W7200 User's Guide

for STM32F10x Standard Library

- USART
- GPIO
- Timer

Version 1.0.0e

Table of Contents

1 Introduction

W7200 is WIZnet's newest iMCU product which contains STMicroelectronics STM32F103CB (the MCU core of ARM's 32 bit Cortex-M3) and WIZnet W5200 (hardwired TCP/IP, MAC, PHY). W7200 can be easily applied to peripherals by using the peripheral library provided from STSTMicroelectronics since ST STMicroelectronics's MCU is internally installed in W7200.

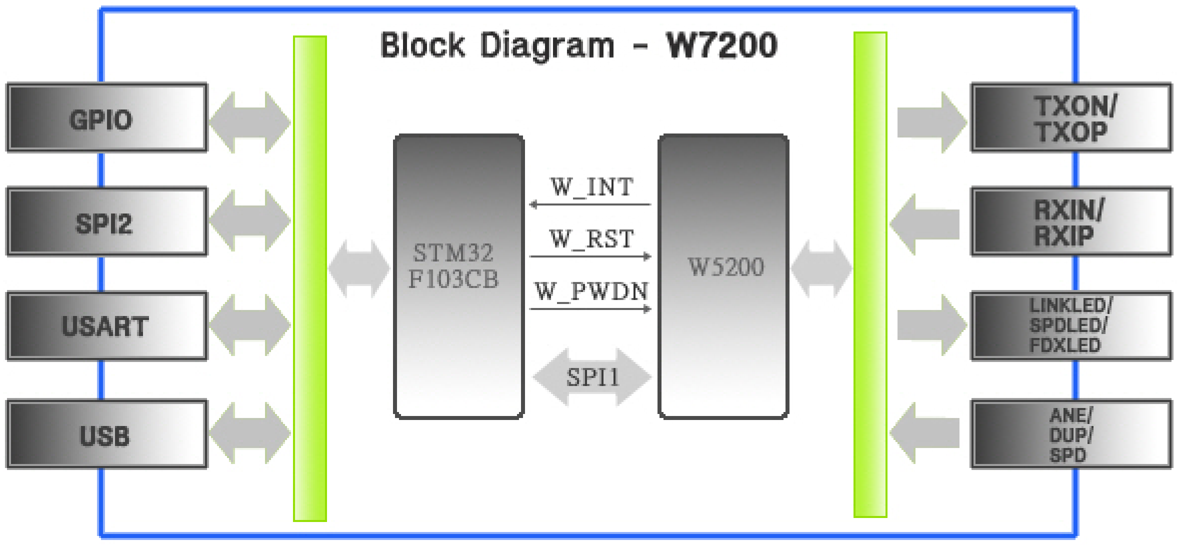


Figure 1. iMCU W7200 Block Diagram

This document will cover the composition of STMicroelectronics' standard library and how to use it. The following basic examples of peripherals will also be covered.

- **USART : printf (+Echoback)**
- **GPIO : IO Toggle (+LED control)**
- **Timer : Time Base**

2 Environment Setting for Development

Refer to the W7200 Startup Guide document when preparing the development environment for IAR EWARM (Embedded Workbench for ARM). This document is written based on STM32 F10x Standard Peripheral Library **version 3.5.0** (STMicroelectronics official website, www.st.com) and IAR EWARM version 6.5(30-day time limited evaluation license) is used.
W7200 Startup Guide can be downloaded by clicking the link below.

⇨ http://www.wiznet.co.kr/UpLoad_Files/ReferenceFiles/W7200_StartUpGuide_EWARM_v100K.pdf

3 STM32F10x Standard Library Structures

STM32 F10x Standard Peripheral Library is composed of three folders: [Libraries], [Project], and [Utilities]. The HTML help file (stm32f10x_stdperiph_lib_um.chm) is located in the top folder of the STM32 F10x Standard Peripheral Library; refer to the HTML help file for more details on the library.

