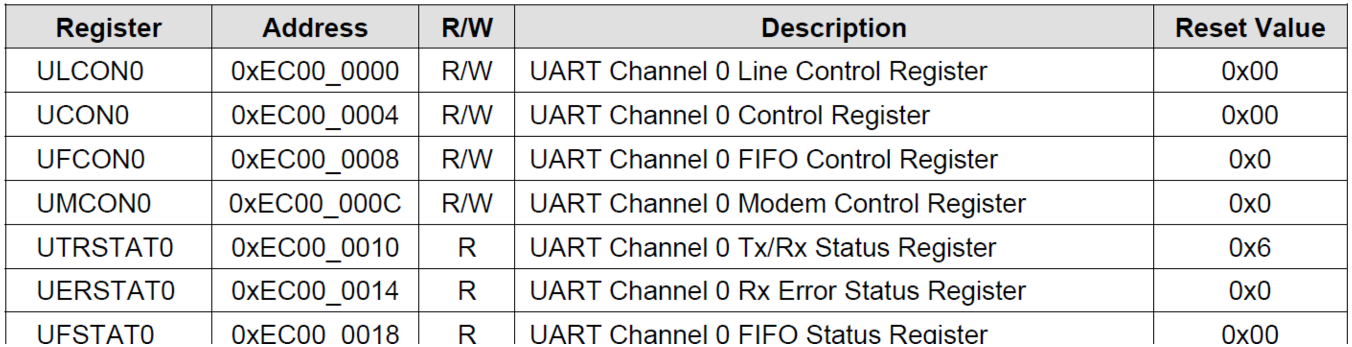
UINTM Setting

UINTP Setting

UBRDIV Setting

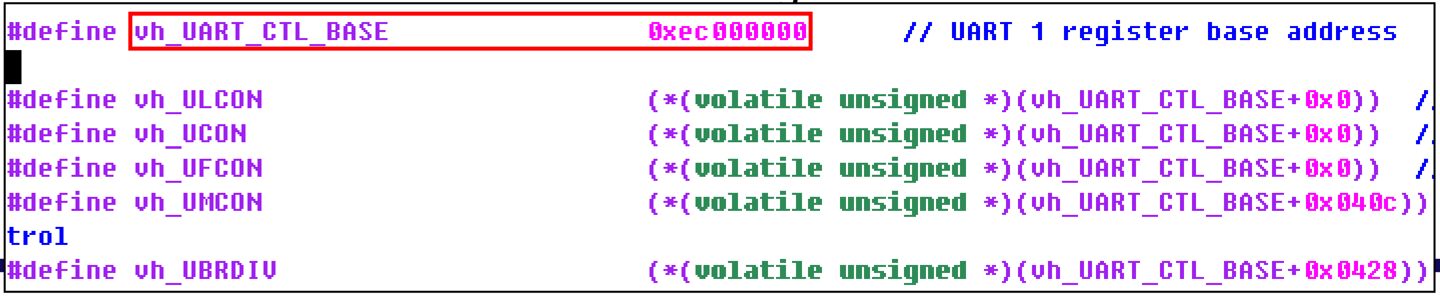
Before setting the UART

- **Define UART Register Base Address**
  - Define vh_UART_CTL_BASE in vh_io_hal.h
  - Then register address of UART is defined as [Base address + Offset]
  - The UART related registers are located in the order of 0xec000000
    - vh_UART_CTL_BASE is 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

