AN3983
Application note

STM32F4DISCOVERY peripheral firmware examples

Introduction

This application note describes the peripheral firmware examples provided for the STM32F4DISCOVERY Kit.

These ready-to-run examples are provided to help the user get started quickly with STM32F4xx peripherals and STM32F4DISCOVERY board hardware. Preconfigured projects for EWARM, MDK-ARM, TrueSTUDIO and TASKING toolchains are provided for each example.

These examples are included in the firmware applications package available for download on www.st.com/stm32f4-discovery.

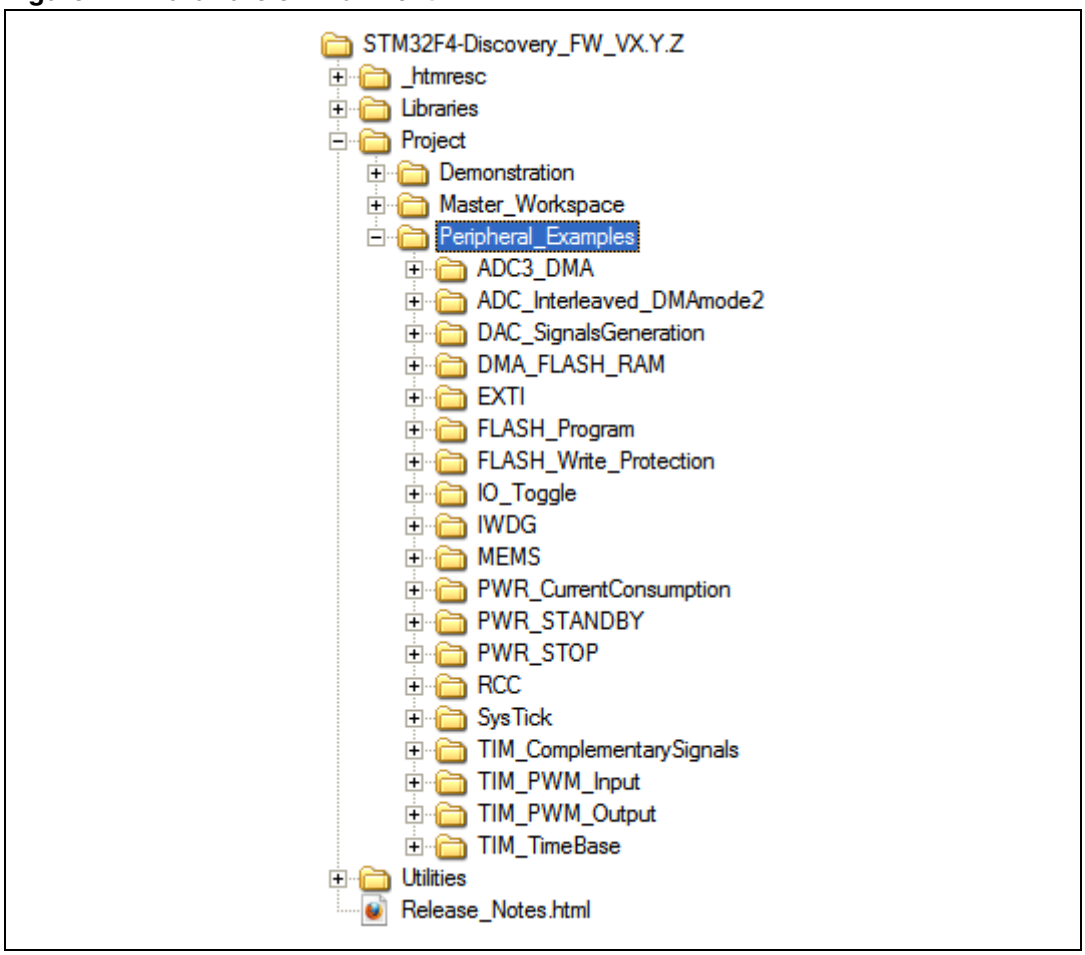
Users are advised to first read the document *Getting started with software and firmware environments for the* STM32F4DISCOVERY *Kit* (UM1467) to familiarize themselves with the STM32F4DISCOVERY Kit.