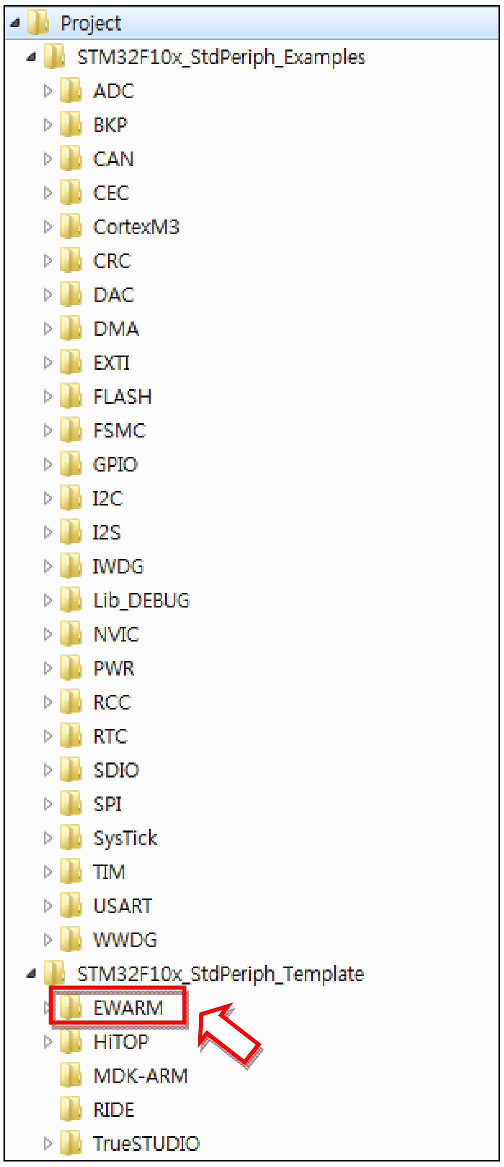
Below are the two folders in the [Project] folder

[STM32F10x_StdPeriph_Examples] :

Example for controlling peripherals by using the STM peripheral drivers

[STM32F10x_StdPeriph_Template] :

Project template for each compiler (W7200: EWARM)



```
Project
  STM32F10x_StdPeriph_Examples
    ADC
    BKP
    CAN
    CEC
    CortexM3
    CRC
    DAC
    DMA
    EXTI
    FLASH
    FSMC
    GPIO
    I2C
    I2S
    IWDG
    Lib_DEBUG
    NVIC
    PWR
    RCC
    RTC
    SDIO
    SPI
    SysTick
    TIM
    USART
    WWDG
  STM32F10x_StdPeriph_Template
    EWARM
    HiTOP
    MDK-ARM
    RIDE
    TrueSTUDIO
```

Figure 2. Folders List of Project

Below are the two folders in the [Libraries] folder.

[CMSIS] : Cortex-M series MCU library provided by ARM

※ *CMSIS : Cortex Microcontroller Software Interface Standard*

[STM32F10x_StdPeriph_Driver] : STMicroelectronics peripheral drivers

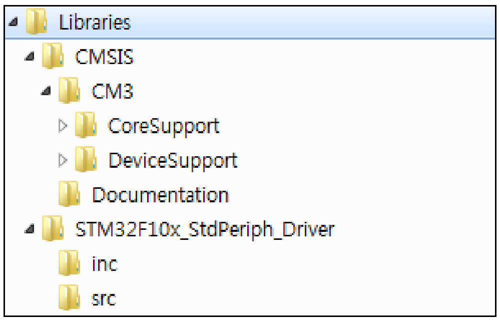


Figure 3. Folders List of Libraries

Below is the only folder in the [Utilities] folder.

[STM32_EVAL] : Setting file for each MCU (EVB) according to the STM MCU line up.

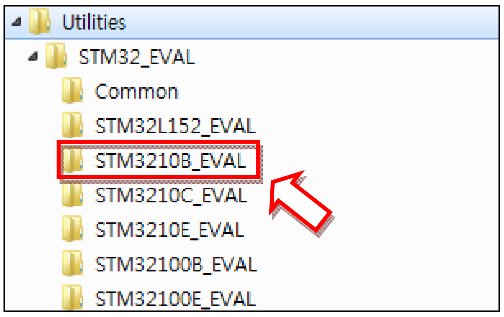


Figure 4. Folders list of STM32_EVAL

4 Configuration

4.1 Check points

STM32 MCU and W5200 is connected through SPI interface inside the W7200 chip.
The pins listed in Table 1 below cannot be used for other purposes.

