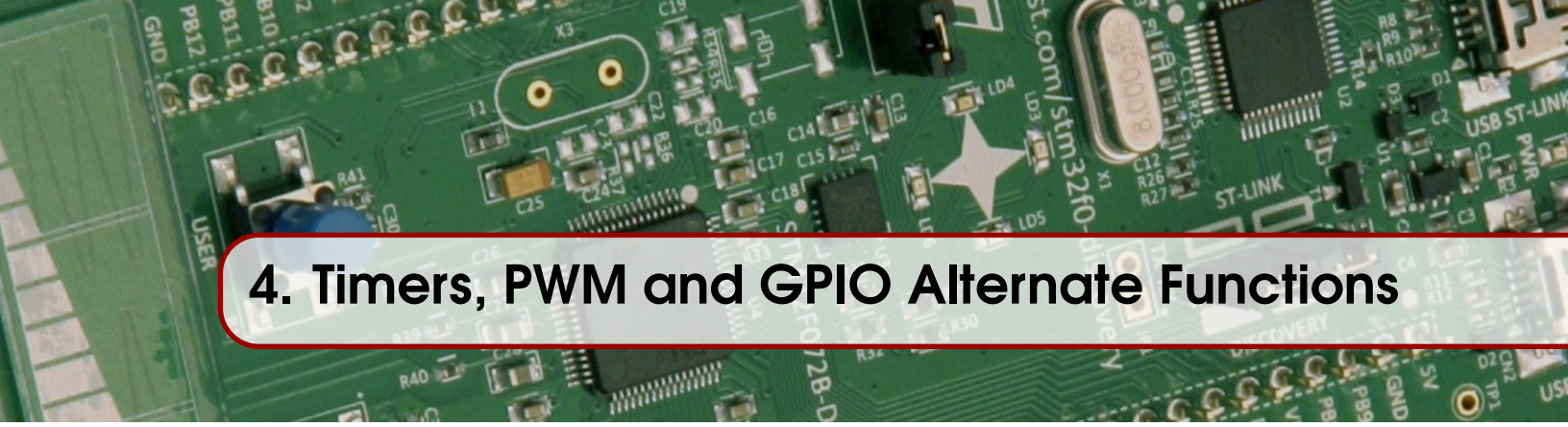

4. Timers, PWM and GPIO Alternate Functions

4.1 Introduction to Timers

The last lab used the SysTick timer to generate periodic interrupts. The SysTick is a simple peripheral with a 24-bit down-counting register that decrements every processor clock cycle. Once this counter reaches zero an interrupt is triggered. The primary purpose of the SysTick timer is to generate a stable system "tick" or timing reference signal.

This reference signal is used to indicate the passage of a unit of time. Some common places where the SysTick is used is within the HAL delay libraries and in the context switcher of real-time operating systems.

An interesting question to ask is why accurate timing so important. Although a processor moves along the applications code in units of the clock cycle, it is difficult and inefficient to measure time simply by counting instruction cycles without dedicated hardware such as a timer. The requirement for accurate timing stems from the fact that many embedded systems require specific timings or schedules for communications and while interacting with the physical world.

At their core, all timers are simple registers which increment or decrement whenever a trigger signal occurs. Upon this basic functionality is built the capability to generate hardware interrupts, like the SysTick. However, the SysTick is the simplest of the onboard timer peripherals.

4.1.1 Systick vs Peripheral Timers

Because the SysTick peripheral has the very simple purpose to generate periodic interrupts, it doesn't offer any features that make it useful for more complicated tasks.

In contrast, the peripheral timers within the STM32F0 are designed to be useful in whatever tasks the user application requires. Because of this they are relatively complicated devices with a large variety of features. Figure 4.1 on the next page shows the block diagram for general-purpose timers 2 & 3 in the STM32F072.

The three main features that peripheral timers offer over simpler devices such as the SysTick are:

1. **Selectable and Prescalable Clock Sources** – Many peripheral timers can choose between multiple sources to use for their clock signal and can divide the input frequency by arbitrary integer values.
2. **Generate Interrupts at Multiple Conditions** – Many peripheral timers can generate interrupts at arbitrary counter values.
3. **Directly Modify GPIO Pins** – Many peripheral timers can directly set or capture GPIO pin states to measure or generate digital signals.

Figure 107. General-purpose timer block diagram (TIM2 and TIM3)

