AN1602
APPLICATION NOTE

16-BIT TIMING OPERATIONS USING ST7262 OR ST7263B MCUs

Microcontroller Division Applications

INTRODUCTION

This Application Note describes how to use the ST7262 or ST7263B for 16-bit timing operations. The intention of this document is to show how to perform pulse measurement and PWM generation using the different timers available on each microcontroller type.

1 16-BIT TIMING MEASUREMENTS

1.1 USING ST7262

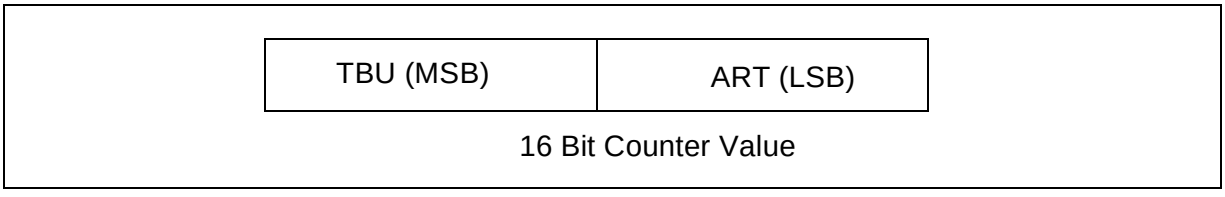
The aim of this section is to describe how to use the ST7262 8-bit timer to perform pulse and frequency measurements with 16-bit accuracy.

1.1.1 16-Bit Capture

1.1.1.1 Principle

The ST7262 architecture allows you to cascade the 8-bit Auto Reload Timer (ART) with the 8-bit Time Base Unit (TBU) to obtain a 16-bit counter (The carry bit of the ART acts as the clock for the TBU). Using this configuration, the ARTCAR register represents the Least Significant Byte and the TBUCV register represents the Most Significant Byte (see Figure 1).

Figure 1. TBU+ART 16-Bit Counter Value



On each valid input capture, the 16-bit counter value is obtained by:

– The ARTCAR register which is automatically latched.

– The TBUCV register which is saved by software in the Input Capture Interrupt routine.

This method ensures a “real-time” capture of the low bits, while the software delay needed to save the TBUCV can be compensated.

1.1.1.2 Delay Compensation method

The software delay introduced by the interrupt routine should be taken into account otherwise the measurement value can be corrupted. This happens if the ART counter rolls over from FFh to 00h while software is storing the TBUCV value after a valid input capture interrupt. In this case, the TBUCV is incremented and a wrong value occurs in the 8th bit of the 16-bit input capture value.

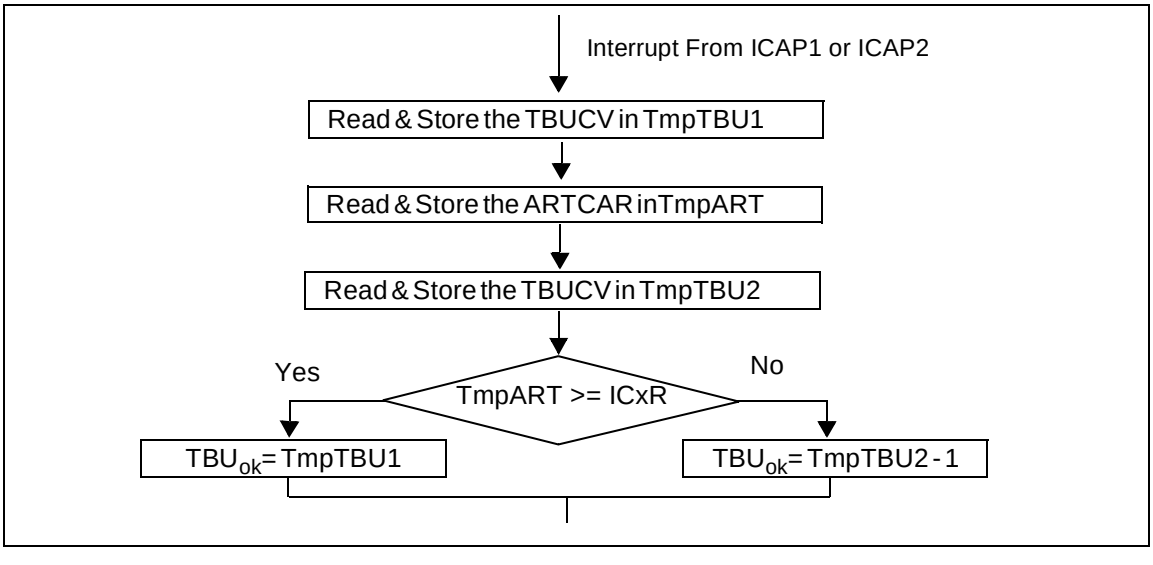
The delay compensation method is described in the following flowchart (Figure 2).