Contents

1 Peripheral firmware examples structure overview

The peripheral firmware examples are provided within the STM32F4DISCOVERY firmware applications package and are located in the \Project folder as shown in *Figure 1*.

Figure 1. Hardware environment



1. 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 Peripheral firmware examples description

2.1 GPIO toggle example

2.1.1 Purpose

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

2.1.2 Description

In this example:

- GPIOD clock is enabled.
- GPIOD pins 12, 13, 14 and 15 are configured.
- In a `while` loop, the ODR12, ODR13, ODR14 and ODR15 bits are set in the GPIOD output data register (ODR) by setting the corresponding bits in the port bit set/reset register (BSRRL). Then the ODR12, ODR13, ODR14 and ODR15 bits are reset in the GPIOD output data register (ODR) by setting the corresponding bits in the port bit reset register (BSRRH).
- A delay is inserted between setting and resetting the GPIOD ODR12, ODR13, ODR14 and ODR15 bits.

When the program is executed, the four LEDs LD3, LD4, LD5 and LD6 are turned ON then OFF in an infinite loop. The duration between the ON and OFF states corresponds to the inserted delay.

2.2 EXTI example

2.2.1 Purpose

This example shows how to configure an external interrupt line.

2.2.2 Description

In this example:

- PA0 pin is configured in input floating.
- PA0 is configured to be used as 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 pushes on the User push button (EXTI0 interrupt routine), the LED LD4 connected to PD12 is toggled.

2.3 SysTick example

2.3.1 Purpose

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

2.3.2 Description

In this example:

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

The four LEDs LD3, LD4, LD5 and LD6 are toggled with a timing defined by the Delay function.

2.4 Standby mode example

2.4.1 Purpose

This example shows how to put the system in Standby mode and wake it up from this mode using the external reset, RTC Alarm A or WKUP pin.

2.4.2 Description

In this example:

- The system tick timer is initialized.
- The system tick timer interrupt is enabled in the NVIC.
- The system tick timer/counter starts in free running mode to generate periodical interrupts. The system tick timer interrupt is triggered every 250 ms. The LD4 is toggled, indicating whether the MCU is in Standby or Run mode.
- The EXTI line 1 is configured to generate an interrupt on each rising/falling edge detected on the PA1 pin. The external interrupt is generated every time the PA1 changes the 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 5 seconds, after which the system enters Standby mode causing the LD4 to stop toggling.
  - A rising edge on WKUP pin (User push button) or an external reset will wakeup the system from Standby. If neither rising edge on WKUP pin (User push button) nor external RESET are generated within 5 seconds, the RTC Alarm A will wakeup the system.

After wake-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 LD4 restarts toggling. As a result there is no need to configure the RTC.

Two LEDs LD4 and LD3 monitor the system state as follows:

- LD3 ON: RTC configuration failed (system will go to an infinite loop)
- LD4 toggling: system in Run mode
- LD4 off: system in Standby mode

These steps are repeated in an infinite loop.

2.5 Stop mode example

2.5.1 Purpose

This example shows how to put the system in Stop mode and wakeup from this mode using the RTC Wakeup Timer Event connected to EXTI Line 22 or EXTI Line 0.

2.5.2 Description

In this example:

- The EXTI Line0 is configured to generate interrupt on rising edge.
- The EXTI Line22 connected internally to the RTC Wakeup event is configured to generate an interrupt on rising edge each 4 s.
- The SysTick is programmed to generate an interrupt each 250 ms. In the SysTick interrupt handler, LD3 is toggled, which indicates whether the MCU is in Stop or Run mode.

The system enters Stop mode and waits for the RTC Wakeup event to be generated each 4 s, or when the User push button is pressed.

  – If the RTC WakeUp event (EXTI_Line22) is the source of wakeup from Stop, LD4 is toggled.
  – If the User push button (EXTI_Line0) is the source of wakeup from Stop, LD6 is toggled.

