AN4157
Application note

STM32F3DISCOVERY peripheral firmware examples

Introduction

This application note describes the peripheral firmware examples provided with the STM32F3DISCOVERY kit.

These ready-to-run examples are provided to help the user get started quickly with STM32F30x or STM32F31x peripherals and STM32F3DISCOVERY board hardware. Preconfigured projects for EWARM, MDK-ARM, TrueSTUDIO and TASKING toolchains are provided for each example.

These examples are included in the firmware application package available for download on www.st.com/stm32f3discovery.

First read the document *Getting started with software and firmware environments for the* STM32F3DISCOVERY *kit* (UM1562) to familiarize ourself with the kit.

Table 1 lists the microcontrollers and development tools concerned by this application note.

Table 1. Applicable products and tools

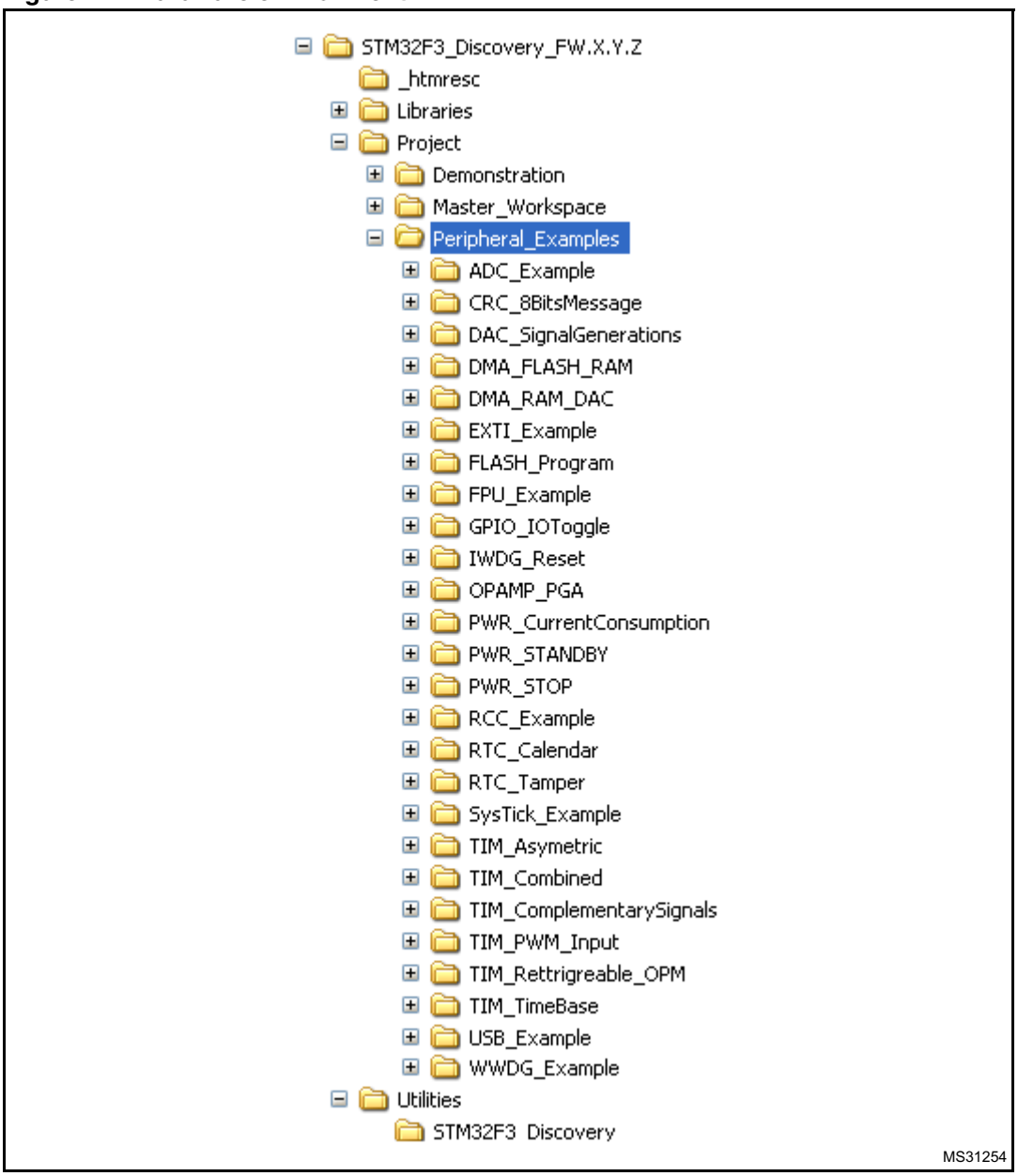
Type	Applicable products
Microcontrollers	STM32 F3 series Entry-level Cortex™-M4 microcontrollers
Development tools	STM32F3DISCOVERY evaluation board and discovery kit