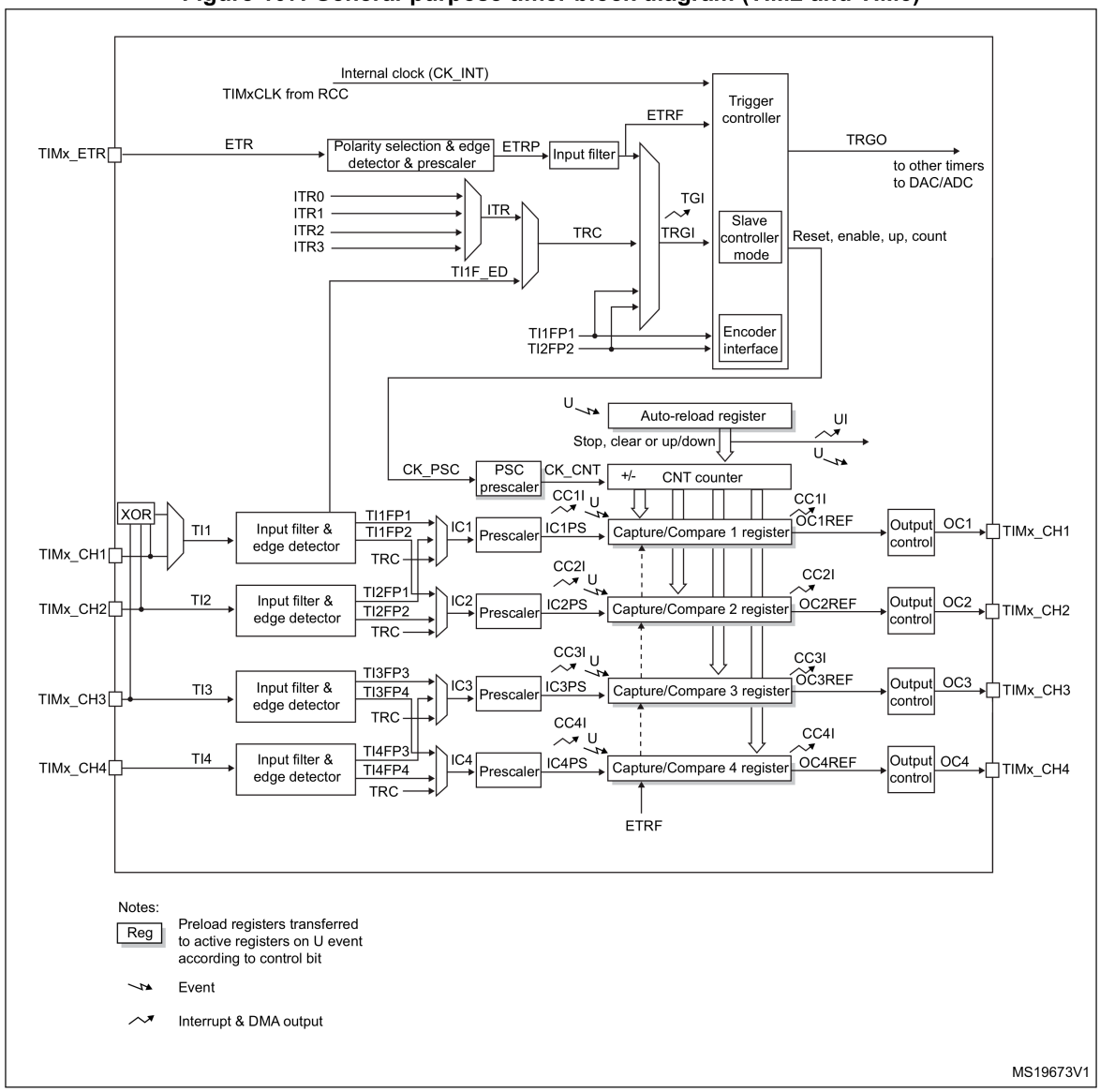

Figure 4.1: Block diagram of timers 2 & 3

Table 7. Timer feature comparison

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
Advanced control	TIM1	16-bit	Up, down, up/down	integer from 1 to 65536	Yes	4	3
General purpose	TIM2	32-bit	Up, down, up/down	integer from 1 to 65536	Yes	4	-
	TIM3	16-bit	Up, down, up/down	integer from 1 to 65536	Yes	4	-
	TIM14	16-bit	Up	integer from 1 to 65536	No	1	-
	TIM15	16-bit	Up	integer from 1 to 65536	Yes	2	1
	TIM16 TIM17	16-bit	Up	integer from 1 to 65536	Yes	1	1
Basic	TIM6 TIM7	16-bit	Up	integer from 1 to 65536	Yes	-	-

Figure 4.2: STM32F072RB hardware timers

4.1.2 Timers in the STM32F072

Figure 4.2 shows the timers available in the specific chip used in the labs. When deciding on a timer to use for an application, it is helpful to understand their capabilities and if they are suitable for the task. A brief breakdown of the information in figure 4.2 is as follows:

- **Timer Type** – There are multiple classes of timers within the STM32F0. These serve as indicators of the appropriate uses for the peripheral. An advanced control timer offers additional and more complex operating modes than a general-purpose one.
- **Timer** – These abbreviated names are used to identify between timers in the documentation. They are also the defined names for the timer structures in the supporting peripheral header files.
- **Counter Resolution** – This indicates how large the hardware counter within the timer is. A 16-bit timer can only count to 2^{16} values before overflowing and wrapping back to zero.
- **Counter Type** – Typically the hardware counter in a timer counts upwards with each clock cycle. However, more advanced timers can also be set to count downwards from a value or to change direction whenever reaching the limit values automatically.
- **Prescaler Factor** – The prescale factor allows the timer to pre-divide the input clock to a slower frequency. The timers within the STM32F0 have the ability to divide the input clock by arbitrary integer factors between 1 and 2^{16}.
- **DMA Request Generation** – Many peripherals have the ability to move data between peripheral registers and system memory without using the processor. This process is called Direct Memory Access (DMA).
- **Capture/Compare Channels** – Most timers have the ability to modify GPIO pins without using the GPIO peripheral. The capture/compare system in a timer is used to generate and measure digital signals on an external pin.

- **Complementary Outputs** – The capture/compare circuity in some timers can generate complementary or opposing outputs on two GPIO pins.

4.2 Using the Timer Documentation

At this point, the peripherals used in the labs are reaching the level of complexity that the lab manual isn't able to provide enough information to complete the lab assignments without additional reading in the datasheets and peripheral reference manual.

Instead, the lab manuals will attempt to provide an overview of the different modes of operation and their relevant registers within the peripheral.