LEDs are used to monitor the system state:

  – LD3 toggling: system in Run mode.
  – LD4 toggled: system woken up from Stop using RTC Wakeup Interrupt.
  – LD6 toggled: system woken up from Stop using EXTI Line0 (User push button).

2.6 Current consumption example

2.6.1 Purpose

This example shows how to configure the STM32F4xx system to measure different low power modes current consumption. The low power modes are:

- Sleep mode
- Stop mode with RTC
- Standby mode without RTC and BKPSRAM
- Standby mode with RTC
- Standby mode with RTC and BKPSRAM

To select the low power modes to be measured, uncomment the corresponding line inside the stm32f4xx_lp_modes.h file.

Note: 1 STM32F4xx consumption can be measured on the STM32F4DISCOVERY board by removing jumper JP1, labeled IDD, and connecting an ampermeter.

2 On the STM32F4DISCOVERY board, an extra power consumption (~500uA) is added because the resistor R31 is connected to BOOT0 pin. Remove this resistor to reach correct power consumption values.

2.6.2 Description

After reset, the program waits for the User push button connected to the PA.00 to be pressed to enter the selected low power mode.

When the RTC is not used in the low power mode configuration, press again the User push button to exit the low power mode.

When the RTC is used, the wakeup from low power mode is automatically generated by the RTC (after 4s).

The different low power mode configurations are:

Sleep mode

- System Running at PLL (168 MHz)
- Flash 3 wait state
- Code running from Internal Flash
- All peripherals disabled
- Wakeup using EXTI Line (User push button PA.00)

Stop mode

- RTC Clocked by LSI
- Regulator in LP mode
- HSI, HSE OFF and LSI if not used as RTC Clock source
- No IWDG
- Flash in deep power down mode
- Automatic Wakeup using RTC clocked by LSI (after ~20s)

Standby mode

- Backup SRAM and 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
- Backup SRAM OFF
- Automatic Wakeup using RTC clocked by LSI (after ~20s)

Standby mode with RTC clocked by LSI and BKPSRAM

- RTC Clocked by LSI
- Backup SRAM ON
- IWDG OFF
- Automatic Wakeup using RTC clocked by LSI (after ~20s)

2.7 Flash program example

2.7.1 Purpose

This example describes how to program the STM32F4xx internal Flash.

2.7.2 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 sector feature. The erase procedure starts with the calculation of the number of sectors to be used. These sectors are erased one-by-one by calling the FLASH_EraseSector function.
- 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 in the MemoryProgramStatus variable.

2.8 Flash write protection example

2.8.1 Purpose

This example describes how to enable and disable the write protection for the STM32F4xx Flash.

2.8.2 Description

In this example:

- By maintaining the User push button pressed at Reset, the program will check the write protection status of FLASH_WRP_SECTORS (defined in main.c)
  - If FLASH_WRP_SECTORS are write protected, write protection is disabled. LD6 turns ON if the protection disable operation is done correctly. If not, LD5 turns ON.
  - If FLASH_WRP_SECTORS are not write protected, write protection is enabled. LD4 turns ON if the protection disable operation is done correctly. If not, LD5 turns ON.
- If the User push button is not pressed after reset, the program turns ON LD3.

2.9 IWDG (independent watchdog) example

2.9.1 Purpose

This example shows how to update the IWDG reload counter at regular periods, and how to simulate a software fault generating an MCU IWDG reset on expiry of a programmed time period.

2.9.2 Description

In this example:

- The independent watchdog timeout is set to 250 ms.
- The system tick is configured to generate an interrupt every 250 ms.
- In the system tick interrupt service routine, the independent watchdog counter is reloaded to prevent an independent watchdog reset, and LD3 is toggled.
- The EXTI line 0 connected to PA0 pin is configured to generate an interrupt on its rising edge.
- In the NVIC, this 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 is 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 STM32F4DISCOVERY board), the corresponding interrupt is served. In the ISR, the LD6 turns off and the EXTI line pending bit is not cleared. The CPU executes the EXTI line ISR indefinitely and the system tick 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, LD4 is turned on after the system resumes operation.