The first step consists of transferring both TBU and ART values into temporary variables (TmpTBU1, TmpART, TmpTBU2) to be used later on. Then in the second step we compare TmpART and ICxR. There are two cases to be considered:

– TmpART >= ICRx: This means that the ART counter value was incremented by a few cycles but has not yet reached the maximum value (FFh). In this case we use TmpTBU1.

– TmpART < ICRx:This means that the ART counter was incremented beyond the maximum value and has been reset. This means we should decrement TmpTBU2 once to get the right value of TBUCV.

Figure 2. TBU Delay Compensation flowchart



1.1.1.3 Example of an interrupt capture routine

The input capture interrupt routine given below can be used to get the precise value of the input capture interrupt instant on ICAP1 pin. It can be used to detect a rising or falling edge depending on the configured edge in ARTICSR.

```
.Inp_Cap_Routine
;---------------------------------------;
      Ld A , TBUCV                      ;Load the TBU counter value
```

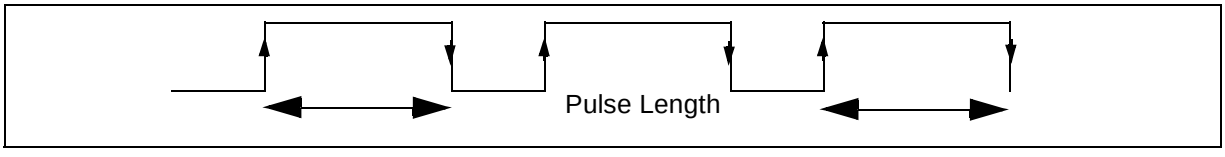
```
      Ld TmpTBU1 , A                          ;into TmpTBU1
      Ld A , ARTCAR                           ; Load the ART counter Value
      Ld TmpART1 , A                          ;into TmpART1
      Ld A , TBUCV                            ; Load the TBU counter value into
      Ld TmpTBU2 , A                          ;TmpTBU2
;-----------------------------------------;
      Ld A , TmpART1                          ; Compare ART & ICR
      Cp A , ARTICR2                          ; If ART > ICR then Case 2
      jrpl Case2                              ;
                  Ld A , TmpTBU2              ; Case 1 : TBU = TBU2 - 1
                  Dec A                       ;
                  jra Next2                   ;
.Case2            Ld A , TmpTBU1              ; Case TBU = TBU1
.Next2            Ld PulseEndHR , A           ;load TBUCV into PulseEndHR
                  Ld A , ARTICR2              ;Load The ART Inp Capt 2 register
                  Ld PulseEndLR , A           ;Value into PulseEndLR
                  Ld A , ARTICCSR             ;Clear The input capture interrupt
                  Iret                        ;Exit From Interrupt
```

1.1.2 Pulse Length measurement

1.1.2.1 Measurement Principles

As first example in this application note, we will try to measure the pulse length of an external signal applied to the input capture pins.

