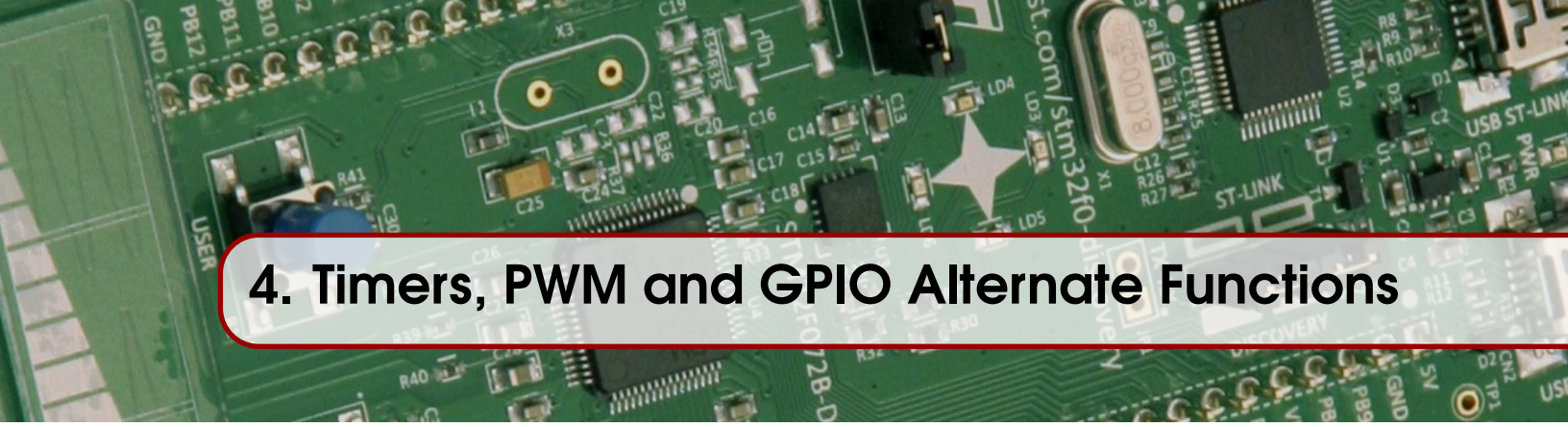

4. Timers, PWM and GPIO Alternate Functions

4.1 Introduction to Timers

The last lab used the SysTick timer to generate a periodic interrupt. The SysTick is a simple peripheral with a 24-bit down-counting counter that triggers an interrupt as it reaches zero. The primary purpose of the SysTick timer is to generate a stable system "tick" or heartbeat/timing reference signal.
this heartbeat signal is used to signal the passage of a unit of time. Some common places where the SysTick is used is in the HAL delay libraries and in the context switcher of real-time operating systems.
Why is getting accurate timing so important? Truth is it's hard and inefficient to measure time simply by the processor without some sort of dedicated hardware like a timer.
Timers are at their core simply a register the increments or decrements whenever a trigger signal occurs. Upon this are built the capability to generate hardware interrupts, like the SysTick. However, the SysTick is the simplest of the on-board timer peripherals.

4.1.1 Systick vs Peripheral Timers

Because the SysTick peripheral has a very simple purpose, to generate periodic interrupts, it doesn't offer any features that make it useful for more complicated tasks.
Unlike the SysTick, peripheral timers within the STM32F0 are designed to be useful in whatever tasks the user application can come up with. Because of this they are fairly complicated devices with a large variety of features. Figure 4.1 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 use multiple signals as clock sources 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.

