

VORAGO Timer (TIM) subsystem application note

Feb 24, 2017, Version 1.2

VA10800/VA10820

Abstract

This application note reviews the Timer (TIM) subsystem on the VA108xx family of MCUs and provides several common use case examples that can be used as a starting point for end applications. The examples progress in complexity and include:

- A. PWM outputs
- B. Periodic interrupt
- C. Input pulse on a port pin measurement
- D. Six channel PWM output suitable for motor control

Table of contents

1.	TIM subsystem overview	2
2.	Register overview.....	4
3.	Examples.....	6
4.	Summary	25
5.	Common questions and issues	25

1. TIM subsystem overview

The TIM subsystem consists of 24 individually programmable blocks. Each unit is a highly configurable circuit that can be used alone or coordinated with other TIM blocks for a variety of functions including:

- Measuring the pulse width or a counting pulses on a port pin
- Generating periodic interrupts to help with scheduling software tasks
- Generating coordinated PWM outputs that can ensure dead time insertion
- Watchdog timer that needs to be updated within a certain period or the device is reset.

The heart of each timer block is a 32-bit down counter. The counter can be used to generate events or measure intervals between events. An event can be activity on a port pin or an interrupt request to or from any TIM block. Thirty-two-bit counter rollover range for various MCU clock rates is shown in Table 1.

Table 1 - Timer roll over interval table for three clock frequencies

Clock Frequency	Roll over interval	Single count interval
50 MHz	85.89 seconds	20 nanosecond
25 MHz	171.8 seconds	40 nanosecond
10 MHz	429.5 seconds	100 nanosecond

There is no dedicated prescaler to divide the clock down to a slower speed but it is possible to use another timer or one of the 7 IOCONFIG clock dividers to act as a prescaler using the cascade function. For instance, if the desired clock rate for TIM03 was 10 msec, an IOCONFIG clock divider could be set to $(10 \text{ msec} / 20 \text{ nsec}) = 0.5 \text{ E6}$. TIM03 would be configured to become active when the clock divider timed out. Every time the 10 msec interval ended, TIM03 would decrement 1 tick.

Timer channels can be linked together to cause a sequence of events. Alternatively, events on port pins can trigger timer events. The mechanism linking events to an individual timer is referred to as cascading. Each timer can have up to 107 input sources to that block's cascade mechanism. See the Cascade Selection Codes table in the programmer's guide for the full list. In order for any

source's output to be cascaded to another block, it must have its interrupts enabled. Note that having a TIM's or GPIO's interrupt enabled does not immediately cause the CPU to be interrupted. In order for the CPU to get an interrupt, the IRQ_SEL peripheral must be configured appropriately and the NVIC must be enabled. In many cases, it is not advised to have the CPU interrupted for every event on a timer.

Cascade inputs 0 and 1 can be used to start an action. An example of this would be triggering an ADC conversion to measure in-rush current exactly 15.7 microseconds after a PWM output gates a power transistor in a motor drive circuit.

Cascade 2 can be used to halt an activity. An example of this would be to perform an emergency shut-down of a motor drive circuit if an over-current event is ever detected. Example 3A of this application note uses cascade 2 to stop a timer when a rising edge on a push button switch is detected.

There are 24 timer blocks on the VA108xx family. Each block is identical and can be configured to 1 of 7 output functions as outlined in Table 2. All of the 24 TIM blocks on the VA108xx are mapped to port pins. All except TIM7, 8 & 9 can be mapped to two port pins as defined in the function select field of the IOCONFIG register. TIM7, 8 and 9 only have one pin assignment option. Each timer also has the ability to generate an interrupt request when the CNT_VALUE = 0. Which input signals are used is set by the cascade control register (CSD_CTRL) and the registers CASCADE0, 1 & 2.

Table 2 - List of output functions (Status_Select) that a TIM block can be configured for

0	<u>A one cycle pulse when the counter transitions to 0.</u> This is normally used for generating interrupts and can be useful for software time keeping and interval management.
1	<u>Output the ACTIVE status bit.</u> This can be useful for debugging software and timer configuration.
2	<u>Toggle between 1/0 every time the counter reaches 0.</u> This can be used for generating 50% duty cycle signals on pins.
3	<u>PWMA output value.</u> 1 (active) when Counter Value \geq PWMA Value, 0 (in-active) when Counter Value $<$ PWMA Value. This is explained in more detail in example 1A of this application note.
4	<u>PWMB output value.</u> (Uses both A and B) 1 when Counter Value $<$ PWMA Value and \geq PWMB, 0 when Counter Value \geq PWMA Value or $<$ PWMB. This is explained in more detail in example 1B of this application note.
5	<u>Output the ENABLED status bit.</u> This can be useful for debugging software and timer configuration.
6	<u>PWMA active mode.</u> 1 when CounterValue \leq PWMA Value and >0 , 0 otherwise

2. Register overview

To provide a great amount of flexibility on inputs and outputs of the timer, the VORAGO TIM provides many options. Details on each bit can be found in the Programmer's Guide. Table 3 - TIM register table can be used as a quick reference to all the registers and bit definitions. This may be useful during software debug and when designing the firmware.

		BIT LOCATION INSIDE THE 32-BIT REGISTER							
	Register bit location	31/23/15/7	30/22/14/6	29/21/13/5	28/20/12/4	27/19/11/3	26/18/10/2	25/17/9/1	24/16/8/0
CTRL	31-24	-	-	-	-	-	-	-	-
	23-16	-	-	-	-	-	-	-	-
	15-8	-	-	-	-	-	-	REQ_STOP	STATUS_INV
	7-0	STATUS_SEL			IRQ_ENAB	AUTO_DEACTIVE	AUTO_DISABLE	ACTIVE	ENABLE
RST_VALUE	31-0	VALUE (32-bits)							
CNT_VALUE	31-0	VALUE (32-bits)							
ENABLE	31-24	-	-	-	-	-	-	-	-
	23-16	-	-	-	-	-	-	-	-
	15-8	-	-	-	-	-	-	-	-
	7-0	-	-	-	-	-	-	-	ENABLE
CSD_CTRL	31-24								
	23-16								
	15-8						CASTRG2	CASINV2	CASEN2
	7-0	CASTRG1	CASTRG0	-	DCASOP	CASINV1	CASEN1	CASINV0	CASEN0
CASCADE0	31-24	-	-	-	-	-	-	-	-
	23-16	-	-	-	-	-	-	-	-
	15-8	-	-	-	-	-	-	-	-
	7-0	-	CASSEL						
CASCADE1	31-24	-	-	-	-	-	-	-	-
	23-16	-	-	-	-	-	-	-	-
	15-8	-	-	-	-	-	-	-	-
	7-0	-	CASSEL						
CASCADE2	31-24	-	-	-	-	-	-	-	-
	23-16	-	-	-	-	-	-	-	-
	15-8	-	-	-	-	-	-	-	-
	7-0	-	CASSEL						
PWMA_VALUE	31-0	VALUE (32-bits)							
PWMB_VALUE	31-0	VALUE (32-bits)							
PERID	31-0	0x0111 07E1							

Table 3 - TIM register table

3. Examples

Example 1: PWM output example

Edge aligned PWM outputs are commonly used for driving circuits that do not require coordination with other channels such as LED indicators or heater control. They are simple to setup and control, also they are easy to implement in silicon. Example 1A shows how to quickly setup a timer to output a PWM on a port pin. For motor control circuits where alignment of high and low side drivers is imperative, center aligned PWMs are required. Center aligned PWMs can also help lower the instantaneous peak current of an application by offsetting when power transistors turn on and off. Center aligned PWMs are explained and demonstrated in example 1B.

Example 1A – PWMA mode (Edge aligned PWM)

Only the duty cycle and period are specified in this mode. As shown in Figure 1, the PWMA (edge aligned) mode will have an active output (=1) when the timer count is greater than the PWMA value. The timer count register (CNT_VALUE) will reset to RST_VALUE when the count reaches zero.