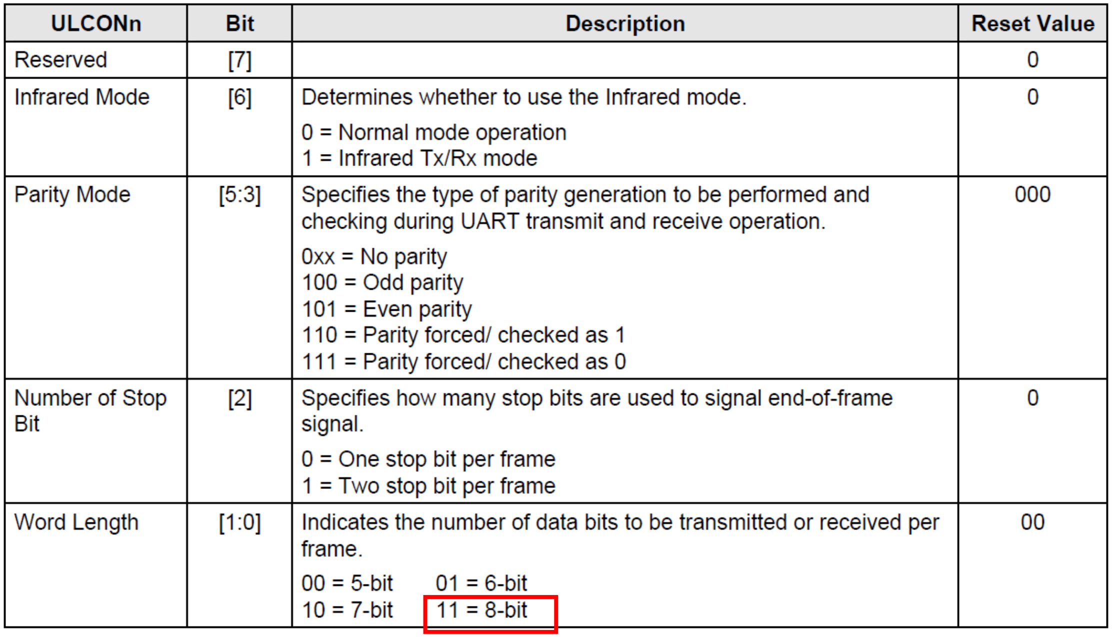
Datasheet Page. 82



```
#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

- **Register : ULCON1 (0xec000400)**
  - UART Line Control Register
    - Data frame setting to transmit/receive UART
- **How to set up**
  - 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

- **Register : UCON1 (0xec000404)**
  - UART Control Register

- **How to set up (Page.638)**
  - Transmit Mode[3:2] & Receive Mode[1:0]
    - Set to interrupt request or polling mode (01)
  - Rx Error Status Interrupt Enable[6]
    - Enable error status interrupt (1)
  - Rx Interrupt Type[8]
    - Set to Pulse (0)
      - In S5PC100, Rx Interrupt Type is set to Pulse unconditionally
  - Tx Interrupt Type[9]
    - Set to level (1)
  - Clock Selection[11:10]
    - Used as PCLK (00)
      - 66MHz

UFCON Setting

- **Register : UFCON1 (0xec000408)**
  - UART FIFO control register
    - Setup FIFO Mode

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

- **Trigger Level determines when interrupt occurs**
  - "Interrupt occurs if there is X byte or less data in FIFO"
    - X byte → Trigger Level

UINTM Setting

- **Register : UINTM1 (0xec000438)**
  - UART Interrupt Mask Register
    - Mask interrupt generated by UART
- **How to set up**
  - Receive Interrupt only and mask the rest.
    - Bit 0 is set to 0 and [3: 1] is set to 1

UINTP Setting

- **Register : UINTP1 (0xec000430)**
  - UART Interrupt Pending Register
    - Has UART Interrupt information

- **How to set up**
  - Generate all interrupts
    - Set all [3: 0] bits to 1

UBRDIV Setting

- **Register : UBRDIV1 (0xec000428)**
  - UART Baud Rate Divisor Register
    - Save Baud rate division value

- **Baud rate**
  - Baud rate : Data transmission speed. Bits per second
  - The baud rate on the transmitting and receiving sides must be the same.
  - Equation of baud rate divisor

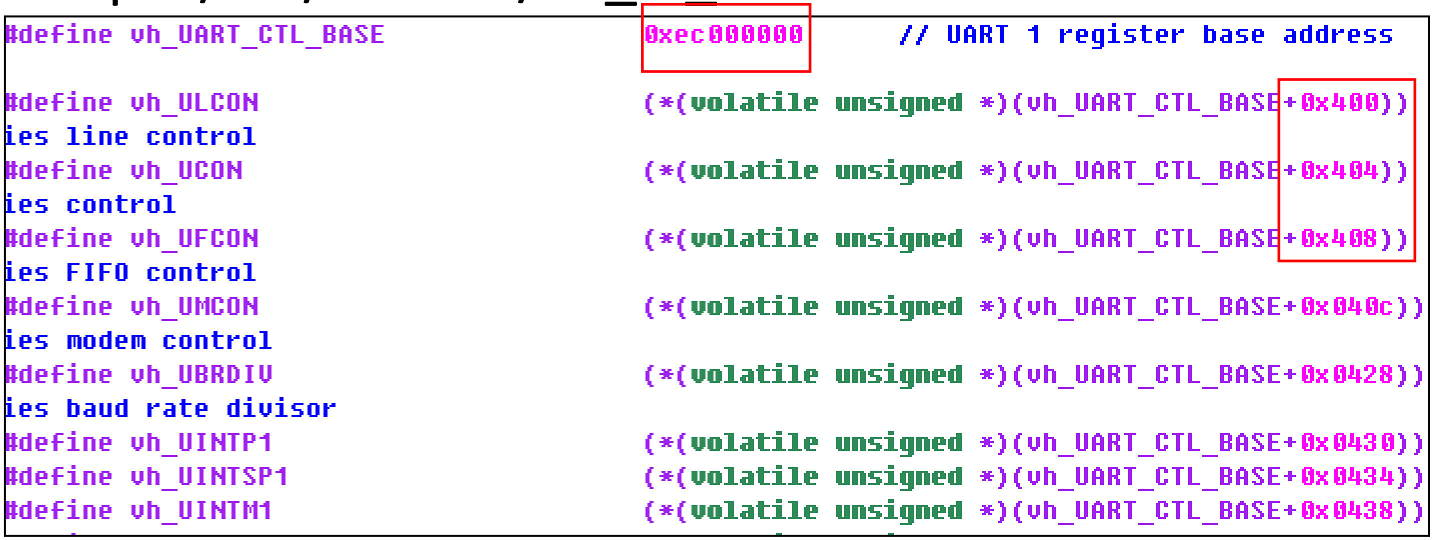
$$\boldsymbol{Divisor = \frac{PCLK}{Baud\ Rate * 16} - 1}$$

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

- **How to set up**
  - Save Divisor to UBRDIV

UART Register Setting

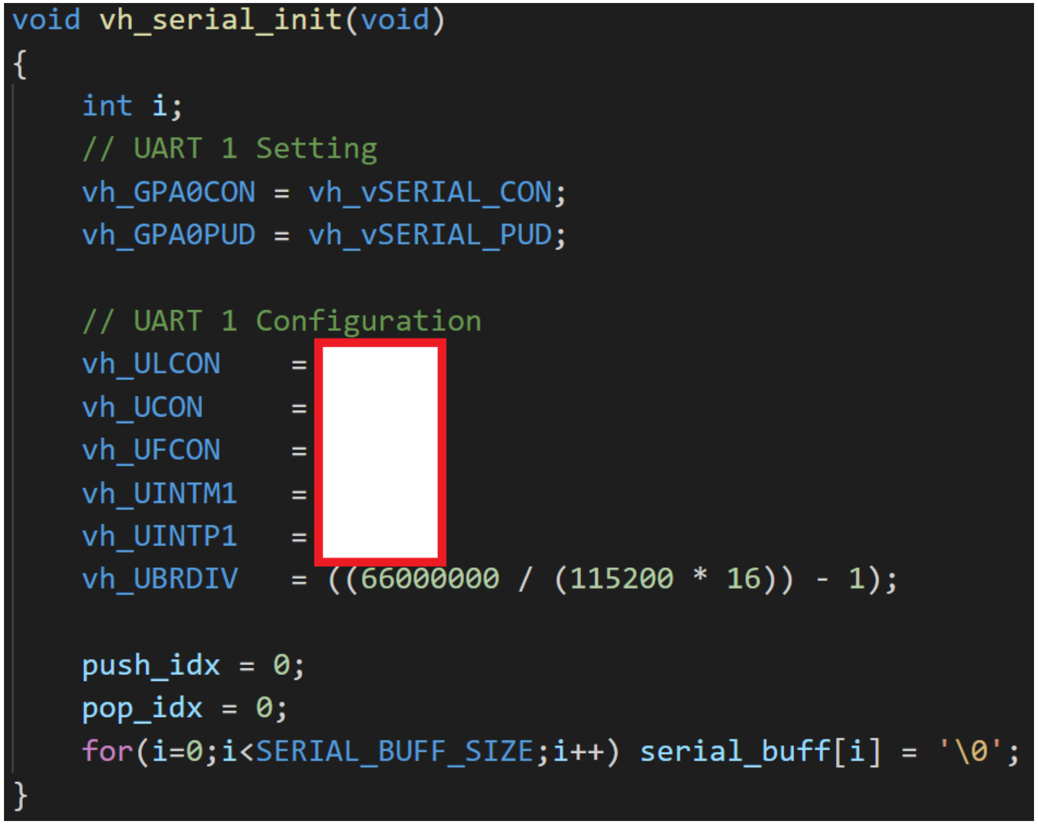
- **Code**
  - vpos/hal/include/vh_io_hal.h



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

#define vh_ULCON                          (*(volatile unsigned *)(vh_UART_CTL_BASE+0x400))
ies line control
#define vh_UCON                           (*(volatile unsigned *)(vh_UART_CTL_BASE+0x404))
ies control
#define vh_UFCON                          (*(volatile unsigned *)(vh_UART_CTL_BASE+0x408))
ies FIFO control
#define vh_UMCON                          (*(volatile unsigned *)(vh_UART_CTL_BASE+0x040c))
ies modem control
#define vh_UBRDIV                         (*(volatile unsigned *)(vh_UART_CTL_BASE+0x0428))
ies baud rate divisor
#define vh_UINTP1                         (*(volatile unsigned *)(vh_UART_CTL_BASE+0x0430))
#define vh_UINTSP1                        (*(volatile unsigned *)(vh_UART_CTL_BASE+0x0434))
#define vh_UINTM1                         (*(volatile unsigned *)(vh_UART_CTL_BASE+0x0438))
```