Contents

1 Peripheral firmware examples structure overview

The peripheral firmware examples are provided within the STM32F3DISCOVERY firmware applications package and are located in the \Project folder as shown in *Figure 1*.

Figure 1. Hardware environment



Note: VX.Y.Z refer to the package version, for example, V1.0.0.

To run an example, open the project with your preferred toolchain, compile, load and run it. Some examples may require additional hardware such as an oscilloscope. For further detail about the required hardware, refer to the readme file provided within each example.

2 Clock configuration

The peripheral examples provided within STM32F3DISCOVERY kit Firmware package are configured to run at 72 MHz, using HSE BYPASS as the clock source.

However, the user can modify this configuration to use HSI or HSE crystal as the clock source, which needs some hardware modification on the discovery kit hardware.

The "system_stm32f30x.c" file provided within each example was customized for use with the discovery kit, allowing the user to select one of the three configurations below (by un-commenting the adequate define).

2.1 PLL_SOURCE_HSI

The HSI clock signal is generated from an internal 8 MHz RC Oscillator and can be used directly as a system clock, or divided by 2 to be used as a PLL input.

The HSI RC oscillator has the advantage of providing a clock source at low cost (no external components). It also has a faster startup time than the HSE crystal oscillator. However, even with a calibration, the frequency is less accurate than an external crystal oscillator or a ceramic resonator.

2.2 PLL_SOURCE_HSE

The high speed external clock signal (HSE) can be generated from two possible clock sources:

- HSE external crystal/ceramic resonator
- HSE user external clock

The HSE crystal is not provided with the discovery kit. Some hardware modifications are necessary to connect this crystal.

For more details, refer to section "4.10 OSC clock" in "STM32F3 discovery kit User manual (UM1570)".

2.3 PLL_SOURCE_HSE_BYPASS

In this mode, the HSE is bypassed with an external clock (fixed at 8 MHz, coming from ST-LINK circuit). It is used to clock the PLL, and the PLL is used as the system clock source.

Some hardware modifications are necessary to bypass the HSE with the clock coming from the ST-LINK circuit.

For more details, refer to section "4.10 OSC clock" in "STM32F3 discovery kit User manual (UM1570)".

Note: PLL_SOURCE_HSE_BYPASS is the default clock configuration.

3 Peripheral firmware examples description

3.1 ADC example

3.1.1 Purpose

This example describes how to use the ADC1 to convert continuously the voltage applied to a given analog input channel.

3.1.2 Description

In this example, detailed steps of ADC configuration are provided starting from GPIO configuration to the analog channel configuration without forgetting the ADC calibration procedure. The ADC1 Channel7(PC1) is used as directly connected to the external power supply (ranges between 0 and 3.3V).

The converted voltage is stored in the ADC1ConvertedVoltage variable.

3.2 CRC 8-bit message example

3.2.1 Purpose

This example describes how to use the CRC peripheral in order to compute 8-bit CRC checksum of a message.

3.2.2 Description

In this example:

- The CRC peripheral is configured to use the 8-bit CRC polynomial x8 + x7 + x6 + x4 + x2 + 1 with the initialization value set to 0.
- The message is declared as an array and labeled "CRCBuffer". The expected CRC value is already computed using an on line CRC tool. Once the CRC value is computed, the computed CRC value is compared to the expected one.
- In case of match the Blue led LD4 is turned on.
- Otherwise the led LD3 is turned on.

3.3 DAC signal generation example

3.3.1 Purpose

This example describes how to use the DAC peripheral to generate several signals using DMA controller.