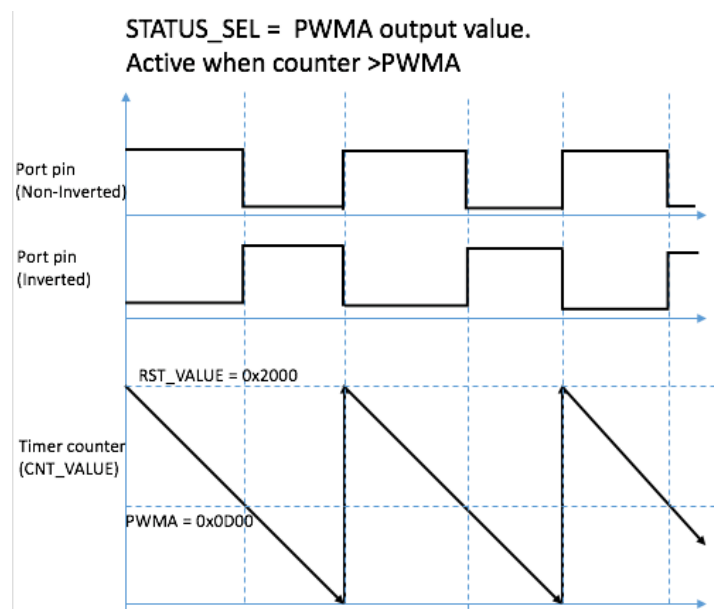


Figure 1 - Pictorial representation of PWMA mode. (aka edge aligned PWM)

Example 1A will have one PWM with a high frequency (20kHz) and 80% duty cycle and one with a relatively low frequency (1 Hz) and a 50% duty cycle. The MCU clock frequency is 50 MHz.

Before setting up registers, a little math is required to determine the overflow and PWMA.

Channel 1: 20 kHz with 10% duty cycle

$$RST_VALUE = 50 \text{ MHz} / 20 \text{ kHz} = 5000 \text{ (0x001388)}$$

$$PWMA = 90\% \text{ of } RST_VALUE = 4500 \text{ (0x001194)}$$

Steps to setup the module

1. In the SYSCONFIG peripheral
 - a. Enable peripheral clocks
 - b. Enable timer clocks and
2. In the IOCONFIG peripheral set pin function selection.
3. In the GPIO peripheral configure the port pins as an output
4. Set RST_VALUE and PWMA_VALUE.
5. Enable the TIM blocks and select the Timer mode (STATUS_SEL)

See Figure 2 for code that performs the above steps.

```

54 //*****
55 // Routine to configure TIM1 as PWMA mode
56 // - Cascade inputs not used
57 // - RESET_VALUE set to "PWM_period*2" = 5000
58 // - PWM edge aligned with PWMA = 90% of RESET VALUE
59 //*****
60 uint32_t CONFIG_TIM1_PWMA(void)
61 {
62     VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE = 0xFFFFFFFF ; // enable all peripheral clocks
63     VOR_SYSCONFIG->TIM_CLK_ENABLE |= 0xFFFF ; // enable clocks to lower 16 TIM blocks
64
65     VOR_IOCONFIG->PORTA[1]= IOCONFIG_PORTA_PLEVEL_Msk | IOCONFIG_PORTA_PEN_Msk |(FUNCSEL1 << IOCONFIG_PORTA_FUNSEL_Pos); //
66
67     VOR_GPIO->BANK[0].DIR |= (1 << 1); // set PA[1] to an output
68
69     VOR_TIM1->CTRL = (TIM1_CTRL_IRQ_ENB_Msk | ( STAT_SEL_PWMA << TIM1_CTRL_STATUS_SEL_Pos)) ; //
70     VOR_TIM1->RST_VALUE = PWM_period*2 ;
71     VOR_TIM1->CNT_VALUE = 0 ;
72
73     VOR_TIM1->CSD_CTRL |= ( TIM1_CSD_CTRL_CSDEN0_Msk ) ; //
74     VOR_TIM1->CSD_CTRL |= ( TIM1_CSD_CTRL_CSDEN0_Msk | TIM1_CSD_CTRL_CSDINV0_Msk ) ; // no cascade for TIM1
75     VOR_TIM1->CASCADE0 = (TIM_CAS_SRC_PORTA_2) ; // set cascade 0 source to PORTA2
76     VOR_TIM1->CASCADE1 = 0x00 ;
77     VOR_TIM1->CASCADE2 = 0x00 ;
78     VOR_TIM1->PWMA_VALUE = .9* (PWM_period*2) ;
79     VOR_TIM1->PWMB_VALUE = 0x88 ;
80     VOR_TIM1-> CTRL = 0x88 ;
81
82     VOR_TIM1->ENABLE = 0x1 ; // enable TIM1
83 }
84 //*****

```

Figure 2 - Code snippet to show TIM1 being configured for PWMA mode

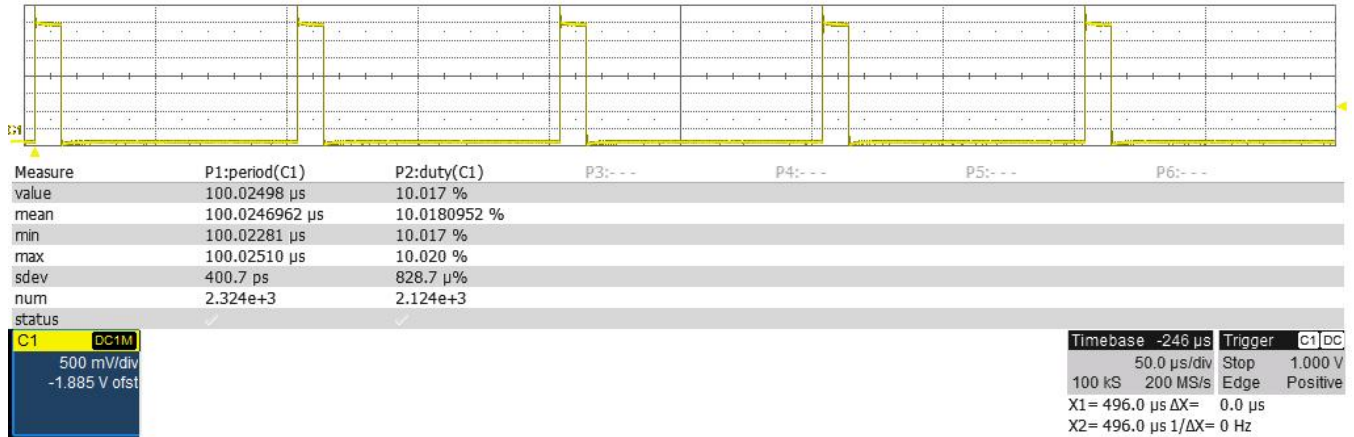


Figure 3 - Scope capture showing the waveform generated on TIM1

Example 1B – PWMB mode (Center aligned PWM)

In PWMB mode, the duty cycle along with both the rising and falling edge times are specified which allows center aligned PWM outputs. As shown in Figure 4, the PWMB (center aligned) mode will have an active output (=1) when the timer count is less than the PWMA value and greater than PWMB. The timer count register (CNT_VALUE) will reset to RST_VALUE when the count reaches zero.

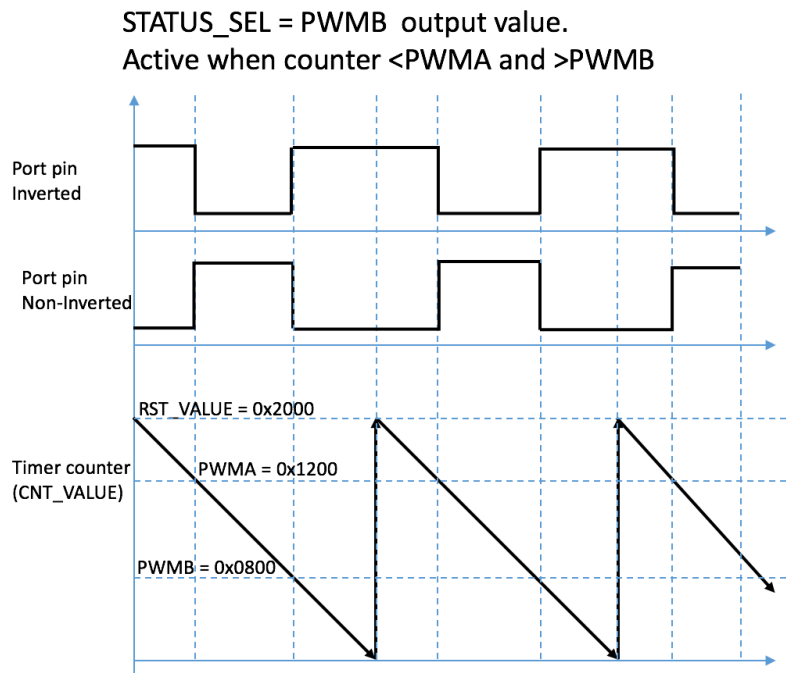


Figure 4 - PWMB operation in pictorial format

Example 1B has two center-aligned PWM channels with a high frequency (~20kHz). One has an 80% duty cycle and one has a 20% duty cycle. A port pin input (switch button on the REB1 board) will be used to trigger the start of both TIM blocks.