UART Register Setting

- **Code**
  - vpos/hal/io/serial.c



```
void vh_serial_init(void)
{
    int i;
    // UART 1 Setting
    vh_GPA0CON = vh_vSERIAL_CON;
    vh_GPA0PUD = vh_vSERIAL_PUD;

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

    push_idx = 0;
    pop_idx = 0;
    for(i=0;i<SERIAL_BUFF_SIZE;i++) serial_buff[i] = '\0';
}
```

UART Register Setting

- **What we modified**

  - vpos/kernel/kernel_start.c
    - Uncomment 3 of 4 pthread_create (see p41)
  - vpos/hal/include/vh_io_hal.h
    - vh_vSERIAL_CON
    - vh_vSERIAL_PUD
    - vh_UART_CTL_BASE
    - vh_ULCON
    - vh_UCON
    - vh_UFCON
  - vpos/hal/io/serial.c
    - modify vh_serial_init()

UART DATA TRANSMIT & RECEIVE

Data Transmit

- **Data : S5PC100 → Host PC**
  - Output character data of S5PC100 to minicom screen of host PC

  → printk()

- **printk()**
  - Functions to display letters, numbers, etc. on screen
  - Call putc(c)
    - Located in serial.c
  - The code of putc(c) is defined as vh_SERIAL_PUTC(c)
    - Located in vh_io_hal.h

vh_SERIAL_PUTC(c)

- **Code**

```
while (!vh_SERIAL_WRITE_READY());

vh_SERIAL_WRITE_CHAR(c);
```

- **vh_SERIAL_WRITE_READY()**
  - Check the UTRSTAT1 register to make sure that the buffer and shift register of the current transmitter are empty
  - UTRSTAT1 : UART TX/RX Status Register

- **vh_SERIAL_WRITE_CHAR(c)**
  - Input data into UTXH1 register
  - UTXH1 : UART Transmit Buffer Register

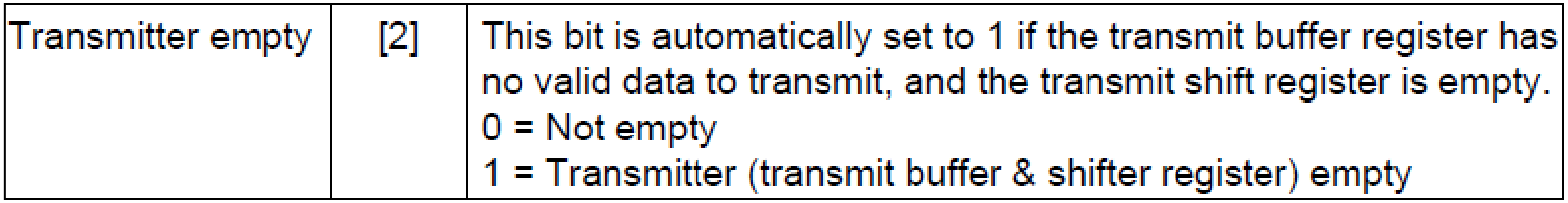
vh_SERIAL_PUTC(c)

- **Check status register → Write data to register**

- **Code**

```
while (!((vh_UTRSTAT1) & vh_UTRSTAT_TX_EMPTY));

vh_UTXH1 = c;
```

(1 << 2) : 2nd bit



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

- **This type of code can also be seen in Linux device drivers**

Data Receive

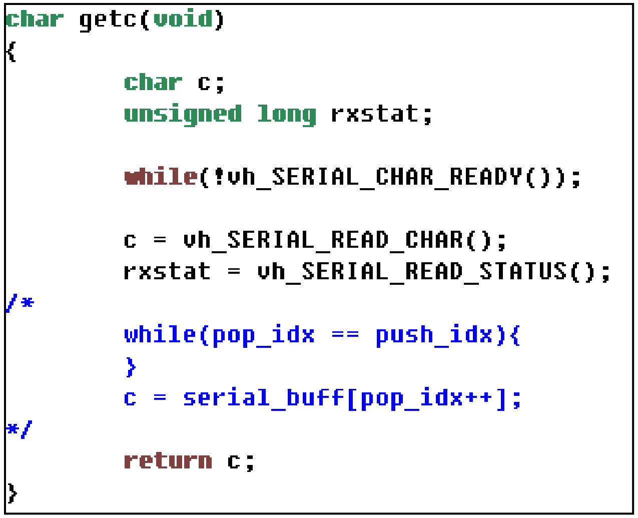
- **Data : Host PC → S5PC100**
  - Keyboard input of PC is transmitted to S5PC100
  - Polling and Interrupt must be selected when receiving data.
    - This week, we use polling