Figure 4.2 on the previous page introduced the set of available timers and indicated their characteristics. For the remainder of this lab manual, we will be focusing on the documentation materials for TIM2 & TIM3.

Timers 2 & 3 are the moderately-advanced versions of the general purpose timers in the STM32F0. Although not as complex as the advanced control timer 1, they feature additional modes and more output channels. Because all the timers in the STM32F0 have a nearly identical interface, by reading through the documentation of the more complex peripherals, you should be able to move to one of the simpler devices without trouble in the assignment if desired.

Organization of Peripheral Documentation

Each section of the peripheral reference manual follows a similar organization regardless of the peripheral documented. These begin with a brief introduction of the purpose and features of the peripheral, followed by a block diagram showing the basic architectural design of the hardware.

Within a peripheral's documentation the section titled *Functional Description* is the most helpful piece of material used when figuring out a new peripheral. The functional description of a peripheral provides the following information:

- Explanation of the purpose of each operating mode
- Basic theory of operation
- Relevant registers and configuration options
- Summarized configuration information
  - Typically the summary provides generic steps for peripheral configuration.
  - These provide a foundation when searching the register documentation for additional details.
- Summary of the output data produced
  - Indicates where data is written for application use.
  - Typically includes figures and graphs depicting peripheral operation.

After the functional description, the reference manual documents each of the peripheral's registers in detail. Assuming that you have read and understand the information within the functional description section, you will spend the majority of your time using the register maps and bit descriptions to configure the peripheral into the desired behavior.

4.3 Using Timers to Generate Interrupts and Events

One of the most basic uses of a timer is to generate periodic interrupts. Unlike the SysTick timer which is typically configured once to a specific base unit of time, the peripheral timers are available to use with arbitrary and varying timing periods. Because their input prescaler can divide the input clock by arbitrary integer values, it becomes possible to generate very long timer periods or match strange timing requirements that aren't multiples of the system clock. Section 18.3 *TIM2 and TIM3 Functional Description* of the peripheral reference manual documents the following sections in greater detail.

Three registers form the core operations of a timer peripheral.

Timer Counter Register (CNT)

The CNT register holds the current value of the counter in the timer. Its size depends on the counter resolution (16/32-bit) indicated in the timer's documentation. Although this register is automatically updated as the timer operates it is possible to read and edit its value manually. The CNT register can be read within the main application loop as a method of counting time or written to modify the timer's operation.

Timer Prescaler Register (PSC)

The PSC register is used to divide the input clock frequency to the timer. Its value is 0-indexed meaning that a value of 0 in the PSC register divides the clock source by 1. (no frequency scaling) Likewise a value of 1 in the PSC register divides the clock source by 2 and causes the timer to count at half the clock frequency. The PSC can divide the input clock by any integer value that fits in its 16-bit width.

Timer Auto-Reload Register (ARR)

Although a timer has a physical hardware size to its counter register, it is possible to set a lower top limit. The value in the ARR register is the trigger point where the timer resets and begins to count a new period. The actual behavior of the timer depends on its counting mode and is discussed more in a later section.

4.3.1 Basic Timer Operation

There are three different counting modes that a timer can operate under; up, down, and center-aligned (up/down). Figure 4.3 on the next page shows a graphical representation of these counting modes.

- **Upcounting Mode** – In upcounting mode, the timer starts at 0 and counts upwards to the auto-reload value in the ARR register. After reaching the ARR limit, the counter resets back to 0.
- **Downcounting Mode** – In downcounting mode, the timer starts at the auto-reload value and counts downwards until reaching 0. After reaching zero, it resets back to the ARR value.
- **Center-aligned Mode (up/down counting)** – In center-aligned mode, the timer starts by counting upward from 0 until the auto-reload value. Once the upper limit is reached, the counting direction reverses, and the timer counts downwards towards zero.

Basic Timer Interrupts

Regardless of the counting mode used, whenever the timer reaches a limit and is reset to a new value is called an *Update Event* (UEV). Additionally, center-aligned counters can be configured to trigger update events whenever the timer changes to a specific direction.

The UEV is one of the possible interrupt sources in a peripheral timer. Typically this source is used to generate periodic interrupts, but with a very flexible and potentially long duration period.