Steps to setup the modules

1. Enable peripheral clocks and set pin function selection.
2. Configure the port pins as outputs and the button as an input with interrupt enabled
3. Set Cascade 0 source as button input (PA11)
4. Set RST_VALUE, PWMA & PWMB.
5. Enable the TIM blocks and select the Timer mode (STATUS_SEL= PWMB)
6. Enable interrupts on the button input

```

119 //*****
120 // Routine to configure TIM3 as PWMB mode
121 // - Cascade 0 selects PA11 (push button)
122 // - RESET_VALUE set to "PWM_period" = 2500
123 // - PWM centered with PWMA = 40% and PWMB = 60% of FWM_period
124 //
125 //*****
126 #define TIM_CAS_SRC_PORTA_11 11 ///
127 uint32_t CONFIG_TIM3_PWMB(void)
128 {
129     VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE = 0xFFFFFFFF ; // enable all peripheral clocks
130     VOR_SYSCONFIG->TIM_CLK_ENABLE |= 0xFFFF ; // enable clocks to lower 16 TIM blocks
131
132     VOR_IOCONFIG->PORTA[3]= IOCONFIG_PORTA_PLEVEL_Msk | IOCONFIG_PORTA_PEN_Msk |(FUNCSEL1 << IOCONFIG_PORTA_FUNSEL_Pos); //
133
134     VOR_GPIO->BANK[0].DIR |= (3 << 1); // set PA[3] to an output |
135
136     VOR_TIM3->CTRL = (TIM1_CTRL_IRQ_ENB_Msk | ( STAT_SEL_PWMB << TIM1_CTRL_STATUS_SEL_Pos)) ; //
137     VOR_TIM3->RST_VALUE = FWM_period ;
138     VOR_TIM3->CNT_VALUE = FWM_period ;
139
140     VOR_TIM3->CSD_CTRL |= ( TIM1_CSD_CTRL_CSDEN0_Msk | TIM1_CSD_CTRL_CSDTRG0_Msk ) ;
141     // Setup cascade control so timer triggers when PA11(button) is pressed
142     VOR_TIM3->CASCADE0 = (TIM_CAS_SRC_PORTA_11) ; // set cascade 0 source to PORTA11
143     VOR_TIM3->CASCADE1 = 0x00 ; // not used
144     VOR_TIM3->CASCADE2 = 0x00 ; // not used
145     VOR_TIM3->PWMA_VALUE = FWM_period * .6 ;
146     VOR_TIM3->PWMB_VALUE = FWM_period * .4 ;
147     VOR_TIM3->ENABLE = 0x1 ; // enable TIM3
148
149     return(FWM_period) ;
150 }
151 //*****

```

Figure 5 - Code snippet showing TIM3 being setup for PWMB with cascade 0 trigger set to PA11

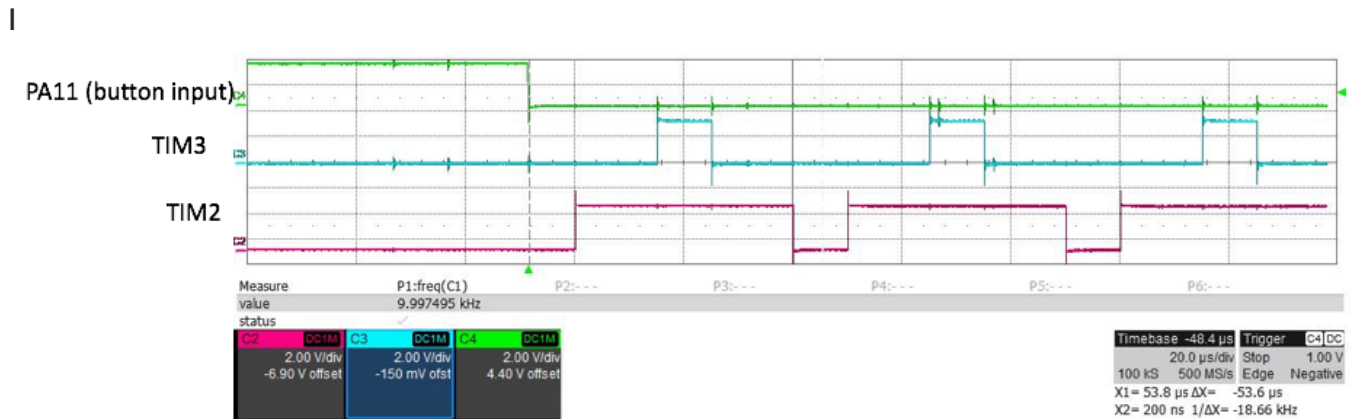


Figure 6 - Scope trace showing PWMB mode generating center-aligned PWM

There is a separate project available with the software download of this AN. If the Keil IDE is already installed, just double click on "AN1201_simple_PWM.uvprojxx" in the project folder. A project should open after a few moments. Perform a "Build All" operation by either clicking on the build all icon or using the "Project" pull down menu. After the project is built and a REB1 board

connected to the host PC, enter debug mode by clicking the icon or using the “Debug” pull down menu. This will move the code to the instruction RAM of the MCU. Run the program by either clicking on the run icon or using the Debug pull down menu.

Caution: Setting breakpoints is an excellent way to monitor the progress of instructions in a program. It needs to be completely understood that the peripheral subsystem is not impacted by a breakpoint. For example, the timers will all continue to run if the CPU is halted at a breakpoint. In most cases this is not an issue but when dealing with interrupts, halting the CPU while the peripherals continue to run can cause unpredictable results.

Example 2: Periodic interrupt

One very useful function of timers is creating a periodic interrupt that software can use for scheduling tasks and keeping track of time. Example 2 will generate an interrupt every 10 msec using TIM23. The VA108xx has an IRQSEL peripheral that routes the >100 interrupt sources on the device to the 32 inputs of the NVIC controller. A small interrupt subroutine will be used to count to 100 and toggle a port pin controlling one of the User LEDs on the REB1 board.

In order for the compiler to comprehend NVIC interrupt 30 being associated with a particular ISR, the VA108xx_startup.c file must be modified. Changing all instances of OC30_IRQHandler to OC30_IRQHandlerX will vector the CPU to the proper ISR when an interrupt on IRQ30 occurs. This has already been done in the example file.

Steps to setup the module

1. Enable peripheral and timer clocks in SYSCONFIG->PERIPHERAL_CLK_ENABLE and SYSCONFIG->TIM_CLK_ENABLE respectively
2. Configure the TIM23
 - a. Set STATUS_SEL to 0 which will generate a 1 cycle pulse every time the count reaches zero.
 - b. Enable the TIMER interrupt
 - c. Set RESET_CNT to $(10 \text{ msec} / 20 \text{ nsec}) = 500e+3$
 - d. Enable TIM23
3. Assign the TIM23 interrupt to NVIC input 30 in the IRQSEL block.
4. Enable interrupts on the NVIC

See Figure 7 for working code that implements the above steps.

```

168 uint32_t CONFIG_TIM23_periodic_int(void)
169 {
170     VOR_TIM23->CTRL = (TIM23_CTRL_IRQ_ENB_Msk | ( STAT_SEL_1CYC << TIM1_CTRL_STATUS_SEL_Pos)) ; //
171     VOR_TIM23->RST_VALUE = (10e-3 * 50e6) ; // setup RST value for 10 msec
172     VOR_TIM23->CNT_VALUE = (10e-3 * 50e6) ;
173
174     //     VOR_TIM23->CSD_CTRL |= NA ;
175     //     // Setup cascade control so timer triggers when PA11(button) is pressed
176     //     VOR_TIM23->CASCADE0 = NA ;
177     //     VOR_TIM23->CASCADE1 = NA ;
178     //     VOR_TIM23->CASCADE2 = NA ;
179     //     VOR_TIM23->PWMA_VALUE = NA ;
180     //     VOR_TIM23->PWMB_VALUE = NA ;
181     VOR_TIM23->ENABLE = 0x1 ; // enable TIM23
182
183     VOR_IRQSEL->TIM[23] = IRQ_TIM23_IRQn; // IRQSEL redirects TIM23 to NVIC IRQ input 30
184
185     NVIC_SetPriority(IRQ_TIM23_IRQn,IRQ_TIM23_IRQ_PRIORITY);
186     NVIC_EnableIRQ(IRQ_TIM23_IRQn);
187
188     SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
189                   SysTick_CTRL_TICKINT_Msk |
190                   SysTick_CTRL_ENABLE_Msk;
191
192     VOR_IOCONFIG->PORTA[5] |= IOCONFIG_PORTA_IOWO_Msk ;
193     VOR_IOCONFIG->PORTA[6] |= IOCONFIG_PORTA_IOWO_Msk ;
194     VOR_GPIO->BANK[0].DIR |= (1 << 6); // set PA[6] to an output. this is LED4 on REB1 board, toggle every 1 sec
195     VOR_GPIO->BANK[0].DIR |= (1 << 5); // set PA[5] to an output. this is available I/O on REB1, toggle every 1 sec
196     VOR_GPIO->BANK[0].DIR |= (1 << 4); // set PA[4] to an output. This will Toggle every 10 msec
197 }