4.1.2 Timers in the STM32F072

Figure 4.2 shows the timers available in the STM32F072 chip we are using. When deciding on a timer to use for an application, it is helpful to understand their capabilities and if they are suitiable for the task. A brief breakdown of the information in figure 4.2 is as follows:

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

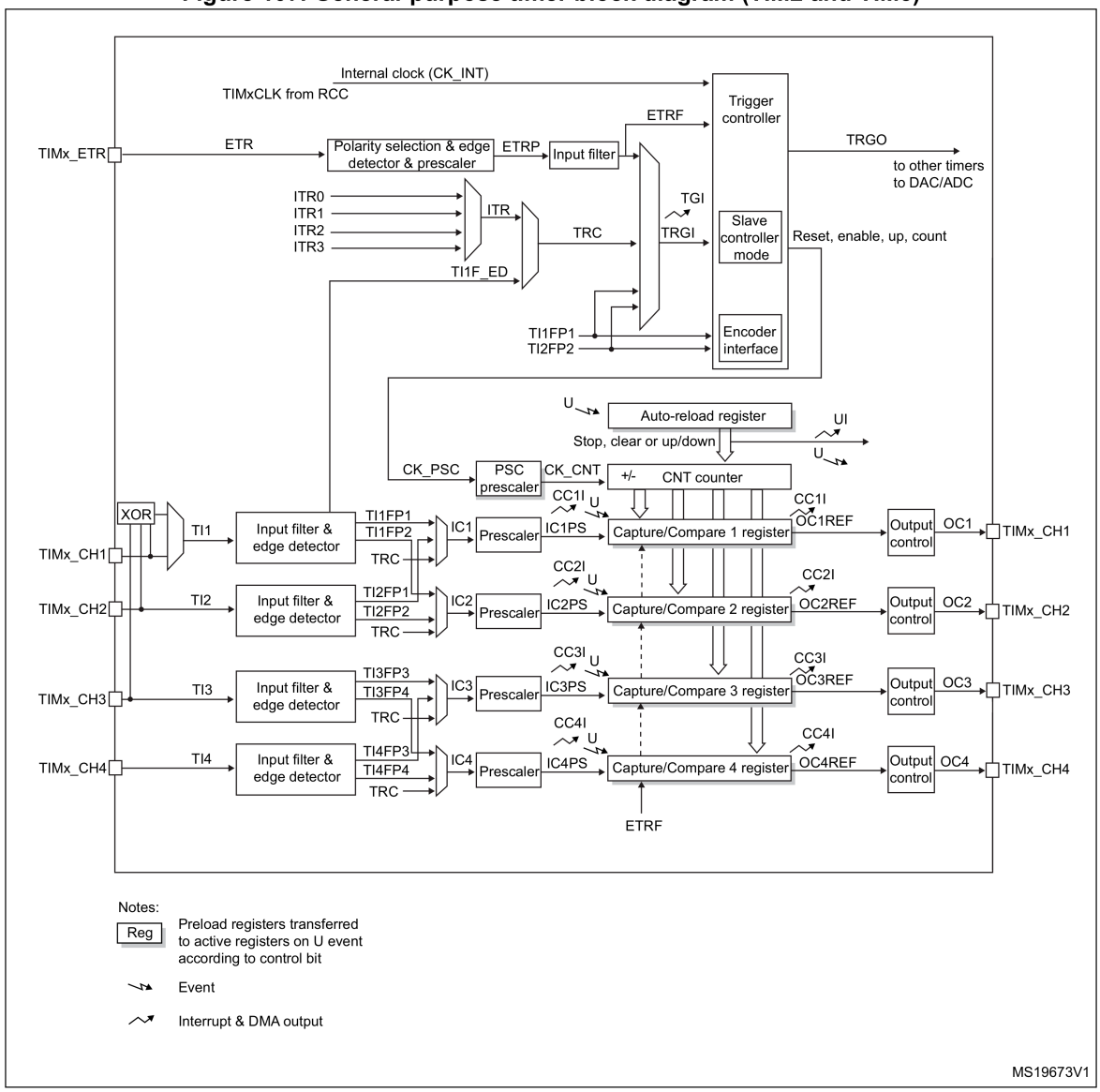

Figure 4.1: Block diagram of timers 2 & 3

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

- **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 automatically change direction whenever reaching the limit values.
- **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 main processor. This process is called direct memory access (DMA).
- **Capture/Compare Channels** – Most timers have the ability to directly 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.
- **Complimentary Outputs** – Some capture/compare circuity has the ability to generate complimentary or opposite outputs on GPIO pins. This generates an effect similar to differential signaling