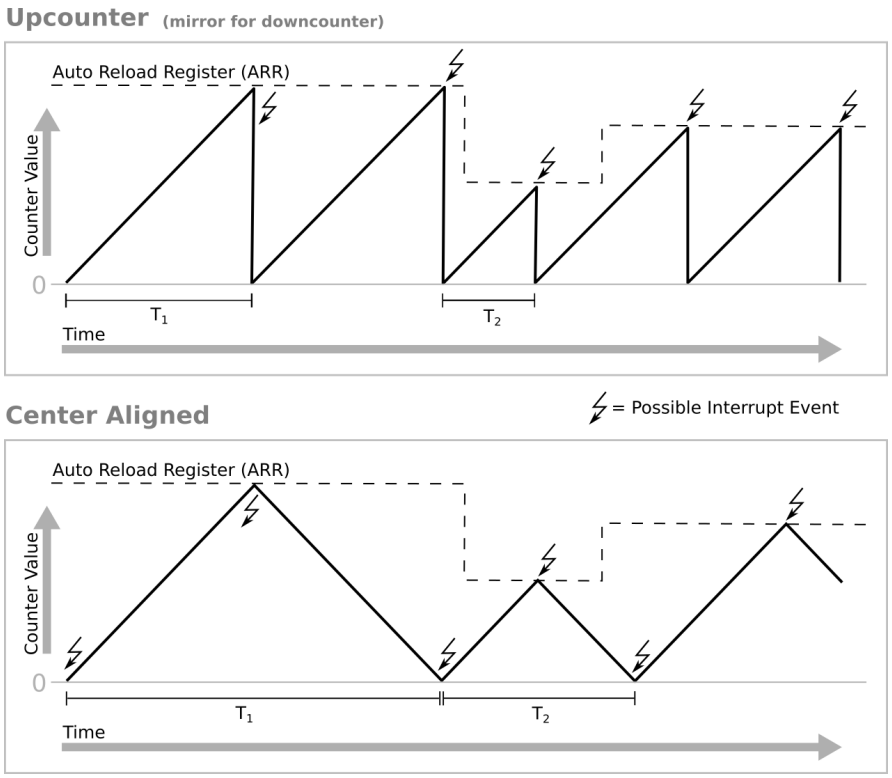

Figure 4.3: Timer counting modes and update interrupts

Configuring Basic Timer Settings

In addition to the three main timer registers, there are four control registers used to set timer parameters. These are used to enable the timer, set the counting direction, enable the UEV interrupt, and clear the pending interrupt flag in a handler.

- **Control Registers 1 & 2 (CR1 & CR2)** – These hold the primary configuration options of the timer. For example, before the timer can operate it must be enabled using the *Counter Enable* (CEN) bit.
- **DMA/Interrupt Enable Register (DIER)** – Controls the possible interrupt sources in the timer.
- **Status Register (SR)** – Contains pending flags that must be cleared in interrupt handlers.

4.3.2 Selecting Timer Frequency and Interrupt Period

The peripheral timers in the STM32F0 have the ability to use a few different sources for their clock input.

- **Internal Peripheral Clock** – This is the most commonly used clock source and is determined by the clock distribution system. In our toolchain, this frequency is configured by default to be 8Mhz by the STMCubeMX program during project generation.
- **External Clock Pin** – Many timers have the ability to use an external clock source provided by an input pin. This source may have a unique frequency or higher accuracy than the processor clock.

- **Internal Trigger Inputs** – Some peripherals can generate trigger events that are visible throughout the device. This mode is used to chain timers together or counting events produced by other types of peripherals.

Using the Prescaler

The primary purpose of the timer prescaler is to determine the base unit of time that a single "tick" or count within the actual timer represents. Consider a timer operating at 8Mhz, at this frequency the period between each update of the timer's value would be one-eighth of a micro-second or 125ns. While it is possible to use this unit of time as a basis for counting longer periods, it may not be convenient.

The prescaler allows us to convert the input clock frequency into more practical units. For example, if we wanted to count something with a duration in milliseconds, it would be reasonable to divide the input 8Mhz clock by a factor of 8000, giving us a base unit of 1ms between each timer count.

Within the timer hardware, the value of the prescaler register (PSC) is 0-indexed. This means that the value written to the PSC register should always be one less than the desired prescaler value. Conveniently this also means that the default value of 0 in the PSC register divides by 1, performing no frequency division.

Calculating Register Values

Using both the prescaler and auto-reload register makes configuring a timer to have a specific period trivial. First, consider the unit of time that your desired period is in, and use the prescaler to convert the timer's base unit to either the same or a convenient multiple/divisor. Afterward, use the auto-reload register to count the desired number of time units to make the target period.

If you are attempting to generate a very long period, you may have to use larger or more granular base units because the physical size of the timer may not be sufficient to count that high at finer resolutions. However, in many cases, it is possible to use small prescaler values to have finer granularity when counting. The following equations may be helpful when selecting prescaler and auto-reload values from a target period or frequency.

$$T_{TARGET} = \frac{(PSC-1)}{f_{CLK}} * ARR \qquad f_{TARGET} = \frac{f_{CLK}}{(PSC-1) * ARR}$$

$$PSC = \frac{f_{CLK}}{ARR * f_{TARGET}} - 1 \qquad ARR = \frac{f_{CLK}}{(PSC+1) * f_{TARGET}}$$

■ **Example 4.1 — Calculating ARR and PSC Values.** For this example, we will set a timer to produce an interrupt at 20Hz assuming that the timer's input clock is 8Mhz.
A target frequency of 20Hz gives a 50ms period. To achieve this, we can divide the 8Mhz clock by 8000 to reduce our timer's frequency to 1kHz. Counting at 1 kHz gives us 1 ms per timer count, so we need 50 counts to reach our target period. We can get our desired interrupt by setting the PSC to 7999, the ARR to 50, and enabling the UEV interrupt in the control registers. ■

Exercise 4.1 — Using Timer Interrupts. In this exercise, you'll set up a timer such that the update event (UEV) triggers an interrupt at a specific period. Timer peripherals allow for greater flexibility in choosing an interrupt period over manually counting in the SysTick handler. To complete this exercise, you will need to carefully read through the control register maps in the peripheral reference manual to determine the proper option bits to set.