```

Figure 7 - Code snippet to configure timer for periodic interrupt

```

203 //*****
204 // Interrupt Subroutine for TIM23 (NVIC IRQ30)
205 //*****
206 void OC30_IRQHandlerX(void)
207 {
208     uint32_t elapsed_time ;
209
210     VOR_GPIO->BANK[0].TOGOUT = (1 << 4); // toggle PA4 every 10 msec
211
212     if(cnt_10ms++ >= 99)
213     {
214         cnt_sec ++ ;
215         VOR_GPIO->BANK[0].DATAOUT ^= ((1 << 5)|(1<<6)) ; // toggle PA5 and PA6
216         cnt_10ms = 0 ;
217     }
218
219     new_SYSTICK = SysTick->VAL ; // Use Cortex-M0 systick to check TIM accuracy
220     elapsed_time = ((old_SYSTICK - new_SYSTICK) & 0xFFFFF) ; // systick only has 24 bits, need to ignore top 8 bits
221     old_SYSTICK = new_SYSTICK ;
222
223     if(i>16) i = 0 ; // keeping rolling array of elapsed time intervals
224     elapsed_time_array[i++] = elapsed_time ; // store elapse tick count for the last 16 interrupts
225 }
226

```

Figure 8 - Example code snippet of timer ISR for periodic interrupt

A separate project "AN1202_periodic_int.uvprojx" accompanies this application note in the SW attachment. Open, compile, download and run the project similarly to what was done in example 1. When the program is running, LED3 on the REB1 board should blink with 1 second on and 1 second off. The ISR includes a check against the SYS_TICK counter of the Cortex-M0 which is a 24-bit counter that decrements once every system clock cycle.

Note: The VA108xx has several circuits that can be used to generate periodic interrupts: a) any of the 24 timers, b) the M0's 24-bit SYS_TICK counter and c) 7 of the pin filter timers. Using one of the pin filter timers for this simple purpose is recommended over a timer channel which can perform much more complex actions.

Example 3: Input signal pulse width measurement and pulse count

Many applications require the accurate measurement of activity on a port pin. Using a timer to do this is an easy way to provide a very accurate measurement and keep the CPU load for this computation to a minimum. It is also possible to put the processor in a WAIT mode such that the CPU is idle while the timer continues to monitor a port pin status. The CPU can exit WAIT mode by any TIM or GPIO interrupt. Each port pin can be set to trigger an interrupt on rising, falling or both edges. In turn, any timer can be configured to react to an interrupt on any port pin. When the trigger event occurs, the timer can be started or stopped. Alternatively, the timer can be enabled when a port pin is high or low. The edge or level configuration is set for the port pin using the GPIO control registers.

Example 3A: Input pulse width measurement

Example 3A will configure a timer to calculate the time a port pin is pulled low. This can be used for de-bouncing a switch or accurately measuring a pulse width. The example will route the push button switch input to TIM1 with the cascade feature. An interrupt will be generated on both the rising and falling edge. On the rising edge, the timer will start automatically and the interrupt will be used to setup the cascade 2 input so the timer is stopped when the pin rises.

Some care must be taken to avoid long pulses (>85 seconds) that cause overflow errors when the counter rolls over from 0 to the RST_VALUE. In that case, software must keep track of the number of time-outs that occur (CNT_VALUE = 0) and add the appropriate number of RST_VALUE counts to the pulse width.

Steps to setup the module:

1. In the SYSCONFIG peripheral:
 - a. Enable peripheral clocks
 - b. Enable timer clock
2. Setup PA[11] to be input pin with filtering and generate IRQ on both edges
 - a. In the SYSCONFIG setup Filter1 for a divide of 0xFF
 - b. In the IOCONFIG peripheral configure PA[11] as an input with a pull-up enabled and filter 1 enabled
 - c. In the GPIO peripheral set PA[11] to have interrupts on both rising and falling edges
 - d. In the IRQ_SEL peripheral assign PA[11] to IRQ30
3. Configure timer 1 to output active status and to be triggered when PA[11] interrupt fires
 - a. Setup PA[1] as an output
 - b. Set Function select for PA[1] as TIM1
 - c. Setup TIM1
 - i. STATUS_SEL = output active status
 - ii. RST_VALUE = 0xFFFFFFFF, this allows up to an 85 second pulse
 - iii. Set both cascade 0 and cascade 2 source to be PA[11]
 - iv. Set Cascade control to be enabled when Cascade0 is active
4. Enable TIM1
5. Enable the NVIC IRQ30 and set Priority to 0. (Zero is highest priority)
6. Enable interrupts on PA[11]

See Figure 9 for working code that implements the above steps.

```

52 //*****
53 // * Routine to setup TIM1 to count whenever PA[11] is low (Pa11 is the push button on the REB1 board
54 // * - A pin interrupt will be generated for falling and rising edges
55 // * - PA[11] will have the input filter enabled and uses CLKDIV1
56 // * - TIM1 will be setup to start counting on a falling edge int of PA[11]
57 // * - inside the PA11 ISR, the TIM1 cascade 2 will be setup to shut the TIM down when a falling edge int on PA11 occurs
58 //*****
59 uint32_t CONFIG_TIM1(void)
60 {
61     uint32_t temp, t2 ;
62
63     VOR_SYSCONFIG->TIM_CLK_ENABLE |= 0xFFFF ; // enable clocks to lower 16 TIM blocks
64
65     // *** setup condition for cascade 0 input - PA11 edge(s)
66     VOR_SYSCONFIG->IOCONFIG_CLKDIV1 = 0xff ; // set clkdiv0 to div256
67
68     VOR_IOCONFIG->PORTA[11]= ((5<<IOCONFIG_PORTA_FLTTYPE_Pos) | (1<<IOCONFIG_PORTA_FLTCLK_Pos)
69     | IOCONFIG_PORTA_PLEVEL_Msk | IOCONFIG_PORTA_PEN_Msk );
70
71     VOR_GPIO->BANK[0].DIR &= ~(1 << 11); // set PA[11] to an input
72     VOR_GPIO->BANK[0].IRQ_SEN &= ~(1 << 11); // set SEN for PA11 to 0 to stat gen from signal transition
73     VOR_GPIO->BANK[0].IRQ_EDGE |= 1 << 11 ; // set EDGE for PA11 to 1 to gen stat for either edge
74
75     VOR_IRQSEL->PORTA[11] = IRQ_PA11_IRQn; // IRQSEL redirects PA11 int to NVIC IRQ input 31
76
77     VOR_GPIO->BANK[0].DIR |= (1 << 1); // set PA[1] to an output
78     VOR_IOCONFIG->PORTA[1]= (FUNCSEL1 << IOCONFIG_PORTA_FUNSEL_Pos); // set pin func to TIM1
79
80     for(temp=0; temp < 1000 ; temp ++){
81         ; // Wait for PA[11] pin and filter to settle
82     }
83
84     VOR_TIM1->CTRL = ( TIM_STATUS_SEL_1<< TIM1_CTRL_STATUS_SEL_Pos) ; // Output when active for debug purposes only
85     VOR_TIM1->RST_VALUE = T1_rollover ;
86     VOR_TIM1->CNT_VALUE = T1_rollover ;
87
88     VOR_TIM1->CSD_CTRL |= ( TIM1_CSD_CTRL_CSDEN0_Msk ) ;
89     // Setup cascade control so timer counts PORTA[11] transitions low
90     VOR_TIM1->CASCADE0 = (TIM_CAS_SRC_PORTA_11) ; // set cascade 0 source to PORTA[11]
91     VOR_TIM1->CASCADE1 = 0x00 ;
92     VOR_TIM1->CASCADE2 = (TIM_CAS_SRC_PORTA_11) ; // set cascade 0 source to PORTA[11]
93
94     VOR_TIM1->ENABLE = 0x1 ; // enable TIM1
95
96     NVIC_SetPriority(IRQ_PA11_IRQn,IRQ_PA11_IRQ_PRIORITY);
97     NVIC_EnableIRQ(IRQ_PA11_IRQn);
98
99     VOR_GPIO->BANK[0].IRQ_ENB |= 1 << 11 ; // set PA[11] Enable bit
100
101     return(temp) ;
102 }