4.2 Using the Timer Documentation

At this point the peripherals that the labs are going to be discussing are going to be reaching the level of complexity that the lab manuals are not going to be able to provide enough documentation to complete the lab assignments without additional reading in the datasheets and peripheral reference manual.
Instead the lab manual will provide an overview of the different modes of operation of the peripheral And provide some insight as to what registers are important in that mode. the assignment will provide meaningful steps to accomplish the end goal.
In figure 4.2 we discussed the characteristics of the available timers in the STM32F072. For the remainder of this lab manual we will be focusing on the documentation materials for TIM2 & TIM3. Timers 2 & 3 are the more-advanced versions of the general purpose timers in the STM32F0. They have a nearly identical interface as the simpler timers, except that they feature additional configuration options and more output channels. By reading through the documentation of these timers, you should be able to move to one of the simpler devices without trouble in the assignment

Organization of Peripheral Documentation

The each section of the peripheral reference manual follows a similar organization when documenting a peripheral.

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 and to a specific base unit of time, the peripheral timers are available to use for arbitrary timing periods and can be used as event counters when using alternate signals as clock sources. Because their input prescaler can be used to divide the input clock by arbitrary integer values, it is 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.
There are three registers that form the basic 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. Athough this register is automatically updated as the timer operates, it is possible to manually read and edit its value. The CNT register can be read within the main application loop as a method of counting time, or written in order to modify the timer's operation.

Timer Prescaler Register (PSC)

The PSC register is used to divide the input clock frequency to the timer. It's 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 will fit 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 an artificial 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 will be discussed more in a later section.