3.3.2 Description

In this example, when the user presses the push-button, DMA transfers two selected waveforms to the DAC.

Each time the push-button is pressed, two signals have been selected and monitored on the DAC Channels:

- Escalator waveform (Channel 1) and Sine waveForm (Channel 2).
- Noise waveform (Channel 1) and Triangle waveform (Channel 2).

Note: Use the push-button *connected to PA0.*

Connect PA4 (DAC Channel1) and PA5 (DAC Channel2) pins to an oscilloscope to monitor the DAC out waves.

3.4 DMA Flash RAM example

3.4.1 Purpose

This example describes how to use a DMA channel to transfer a word data buffer from the Flash memory to the embedded SRAM memory.

3.4.2 Description

DMA1 Channel1 is configured to transfer the content of a 32-word data buffer, stored in the Flash memory, to the reception buffer declared in RAM.

- The start of transfer is triggered by software. DMA1 Channel1 memory-to-memory transfer is enabled as well as source and destination address incrementing.
- The transfer is started by setting the Channel enable bit for DMA1 Channel1.
- At the end of the transfer, a Transfer Complete interrupt is generated since it is enabled. Once the interrupt is generated, the remaining data to be transferred is read, it must be equal to 0. The Transfer Complete Interrupt pending bit is then cleared. A comparison between the source and destination buffers is done to check that all data have been correctly transferred.

3.5 DMA RAM DAC example

3.5.1 Purpose

This example provides a description of how to use a DMA channel to transfer data buffer from RAM memory to DAC.

3.5.2 Description

DMA2 channel3 is configured to transfer continuously, word by word, a 32-word buffer to the dual DAC register DAC_DHR12RD. Both DAC channels conversions are configured to be triggered by TIM2 TRGO triggers and without noise/triangle wave generation. 12-bit right data alignment is selected to be able to access DAC_DHR12RD register.

Note: Connect the oscilloscope to PA04 and PA05.

3.6 EXTI example

3.6.1 Purpose

This example shows how to configure an external interrupt line.

3.6.2 Description

In this example:

- PA0 pin is configured in input floating.
- PA0 is configured to be used as an external interrupt source line 0 (EXTI0).
- The EXTI line 0 is configured to generate an interrupt on each rising edge detected on the PA0 pin. The interrupt is triggered every time the user push-button is pressed.
- In the NVIC (nested vectored interrupt controller), the EXTI line 0 interrupt priority is configured and the interrupt is enabled.

When the program is executed and the user push-button is pressed (EXTI0 interrupt routine), LED4, LED6, LED9 and LED10 are toggled.

3.7 Flash program example

3.7.1 Purpose

This example describes how to program the STM32F30xor STM32F31x internal Flash memory.

3.7.2 Description

In this example:

- After Reset, the Flash memory Program/Erase Controller is locked. The FLASH_Unlock function is used to unlock it.
- Before programming the desired addresses, an erase operation is performed using the Flash erase page feature. The erase procedure starts with the calculation of the number of pages to be used. Then all these pages are erased one by one by calling FLASH_ErasePage function.
- Once this operation is finished, the programming operation is performed by using the FLASH_ProgramWord function. The written data is then checked and the result of the programming operation is stored into the MemoryProgramStatus variable.

3.8 FPU example

3.8.1 Purpose

This example provides information on FPU instructions.

3.8.2 Description

This example shows the benefits of using the STM32F3 FPU. It computes simple mathematical and standard operations with floating point input numbers.

- The time spent for the calculation of these operations wheen FPU is enabled is about 30 µs while it is about 50 µs when FPU is disabled. The gain is consequently of 40% .
- In term of code size, using the IAR toolchain, the gain is about 15%:
  - FPU enabled: 4264 bytes of Flash memory
  - FPU disabled: 4920 bytes of Flash memory

3.9 GPIO toggle example

3.9.1 Purpose

This example shows how to use the GPIO port bit set/reset registers (BSRR and BRR) for I/O toggling.

3.9.2 Description

In this example:

- GPIOE clock is enabled.
- GPIOE pins 14 and 15 are configured.
- In a `while` loop, the ODR14 and ODR15 bits are set in the GPIOE output data register (ODR) by setting the corresponding bits in the port bit set/reset register (BSRR). Then the ODR14 and ODR15 bits are reset in the GPIOC output data register (ODR) by setting the corresponding bits in the port bit reset register (BRR).
- A delay is inserted between setting and resetting the GPIOE ODR14 and ODR15 bits.