```

Figure 9 - Subroutine to setup TIM1 for input capture of a pulse on PA[11]

The IRQ will be generated on each rising and falling edge of PA11. In the ISR, software must determine if a falling or rising edge occurred. If it was a falling edge, this is the beginning of the measurement and the timer already started decrementing. All that is needed to do is to setup

cascade control so cascade 2 is active and will stop the clock at the next event. If it was a rising edge, the pulse width needs to be calculated by subtracting the CNT_VALUE from the RST_VALUE and the cascade control must be setup to use cascade 0 to trigger an event when the next falling edge occurs. See Figure 10 for working ISR code.

```

204 //*****
205 // Interrupt Subroutine for PA11 (NVIC IRQ31)
206 // - increment button_press count
207 // - if (falling edge cause int), then
208 //     a) setup cascade control 2 to shutdown on rising edge
209 // - else ( if rising edge caused int), then
210 //     a) setup cascade control 0 to start
211 //     b) calculate pulse width and
212 //*****
213 void OC31_IRQHandlerX(void)
214 {
215     uint32_t elapsed_time ;
216
217     cnt_button_presses ++ ;
218     new_SYSTICK = SysTick->VAL ; //
219     elapsed_time = old_SYSTICK - new_SYSTICK ;
220     old_SYSTICK = new_SYSTICK ;
221
222     if(VOR_TIM1->CSD_CTRL & TIM1_CSD_CTRL_CSDEN0_Msk ) // if yes, this was falling edge
223     {
224         VOR_TIM1->CSD_CTRL = TIM1_CSD_CTRL_CSDEN2_Msk ; // setup PA11 next edge to stop TIM1
225         if((VOR_GPIO->BANK[0].DATAIN & (1<<11)) )
226             { VOR_TIM1->CSD_CTRL = TIM1_CSD_CTRL_CSDEN0_Msk; } // precautionary measure if ever rise/fall edges get out of sync
227     }
228     else
229     {
230         VOR_TIM1->CSD_CTRL = TIM1_CSD_CTRL_CSDEN0_Msk ; // setup PA11 next edge to start TIM1
231         VOR_TIM1->ENABLE = 0x1 ; // enable TIM1
232         input_pulse[k] = (VOR_TIM1->RST_VALUE - VOR_TIM1->CNT_VALUE) ; // calc pulse width and place in array
233         k++ ;
234         VOR_TIM1->CNT_VALUE = VOR_TIM1->RST_VALUE ;
235         if(k>16) { k = 0 ; }
236     }
237 }

```

Figure 10 - Interrupt subroutine to assist with input capture. This ISR is called for every edge detected on PA[11]

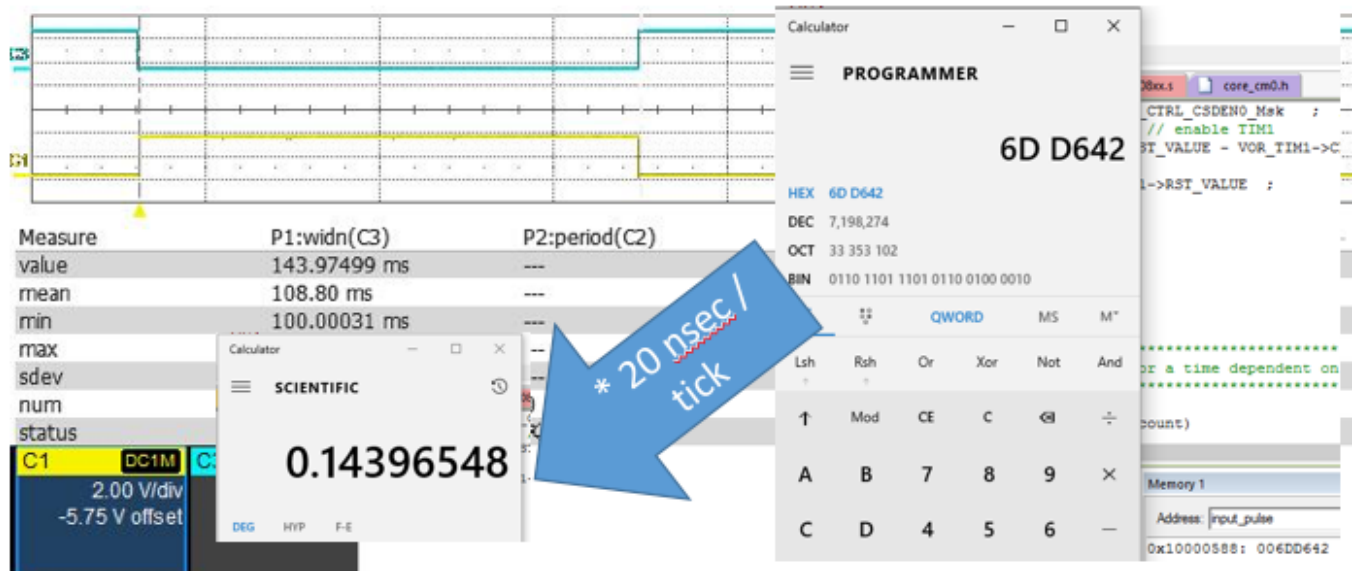


Figure 11 - Scope capture of input capture example along with measured "input_pulse" variable. Scope measured pulse to be 143.97499 msec. The MCU measured the pulse to be 143.96548 msec

Example 3B – Pulse counter

Some applications use encoder wheels that provide a pulse train with a speed related to the angular velocity of the wheel. In some cases, it is better to count the pulses during a certain interval to indicate the speed. This can help average the speed and allow for different rise and fall times on the sensor output. Example 3B will use two channels to accurately count the number of pulses in a 0.1 second window. The input filter capability of the GPIO will be used to eliminate noise on the input or transitions with multiple edges.

Steps to setup the module

1. Enable peripheral clocks in SYSCONFIG->PERIPHERAL_CLK_ENABLE
2. Enable timer clocks in SYSCONFIG->TIM_CLK_ENABLES
3. Setup 1 timer (TIM23) to create a periodic interrupt every 100msec.
 - a. Enable interrupt
 - b. Set RST_COUNT to $100\text{msec}/20\text{nsec} = 500\text{E}+6$
4. Setup second timer (TIM3) to be active when port pin PA[8] sees a falling edge.
5. Setup IOCONFIG peripheral to:

- a. set pin function selection,
 - b. configure the port pin(s) as inputs with a pull-up enabled,
 - c. select input filter source (FLT_TYPE and FLT_NUM)
 - d. enable interrupt on falling edge of port pin
6. Setup filter count interval in SYSCONFIG->IOCONFIG_CLKDIV1 register
 7. In GPIO block, setup PA[8] to generate an interrupt on each falling edge
 8. Enable the TIMER blocks