Figure 3. Pulse Length Measurement



To perform this operation, two input captures are used together and they are configured as follows:

– Input capture1 pin (ARTIC1) configured to detect rising edges

– The input capture2 pin (ARTIC2) configured to detect falling edges

The ARTIC1 & ARTIC2 are connected together (Figure 4).

Two 16-bits variables are used to hold the 16-bits timer values on each valid input capture edge (PulseStart for the rising edge and PulseEnd for the edge).The pulse length value can be deduced later using a 16-bits substruction between PulseStart and PulseEnd

Figure 4. Input Capture Configuration

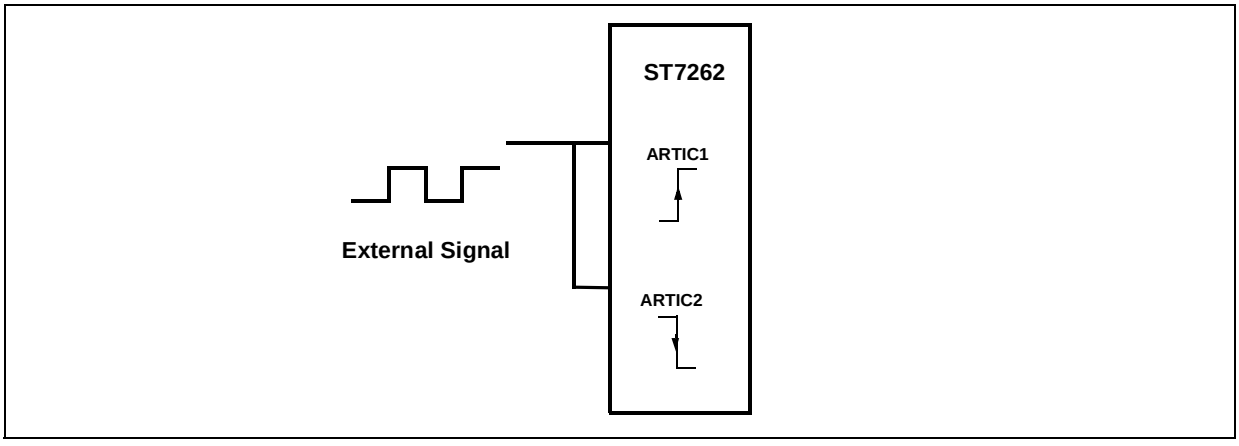
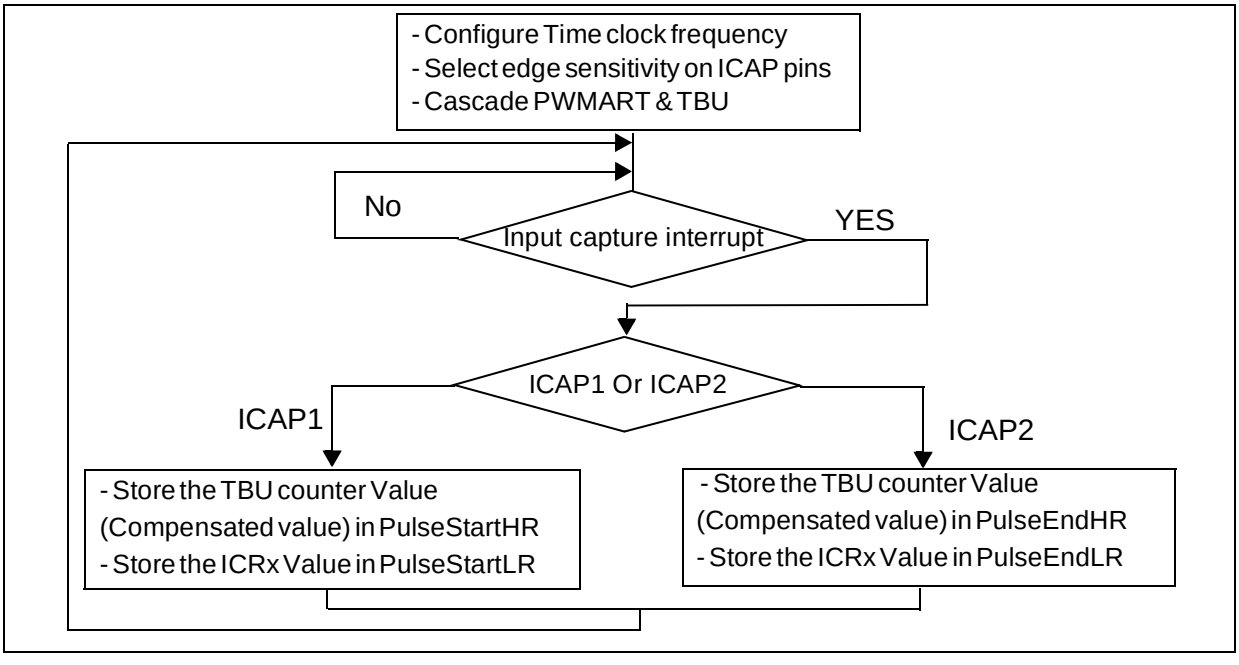