When the program is executed, the two LEDs, LED7 and LED10, are turned ON then OFF in an infinite loop. The duration between the ON and OFF states corresponds to the inserted delay.

Note: Connect the oscilloscope to the pins PE14 and PE15.

3.10 IWDG reset example

3.10.1 Purpose

This example shows how to update the IWDG reload counter at regular periods, and how to simulate a software fault by generating an MCU IWDG reset when the programmed time period has expired.

3.10.2 Description

In this example:

- The independent watchdog timeout is set to 250 ms.
- The SysTick timer is configured to generate an interrupt every 250 ms.
- In the SysTick interrupt service routine, the independent watchdog counter is reloaded to prevent an independent watchdog reset, and LED4 is toggled.
- The EXTI line 0 connected to PA0 pin is configured to generate an interrupt on its falling edge.
- In the NVIC, the EXTI line 0 corresponding interrupt vector is enabled with a priority equal to 0, and the systick interrupt vector priority is set to 1 (EXTI interrupt occurs prior to systick interrupt).
- The EXTI line is used to simulate a firmware failure: when the EXTI line event is triggered (after pressing the user push-button on the STM32F3DISCOVERY board), the corresponding interrupt is serviced. In the ISR, LED3 is switched off and the EXTI line pending bit is not cleared. The CPU executes the EXTI line ISR indefinitely and the SysTick interrupt routine is never entered, so the independent watchdog counter is not reloaded. As a result, when the independent watchdog counter reaches 00, the independent watchdog generates a reset.

When the program is running and the independent watchdog reset is generated, LED3 is turned on after the system resumes operation.

3.11 OPMAP in programmable gain amplifier example

3.11.1 Purpose

This example shows how to configure OPAMP1 peripheral to amplify a signal applied to OPAMP1 non inverting input using the built-in PGA.

3.11.2 Description

In this example, OPAMP1 is configured as following:

- PGA mode enabled with gain set to 2
- Non Inverting input is connected to PA5
- Output is available on PA2

In this configuration the OPAMP1 amplifies the signal on PA5. The DAC peripheral is configured to generate a sine wave signal on DAC_OUT2 (PA5) which will be amplified by the OPAMP1. To monitor the signal:

- Connect an oscilloscope to DAC_OUT2 (PA5) to display the sine wave signal generated by the DAC.
- Connect an oscilloscope to OPAMP1 Out (PA2) to display the amplified (by 2) sine wave signal.

3.12 Current consumption example

3.12.1 Purpose

This example shows how to configure the STM32F30x or STM32F31x system to measure different low-power mode current consumption. The low-power modes are:

- Sleep mode
- Stop mode with RTC
- Standby mode with wake up pin (without RTC)
- Standby mode with RTC

To select the low power modes to be measured, uncomment the corresponding line inside the *stm32f30x_lp_modes.h* file.

Note: STM32F30x or STM32F31x *consumption can be measured on the STM32F3DISCOVERY board by removing jumper JP3, labeled IDD, and connecting an ampermeter.*

Note: When using small packages (48 and 64 pin), the GPIO pins which are not present on these packages must not be configured in analog mode.

3.12.2 Description

After reset, the program waits for the User button connected to PA0 to be pressed to enter the selected low power mode.When the RTC is used, the wakeup from low power mode is automatically generated by the RTC (after 5s).In Sleep mode and Standby mode, press again the User button to exit the low power mode.

The different low power mode configurations are described below:

Sleep mode

- System running at PLL (72 MHz)
- A Flash memory wait state
- Code running from Internal Flash memory
- All peripherals disabled
- Wakeup using EXTI Line (User push-button PA0)

Stop mode

- RTC clocked by LSI
- Regulator in low-power mode
- HSI and HSE OFF. LSI if not used as RTC clock source
- No IWDG
- Flash memory in deep power-down mode
- Automatic Wakeup using RTC clocked by LSI

Standby mode

- RTC OFF
- IWDG and LSI OFF
- Wakeup using Wakeup Pin (PA.00)

Standby mode with RTC clocked by LSI

- RTC clocked by LSI
- IWDG OFF and LSI OFF if not used as RTC clock source
- Automatic Wakeup using RTC clocked by LSI

3.13 Standby mode example

3.13.1 Purpose

This example shows how to put the STM32F30x/31x in Standby mode and wake it up from this mode using RTC Alarm A external reset.