2.10 ADC interleaved DMA mode 2 example

2.10.1 Purpose

This example provides a short description of how to use the ADC peripheral to convert a regular channel in triple interleaved mode using DMA in mode 2 with 7.2 Msps.

2.10.2 Description

In this example three DMA requests are generated:

- ADC2 and ADC1 data is transferred (ADC2 data takes the upper half-word and ADC1 data takes the lower half-word).
- ADC1 and ADC3 data is transferred (ADC1 data takes the upper half-word and ADC3 data takes the lower half-word).
- ADC3 and ADC2 data is transferred (ADC3 data takes the upper half-word and ADC2 data takes the lower half-word) and so on.

On each DMA request (two data items are available) two half-words representing two ADC-converted data items are transferred as a word.

A DMA request is generated each time 2 data items are available:

- 1st request: ADC_CDR[31:0] = (ADC2_DR[15:0] << 16) I ADC1_DR[15:0] (step1)
- 2nd request: ADC_CDR[31:0] = (ADC1_DR[15:0] << 16) I ADC3_DR[15:0] (step2)
- 3rd request: ADC_CDR[31:0] = (ADC3_DR[15:0] << 16) I ADC2_DR[15:0] (step3)
- 4th request: ADC_CDR[31:0] = (ADC2_DR[15:0] << 16) I ADC1_DR[15:0] (step1)
- and so on.

The conversion is triggered by software.

ADC1, ADC2 and ADC3 are configured to convert ADC Channel 12. In this way, the ADC can reach 7.2 Msps. The same channel is converted every five cycles. In this example, the system clock is 144 MHz, APB2 = 72 MHz and ADC clock = APB2/2.

Because ADCCLK= 36 MHz and conversion rate = five cycles, the conversion time = 36 MHz/5cyc = 7.2 Msps.

2.11 ADC DMA example

Note: Connect an external signal (ranges from 0 to 3.3 V) to the ADC3 pin (PC.02) to be converted.

2.11.1 Purpose

This example describes how to use the ADC3 and DMA to transfer continuously converted data from ADC3 to memory.

2.11.2 Description

In this example:

- The ADC3 is configured to convert channel7 continuously.
- Each time an end of conversion occurs, the DMA transfers the converted data from the ADC3 DR register to the ADC3ConvertedValue variable in circular mode.
- The system clock is equal to 144 MHz, APB2 clock is equal to 72 MHz and ADC clock is equal to APB2/2.
- Since ADC3 clock is 36 MHz and sampling time is set to 3 cycles, the conversion time to 12-bit data is 12 cycles, so the total conversion time is (12+3)/36 = 0.41 us (2.4 Msps).

2.12 MEMS accelerometer (LIS302DL) example

2.12.1 Purpose

This example shows how to configure the MEMS accelerometer to detect acceleration on X/Y axes and to detect the click/double-click on the Z axis.

2.12.2 Description

After startup, the program checks the MEMS accelerometer status registers and behaves as follows:

- If the board is moved, the acceleration is detected on the X/Y axis and LEDs toggle according to the motion direction and speed.
- If a click is detected on Z axis, LED3 and LED6 toggle during 3 s.
- If a double-click is detected on Z axis, all LEDs toggle during 3 s.

The LED lighting is managed by TIM4 capture compare channels.

2.13 RCC (reset and clock control) example

2.13.1 Purpose

This example shows how to:

- Configure the HSE (High Speed Clock) as RCC clock
- Use the Clock Security System (CSS) feature to generate NMI interrupt
- Output the system clock on MCO2

2.13.2 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 handles also 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 and 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 MCO2 pin (PC9).

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

2.14 DMA Flash RAM example

2.14.1 Purpose

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

2.14.2 Description

DMA2 Stream0 channel0 is configured to transfer the contents of a 32-word data buffer stored in Flash memory to the reception buffer declared in RAM.