Table 1. W7200 Internal Connected Pin List

No	W7200		Description
	STM32F103CB Pins, Pin name	W5200 Pins, Pin name	
1	2, PC13/TAMPER_RTC	49, W_INT	W5200 Interrupt pin
2	14, PA4/SPI_NSS	50, CSN	STM32 MCU(SPI1) – W5200 SPI Interface pin
3	15, PA5/SPI_SCK	51, SCLK	
4	16, PA6/SPI_MISO	53, MISO	
5	17, PA7/SPI_MOSI	52, MOSI	
6	45, PB8	53, W_RESET	W5200 Reset pin
7	46, PB9	54, PWDN	W5200 Power-down pin

4.2 How to use it

Follow the steps below to use the examples provided from STMicroelectronics.

1. Extract the zipped STM32 F10x Standard Peripheral Library file.
2. Copy/paste the example source code file from the Examples folder to template folder.

   $PROJ_DIR$ \ STM32F10x_StdPeriph_Template

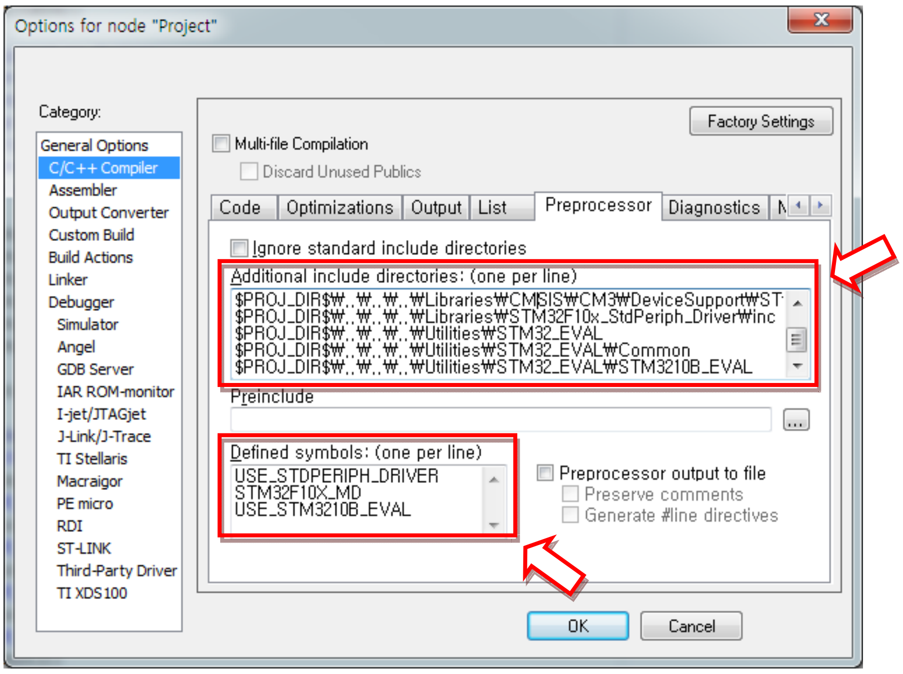
3. Use IAR EWARM to open the EWARM project file in the Template folder.
4. Select 'Rebuild' to create a binary image file and program it on the W7200 board.
5. Run the example.

W7200 is considered as a 'medium-density device' since STM32F103CB MCU(128Kbytes Flash memory) is inside. Process the '**STM3210B-EVAL** Set-up' from the IAR EWARM workspace. Please refer to Chapter 2 '**Configuration and Start the Project**' of the W7200 Startup Guide for more details on creating binary image files and programming the W7200 board.