3.13.2 Description

In this example:

- The SysTick timer is initialized.
- The SysTick timer interrupt is enabled in the NVIC.
- The SysTick timer/counter starts in free running mode to generate periodic interrupts. The SysTick timer interrupt is triggered every 250 ms. In the SysTick interrupt handler, LED3 is toggled; this is used to indicate whether the MCU is in Standby or Run mode.
- The EXTI line 0 is configured to generate an interrupt on each rising/falling edge detected on the PA0 pin. The external interrupt is generated every time the PA0 changes level (GND or VDD).

  When a falling or rising edge is detected on the EXTI line, an interrupt is generated. In the EXTI handler routine, the RTC is configured to generate an Alarm event in 3 seconds, after which the system enters Standby mode and LED3 is Off.

After waking-up from Standby mode, the program execution restarts in the same way as after a reset; the RTC configuration (clock source, prescaler) is kept and LED3 toggles again. As a result, there is no need to configure the RTC.

LED3 is used to monitor the system state as following:

- LED3 toggling: system in Run mode
- LED3 off: system in Standby mode
- LED3 toggling: system resumed from Standby mode

These steps are repeated in an infinite loop.

Note: To measure the current consumption in Standby mode, please refer to Chapter 3.12: Current consumption example.

3.14 Stop mode example

3.14.1 Purpose

This example shows how to enter Stop mode and wakeup using EXTI Line interrupts. The EXTI Line sources are PA0 and RTC Alarm.

3.14.2 Description

In this example:

- The EXTI Line0 is configured to generate interrupt on falling edge.
- The EXTI line17 (RTC Alarm) is configured to generate interrupt on rising edge.
- The SysTick timer is programmed to generate an interrupt every 250 ms. In the SysTick interrupt handler, LED3 is toggled, which indicates whether the MCU is in Stop or Run mode.

The system enters Stop mode and waits for the RTC Alarm to be generated every 5s, or for the user push-button to be pressed.

- If the RTC Alarm (EXTI_Line17) has been configured to wakeup the microcontroller from Stop mode, LED4 and LED3 is On.
- If the User push-button (EXTI_Line0) has been configured to wakeup the microcontroller from Stop mode, LED5 and LED3 is On.

LEDs are used to monitor the system state:

- LED3 ON: system in Run mode
- LED3, LED4 and LED5 OFF: system in Stop mode
- LED5 ON if EXTI Line0 is used to exit from Stop
- LED4 ON if EXTI line17(RTC Alarm) is used to exit from Stop.

3.15 RCC (reset and clock controller) example

3.15.1 Purpose

This example shows how to:

- Configure the HSE (High Speed Clock) as an RCC clock
- Use the Clock Security System (CSS) feature to generate an NMI interrupt
- Output the system clock on MCO

3.15.2 Description

For debug purposes, the RCC_GetClocksFreq() function is used to retrieve the current status and frequencies of different on-chip clocks.

You can see the RCC_ClockFreq structure content, which holds the frequencies of different on-chip clocks, using your toolchain debugger.

This example also handles the High Speed External clock (HSE) failure detection: when the HSE clock disappears (broken or disconnected external Quartz), HSE and PLL are disabled (but no change to PLL configuration), HSI is selected as a system clock source and an interrupt (NMI) is generated. In the NMI ISR, the HSE and HSE ready interrupt are enabled. Once the HSE clock recovers, the HSERDY interrupt is generated and, in the RCC ISR routine, the system clock is reconfigured to its previous state (before HSE clock failure). You can monitor the system clock on MCO pin (PA8).

Four LEDs(LED3, LED5, LED7 and LED8) are toggled with a timing defined by the Delay function.

3.16 RTC calendar example

3.16.1 Purpose

This example shows how to setup the RTC prescaler and interrupts used to keep date, time and to generate alarm interrupt.