1. Enable the timer 2 peripheral (TIM2) in the RCC.
   - This lab will use timers 2 and 3. While all of the timer peripherals can produce interrupts, their configuration registers often have slight differences.
2. Configure the timer such that its update event (UEV) occurs at 4 Hz.
   - **The default processor frequency of the STM32F072 is 8 MHz. Use this value when calculating the timer parameters.**
   - A typical approach is to set the timer's base frequency and the target period into the same units.
     - See example 4.1 on the previous page and use the prescaler (PSC) register.
   - Once the timer and target period are in similar units, set the auto reload register (ARR) to count the number of units to achieve the target.
     - Pay attention to the bit-size of the timer. For example, a 16-bit timer can only count up to 65535. If your target ARR is outside of that range, you'll need to adjust the prescaler (change units) to scale the ARR required.
3. Configure the timer to generate an interrupt on the UEV event.
   - Use the DMA/Interrupt Enable Register (DIER) to enable the update interrupt.
4. Configure and enable/start the timer
   - Although the timer's clock source has been enabled in the RCC, the timer has its own enable/start bit in the control registers.
   - Note that you should not enable a timer until you've finished setting all the basic parameters and options. (Not referring to the RCC enable, always enable the peripheral clock in the RCC first!)
5. Set up the timer's interrupt handler, and enable in the NVIC.
   - The timer will only have a single generic interrupt reserved for it in the vector table. (not a specific interrupt about the UEV)
   - Look for an interrupt named after the timer you are using.
6. Toggle between the green (PC8) and orange (PC9) LEDs in the interrupt handler.
   - Remember to initialize the LED GPIO pins in your main function.
   - To get the alternating flash pattern, set one of the LEDs active in the GPIO initialization.
   - Don't forget to clear the update interrupt pending flag in the status register of the timer peripheral.
7. Compile and load the application onto the Discovery board.

■

4.4 Using Timers to Generate or Measure Signals

The basic modes of operation in a timer allow for a very flexible method of generating interrupts by modifying the timer's frequency and top value. However, the timers in the STM32F0 have the additional capability of generating and measuring physical waveforms by directly interfacing with GPIO pins. These features are made possible by the Capture/Compare Unit in the timer.

When configured in a capture/compare mode, the timer has two additional registers with important functions. In figure 4.2 on page 3 one of the columns indicated how many capture/compare channels were present within each timer. Each capture/compare channel functions independently but has an identical interface with its own set of registers.

- **Capture/Compare Registers (CCRx)** – This register holds capture/compare data. Its specific use depends on the selected mode.
- **Capture/Compare Mode Registers (CCMRx)** – Selects between capture/compare modes and configures their operation.
- **Capture/Compare Enable Register (CCER)** – Enables/disables the capture/compare channel outputs.

The three basic modes that the capture/compare channels can be configured into are input capture mode, output compare mode, and pulse-width modulation mode. Although these modes are primarily intended to interface with their associated GPIO pins, they can generate interrupts on their respective trigger conditions.

4.4.1 Input Capture Mode

When configured into input capture mode, a capture/compare channel latches the current value of the timer's counter into the appropriate CCRx register whenever a connected external GPIO pin changes. This mode allows very accurate measurement of the time elapsed between changes on the pin.

Similar to the EXTI, the input capture hardware can be set to trigger on either the rising or falling edges of a signal. However, the input capture system has a configurable digital filter which can be used to remove glitches and other noise from the signal. The primary purpose of the input capture mode is to measure precise timing-based signals such as those from a single-wire serial bus or a motor encoder.

4.4.2 Output Compare Mode

The output compare mode directly modifies the output of a GPIO pin whenever the timer's counter matches the value stored in the CCRx register. Depending on the configuration, an output compare channel can set, clear or toggle its pin on a counter match.

Using the output compare mode it is possible to create arbitrary digital waveforms on the output. Typically, the ARR register is set to provide a constant period to the timer, and the user's application produces the desired output by modifying the CCRx register while the timer is running. Figure 4.4 on the next page shows an up-counting timer toggling an output compare pin on every match of the CCRx register.

4.4.3 Pulse-Width Modulation (PWM) Mode

The pulse-width modulation mode of the capture/compare system is a more advanced form of output compare. Although pulse-width modulation (PWM) is simple once understood, it can be confusing at first.

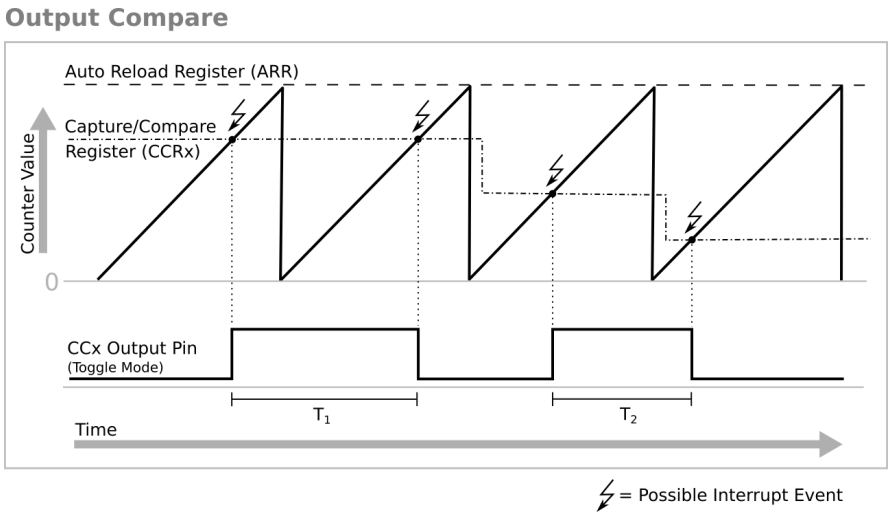

Figure 4.4: Output Compare mode with pin toggle on compare match.

About Pulse-Width Modulation

PWM is a method of approximating an analog signal using only digital hardware. It operates by using a high-frequency rectangular-wave signal where every period is a ratio of on and off time. This ratio of on/off timing is known as the *duty-cycle* and represents an analog voltage ranging between the low and high voltages of the digital signal.