See Figure 12 for working code that implements the above steps.

```

96 //*****
97 // * Routine to setup TIM3 to count rising edges on PA[8]
98 // *   - PA8 is setup to generate an interrupt for every falling edge
99 // *   - TIM3 is setup to use cascade 0 (PA9) for gating the counter
100 // *   (Note: For this program to run, a jumper must be placed between PA[8] and PA[1].)
101 //*****
102 #define T3_rollover 0x20000
103 uint32_t CONFIG_TIM3(void)
104 {
105     uint32_t temp ;
106
107     VOR_TIM3->CTRL = (TIM1_CTRL_IRQ_ENB_Msk | ( 2 << TIM1_CTRL_STATUS_SEL_Pos)) ; //
108     VOR_TIM3->RST_VALUE = T3_rollover ;
109     VOR_TIM3->CNT_VALUE = T3_rollover ;
110
111     VOR_TIM3->CSD_CTRL |= ( TIM1_CSD_CTRL_CSDEN0_Msk ) ;
112     // Setup cascade control so timer only counts when input is low
113     VOR_TIM3->CASCADE0 = (TIM_CAS_SRC_PORTA_8) ; // set cascade 0 source to PORTA11
114     VOR_TIM3->CASCADE1 = 0x00 ;
115     VOR_TIM3->CASCADE2 = 0x00 ;
116     VOR_TIM3->ENABLE = 0x1 ; // enable TIM1
117
118     // *** setup condition for cascade 0 input
119     VOR_SYSCONFIG->IOCONFIG_CLKDIV1 = 0x80 ; // set clkdiv0 to div128
120
121     VOR_IOCONFIG->PORTA[1]= IOCONFIG_PORTA_PLEVEL_Msk | IOCONFIG_PORTA_PEN_Msk | (FUNCSEL1 << IOCONFIG_PORTA_FUNSEL_Pos) ; //
122     VOR_IOCONFIG->PORTA[8]= ((5<<IOCONFIG_PORTA_FLTTYPE_Pos) | (1<<IOCONFIG_PORTA_FLTCLK_Pos) | IOCONFIG_PORTA_PLEVEL_Msk |
123
124     VOR_GPIO->BANK[0].IRQ_EVT &= ~(1 << TIM_CAS_SRC_PORTA_8) ; // clr EVT -> STAt generated when H->L transition in IRQ
125     VOR_GPIO->BANK[0].IRQ_SEN &= ~(1 << TIM_CAS_SRC_PORTA_8) ; // clr SEN -> Status is generated from signal transition
126     VOR_GPIO->BANK[0].IRQ_EDGE &= ~(1 << TIM_CAS_SRC_PORTA_8) ; // clr EDGE -> Status is generated from signal transition
127     VOR_GPIO->BANK[0].IRQ_ENB = 1 << TIM_CAS_SRC_PORTA_8 ; // set PA[8] IRQ Enable bit
128
129     return(temp) ;
130 }

```

Figure 12 - Code Snippet to setup Timer 3 to only count down during a falling edge of PA[8]

See Figure 13 for code used in the ISR.

```

227 //*****
228 // Interrupt Subroutine for TIM23 (NVIC IRQ30)
229 // - calculate pulse count
230 // - toggle PA[4] to show interval
231 //*****
232 void OC30_IRQHandlerX(void)
233 {
234     uint32_t elapsed_time ;
235
236     cnt_10ms ++ ;
237     new_SYSTICK = SysTick->VAL ; //
238     elapsed_time = old_SYSTICK - new_SYSTICK ;
239     old_SYSTICK = new_SYSTICK ;
240
241     if(i>16) { i = 0 ; }
242
243     PA8_pulse_cnt_array[i++] = (VOR_TIM3->RST_VALUE - VOR_TIM3->CNT_VALUE) ; // store pulse count in 16 unit array
244     VOR_TIM3->CNT_VALUE = VOR_TIM3->RST_VALUE ;
245
246     VOR_GPIO->BANK[0].DIR |= (1 << 4); // set PA[4] to an output
247     VOR_GPIO->BANK[0].TOGOUT |= (1 << 4); // Toggle PA[4] to show interval of interrupt
248 }
    
```

Figure 13 - Code Snippet to show ISR used for calculating number of pulses on TIM3

Figure 14 shows a scope capture of pulses being input to PA3 and the interval used on TIM3. On the right side, the TIM3 register values are shown. In the lower left, a 16 element array is shown with the last 16 interval edge counts.



Figure 14 - Scope capture with MCU RAM and register information for input pulse count. A) Waveform showing the interval for counting pulses and the actual pulse train. (Note 10 pulses per half period of the window), B) Memory display window from debugger showing the 16 entry array with pulse count and C) TIM3 register contents to count pulses on PA[8]

A separate project, AN1202_input_capture.uvprojx, accompanies this application note in the SW attachment. Open, compile, download and run the project similarly to what was done in example 1. A jumper will need to be placed on the REB1 board between PA[1] and PA[8] for the demonstration program to run properly.

Example 4: 3-phase motor control driver example

Many motor control applications configure their power switches as shown here. This configuration allows 6 distinct magnetic field vectors to generated in the motor windings. With proper control the switches can be turned on and off to creating a rotating field in the motor’s stator which applies torque to the rotor.

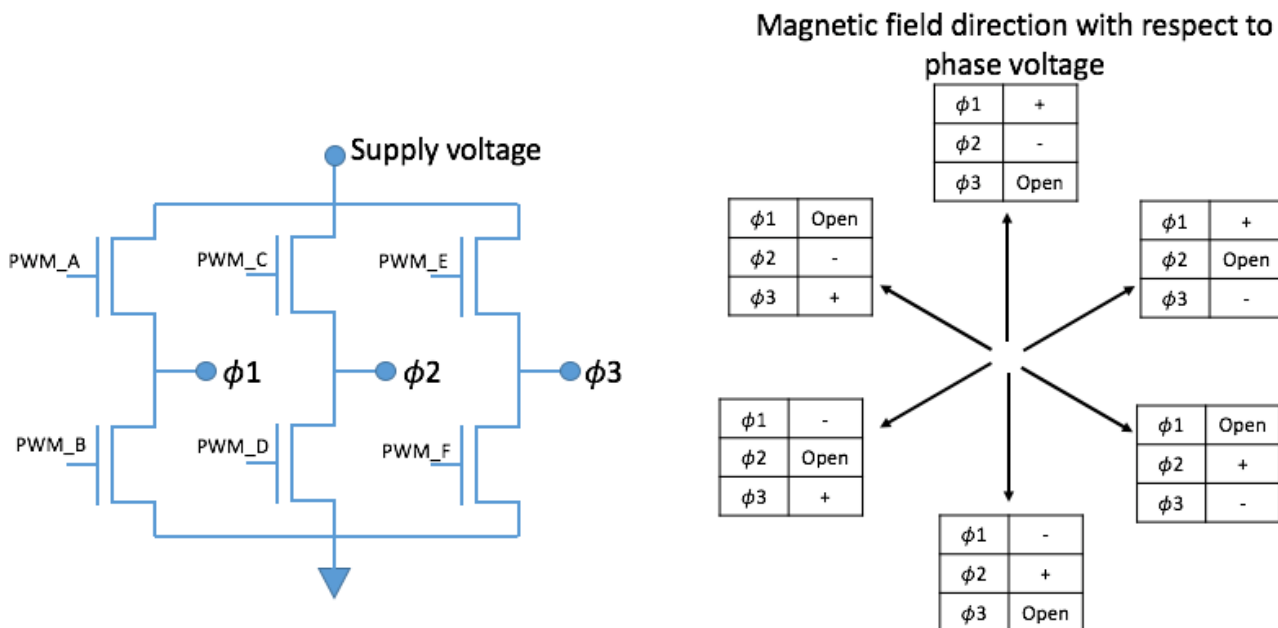


Figure 15 - Common power switch topology for 3-phase motor control & magnetic field direction diagram.

This configuration requires coordinated PWM outputs to control both torque and speed. Care must be taken to avoid having the top and bottom switching devices of a single leg turned on at the same time. This is commonly referred to as “shoot-through” and can cause permanent damage. The VA108x0 timer can produce the necessary PWM signals and include an immediate shutdown mechanism if a fault is detected.

The timer topology for this example is shown in Figure 16. To synchronize all the timers we use TIM0 as a trigger. TIM1-6 are setup in PWMB mode to create 120 degree intervals as shown in Figure 17. TIM7-9 are used to provide higher speed PWM switching to control the energy provided to the motor. For this example, TIM7-9 use a simple PWMA mode to implement a chopper type of control. It is possible to use PWMB mode with interrupts to change the duty cycle after each PWM period to more closely resemble an ideal sinusoid. A future application note will show a full motor control application with current monitoring and emergency shutdown if an overcurrent condition exists.