Figure 5 shows the steps that have to be followed to successfully measure the length of a pulse in 16-bit accuracy.

Figure 5. Input Capture Sequence



1.1.2.2 Example of input capture interrupt routine

```
.Inp_Cap_Routine                    ;
      Btjt ARTICCSR,#0,Flag1        ; Test If is ICAP(1 Or 2)
.Flag2 Ld A , ARTICR2               ; Latch the Input Cap Reg into
      Ld IcapReg2 , A               ; IcapReg2
      Ld A , ARTICCSR               ; Clear Interrupt
```

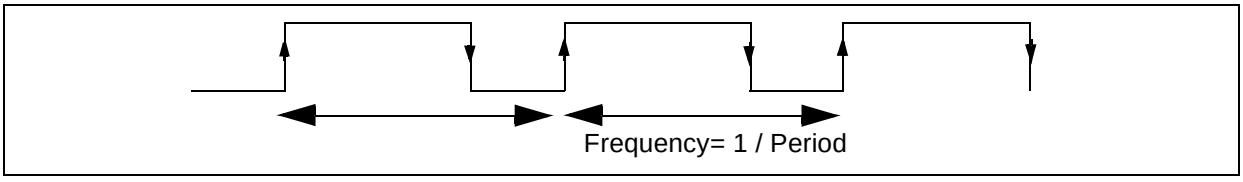
```
       Ld A , TBUCV                    ; Load the TBU counter value
       Ld TmpTBU1 , A                  ; into TmpTBU1
       Ld A , ARTCAR                   ; Load the ART counter Value
       Ld TmpART1 , A                  ; into TmpART1
       Ld A , TBUCV                    ; Load the TBU counter value into
       Ld TmpTBU2 , A                  ; TmpTBU2
                     Ld A , TmpART1    ; Compare ART & ICR
                     Cp A , IcapReg2   ; If ART > ICR then Case 2
                     jrpl Case2        ;
                           Ld A ,TmpTBU2; Case 1 : TBU = TBU2 - 1
                           Dec A       ;
                           jra Next2   ;
.Case2               Ld A , TmpTBU1    ; Case TBU = TBU1
.Next2               Ld PulseEndHR , A ; load TBUCV into PulseEndHR
                     Ld A , IcapReg2   ; Load The ART Inp Capt 2 register
                     Ld PulseEndLR , A ; Value into PulseEndLR
                     Iret              ; Exit from Interrupt
.Flag1 Ld A , ARTICR1                  ; Latch the Input Cap Reg into
       Ld IcapReg1 , A                 ; IcapReg1
       Ld A , ARTICCSR                 ; Clear Interrupt
       Ld A , TBUCV                    ; Load the TBU counter value
       Ld TmpTBU1 , A                  ; into TmpTBU1
       Ld A , ARTCAR                   ; Load the ART counter Value
       Ld TmpART1 , A                  ; into TmpART1
       Ld A , TBUCV                    ; Load the TBU counter value into
       Ld TmpTBU2 , A                  ; TmpTBU2
                     Ld A , TmpART1    ; Compare ART & ICR
                     Cp A , IcapReg1   ;
                     jrpl Case1        ;
                           Ld A , TmpTBU2; Case : TBU = TBU2 - 1
                           Dec A       ;
                           jra Next1   ;
.Case1               Ld A , TmpTBU1    ; Case TBU = TBU1
.Next1               Ld PulseStartHR , A; load TBUCV into PulseEndHR
                     Ld A , IcapReg1   ; Load The ART Inp Capt 2 register
                     Ld PulseStartLR , A; Value into PulseEndLR
                     Iret              ; Exit from Interrupt
```