The start of the transfer is triggered by software. DMA2 Stream0 channel0 memory-to-memory transfer is enabled. Source and destination addresses incrementing is also enabled.

The transfer is started by setting the Channel enable bit for DMA2 Stream0 channel0. At the end of the transfer, a Transfer Complete interrupt is generated as it is enabled. The Transfer Complete Interrupt pending bit is then cleared.

When the DMA transfer is completed, the DMA Stream is disabled by hardware.

The main application can check the Stream Enable status to detect the end of transfer or it can check the number of remaining transfers, which should be equal to 0 at the end of the transfer.

A comparison between the source and destination buffers is done to check that all data has been correctly transferred.

STM32 Evaluation board's LEDs can be used to monitor the transfer status:

- LD4 is ON when the program starts.
- LD3 is ON when the configuration phase is done and the transfer is started.
- LD5 is ON when the transfer is complete (into the transfer complete interrupt routine)
- LD6 is ON when the comparison result between source buffer and destination buffer is passed.

It is possible to select a different stream and/or channel for the DMA transfer example by modifying defines values in the file main.h.

Note: Only DMA2 Streams are able to perform memory-to-memory transfers.

There are different options to check on the DMA end of transfer:

- Use DMA transfer complete interrupt.
- Use DMA enable state (the DMA stream is disabled by hardware when transfer is complete).
- Use DMA stream transfer counter value (the counter value is decremented when transfer is ongoing and is equal to 0 at the transfer end).
- Use DMA transfer complete flag (polling mode).
- In this example methods 1, 2 and 3 are provided (you can select between method 2 and 3 by uncommenting relative code in waiting loop in the main.c file).

2.15 DAC signals generation example

2.15.1 Purpose

This example provides a short description of how to use the DAC peripheral to generate several signals using DMA controller.

2.15.2 Description

When the user presses the User push button, DMA transfers the two selected waveforms to the DAC.

For each User push button press, two signal are selected and can be monitored on the two DAC channels:

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

2.16 TIM complementary signals example

2.16.1 Purpose

This example shows how to configure the TIM1 peripheral to generate three complementary TIM1 signals, to insert a defined dead time value, to use the break feature and to lock the desired parameters.

2.16.2 Description

TIM1CLK is fixed to SystemCoreClock, TIM1 Prescaler is equal to 0, so the TIM1 counter clock used is SystemCoreClock (168 MHz).

The objective is to generate PWM signal at 17.57 kHz:

TIM1_Period = (SystemCoreClock / 17570) - 1

The three duty cycles are computed as follows:

- Channel 1 duty cycle is set to 50% so channel 1N is set to 50%.
- Channel 2 duty cycle is set to 25% so channel 2N is set to 75%.
- 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 used at High level.

The TIM1 waveform can be displayed using an oscilloscope.

2.17 TIM time base example

2.17.1 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 four different time bases.

2.17.2 Description

The TIM3CLK frequency is set to SystemCoreClock / 2 (Hz), in order to get TIM3 counter clock at 500 kHz:

- Prescaler is computed as follows:

  Prescaler = (TIM3CLK / TIM3 counter clock) - 1
- SystemCoreClock is set to 168 MHz
- The TIM3 CC1 register value is equal to 54618

  CC1 update rate = TIM3 counter clock / CCR1_Val = 9.154 Hz, so the TIM3 Channel 1 generates an interrupt each 109.2 ms
- The TIM3 CC2 register is equal to 27309

  CC2 update rate = TIM3 counter clock / CCR2_Val = 18.31 Hz so the TIM3 Channel 2 generates an interrupt each 54.6 ms
- The TIM3 CC3 register is equal to 13654

  CC3 update rate = TIM3 counter clock / CCR3_Val = 36.62 Hz so the TIM3 Channel 3 generates an interrupt each 27.3 ms
- The TIM3 CC4 register is equal to 6826

  CC4 update rate = TIM3 counter clock / CCR4_Val = 73.25 Hz so the TIM3 Channel 4 generates an interrupt each 13.65 ms.