For example, a period of the rectangular wave with a 50% duty cycle would spend half the period at the high (on) output voltage and the other half at the low (off). A period with 0% duty cycle remains low for the entire duration, and one with a 100% duty cycle remains high. It is possible to see how the duty cycle of a PWM signal represents an analog voltage by considering what the average voltage of the digital signal was over the entire period. A digital waveform that was high (on) for half the time and off for the rest would average to one-half the original voltage.

Naturally, the concept of PWM does not make sense at low frequencies; it would just appear as a series of different length pulses. However, at high frequencies many physical systems cannot respond quick enough to shut-off or turn-on with each output transition. This is the basis of a mechanical and electrical low-pass filter which averages or smooths out the high-frequency transitions of the PWM signal into an approximated analog voltage.

PWM is used for a variety of purposes. A few common examples are audio, light dimming, and motor speed control. Figure 4.5 on the following page shows how an analog sine-wave is approximated by a digital PWM signal.

Operation of Capture/Compare PWM Mode

The PWM mode operates almost exactly as the output compare mode of the timer. The difference is that the output pin state is also changed whenever the timer counter resets to begin a new period. There are different modes of PWM that the capture/compare system can generate. Figure 4.6 on page 12 demonstrates one of the possible PWM modes. In the mode shown in the figure, the output pin begins the PWM period at a low state and is set high once the timer's counter matches the CCRx register. This output is set low again when the next period starts, and the overall ratio of on/off is set by the location of the CCRx value in comparison to the ARR register.

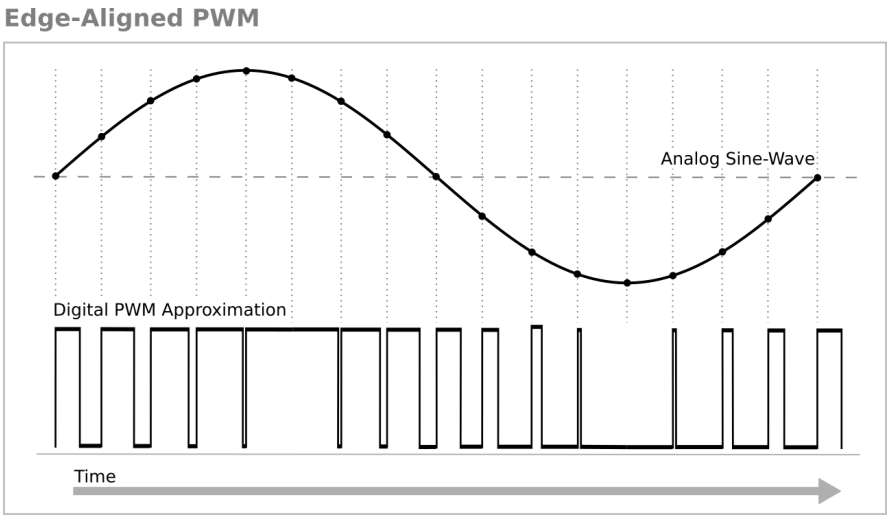

Figure 4.5: PWM approximation of an analog sine-wave.

The important thing to note about PWM is that the signal has both a duty-cycle and frequency. The frequency of the PWM is set by the timer's frequency and the ARR register. The duty-cycle is set by the ratio of the CCRx and ARR registers. Typically the frequency of the PWM signal is irrelevant to the approximated analog voltage. However, it must remain high enough such that the physical system can't directly respond to the transitions in a PWM period.

Exercise 4.2 — Configuring Timer Channels to PWM Mode. In this exercise you will set up PWM output to dim LEDs on the Discovery board. By changing the duty-cycle of the generated waveform you can directly control the apparent brightness of the LED. For this exercise you will be using capture/compare channels 1 & 2 of timer 3.

1. Enable the timer 3 peripheral (TIM3) in the RCC.
2. The period of the PWM signal is determined by the timer's update period. Configure the timer to have a UEV period of 800 Hz.
   - Follow the previous exercise as a template, but do not enable/start the timer.
   - Set the prescaler such that you have a reasonable range of counter values between zero and the top limit. Otherwise your timer will be too granular to be able to make fine adjustments to the duty-cycle of the PWM.
   - Do not enable the update interrupt or set up the handler, we will not be using interrupts.
3. Use the Capture/Compare Mode Register 1 (CCMR1) register to configure the output channels to PWM mode.
   (a) Channels 1 & 2 are configured by the CCMR1 register, channels 3 & 4 by the CCMR2 register.
   (b) Examine the bit definitions for the CC1S[1:0] and CC2S[1:0] bit fields. Ensure that the channels are set to output type.
   (c) Examine the bit definitions for the OC1M[2:0] bit field. Set output channel 1 to *PWM Mode 2*.

(d) Use the OC1M[2:0] bit field to set channel 2 to *PWM Mode 1*.
- The OC2M bits operate identically to the OC1M and are not documented separately.
- You will see the difference between the different PWM modes in a later exercise.

(e) Enable the output compare preload for both channels.

4. Set the output enable bits for channels 1 & 2 in the CCER register.
5. Set the capture/compare registers (CCRx) for both channels to 20% of your ARR value.

The previous sections mention that the frequency of a PWM signal isn't nearly as important as the duty-cycle, provided that the frequency is high enough that the system can't directly respond to the on-off periods. This is true when dimming LEDs, although the lower-limit isn't how fast the LED can respond to the electrical signal, but how fast the human eye can distinguish separate blinks.