1.1.3 Frequency Measurement

1.1.3.1 Measurement Principles

Frequency measurement is a little bit different from pulse length measurement. To get the frequency, we measure the full signal period which can be defined by the time interval between two consecutive rising edges (Figure 6)

Figure 6. Frequency Measurement



In this example we will also use two 16-bit variables to hold the 16-bit timer value each time an input capture occurs (StartPeriod: for the first rising edge and EndPeriod for the second falling edge). In addition to these two variables, we will need an extra variable (FirstEdge) which is used to indicate the first rising edge or the second edge.

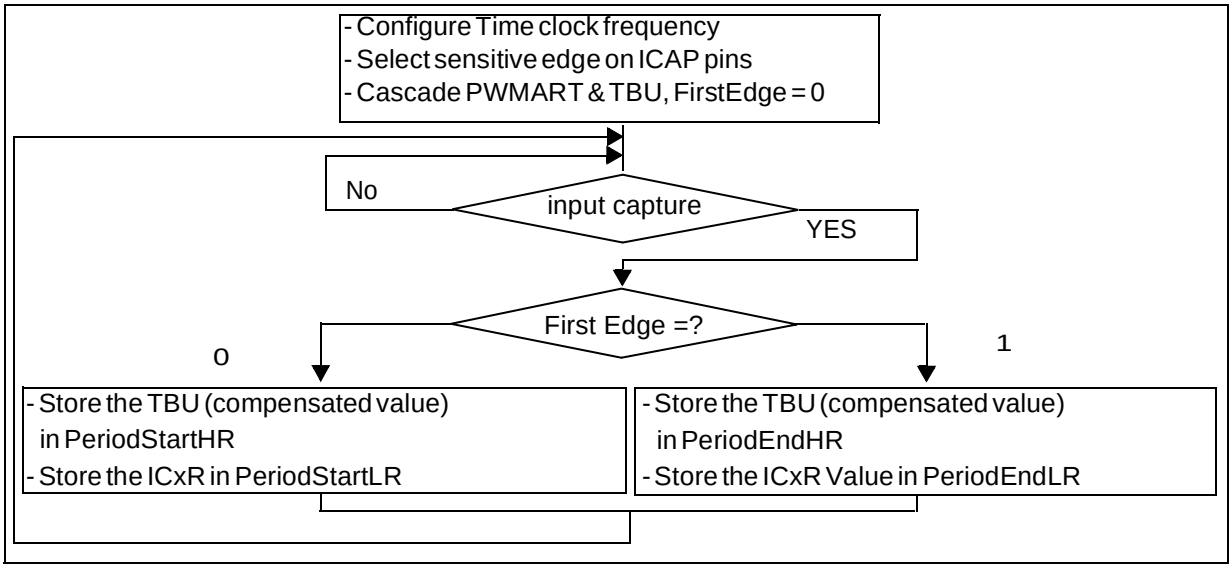
– FisrtEdge =0: First Rising edge (PeriodStart)

– FisrtEdge =1: Second Rising edge (PeriodEnd)

In this case we will need only one input capture pin (ARTIC1) configured to detect only rising edge.