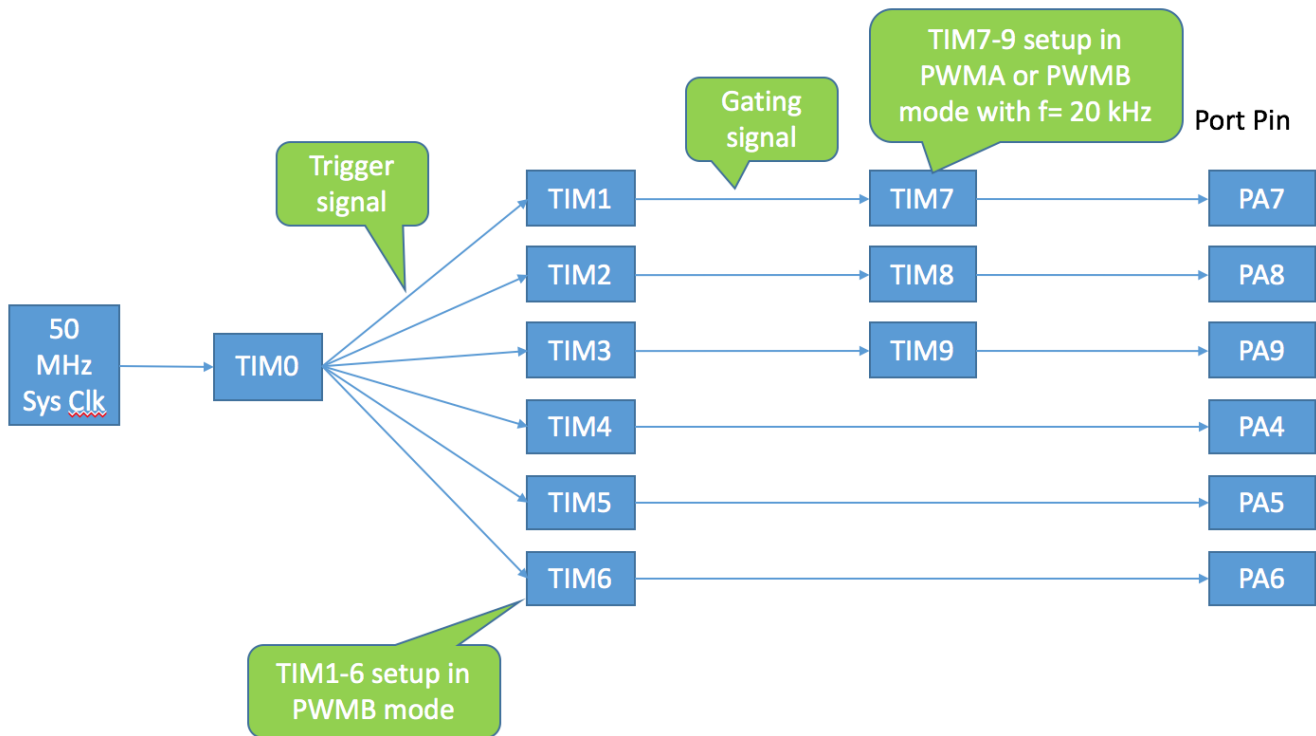


Figure 16 - Timer topology to create the 6 PWM outputs to feed a three phase motor

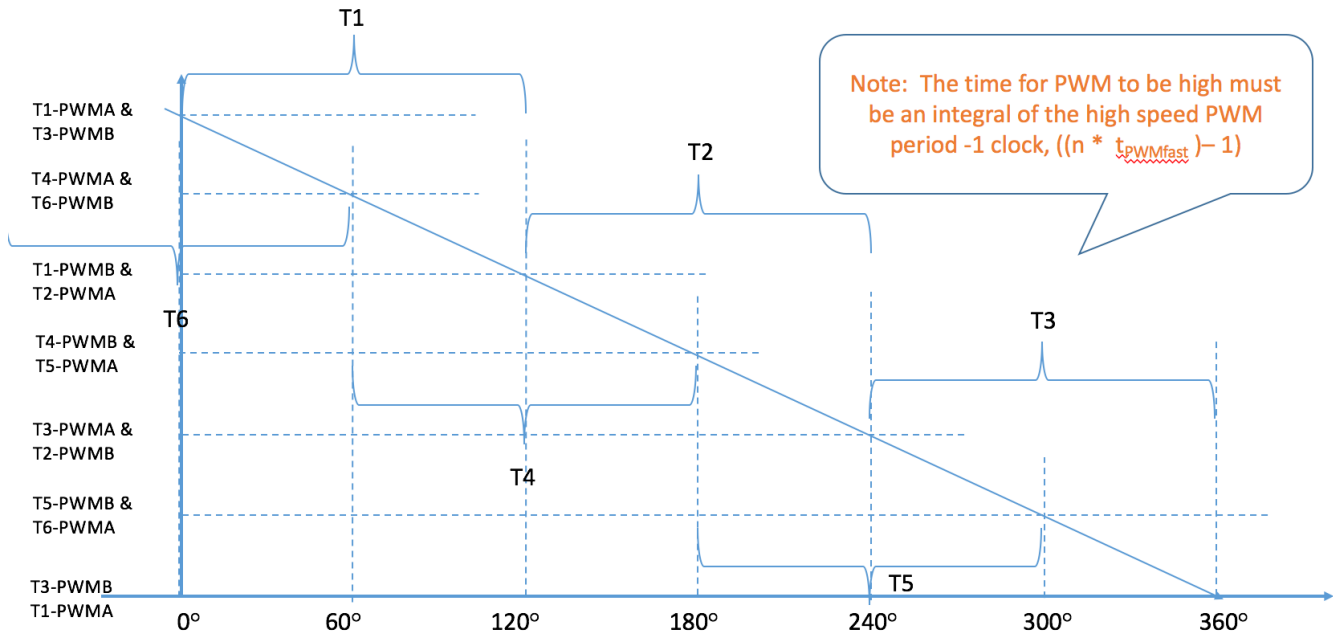


Figure 17 - Timeline of 6 channel output generation for 3-phase motor. Note that all 6 outputs are re-synchronized each cycle at 0 degrees.

The software for this example uses nested structures for each of the 6 phases and timer driver library as shown in Figure 18 and Figure 19. This reduces error prone redundant coding for direct register writes and forces consistency between channel setup.

```

28 /**
29  \brief Signal type enum for timer.
30  */
31 typedef enum {
32     SIGNAL_TYPE_TRIGGER,           ///< Trigger signal type.
33     SIGNAL_TYPE_PULSE,            ///< Pulse signal type.
34     SIGNAL_TYPE_PWM_PULSE         ///< PWM pulse signal type.
35 } VOR_TIM_SIGNALTYPE;
36
37 /**
38  \brief Allocated timers structure for timer.
39  */
40 typedef struct {
41     uint8_t timers[MAX_TIMERS];    ///< An array of allocated timers.
42     uint8_t num_timers;           ///< Specifies number of timers allocated.
43     uint8_t trigger_timer_index;  ///< Specifies the trigger timer index in allocated timers array
44 } VOR_TIM_ALLOCTIMERS;
45
46 /**
47  \brief Trigger signal structure for timer, where all attributes for that signal are specified.
48  */
49 typedef struct {
50     uint8_t timer_instance;       ///< Specifies trigger timer instance.
51     uint8_t port_num;            ///< Specifies the port number for the port pin used to route this signal.
52     uint8_t pin_num;            ///< Specifies the pin number to which the signal will be routed.
53     uint32_t cycle;              ///< Specifies the cycle period in micro seconds.
54 } VOR_TIM_TRIGSIGNAL;
55

```

Figure 18 - Code excerpt #1 from `reb_timer.h` showing structures used to organize timer configuration.

```

56 /**
57  \brief Pulse signal structure for timer, where all attributes for that signal are specified.
58  */
59 typedef struct {
60     uint8_t timer_instance;       ///< Specifies pulse timer instance.
61     uint8_t port_num;            ///< Specifies the port number for the port pin used to route this signal.
62     uint8_t pin_num;            ///< Specifies the pin number to which the signal will be routed.
63     uint32_t cycle;              ///< Specifies the cycle period in micro seconds.
64     uint32_t start_offset;       ///< Specifies start offset for this signal.
65     uint32_t end_offset;        ///< Specifies the end offset for this signal.
66     bool inverse;                ///< Specifies whether signal has to be inverted.
67 } VOR_TIM_PULSE SIGNAL;
68
69 /**
70  \brief PWM pulse signal structure for timer, where all attributes for that signal are specified.
71  */
72 typedef struct {
73     uint8_t timer_instance;       ///< Specifies PWM pulse timer instance.
74     uint8_t port_num;            ///< Specifies the port number for the port pin used to route this signal.
75     uint8_t pin_num;            ///< Specifies the pin number to which the signal will be routed.
76     uint32_t cycle;              ///< Specifies the cycle period in micro seconds.
77 } VOR_TIM_PWM_PULSE SIGNAL;
78
79 /**
80  \brief Signal structure for timer, which encompasses various other signal structures such as trigger, pulse and PWM pulse.
81  */
82 typedef struct {
83     VOR_TIM_TRIGSIGNAL trig;      ///< Trigger signal attributes
84     VOR_TIM_PULSE SIGNAL pulse;   ///< Pulse signal attributes
85     VOR_TIM_PWM_PULSE SIGNAL pwpulse; ///< PWM pulse signal attributes
86     VOR_TIM_SIGNALTYPE type;     ///< Specifies signal type - trigger, pulse or PWM pulse
87 } VOR_TIM_SIGNAL;
88

```

Figure 19 - Code excerpt #2 from `reb_timer.h` showing structures used to organize timer setup.