- **getc()**
  - Shell receives keyboard character via getc()
  - getc() code is located in serial.c

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()**
  - Check the UTRSTAT1 register and check if there is data in the buffer register of Receiver now
  - UTRSTAT1 : UART TX/RX Status Register

- **vh_SERIAL_READ_CHAR()**
  - Points to the URXH1 register
  - URXH1 : UART Receive Buffer Register

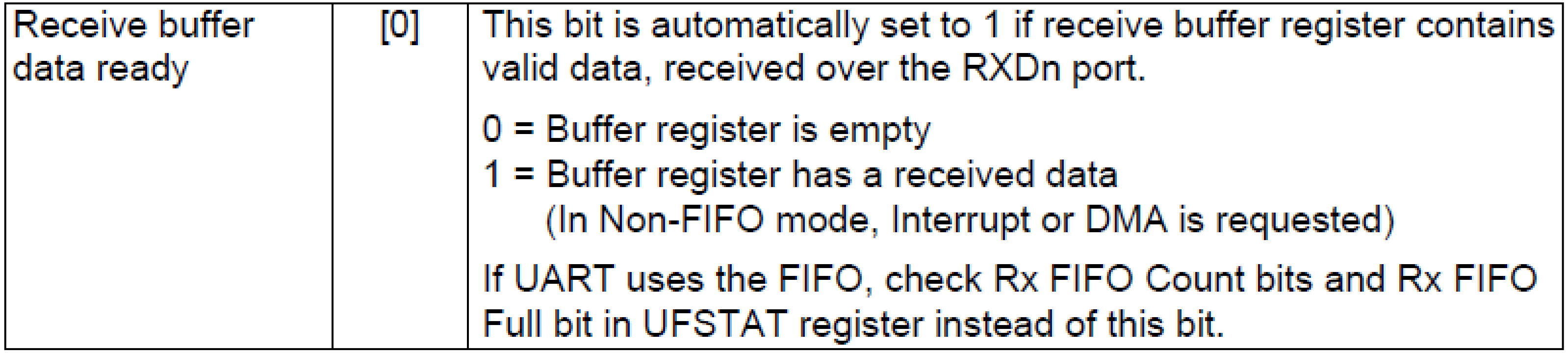
Data Receive

- **Check the status register → Read the data in the register**

- **Code**

```
while (!(vh_UTRSTAT1 & vh_UTRSTAT_RX_READY));

c = vh_URXH1;
```

(1 << 0) : bit 0



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

TIMER

Preparation for the VPOS kernel porting

1. **Implement Startup code**

2. **UART Settings**

3. **TIMER Settings**

4. **Implement Hardware Interrupt Handler**
   (1) UART Interrupt
   (2) Timer Interrupt

5. **Implement Software Interrupt Entering/Leaving Routine**

6. **Kernel compile + load kernel image in RAM**

Contents

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

Timer

- **What is timer?**
  - Counts up or down at regular intervals over time
  - A hardware device that generates an interrupt when a predetermined count is reached
  - Used to generate an interrupt at a desired cycle

- **Timer example**
  - Watchdog Timer
  - PWM Timer
  - System Timer
  - …

Basic Elements of Timer

- **Clock Source**
  - Timer needs Clock Source
  - Clock Source, prescaler, and divider make the desired count cycle
    - S5PC100 using PCLK(66MHz)

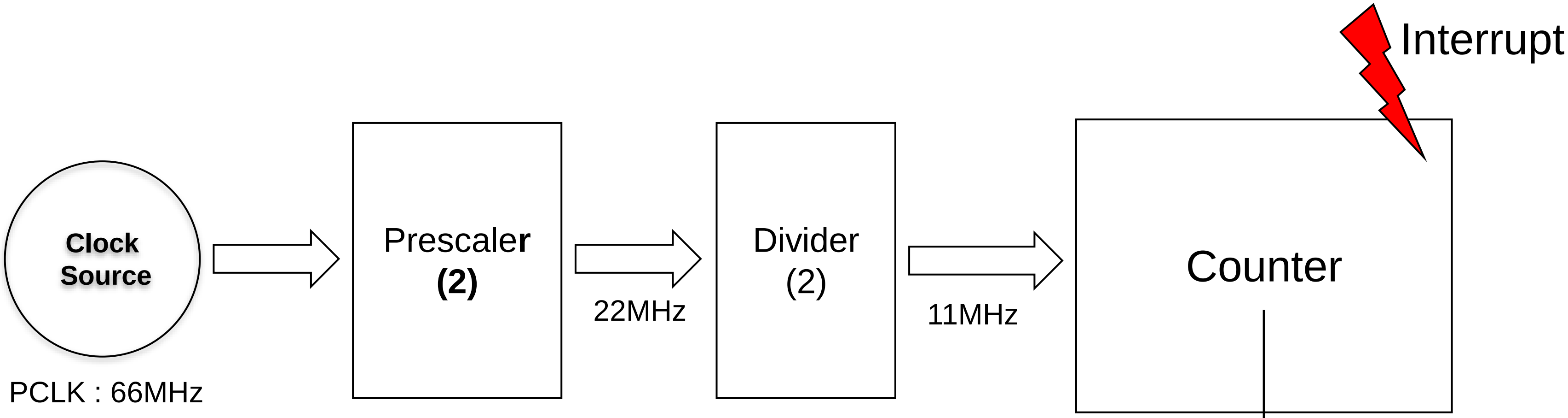
- **Counter**
  - Increment or decrement by 1 every clock cycle
  - If count is incrementing type, count up from 0 to Reload value
  - If count is decrementing type, count down from Reload value to 0

- **Reload value & Refresh**
  - Timer refresh occurs when Counter is incremented or decremented by 1 and reaches a specified value
    - If the counter is a decrementing type, it is refreshed when it becomes 0.
  - Interrupt occurs when refresh occurs