The capture steps are shown in Figure 7. After selecting the timer frequency and the input capture edge we wait for an input capture interrupt. Depending on the FirstEdge value the input capture parameters (ICxR and the TBU) are transferred into PeriodStart or PeriodEnd.

Figure 7. Input Capture Sequence



1.1.3.2 Example of input capture interrupt routine

```
.Inp_Cap_Routine                          ;
      Ld A , ARTICR1                      ;
      Ld CurrentICAP , A                  ;
;-----------------------------------------;
      Ld A, ARTICCSR                      ;
;-----------------------------------------;
      Ld A , TBUCV                        ;Load the TBU counter value
      Ld TmpTBU1 , A                      ;into TmpTBU1
      Ld A , ARTCAR                       ;Load the ART counter Value
      Ld TmpART1 , A                      ;into TmpART1
      Ld A , TBUCV                        ;Load the TBU counter value into
      Ld TmpTBU2 , A                      ;
;-----------------------------------------;
      Btjf FirstEdge,#0,FirstRisingEdge   ;TmpTBU2
;-----------------------------------------;
;.SecondRisingEdge                        ;
                  Ld A , TmpART1          ;Compare ART & ICR
                  Cp A , CurrentICAP      ;If ART > ICR then Case 2
                  jrpl Case2              ;
                        Ld A , TmpTBU2    ;Case 1 : TBU = TBU2 - 1
                        Dec A             ;
                        jra Next2         ;
.Case2            Ld A , TmpTBU1          ;Case TBU = TBU1
.Next2            Ld PeriodEndHR , A      ;load TBUCV into PeriodEndHR
                  Ld A , CurrentICAP      ;Load The ART Inp Capt 1 register
                  Ld PeriodEndLR , A      ;Value into PeriodEndLR
                  Clr FirstEdge           ;
                  Iret                    ;
.FirstRisingEdge                          ;
                  Ld A , TmpART1          ;Compare ART & ICR
                  Cp A , CurrentICAP      ;
                  jrpl Case1              ;
                        Ld A , TmpTBU2    ;Case : TBU = TBU2 - 1
                        Dec A             ;
                        jra Next1         ;
.Case1            Ld A , TmpTBU1          ; Case TBU = TBU1
.Next1            Ld PeriodStartHR , A    ;load TBUCV into PulseEndHR
                  Ld A , CurrentICAP      ;Load The ART Inp Capt 1 register
                  Ld PeriodStartLR , A    ;Value into PulseEndLR
                  Inc FirstEdge           ;
                  Iret                    ;
```

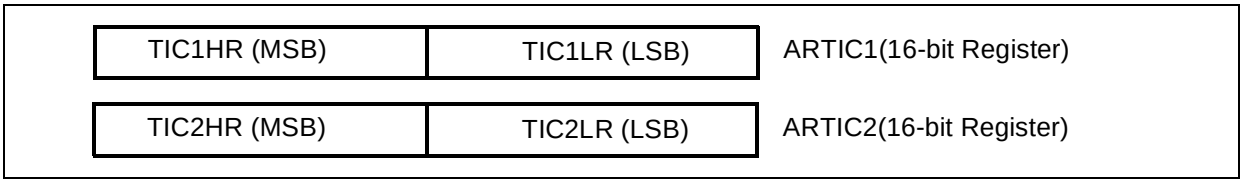
1.2 USING ST7263B

This section describe how to use the ST7263B 16-bit timer to perform 16-bit timing operations such as period and pulse length measurement.