Note

Setting for C Compiler Preprocessor

The correct directories of the library that are wished to be included must be marked in order to compile; if they are not marked in other versions than STM32 F10x Standard Peripheral Library Version 3.5.0, add the directory information into Project > Options / C/C++ Compiler / Preprocessor.



Multi-file Compilation
Discard Unused Publics
Ignore standard include directories
Additional include directories: (one per line)

```
$PROJ_DIR$₩..₩..₩..₩Libraries₩CMSIS₩CM3₩DeviceSupport₩ST
$PROJ_DIR$₩..₩..₩..₩Libraries₩STM32F10x_StdPeriph_Driver₩inc
$PROJ_DIR$₩..₩..₩..₩Utilities₩STM32_EVAL
$PROJ_DIR$₩..₩..₩..₩Utilities₩STM32_EVAL₩Common
$PROJ_DIR$₩..₩..₩..₩Utilities₩STM32_EVAL₩STM3210B_EVAL
```

Preinclude

Defined symbols: (one per line)

```
USE_STDPERIPH_DRIVER
STM32F10X_MD
USE_STM3210B_EVAL
```

Preprocessor output to file
Preserve comments
Generate #line directives

Figure 5. Setting for C/C++ Compiler Preprocessor

Additional include directories: (one per line)

$PROJ_DIR$\..

$PROJ_DIR$\..\..\..\Libraries\CMSIS\CM3\CoreSupport

$PROJ_DIR$\..\..\..\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x

$PROJ_DIR$\..\..\..\Libraries\STM32F10x_StdPeriph_Driver\inc

$PROJ_DIR$\..\..\..\Utilities\STM32_EVAL

$PROJ_DIR$\..\..\..\Utilities\STM32_EVAL\Common

$PROJ_DIR$\..\..\..\Utilities\STM32_EVAL\STM3210B_EVAL

Defined symbols: (one per line)

USE_STDPERIPH_DRIVER

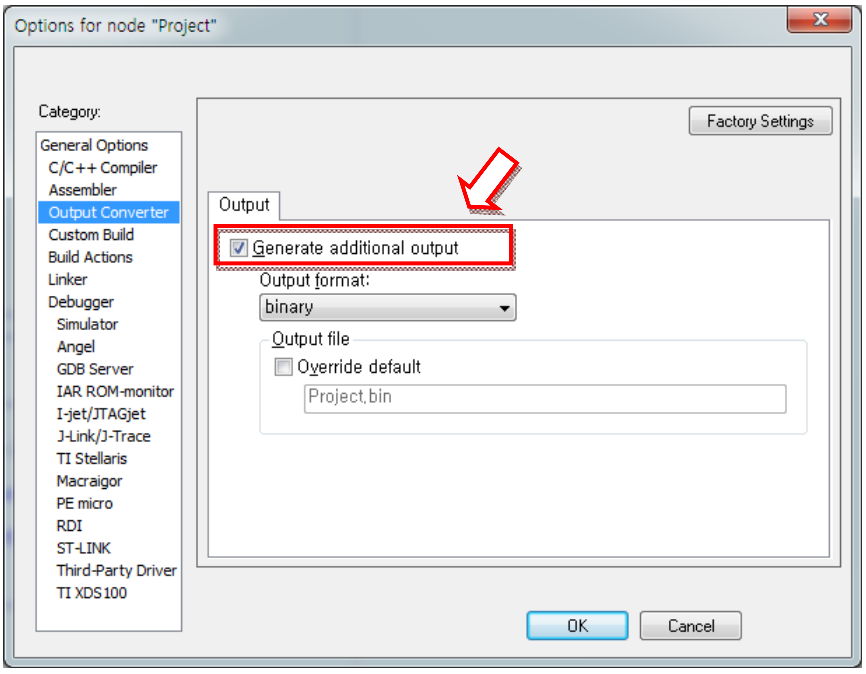
STM32F10X_MD

USE_STM3210B_EVAL

Note

Make the Binary output file

Binary image file is necessary when programming the W7200 board by using the Flash Loader Demonstrator. In STM32 F10x Standard Peripheral Library Version 3.5.0, it is default to not create output binary; select the '**Generate additional output'** option from Project > Options / Output Converter menu to activate.