Basic Elements of Timer

- **Prescaler**
  - The first factor that reduces the frequency of the clock source
  - Divide the frequency of the clock source by (factor +1)
  - 1 ~ 255 (S5PC100)
  - ex) 66MHz, pre-scaler value : 1 ➔ 33MHz
- **Divider**
  - The second factor that reduces the frequency of the clock source
  - Divide the frequency of the clock source by the factor value
  - 1, 2, 4, 8, 16, TCLK(external clock)
  - ex) 66MHz, divider value : 2 ➔ 33MHz
- **Timer Input Clock Frequency**
  - The actual clock frequency input to the timer(counter)
  - Calculated by (CLK Source / {prescaler value + 1}) / {divider value}

Timer



If the Reload Value is 11,000
1,000 interrupts occur per second

TIMER REGISTER SETTING

Timer of S5PC100

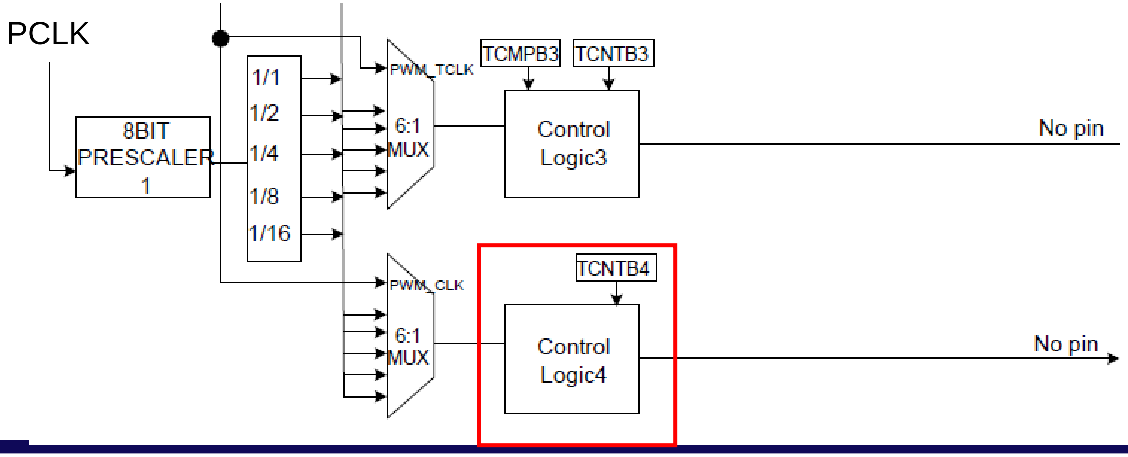
- **Type of Timer**
  - PWM Timer
  - System Timer

- **PWM Timer**
  - Has 5 32bit Timers
    - Timer 0, 1, 2
      - Timer that can send out signal
    - Timer 3, 4
      - Timer that does not have an output pin and only used internally
  - Clock Source is PCLK(66MHz)
  - Down-Counter
    - Starting from the value(Reload value) stored in the Timer Count Buffer Register(TCNTBn), count down to 0
    - When it becomes 0, it generates an interrupt to notifies CPU

Timer in this practice

- **Usage**
  - Scheduler calls every cycle(1 second)

- **Using Timer 4**
  - No need to send out signal through output pin
    - Using internal timer
      - Timer 3 or 4
  - Used as Timer 4 which does not need to adjust duty cycle



Timer registers

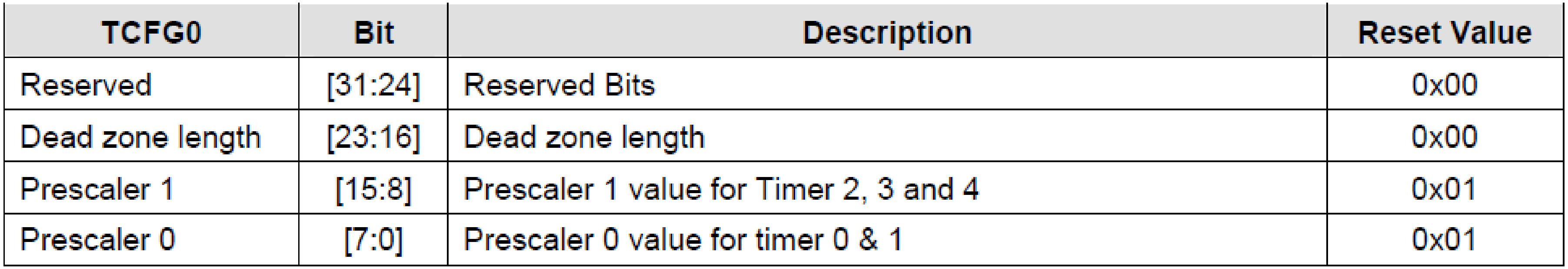
- **Control registers**
  - TCFG0
  - TCFG1
  - TCON
  - TINT_CSTAT (Interrupt Control & Status Register)

- **Buffer register**
  - TCNTB4

- **Other register**
  - TCNTO4

TCFG0

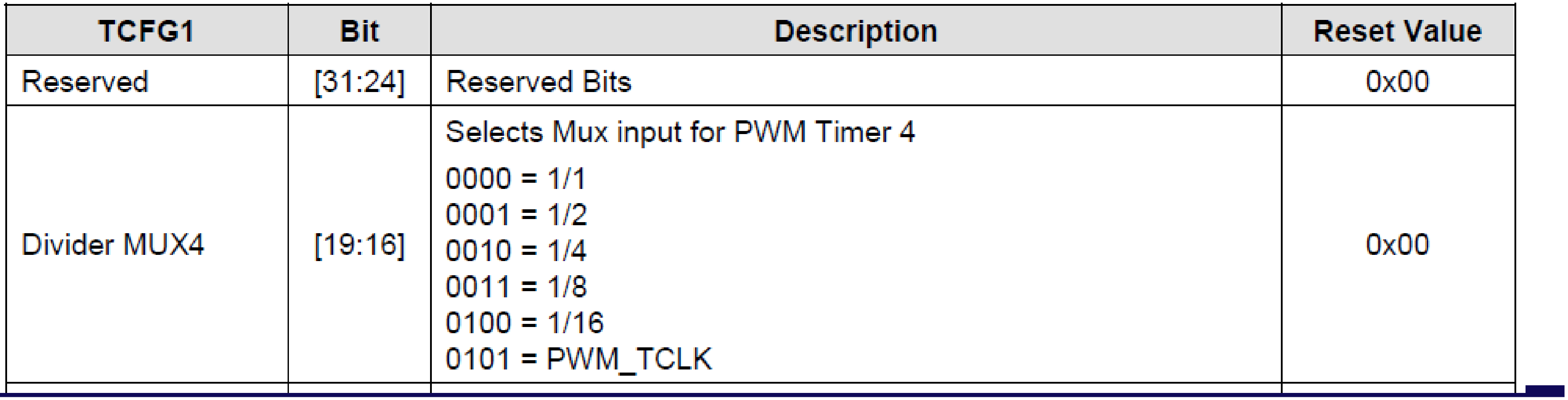
- **Register : TCFG0 (0xea000000)**
  - Timer Configuration Register
    - Prescaler value setting (1~255)
    - You can set the desired timer frequency (frequency) by dividing prescaler value in PCLK (66MHz)
      - When dividing prescaler value, add 1 first : (PCLK / (prescaler +1))
  - Datasheet
    - 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