1.2.1 The Input Capture operation

In the case of the ST7263B both ICAP1 and ICAP2 are associated with 16-bit registers (TIC1HR, TIC1LR, TIC2HR and TIC2LR). These registers are used to hold the corresponding 16-bit timer value at each valid input capture (see Figure 8).

Figure 8. Input Capture Registers

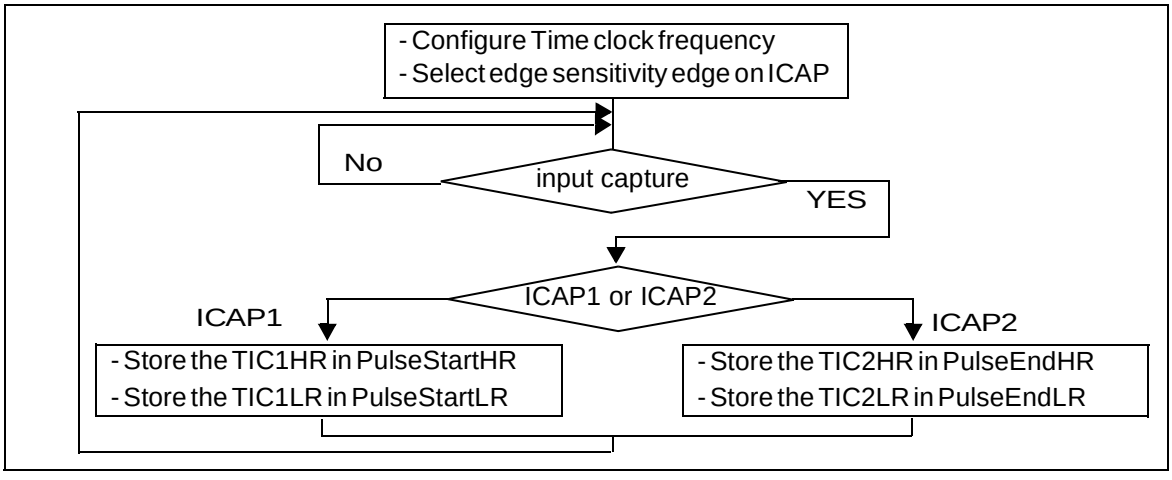


1.2.2 Pulse Length measurement

1.2.2.1 Measurement principles

To measure pulse length, two input captures are used together. The input capture1 pin (ICAP1) is configured to detect rising edge and the input capture2 pin (ICAP2) is configured to detect falling edge. The ICAP1 & ICAP2 pins are connected together and on each input capture: rising edge or falling edge two 16-bit variables are used to hold the 16-bit timer value.

Figure 9. Input Capture sequence



Whenever a valid rising edge is captured, the timer counter value is automatically stored in the TIC1HR & TIC1LR registers. These two registers act together as one 16-bit register that holds the counter value to be transferred into PulseStart (PulseStartHR & PulseStartLR).Whenever a valid falling edge is captured, the timer counter value is automatically stored in the TIC2HR

& TIC2LR registers. These two registers act together as one 16-bit register that holds the counter value to be stored in PulseEnd (PulseEndHR & PulseEndLR).