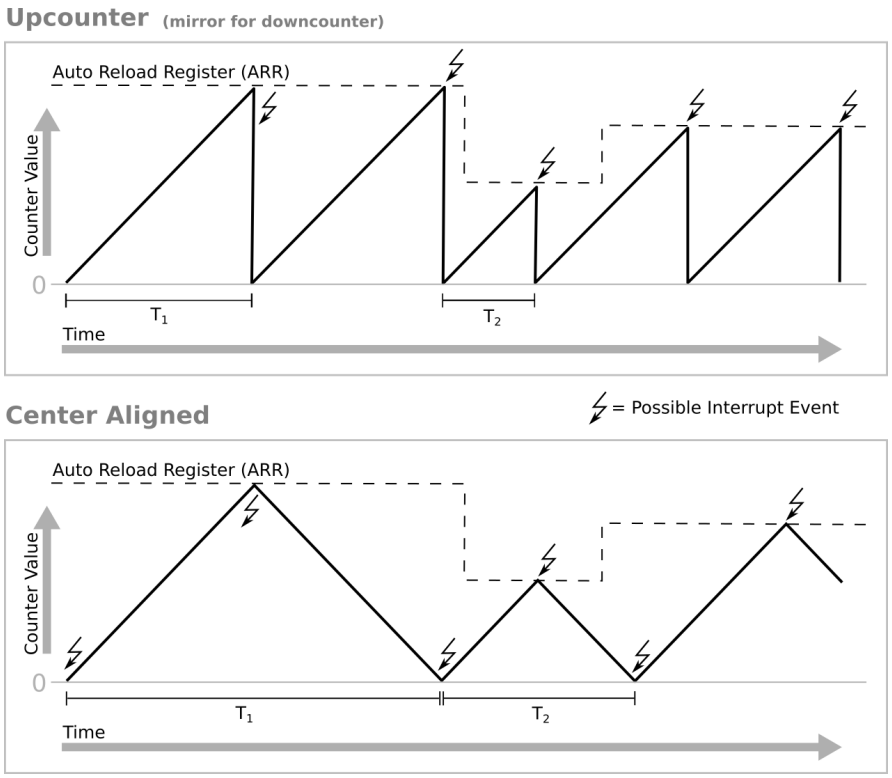

Figure 4.3: Timer counting modes and update interrupts

4.3.1 Basic Timer Operation

There are three different counting modes that the timers can operate under: up, down, and center-aligned (up/down). Note that all but three of the hardware timers in the STM32F072 are only upcounters. Figure 4.3 shows a graphical representation of the 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 the lower limit 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 interrupt source is used to generate periodic interrupts, but with a very flexible and potentially long duration period.

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 crucial 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 ablility 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 out toolchain, this frequency is configured 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 count events generated 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 of time 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 convenient 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 on the input clock.

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 a fine resolution. However in many cases it is possible to use small prescaler values in order to have finer granularity when counting. The following four equations may be helpful when selecting prescalar 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'll 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 1kHz gives us 1ms 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. ■

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 a two additional registers with important functions. In figure 4.2 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 hold capture/compare data. Its specific use depends on the selected mode.
- **Capture/Compare Mode Registers (CCMRx)** – Contains additional options to select between specific capture/compare modes and their operation.

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. Athough these modes are primarily intended to interface with their associated GPIO pins, they all can generate interrupts on their respective trigger conditions.

4.4.1 Input Capture Mode

When configured into input capture mode, a capture/compare channel will latch 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.

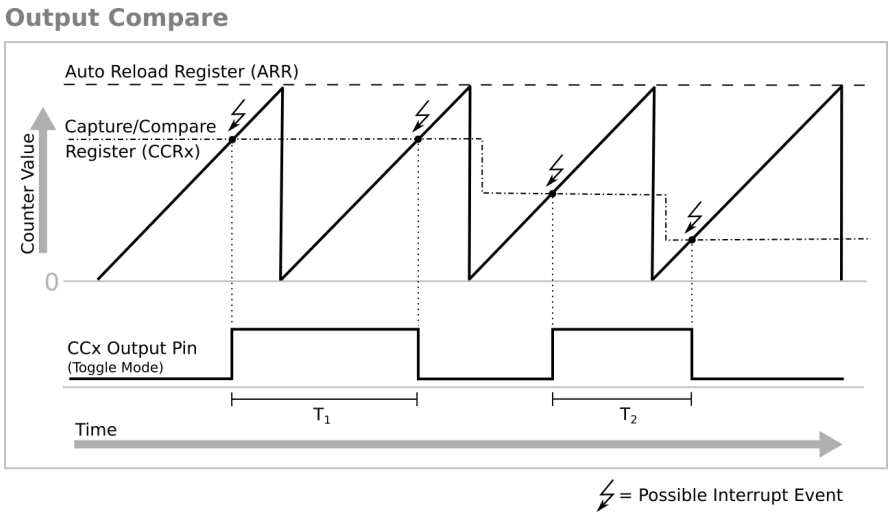

Figure 4.4: Output Compare mode with pin toggle on compare match.

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. Although the assignments in this section will not require you to use the input capture mode of a timer, we will be using an advanced mode of the input capture system to interface with the quadrature encoder of a DC motor during a later lab.

4.4.2 Output Compare Mode

The output compare mode directly modifies the output of a GPIO pin whenever the timers 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.
By 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 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 really a slightly more advanced form of the output compare mode. However, it's helpful to first discuss pulse-width modulation (PWM) and it's uses.

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.