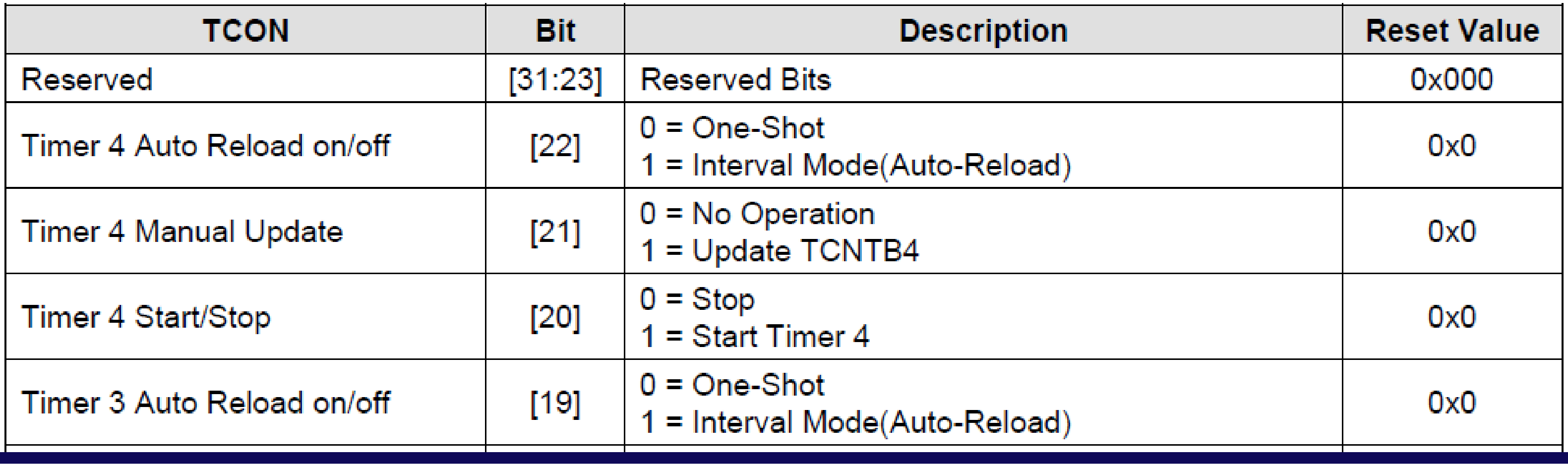
- **Register : TCFG1 (0xea000004)**
  - Timer Configuration Register
    - Set Divider value (1,2,4,8,16 or PWM_TCLK)
    - Divide frequencies divided by prescaler values into divider values again
      - PCLK / ({prescaler value + 1}) / {divider value} = Timer Freq.
  - Datasheet
    - Page. 592



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)

- **Register : TCON (0xea000008)**
  - Timer Control Register
    - Timer start & stop
    - Update TCNTB4
    - Auto Reload On/Off
  - Datasheet
    - Page. 593



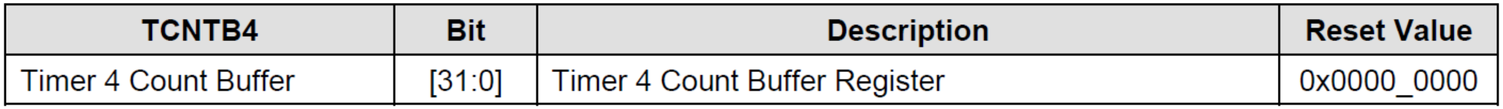
TCON	Bit	Description	Reset Value
Reserved	[31:23]	Reserved Bits	0x000
Timer 4 Auto Reload on/off	[22]	0 = One-Shot 1 = Interval Mode(Auto-Reload)	0x0
Timer 4 Manual Update	[21]	0 = No Operation 1 = Update TCNTB4	0x0
Timer 4 Start/Stop	[20]	0 = Stop 1 = Start Timer 4	0x0
Timer 3 Auto Reload on/off	[19]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

TCON (2/2)

- **Register : TCON (0xea000008)**
  - Bit [20] Timer 4 Start/Stop
    - 1 : Start Timer 4
    - 0 : Stop Timer 4
  - Bit [21] Timer 4 Manual Update
    - 1 : Update TCNTB4
    - 0 : No operation
  - Bit [22] Timer 4 Auto Reload On/Off
    - 1 : When the counter of Timer 4 reaches 0, reload automatically
    - 0 : Timer 4's counter only works once

TCNTB4

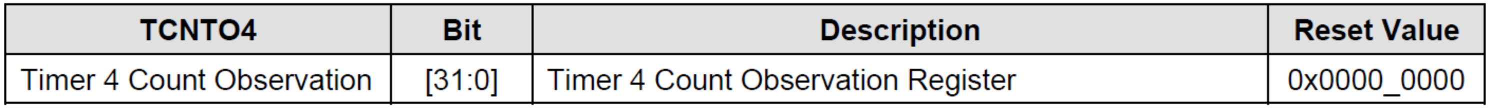
- **Register : TCNTB4 (0xea00003c)**
  - Timer4 Counter Register
    - Set up Reload Value
    - Timer counts down from the value set in TCNTB4. When it reaches 0, it generates an interrupt
    - Interrupt is generated and counter is reset to TCNTB4 value again
  - Example
    - Timer 4's clock is set to 1000Hz through TCFG0 and TCFG1
    - Input 1000 (decimal) or 0x3e8 in the TCNTB4 register
    - Interrupts occur once per second
  - Datasheet
    - Page. 596