3.16.2 Description

In this example:

- The RTC prescalers are configured to generate the adequate base time.
- The RTC Alarm is set and configured to generate alarm interrupt each 1s.
- The RTC Time (hh:min:ss) is set.
- The RTC Date (DD-MM-YY) is set

When an RTC Alarm A event occurs, the RTC Alarm interrupt is generated each 1s. At each Alarm interrupt, one LED is toggled.

The current time and date are stored in the showtime and showdate variables, respectively.

3.17 RTC tamper example

3.17.1 Purpose

This example shows how to write/read data to/from RTC Backup data registers and demonstrates the Tamper detection feature.

3.17.2 Description

In this example:

- The RTC clock source is the LSI oscillator clock.

The associated firmware performs the following operations:

- It configures the Tamper pin to be falling edge, and enables the Tamper interrupt.
- It writes the data to all RTC Backup data registers, then check whether the data were correctly written. If yes, LED3 turns on, otherwise LED4 turns on.
- It applies a low level on the TAMPER2 pin (PA00) by pressing TAMPER push button, the RTC backup data registers are reset and the Tamper interrupt is generated.

The corresponding ISR then checks if the RTC Backup data registers are cleared. If so, LED5 is switched ON, otherwise LED6 is switched ON.

3.18 SysTick timer example

3.18.1 Purpose

This example shows how to configure the SysTick timer and use it to generate a 1 ms time base.

3.18.2 Description

In this example:

- The SysTick timer is initialized.
- The SysTick timer interrupt is enabled in the NVIC.
- The SysTick timer/counter starts in free running mode to generate periodical interrupts.
- The SysTick timer interrupt is triggered every 1 ms.
- A Delay function is implemented, based on the SysTick timer end-of-count event.

The eight LEDs are toggled with a timing defined by the Delay function.

3.19 TIM asymmetric example

3.19.1 Purpose

This example describes how to configure the timer 1 and 8 peripheral to generate an asymmetric signal.

3.19.2 Description

TIM8 is configured to generate an asymmetric signal with a programmable phase-shifted signal on TIM8_CH2:

- TIM8 Channel 1 is configured in PWM2 mode
- TIM8 Channel 2 is configured in Asymmetric PWM2 mode
- The counter mode is center aligned mode
- The pulse length and the phase shift are programmed consecutively in TIM8_CCR2 and TIM8_CCR1.

TIM1 is configured to generate the reference signal used by TIM8 on Channel1:

- TIM1 generates a PWM signal with a frequency equal to 1.4560 kHz.
- TIM1 is used as master for TIM8: TIM1 update event resets TIM8 counter to synchronize the reference signal (TIM1_CH1) and the shifted signal (TIM8_CH2).

TIM1 and TIM8 input clock (TIM18CLK) is set to APB2 clock (PCLK2). TIM1 and TIM8 signals are at frequency of (SystemCoreClock / (Period + 1)).

TIM8 generates a signal with the following characteristics:

Pulse length = (TIM8_CCR1 + TIM8_CCR2) / TIM8_CLK

Phase shift = TIM8_CCR1/TIM8_CLK

where TIM8_CLK = (SystemCoreClock / (Period + 1)), as the prescaler is equal to zero.

Connect the following pins to an oscilloscope to monitor the different waveforms:

- PA08 (TIM1_CH1)
- PB08 (TIM8_CH2)
- PB06 (TIM8_CH1)

Note: The shift is measured using the TIM1_CH1 as reference signal.

3.20 TIM combined example

3.20.1 Purpose

This example provides a short description of how to configure the TIM1 peripheral to generate 3 PWM combined signals with TIM1 Channel5.

3.20.2 Description

The TIM1 input clock (TIM1CLK) with prescaler 0 is set to APB2 clock (PCLK2), which is equal to system clock (SystemCoreClock).

SystemCoreClock is set to 72 MHz for STM32F30x or STM32F31x devices.