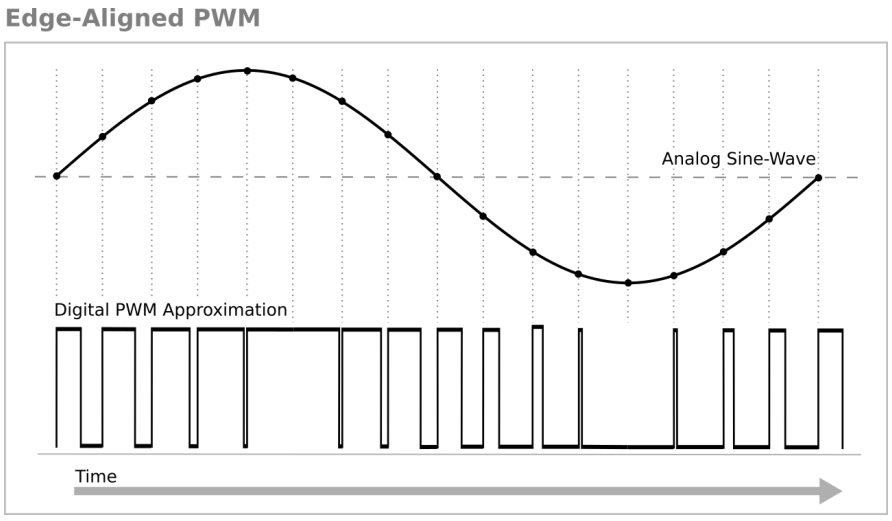

Figure 4.5: PWM approximation of an analog sine-wave.

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 will remain low for the entire duration and one with a 100% duty cycle will remain 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 doesn't make sense at low frequencies, it would just appear as a series of different length flashes. However at high frequencies, many physical systems can't respond quick enough to shut-off or turn-on with each output transition. This is the basis of mechanical and electrical low-pass filters which average or smooth out the high-frequency transitions of the PWM signal into an approximated analog signal.
PWM is used for a variety of purposes. A few common examples are audio, light dimming, and motor speed control. Figure 4.5 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 to the output compare mode of the timer. The difference is that the output pin state is also modified 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 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 once 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.
The important ting to note about PWM is that the signal has both a duty-cycle and a 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 low-pass filter smooths out the transitions.

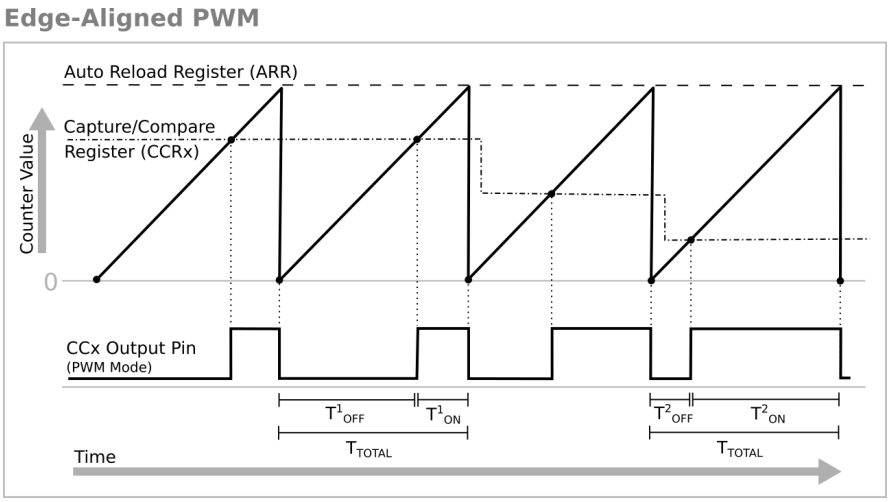

Figure 4.6: Edge-aligned PWM mode and output pin state.

4.5 Configuring GPIO pins to Alternate Function Mode

4.5.1 Finding Available Pins on a Device Package

- **Example 4.2 — Finding Pins With an Alternate Function.** ■

4.5.2 Selecting an Alternate Function

- **Example 4.3 — Determining an Alternate Function's Number.** ■

4.6 Lab Assignment:

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

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