When both the light and viewer are stationary, the human eye has difficulty seeing the blinking transitions past 70 Hz. However, many people can see noticeable flicker in moving lights even above 500 Hz. Because of this, you'll be using 800 Hz as the base frequency for the PWM.

■

4.5 Configuring GPIO pins to Alternate Function Mode

In normal operating conditions each GPIO peripheral controls the output state of its pins. Because of this, how does another peripheral such as a timer modify a pin's state? The answer lies within one of the four available pin modes in the GPIO MODER register. In the first lab we used the Input and Output modes of a GPIO pin. However there are two others that we haven't yet discussed. These are the Analog and Alternate Function modes. Allowing a pin to connect directly to internal peripherals of the STM32F0 is the purpose of the alternate function mode.

Unlike the EXTI which has the SYSCFG multiplexers, peripherals such as the timers do not have access to arbitrary pins. Instead, the peripheral's internal signals are hardwired only to a handful of pins scattered around the chip.

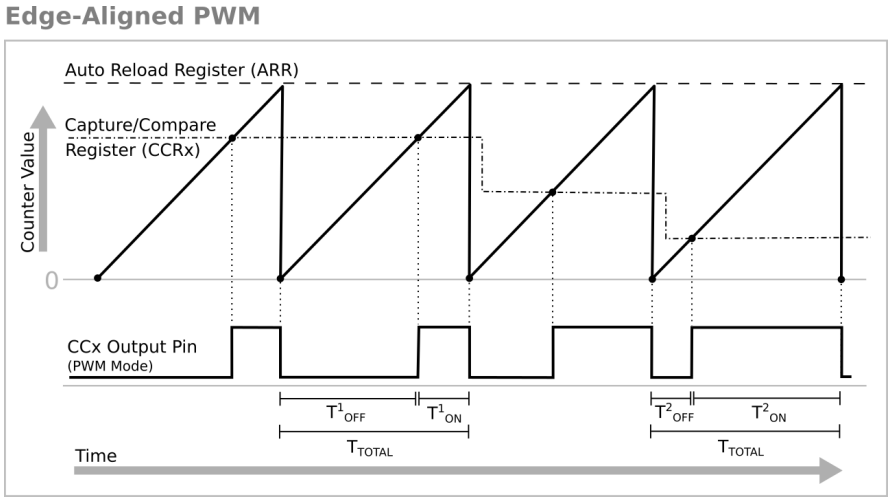

Figure 4.6: Edge-aligned PWM mode and output pin state.

The GPIO alternate function system allows the user to have a few different options when selecting pins for a peripheral. Because there are a large number of possible peripheral signals in relation to the number of pins, many pins have multiple alternate functions shared across multiple peripherals. Although a pin may have multiple functions, only a single one can be selected at a time. When using many peripherals in an application you may need to carefully plan the pins used to ensure that all of them can reach one of their limited output pins.

4.5.1 Finding Available Pins on a Device Package

Alternate function assignments are specific to the STM32F0 device used. This means that the information on pin alternate functions is found within the device datasheet and not the peripheral reference manual.

Open the STM32F072xB datasheet and navigate to Section 4 *Pinouts and Pin Descriptions*. This section provides documentation on all of the chip packages available for the device. It also provides tables with pinout details, including alternate functions.

Navigate past the package maps until you find Table 13 *STM32F072x8/xB pin definitions*. This table provides pin number to pin name mappings, pin characteristics and limitations, and available alternate functions. Figure 4.7 on the next page shows an annotated portion of table 13.

Physical Pin Numbering

Beginning from the left-side of the table, the first set of column headers list the different packages available for the chip. These columns provide the physical pin numbering of the chip for that specific package. The STM32F072xB on the Discovery board we are using has a 64-pin low-quad-flat-pack (LQFP64) chip package; this column has been highlighted in orange. Notice that the lowest two rows do not have a number and are highlighted in red, this indicates that the specific pin on that row does not physically exist in the given package.

When selecting pins, make sure they exist physically on the device package you are using! All GPIO outputs exist on the silicon die of the STM32F0. However, there aren't enough pins available on every type of chip packaging, and they are not all wired out.

Pin Name and Characteristics

The next column titled "Pin Name" (highlighted in green) gives the conventional pin name which indicates the specific GPIO port and location within the peripheral registers. When designing a hardware circuit around an STM32F0 device the information in this table is required to map pin names used in software to physical pin numbers on the chip.

The next three columns give information about the input/output circuitry driving the pin. The definitions of the acronyms used are listed in Table 12.

Alternate and Additional Functions

The "Alternate Functions" column (highlighted in blue) lists the various peripheral signals available for the pin. Once configured into alternate function mode the GPIO peripheral connects one of these signals. The specific function connected is selected by the GPIO Alternate Function Registers (AFR).

The "Additional Functions" column lists pin capabilities while in analog mode. These are discussed in a later lab which introduces the analog peripherals of the STM32F0.

■ **Example 4.2 — Finding Pins With an Alternate Function.** This example demonstrates selecting a pin which connects to the fourth capture/compare channel of timer 3.