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

TCNTO4

- **Register : TCNTO4 (0xea000040)**
  - Timer4 Observation Register
    - Can read current value of Counter
  - Datasheet
    - Page. 596

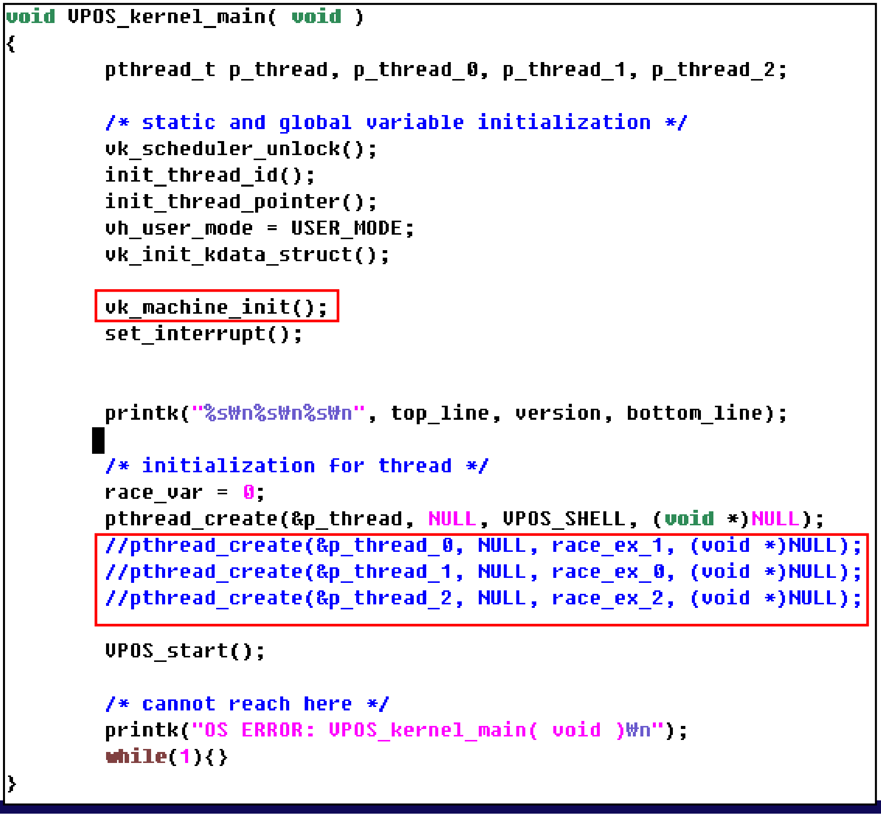


TCNTO4	Bit	Description	Reset Value
Timer 4 Count Observation	[31:0]	Timer 4 Count Observation Register	0x0000_0000

VPOS_kernel_main()

- **Functions**
  - Initialize the VPOS kernel data structure
  - Initialize hardware such as serial device and timer
  - Enable interrupt
  - Print boot message
  - Create a shell thread
  - Enter scheduler calling VPOS_start routine

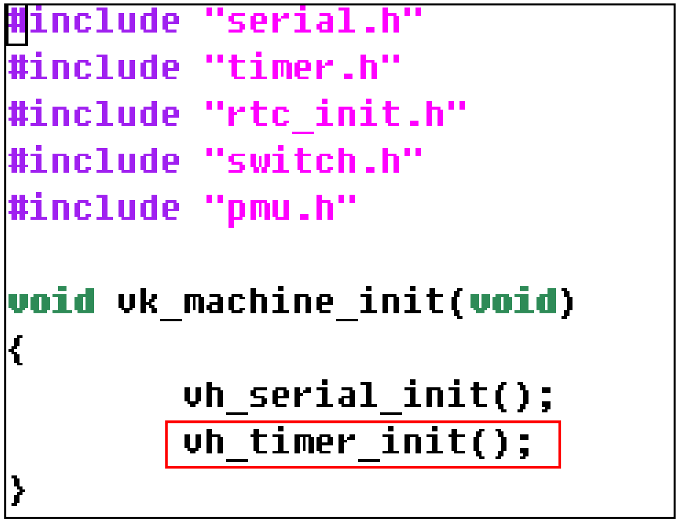
- **Source code location**
  - vpos/kernel/kernel.start.c



```
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, 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();

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

vk_machine_init()

- **Code**
  - Initialize hardware device
  - vh_serial_init() : Initialize UART
  - vh_timer_init() : Initialize Timer