☑ Generate additional output

Output format: binary

Output file

☐ Override default

Project.bin

Figure 6. Enable the Generate Additional Output Option

5 Examples Demonstration

5.1 USART

In this document, example 'printf'(this examples uses USARTx to print serial terminal messages) will be demonstrated.

As explained in chapter 4.2 'How to use it,' copy the USART > printf example to the *STM32F10x_StdPeriph_Template* folder. Below are the source codes in the 'printf' example.

Table 2. Source code list of the 'printf' example

File name	Description
main.c	main function, USART configuration and printf included
stm32f10x_conf.h	Library configuration file
stm32f10x_it.c	Interrupt service routine file
stm32f10x_it.h	Interrupt handler header file
system_stm32f10x.c	System clock configuration and system initialization function file
readme.txt	File of explaining 'printf' example

Note

※ Handling routine about USART interrupt is not included in this example.

Please take note that USART interrupt handling routine is implemented in the 'Interrupt' example in STM32 standard peripheral library, USART examples.

The main function includes USART initialization code. This initialization process saves the usart configuration value in the USART_InitStructure and uses STM_EVAL_COMInit function to configure and enable USART.

Example code: USART Initialization

```
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

STM_EVAL_COMInit(COM1, &USART_InitStructure);
```

The fputc function must be defined in order for the printf function character string can be printed in USART.

Example code: Retarget the C library printf function to the USART

```
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART */
  USART_SendData(EVAL_COM1, (uint8_t) ch);

  /* Loop until the end of transmission */
  while (USART_GetFlagStatus(EVAL_COM1, USART_FLAG_TC) == RESET)
  {}

  return ch;
}
```

Once the initialization process is done, use the printf() function to print the character string in the serial terminal. Use the settings shown in Table 3 below for the serial terminal.

Table 3. Serial Terminal Setting

Port Setting	Value
Baud rate	115200
Data bits	8
Parity	none
Stop bits	1
Flow control	None

The Echoback function prints out the input received from serial to prove the input/output operation of USART in the main code.

Insert the following code in 'while() loop' as shown below to implement Echoback. The USART related functions used in the code below are included in stm32f10x_usart.c.

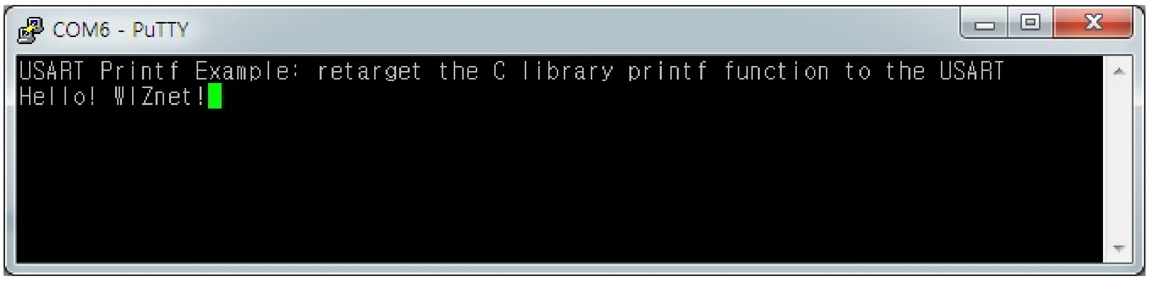
Note

※ The shaded parts in this document are additional info or modified for iMCU7200EVB.
All example codes in this document are identical.

Example code: Main Function

```
Int main(void)
{
    unsigned char c;

    /* USART1 Initialization */
    …
    STM_EVAL_COMInit(COM1, &USART_InitStructure);

    /* Output a message on Hyperterminal using printf function */
    printf("\n\rUSART Printf Example: retarget the C library printf function to the USART\n\r");

    // USART Echoback
    while(1)
    {
        while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET) { }

        c = USART_ReceiveData(USART1);     // Read a character from the USART
        USART_SendData(USART1, c);         // Send a character to the USART
    }
}
```

Figure 7 is the result of Echoback added to the example code in iMCU7200EVB; the message was printed in serial terminal using printf function, and was checked by typing in 'Hello! WIZnet!'



```
USART Printf Example: retarget the C library printf function to the USART
Hello! WIZnet!
```

Figure 7. USART Demonstration on iMCU7200EVB

5.2 GPIO

The GPIO example provided in this document is 'IO Toggle,' which prints the set and reset status alternately using the GPIO pin.