When the counter value reaches the output compare registers values, the Output Compare interrupts are generated and four pins (PD.12, PD.13, PD.14 and PD.15) in the handler routine are toggled with the following frequencies:

- PD.12: 4.57 Hz (CC1)
- PD.13: 9.15 Hz (CC2)
- PD.14: 18.31 Hz (CC3)
- PD.15: 36.62 Hz (CC4)

2.18 TIM PWM input example

2.18.1 Purpose

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

2.18.2 Description

- The TIMxCLK frequency is set to SystemCoreClock/4 (Hz). The Prescaler is 0, so the counter clock is SystemCoreClock/2 (Hz).
- SystemCoreClock is set to 168 MHz for STM32F4xx Devices Revision A.
- TIM4 is configured in PWM Input Mode: the external signal is connected to TIM4 Channel2 used as input pin.

To measure the frequency and the duty cycle, the TIM4 CC2 interrupt request is used, so the frequency and the duty cycle of the external signal are computed in the TIM4_IRQHandler routine.

The Frequency variable contains the external signal frequency:

- TIM4 counter clock = SystemCoreClock / 2
- Frequency = TIM4 counter clock / TIM4_CCR2 in Hz
- The DutyCycle variable contains the external signal duty cycle:
  DutyCycle = (TIM4_CCR1*100)/(TIM4_CCR2) in %.

The minimum frequency value to measure is 1280 Hz (TIM4 counter clock / CCR MAX).

2.19 TIM PWM output example

2.19.1 Purpose

This example shows how to configure the TIM peripheral in PWM (Pulse Width Modulation) mode.

2.19.2 Description

The TIM3CLK frequency is set to SystemCoreClock / 2 (Hz). To get TIM3 counter clock at 28 MHz, the Prescaler is computed as follows:

- Prescaler = (TIM3CLK / TIM3 counter clock) - 1
- SystemCoreClock is set to 168 MHz for STM32F4xx Devices Revision A.
- TIM3 is running at 42 kHz:

  TIM3 Frequency = TIM3 counter clock/(ARR + 1) = 28 MHz / 666 = 42 kHz
- The TIM3 CCR1 register value is equal to 333, so the TIM3 Channel 1 generates a PWM signal with a frequency equal to 30 kHz and a duty cycle equal to 50%:

  TIM3 Channel1 duty cycle = (TIM3_CCR1/ TIM3_ARR + 1)* 100 = 50%
- The TIM3 CCR2 register value is equal to 249, so the TIM3 Channel 2 generates a PWM signal with a frequency equal to 30 kHz and a duty cycle equal to 37.5%:

  TIM3 Channel2 duty cycle = (TIM3_CCR2/ TIM3_ARR + 1)* 100 = 37.5%
- The TIM3 CCR3 register value is equal to 166, so the TIM3 Channel 3 generates a PWM signal with a frequency equal to 30 kHz and a duty cycle equal to 25%:

  TIM3 Channel3 duty cycle = (TIM3_CCR3/ TIM3_ARR + 1)* 100 = 25%
- The TIM3 CCR4 register value is equal to 83, so the TIM3 Channel 4 generates a PWM signal with a frequency equal to 30 kHz and a duty cycle equal to 12.5%:

  TIM3 Channel4 duty cycle = (TIM3_CCR4/ TIM3_ARR + 1)* 100 = 12.5%

The PWM waveform can be displayed using an oscilloscope.

3 Revision history

Table 1. Document revision history

Date	Revision	Changes
16-Sep-2011	1	Initial release.
23-Sep-2011	2	Changed STM32F-Discovery to STM32F4DISCOVERY Changed *Figure 1: Hardware environment on page 4* Added data to *Section 2.4: Standby mode example on page 6* Updated *Section 2.5: Stop mode example on page 7* (EXTI Line 22 or EXTI Line 0) Added data to *Section 2.10: ADC interleaved DMA mode 2 example on page 11* Updated *Section 2.11: ADC DMA example on page 11* Updated *Section 2.14: DMA Flash RAM example on page 13*