```
#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();
}
```

- **Source code location**
  - vpos/kernel/machine_init.c

Timer Related Files

- **Timer Source Code**
  - vpos/hal/io/timer.c
    - Initialize Timer
    - Timer Interrupt enable/disable
    - Timer Interrupt Handler
  - vpos/hal/include/vh_io_hal.h
    - Define address of timer related register

Timer Setting

- **Define Timer 4 related register address**
  - vpos/hal/include/vh_io_hal.h

- **Initialize Timer 4**
  - vpos/hal/io/timer.c
    - Initialize Timer 4 in vh_timer_init() function

Timer Setting

- **Define Timer 4 related register address**
  - Defined in vpos/hal /include/vh_io_hal.h (using define statement)
  - Register list
    - TCFG0
    - TCFG1
    - TCON
    - TINT_CSTAT
    - TCNTB4
    - TCNTO4
  - Define constant name as vh_[Regiser Name].

    ex) vh_TCFG0, ….

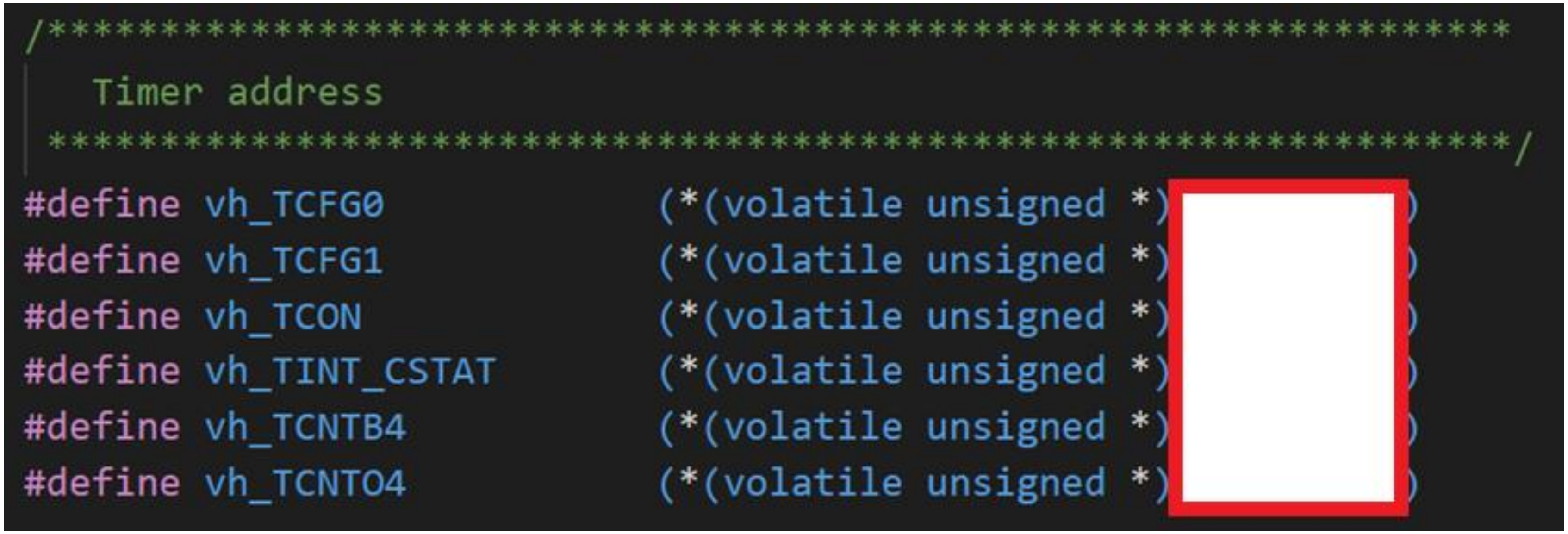
Timer Setting

- **Initialize Timer 4**
  - vpos/hal/io/timer.c
    - Initialize Timer 4 in vh_timer_init() function
  - Clock Source(PCLK) : 66MHz
  - Interrupt occurrence frequency : 1 second

- **Register setting order**
  1. TCFG0 setting
     - Specify prescaler value for Timer 4 : 255
  2. TCFG1 setting
     - Specify divider value of Timer 4 : 1/16
  3. TCNTB4 setting
     - Calculate and set the interrupt occurrence period to 1 second

Timer Setting

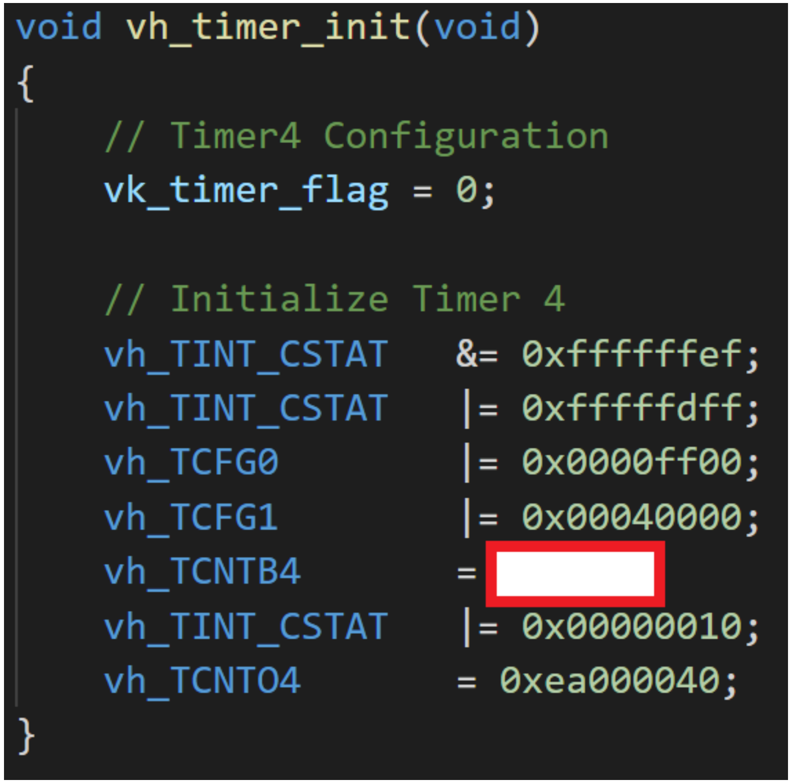
vpos/hal/include/vh_io_hal.h



```
/*****************************************************************
   Timer address
 *****************************************************************/
#define vh_TCFG0            (*(volatile unsigned *)          )
#define vh_TCFG1            (*(volatile unsigned *)          )
#define vh_TCON             (*(volatile unsigned *)          )
#define vh_TINT_CSTAT       (*(volatile unsigned *)          )
#define vh_TCNTB4           (*(volatile unsigned *)          )
#define vh_TCNTO4           (*(volatile unsigned *)          )
```

Timer Setting

vpos/hal/io/timer.c



```
void vh_timer_init(void)
{
    // Timer4 Configuration
    vk_timer_flag = 0;

    // Initialize Timer 4
    vh_TINT_CSTAT   &= 0xffffffef;
    vh_TINT_CSTAT   |= 0xfffffdff;
    vh_TCFG0        |= 0x0000ff00;
    vh_TCFG1        |= 0x00040000;
    vh_TCNTB4       = 
    vh_TINT_CSTAT   |= 0x00000010;
    vh_TCNTO4       = 0xea000040;
}
```

Timer Setting

What we modified

- vh_io_hal.h
  - vh_TCFG0
  - vh_TCFG1
  - vh_TCON
  - vh_TINT_CSTAT → you must find memory address from the datasheet (S5PC100_UM_REV101.pdf)
  - vh_TCNTB4
  - vh_TCNTO4

Timer Setting

What we modified

- vh_timer_init

clock frequency = 66,000,000 / (255 +1) / 16

= 16,113.28125

= 16,000

Thank you