Note

※ The STM32 library example shows toggling PD0 and PD2 but the STM32f103CB-48PIN used in W7200 does not support PD0 and PD2. In this document, PD0 and PD2 is replaced by LED3 and LED 4 of iMCU7200EVB. Therefore, the operation of GPIO toggle can be checked by monitoring the EVB's LED.

The following GPIO Initialization code is included in the main function. Configuration is done by setting the pin selection, pin speed, and operation mode for the InitStructure and then using the structure value for GPIO_Init function. The GPIO Initialization code in this document is modified from the IO Toggle code from the STM32 library.

Example code: GPIO Initialization

```
/* GPIOB Periph clock enable */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

/* Configure PB0 and PB1 in output push-pull mode */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;

GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Use the BSRR and BRR register in order to set/reset PB0 and PB1. Set the mapping pins that are equivalent to the lower 16 bits of the GPIO_BSRR (Port Bit Set/Reset Register). GPIO_BRR (Port Bit Reset Register) is similar to BSRR. If user wishes to set Pin0, set the BSRR register to 0x00000001. If user wishes to reset Pin15, set the BRR register to 0x00008000.

The example code for SET/RESET of PB0 and PB1 are as below.
User can enable SET/RESET by setting 0x00000001 to control PB0 and 0x00000002 to control PB1; set the bit value to BSRR to do SET or set the bit value to BRR to do RESET.

The Delay function is added to check the LED blinking. This Delay function is provided from the ST library.

Example code: GPIO Toggle

```
while (1)
{
    /* Set PB0 and PB1 */
    GPIOB->BSRR = 0x00000003;           // 0x00000001 | 0x00000002
    Delay(10);                          // Delay for LED(I/O) toggling indication
    /* Reset PB0 and PB1 */
    GPIOB->BRR  = 0x00000003;
    Delay(10);                          // Delay for LED(I/O) toggling indication
}
```

Program the example code into iMCU7200EVB and the LED will flash to show IO toggle working.

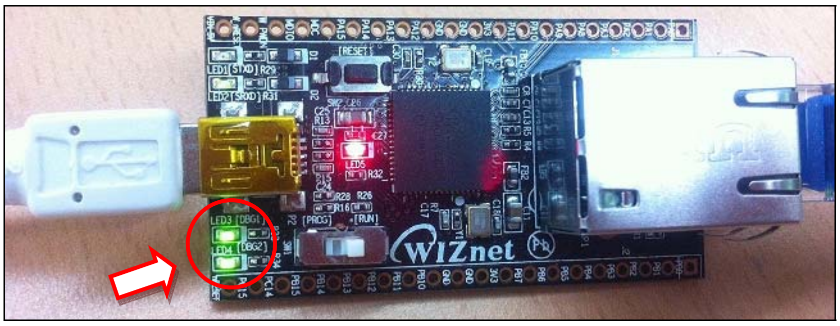


Figure 8. IO Toggle - LED on

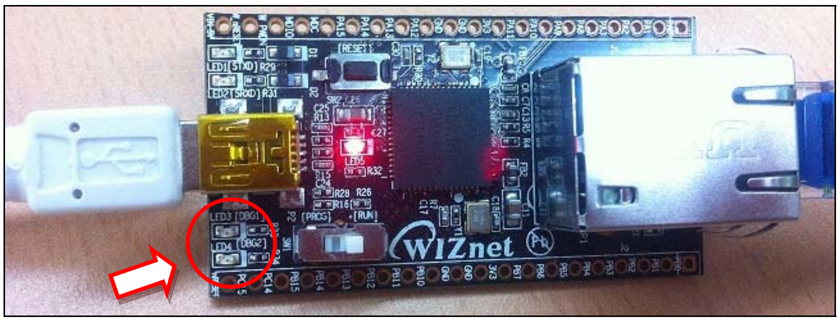


Figure 9. IO Toggle - LED off

5.3 Timer

In this document, TIM Time Base example is shown among the Timer examples from STMicroelectronics. This example uses TIM2 for configuration of TIM peripheral that operate in Output Compare Timing mode. Also, Interrupt request can be created according to the time base of each channel.

Note

※ PC6 ~ 9 are used as the Timer Interrupt channel in the STM32 library example. However, the STM32F103CB-48PIN used in W7200 does not support those pins. In this document, GPIO PB11~14 will be used as output port.

The below table shows the settings for each channel, and the interrupt request of each channel are occurred periodically.

Table 4. Time base Configuration

Pin	TIM2 CC	CC Register value	Description
PB11	CC1	40961	CC1 update rate = TIM2 counter clock / CCR1_Val = 146.48Hz TIM2 Channel 1 generates an interrupt each 6.8ms
PB12	CC2	27309	CC2 update rate = TIM2 counter clock / CCR2_Val = 219.7Hz TIM2 Channel 2 generates an interrupt each 4.55ms
PB13	CC3	13654	CC3 update rate = TIM2 counter clock / CCR3_Val = 439.4Hz TIM2 Channel 3 generates an interrupt each 2.27ms
PB14	CC4	6826	CC4 update rate = TIM2 counter clock / CCR4_Val = 878.9Hz TIM2 Channel 4 generates an interrupt each 1.13ms

CC Register values are declared in main.c file.

Example code: CCR Value Declarations