The objective is to generate 3 combined PWM signal at 8.78 kHz (in center aligned mode):
TIM1_Period = (SystemCoreClock / (8.78*2)) - 1

- The channel 1 duty cycle is set to 50%
- The channel 2 duty cycle is set to 37.5%
- The channel 3 duty cycle is set to 25%

The Timer pulse is calculated as follows:

ChannelxPulse = DutyCycle * (TIM1_Period - 1) / 100

The channel 5 is used in PWM2 mode with duty cycle set to 6.22%

The 3 resulting signals are made of an AND logical combination of two reference PWMs:

- Channel 1 and Channel 5
- Channel 2 and Channel 5
- Channel 3 and Channel 5

The TIM1 waveform can be displayed using an oscilloscope. Connect the following pins to an oscilloscope to monitor the different waveforms:

- PA08 (TIM1_CH1)
- PA09 (TIM2_CH2)
- PA10 (TIM2_CH3)

3.21 TIM complementary signals example

3.21.1 Purpose

This example shows how to perform the following operations using TIM1 timer:

- Configure TIM1 to generate three complementary signals
- Insert a defined dead time value
- Use the break feature
- Lock the desired parameters

3.21.2 Description

TIM1CLK is SystemCoreClock, and TIM1 Prescaler is set to 0.As a result, the TIM1 counter clock is SystemCoreClock (72 MHz).

The objective is to generate a PWM signal at 17.57 kHz:

TIM1_Period = (SystemCoreClock / 17570) - 1

The three Duty cycles are computed as follows:

- The channel 1 duty cycle is set to 50% so channel 1N is set to 50%.
- The channel 2 duty cycle is set to 25% so channel 2N is set to 75%.
- The channel 3 duty cycle is set to 12.5% so channel 3N is set to 87.5%.

The Timer pulse is calculated as follows:

ChannelxPulse = DutyCycle * (TIM1_Period - 1) / 100

A dead time equal to 11/SystemCoreClock is inserted between the different complementary signals, and the Lock level 1 is selected.The break Polarity is set to high level.

The TIM1 waveform can be displayed using an oscilloscope.

- Connect the TIM1 pins to an oscilloscope to monitor the different waveforms:
  - TIM1_CH1 pin (PA08)
  - TIM1_CH1N pin (PA07)
  - TIM1_CH2 pin (PA9)
  - TIM1_CH2N pin (PB00)
  - TIM1_CH3 pin (PA10)
  - TIM1_CH3N pin (PB01)

Note: Connect the TIM1 break pin TIM1_BKIN pin (PB12) to the GND. To generate a break event, switch this pin level from 0V to 3.3V.

3.22 TIM time base example

3.22.1 Purpose

This example shows how to configure the TIM peripheral in Output Compare Timing mode with the corresponding Interrupt requests for each channel, in order to generate 2 different time bases.

3.22.2 Description

TIM3CLK frequency is set SystemCoreClock, to obtain a TIM3 counter clockfrequency of 6 MHz. As a consequence, the prescaler is computed as follows:

- Prescaler = (TIM3CLK / TIM3 counter clock) - 1.
- SystemCoreClock is set to 72 MHz.
- The TIM3_CCR1 register value is equal to 40961, TIM3_CCR1 update rate = TIM3 counter clock / CCR1_Val = 1757.77 Hz.

  As a result, TIM3 Channel 1 generates an interrupt each 6.8ms.
- The TIM3_CCR2 register is equal to 27309, TIM3_CCR2 update rate = TIM3 counter clock / CCR2_Val = 2636.49 Hz.

  As a result, TIM3 Channel 2 generates an interrupt each 4.55ms.
- The TIM3_CCR3 register is equal to 13654, TIM3_CCR3 update rate = TIM3 counter clock / CCR3_Val = 5273.18 Hz.

  As a result, TIM3 Channel 3 generates an interrupt each 2.27ms
- The TIM3_CCR4 register is equal to 6826, TIM3_CCR4 update rate = TIM3 counter clock / CCR4_Val = 10547.9 Hz.

  As a result, TIM3 Channel 4 generates an interrupt each 1.13ms.

When the counter value reaches the Output compare registers values, the Output Compare interrupts are generated and, in the handler routine, 4 leds (LED3, LED4, LED5 and LED6) are toggled at the following frequencies:

- LED3: 878.88 Hz (CC1)
- LED4: 1318.24Hz (CC2)
- LED5: 2636.59Hz (CC3)
- LED6: 5273.95Hz (CC4)

Note: Use LED3, LED4, LED5 and LED6 connected respectively to PE8, PE9,PE10 and PE15 pins and connect them to an oscilloscope to show the different Time Base signals.