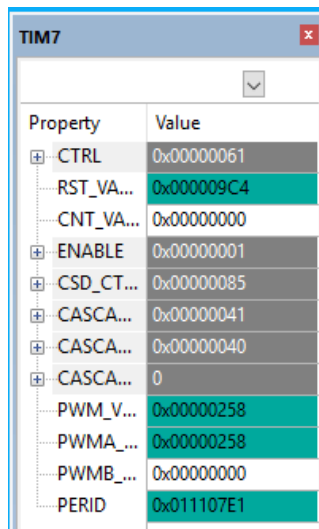
The structures are initialized with code as shown in Figure 20. Note that the start and end offset values are calculated in the pulse structure assignment as a ratio of the 360 degree cycle.

```

267
268 s7.trig = (VOR_TIM_TRIGSIGNAL){.timer_instance = 0, .port_num = 0, .pin_num = 0, .cycle = 18000};
269 s7.pulse = (VOR_TIM_PULSE SIGNAL){.timer_instance = 1, .port_num = 0, .pin_num = 1, .cycle = init_motor_period,
270     .start_offset = (0*init_motor_period/360), .end_offset = (120*init_motor_period/360), .inverse = false};
271 s7.pwmpulse = (VOR_TIM_PWM PULSE SIGNAL){.timer_instance = 7, .port_num = 0, .pin_num = 7, .cycle = init_fpwm_period};
272 s7.type = SIGNAL_TYPE_PWM PULSE;
273 VOR_TIM_Create(s7);
274
275 s8.trig = (VOR_TIM_TRIGSIGNAL){.timer_instance = 0, .port_num = 0, .pin_num = 0, .cycle = 18000};
276 s8.pulse = (VOR_TIM_PULSE SIGNAL){.timer_instance = 2, .port_num = 0, .pin_num = 2, .cycle = init_motor_period,
277     .start_offset = (120*init_motor_period/360), .end_offset = (240*init_motor_period/360), .inverse = false};
278 s8.pwmpulse = (VOR_TIM_PWM PULSE SIGNAL){.timer_instance = 8, .port_num = 0, .pin_num = 8, .cycle = init_fpwm_period};
279 s8.type = SIGNAL_TYPE_PWM PULSE;
280 VOR_TIM_Create(s8);
281
282 s9.trig = (VOR_TIM_TRIGSIGNAL){.timer_instance = 0, .port_num = 0, .pin_num = 0, .cycle = 18000};
283 s9.pulse = (VOR_TIM_PULSE SIGNAL){.timer_instance = 3, .port_num = 0, .pin_num = 3, .cycle = init_motor_period,
284     .start_offset = (240*init_motor_period/360), .end_offset = (360*init_motor_period/360), .inverse = false};
285 s9.pwmpulse = (VOR_TIM_PWM PULSE SIGNAL){.timer_instance = 9, .port_num = 0, .pin_num = 9, .cycle = init_fpwm_period};
286 s9.type = SIGNAL_TYPE_PWM PULSE;
287 VOR_TIM_Create(s9);
288

```

Figure 20 - Code excerpt from main routine that sets up timers 7, 8 and 9.



Property	Value
CTRL	0x00000061
RST_VA...	0x000009C4
CNT_VA...	0x00000000
ENABLE	0x00000001
CSD_CT...	0x00000085
CASCA...	0x00000041
CASCA...	0x00000040
CASCA...	0
PWM_V...	0x00000258
PWMA_...	0x00000258
PWMB_...	0x00000000
PERID	0x011107E1

Figure 21 - TIM7 register content after the VOR_TIM_Create(s7) is executed.

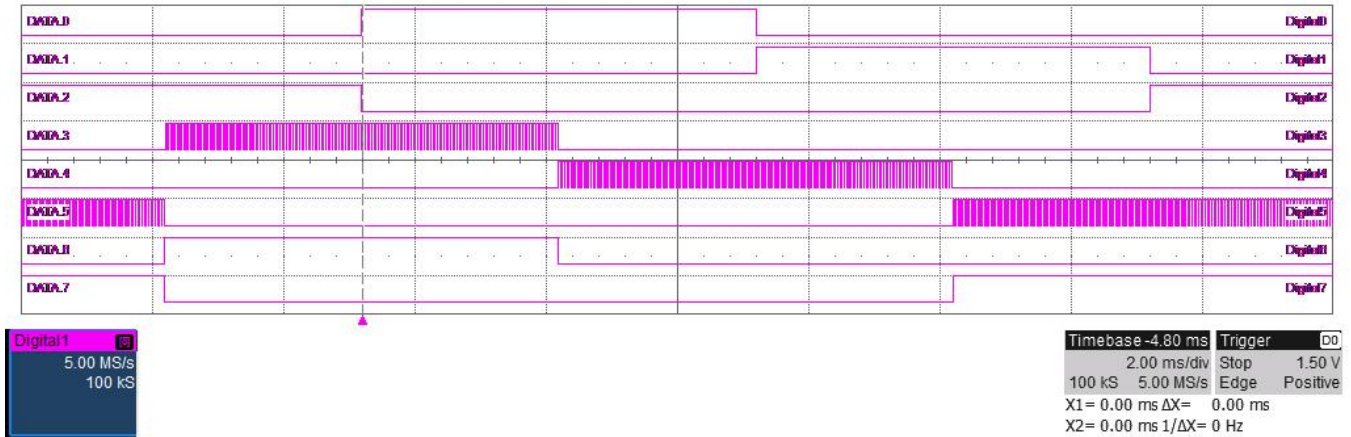


Figure 22 - Waveform of 6 channels (DATA0-DATA5) feeding 6 power transistors in a 3-phase H-bridge. Channels DATA6 and DATA7 are the timers cascaded timers feeding DATA3 and DATA5.

A separate project accompanies this application note in the SW attachment. Open, compile, download and run the "AN1202_3phase_pwm.uvprojx" project similarly to what was done in example 1. Port A 1-9 signals can be monitored on the REB1 board to see all the waveforms. Note that PA[6] and PA[7] are not routed to connectors but can be probed on LED's D3 and D4.

4. Summary

This application note has demonstrated many of the functions possible with the 24 TIM channels on the VA108xx. Compared to many other M0 based devices, the VA108xx has a very powerful timer subsystem using 32-bit counters and has the ability to automatically start or stop when specific activity occurs on GPIO or other timers.

Examples of PWM outputs, input captures, periodic interrupts and a 3-phase motor drive were provided. This should provide a starting platform for most applications making use of the TIM subsystem.

5. Common questions and issues

- Question: How are timer channel interrupts prioritized?

- a. This is determined by the programmable ARM M0 NVIC. ARM Cortex-M uses the “reversed” priority numbering scheme for interrupts, where priority zero corresponds to the highest urgency interrupt and higher numerical values of priority correspond to lower urgency. The M0 has 2 bits to set interrupt priority hence only 4 interrupt priority levels.
- b. If two channels have the same interrupt vector, software will need to handle the potential for consecutive interrupts.
- Question: What is the finest resolution a timer can have?
 - a. All timers will use the MCU bus clock for decrementing the counter. The maximum frequency of the VA108xx is 50 MHz, hence the finest resolution is 20 nsec.
- Question: How to get timer that counts every 10 msec. instead of at the bus frequency?
 - a. There are several ways to do this but the simplest is to use one of the pin filter dividers as a cascade input. The pin filter will create a single bus cycle pulse every time it reaches a count of zero.
- Question: Even when the processor is halted during debug, the timer continues to run. Is that right?
 - a. Yes, all the peripherals continue to run when the CPU is halted. Care should be taken when debugging with active power transistors and loads.

*** Change log: ***

Rev 1.2 Feb 2017

- Corrected unresolved cross reference to Table 3.

For more information, contact below or visit our web site at www.voragotech.com

VORAGO Technologies | 1501 S MoPac Expressway, Suite 350, Austin, Texas, 78746 | info@voragotech.com