```
__IO uint16_t CCR1_Val = 40961;
__IO uint16_t CCR2_Val = 27309;
__IO uint16_t CCR3_Val = 13654;
__IO uint16_t CCR4_Val = 6826;
```

Modify the following in order to use pin other than the example.

1.	RCC_Configuration	: System clocks configuration
2.	GPIO_Configuration	: General purpose I/O configuration
3.	TIM2_IRQHandler	: TIM2 Interrupt service routine
-	NVIC_Configuration	: Nested vectored interrupt controller configuration

Follow the steps above to modify the pin (refer to the below example code).
Modify the GPIOC clock enable to GPIOB clock enable for the RCC_Configuration function.

Example code: RCC Configuration

```
void RCC_Configuration(void)
{
        /* PCLK1 = HCLK/4 */
        RCC_PCLK1Config(RCC_HCLK_Div4);

        /* TIM2 clock enable */
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

        /* GPIOB clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
}
```

The GPIO_Configuration function resets the GPIO Pin when Timer interrupt occurs.
If the user wants to check TIM2 interrupt request using a different pin, modify the pin setting.

Example code: GPIO Configuration

```
void GPIO_Configuration(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;

        /* GPIOB Configuration: Pin4, 5, 6 and 7 as alternate function push-pull */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

        GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

The TIM2_IRQHandler function is used when a Timer interrupt has occurred; the modification of GPIO Pin is applied when there is an interrupt request.

Example code: TIM2_IRQHandler

```
void TIM2_IRQHandler(void)
{
        if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
        {
                TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
                  /* Pin PB.11 toggling with frequency = 73.24 Hz */
                GPIO_WriteBit(GPIOB, GPIO_Pin_11, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_11)));
                capture = TIM_GetCapture1(TIM2);
                TIM_SetCompare1(TIM2, capture + CCR1_Val);
        }
        else if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET)
        {
                TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
                /* Pin PB.12 toggling with frequency = 109.8 Hz */
                GPIO_WriteBit(GPIOB, GPIO_Pin_12, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_12)));
                capture = TIM_GetCapture2(TIM2);
                TIM_SetCompare2(TIM2, capture + CCR2_Val);
        }
        else if (TIM_GetITStatus(TIM2, TIM_IT_CC3) != RESET)
        {
                TIM_ClearITPendingBit(TIM2, TIM_IT_CC3);
                /* Pin PB.13 toggling with frequency = 219.7 Hz */
                GPIO_WriteBit(GPIOB, GPIO_Pin_13, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_13)));
                capture = TIM_GetCapture3(TIM2);
                TIM_SetCompare3(TIM2, capture + CCR3_Val);
        }
        else
        {
                TIM_ClearITPendingBit(TIM2, TIM_IT_CC4);
                /* Pin PB.14 toggling with frequency = 439.4 Hz */
                GPIO_WriteBit(GPIOB, GPIO_Pin_14, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_14)));
                capture = TIM_GetCapture4(TIM2);
                TIM_SetCompare4(TIM2, capture + CCR4_Val);
        }
}
```