3.23 TIM PWM input example

3.23.1 Purpose

This example shows how to use the TIM peripheral to measure the frequency and duty cycle of an external signal.

3.23.2 Description

TIMxclock (TIMxCLK) is SystemCoreClock, and TIMx prescalers are set to 0 so that TIMx counter clock is SystemCoreClock (72 MHz).

TIM2 is configured in PWM Input mode: the external signal is connected to TIM2 Channel2 used as an input pin.

To measure the frequency and the duty cycle, TIM2_CCR2 interrupt request is used, so that the frequency and the duty cycle of the external signal are computed in the TIM2_IRQHandler routine.

The “Frequency” variable contains the external signal frequency:

- TIM2 counter clock = SystemCoreClock,
- Frequency = TIM2 counter clock / TIM2_CCR2 in Hz,

The “DutyCycle” variable contains the external signal duty cycle:

DutyCycle = (TIM2_CCR1*100)/(TIM2_CCR2) in %.

The minimum frequency value to measure is 915 Hz (TIM2 counter clock / CCR MAX).

Note: Connect the external signal to measure the TIM2 CH2 pin (PA01).

3.24 TIM rettrigreable one pulse mode example

3.24.1 Purpose

This example shows how to use the TIM peripheral to generate a Retrigerrable One pulse Mode after a Rising edge of an external signal is received in Timer Input pin.

3.24.2 Description

In this example, we want to obtain aTIM2 counter clock of 36 MHz and a TIM2 inputs clock (TIM2CLK) equal to systemCoreClock (72 MHz):

Prescaler = (TIM2CLK / TIM2 counter clock) - 1

The Autoreload value is 65535 (TIM2->ARR). Aa a result the maximum frequency to trigger the TIM2 input is:

36000000/65535 = 549.3 Hz.

TIM2 is configured as follows:

- The Retrigerrable One Pulse mode is used, the external signal is connected to TIM2 CH2 pin (PA1), the rising edge is used as active edge, and the One Pulse signal is output on TIM2_CH1 (PA0).
- The TIM_Pulse defines the One Pulse value. The pulse value is fixed to:

  Retrigerrable One Pulse value = TIM_Period/TIM2 counter clock

  Retrigerrable One Pulse value = 65535 / 36000000 = 1.8 ms.

Note: Connect the external signal to the TIM2_CH2 pin (PA01)

Connect the TIM2_CH1 (PA00) pin to an oscilloscope to monitor the waveform.

3.25 USB example

3.25.1 Purpose

This example provides a description of how to use the USB-FS-Device on the STM32F30x /STM32F31x devices. It is enumerated as an USB-FS-Device mouse, that uses the native PC Host USB-FS-Device HID driver.

3.25.2 Description

This example shows how the board is recognized as a standard mouse and how its motion will also control the PC cursor.

The LSM303DLHC MEMS accelerometer device mounted on the STM32F3DISCOVERY board is used to emulate the mouse directions.

3.26 WWDG example

3.26.1 Purpose

This example shows how to periodically update the WWDG counter, and how to simulate a software fault which generates an MCU WWDG reset when the programmed time period has expired.

3.26.2 Description

The WWDG timeout is set to 58.2 ms and the refresh window is set to 80 ms. The WWDG counter is refreshed each 43ms in the main program infinite loop to prevent a WWDG reset.

LED4 is also toggled each 43 ms indicating that the program is running. An EXTI Line is connected to a GPIO pin and configured to generate an interrupt on the rising edge of the signal.

The EXTI Line is used to simulate a software failure: once the EXTI Line event occurs by pressing the user push-button, the corresponding interrupt is served. In the ISR, a write to invalid address generates a Hardfault exception containing an infinite loop and preventing to return to main program (the WWDG counter is not refreshed).

As a result, when the WWDG counter falls to 43, the WWDG reset occurs:

- If the WWDG reset is generated, after the system resumes from reset, LED3 turns on.
- If the EXTI Line event does not occur, the WWDG counter is indefinitely refreshed in the main program infinite loop, and there is no WWDG reset.

4 Revision history

Table 2. Document revision history

Date	Revision	Changes
07-Sep-2012	1	Initial release.