Table 13. STM32F072x8/xB pin definitions

Pin numbers						Pin name (function upon reset)	Pin type	I/O structure	Notes	Pin functions	
UFBGA100	LQFP100	UFBGA64	LQFP64	LQFP48/UFQFPN48	WLCSP49					Alternate functions	Additional functions
K5	33	H5	24	-	-	PC4	I/O	TTa	-	EVENTOUT, USART3_TX	ADC_IN14
L5	34	H6	25	-	-	PC5	I/O	TTa	-	TSC_G3_IO1, USART3_RX	ADC_IN15, WKUP5
M5	35	F5	26	18	G5	PB0	I/O	TTa	-	TIM3_CH3, TIM1_CH2N, TSC_G3_IO2, EVENTOUT, USART3_CK	ADC_IN8
M6	36	G5	27	19	G4	PB1	I/O	TTa	-	TIM3_CH4, USART3_RTS, TIM14_CH1, TIM1_CH3N, TSC_G3_IO3	ADC_IN9
L6	37	G6	28	20	G3	PB2	I/O	FT	-	TSC_G3_IO4	-
M7	38	-	-	-	-	PE7	I/O	FT	-	TIM1_ETR	-
L7	39	-	-	-	-	PE8	I/O	FT	-	TIM1_CH1N	-

Figure 4.7: Subset of table 13 - STM32F072x8/xB Datasheet

Begin by browsing through the alternate functions column in Table 13 until you find a signal titled "TIM3_CH4." The names of alternate functions always begin with an abbreviation of their peripheral, followed by a short designator which is indicated somewhere in the functional description. Figure 4.7 shows one of the possible pin choices with the desired function circled in blue. Examining the pin information across the row we can see that this pin belongs to GPIOB (circled in green) and is the 27th physical pin on the LQFP64 package used on the Discovery board (circled in orange).

■

4.5.2 Selecting an Alternate Function

Because a pin can only be connected to a single alternate function at a time, the GIPO peripherals have control registers dedicated to selecting between the possibilities. These are the *Alternate Function High Register* (AFRH) and the *Alternate Function Low Register* (AFRL).

Within these registers are regions of bits that select alternate functions by their alternate function number for the pin. These function numbers are documented in Tables 14-19 of the *Pinouts and Pin Descriptions* section in the device datasheet.

■ **Example 4.3 — Determining an Alternate Function's Number.** Figure 4.8 on the following page shows a portion of Table 15 in the pinouts and pin descriptions section of the device datasheet. This table lists the alternate functions for the GPIOB pins of the device. Continuing from the previous example, we can begin by finding the row representing PB1, the pin chosen with the TIM3_CH4 function found in Table 13.

Table 15. Alternate functions selected through GPIOB_AFR registers for port B

Pin name	AF0	AF1	AF2	AF3	AF4	AF5
PB0	EVENTOUT	TIM3_CH3	TIM1_CH2N	TSC_G3_IO2	USART3_CK	-
PB1	TIM14_CH1	TIM3_CH4	TIM1_CH3N	TSC_G3_IO3	USART3_RTS	-
PB2	-	-	-	TSC_G3_IO4	-	-
PB3	SPI1_SCK, I2S1_CK	EVENTOUT	TIM2_CH2	TSC_G5_IO1	-	-
PB4	SPI1_MISO, I2S1_MCK	TIM3_CH1	EVENTOUT	TSC_G5_IO2	-	TIM17_BKIN
PB5	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM16_BKIN	I2C1_SMBA	-	-
PB6	USART1_TX	I2C1_SCL	TIM16_CH1N	TSC_G5_IO3	-	-
PB7	USART1_RX	I2C1_SDA	TIM17_CH1N	TSC_G5_IO4	USART4_CTS	-

Figure 4.8: Subset of table 15 - STM32F072x8/xB Datasheet

Looking across the columns of the PB1 row lists the same alternate functions listed in Table 13. The difference in this table is that the column header's indicate an alternate function number. Finding the TIM3_CH4 (circled in blue), we can look up to the column header and see the alternate function number is "AF1." (circled in red)

By programming the bit pattern for AF1 into the bit region representing PB1 in the GPIOB alternate function registers we can select the capture/compare channel 4 of timer 3 to output on that pin. You will need to read the register map for the GPIO AFRH & AFRL registers, as well as check the peripheral header files to see how the GPIO structure defines them. ■

Exercise 4.3 — Configuring Pin Alternate Functions. All four of the LEDs on the Discovery board connect to timer capture/compare channels. This enables us to control their apparent brightness using PWM. In the previous exercise you used two of the LEDs in timer 2's interrupt. For this portion of the lab, you'll use the remaining two.

1. Look up the alternate functions of the red (PC6) and blue (PC7) LEDs by following examples 4.2 and 4.3.
   - Alternate functions that connect to the capture/compare channels of timers have the form: "TIMx_CHy".
2. Configure the LED pins to alternate function mode, and select the appropriate function number in alternate function registers.
   - Alternate function numbers for each pin are listed in table 15 of the device datasheet.
   - The alternate function registers are defined as an array in *stm32f0xb.h*. You'll need to check the register map in the peripheral manual to determine what alternate function register to modify for the pins you are using.
3. Although we configured the matching capture/compare channels first in this lab, typically you choose pins first and then work with the timer channels available.
4. Compile and load your application onto the Discovery board.

■

Exercise 4.4 — Measuring PWM Output. In exercise 4.2 on page 11 you configured channel 1 to *PWM mode 2* and channel 2 to *PWM mode 1*. In this exercise you will be exploring the difference between the two modes and the effect of the CCRx register on the output duty cycle.

1. Connect the Saleae logic analyzer to pins PC6 and PC7, and start a capture with the PWM running.
2. Considering that both channels have their CCRx values set to 20% of the ARR, what is the difference between the two PWM modes?
3. Experiment with a variety of CCRx values for both channels.
    - What does increasing the CCRx value do for each PWM mode?
    - The maximum value that an be used in the CCRx register is the ARR value. What is the relationship between PWM duty cycle and the CCRx, ARR registers?

■