If user wants to use another timer than Timer2, use the NVIC_Configuration function to reset the Timer2 global interrupt.

Example code: NVIC Configuration

```
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStructure;

        /* Enable the TIM2 global Interrupt */
        NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

        NVIC_Init(&NVIC_InitStructure);
}
```

Below are the results of observing the Timer Interrupt signal by using oscilloscope when the modified binary image is programmed onto iMCU7200EVB.
The order from top to bottom is; CC1, CC2, CC3, and CC4.

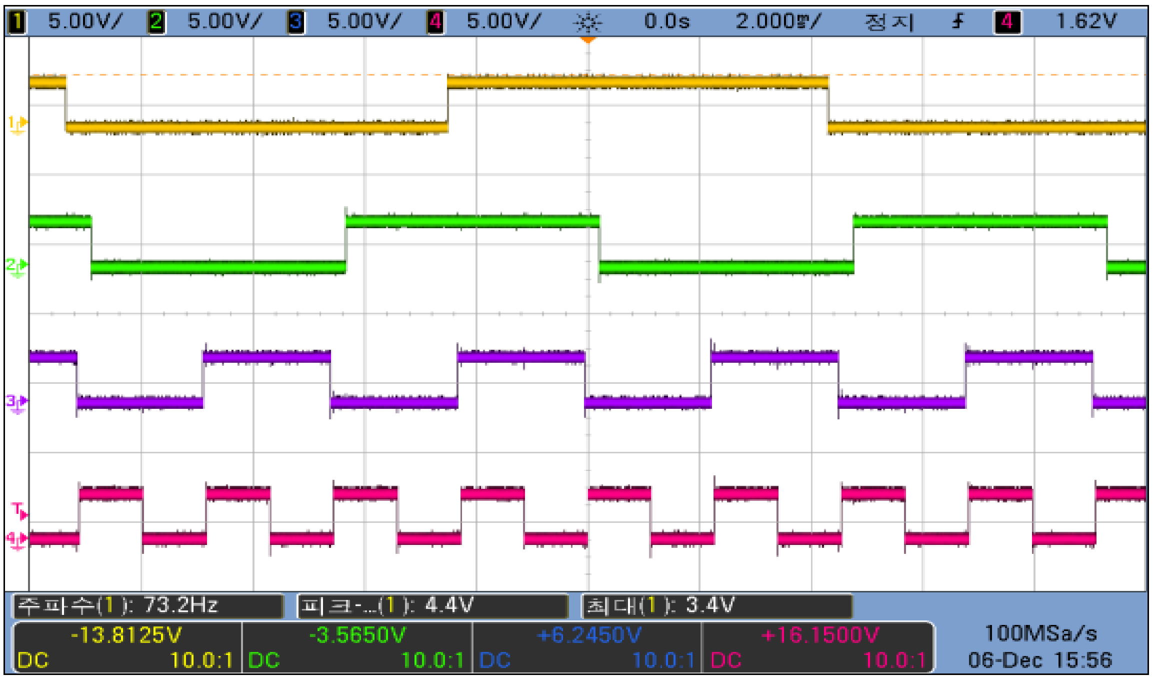


Figure 10. Oscilloscope Measurements : Timer interrupts

Document History Information

Version	Date	Descriptions
Ver. 1.0.0e	23 Apr. 2013	Document Released