1.2.2.2 Example of input capture interrupt routine

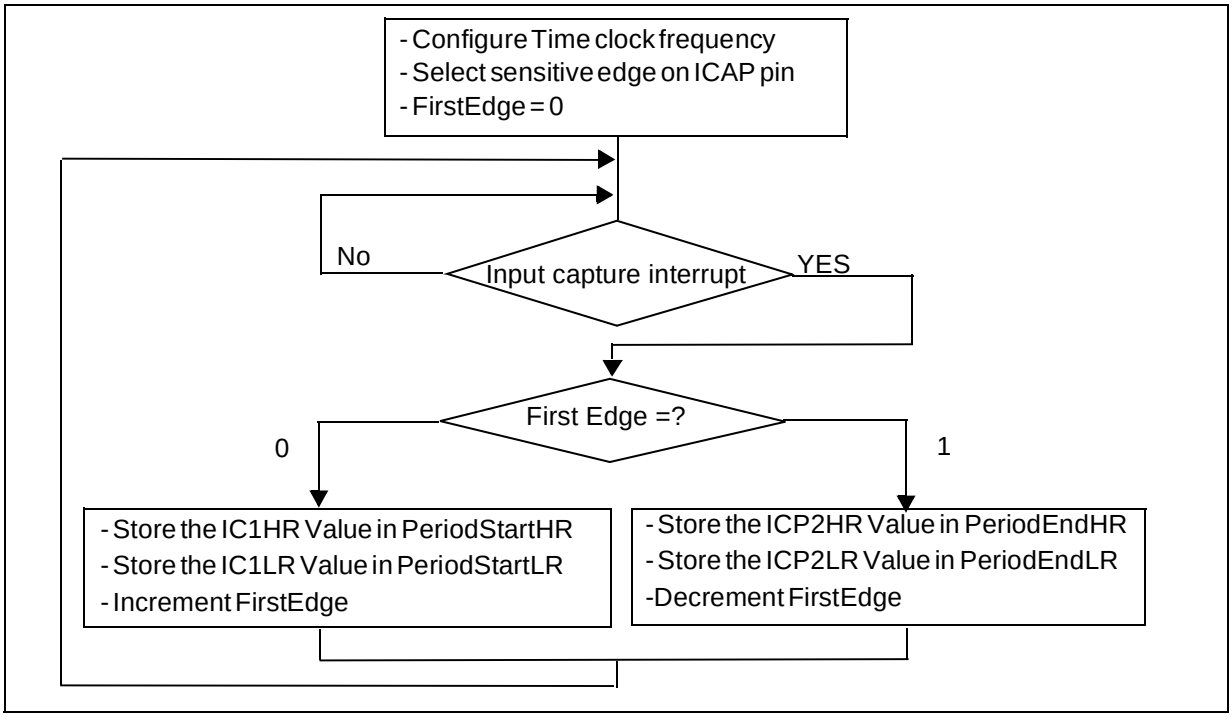
```
.Input_Capt_Rt
      Btjt TSR , #7 , Flag1                ; Test Rising or falling edge ?
.Flag2                                     ; Falling Edge
      Ld           A , TSR                 ; Clear Interrupt Flag
      Ld           A , TIC2HR              ; Load TIC2R ( HR & LR ) into
      Ld           PulseEndHR , A          ; PulseEnd ( HR & LR )
      Ld           A , TIC2LR              ;
      Ld           PulseEndLR , A          ;
      Iret                                 ;
.Flag1                                     ; Rising Edge
      Ld           A , TSR                 ;Load TIC1R ( HR & LR ) into
      Ld           A , TIC1HR              ; PulseStart ( HR & LR )
      Ld           PulseStartHR , A        ;
      Ld           A , TIC1LR              ;
      Ld           PulseStartLR , A        ;
      Iret                                 ;
```

1.2.3 Frequency Measurement

1.2.3.1 Measurement principles

To measure frequency we will use the same technique we used for the ST7262 (see Section 1.1.3). We need only one input capture pin configured to detect rising edges and two 16-bit variables to hold the 16-bit timer value at each input capture occurrence (StartPeriod) and (EndPeriod). We will need also FirstEdge which indicates the first rising edge or the second edge.

Figure 10. Input Capture Sequence



1.2.3.2 Example of input capture interrupt routine

```
.Input_Capt_Rt
       Ld A , TIC1HR                          ; Lock The TICR
       Btjf FirstEdge ,#0, FirstRisingEdge    ; Test First Or Second Edge
.SecondRisingEdge                             ;
       Ld PeriodEndHR , A                     ; Load PriodEnd Values
       Ld A , TIC1LR                          ;
       Ld PeriodEndLR , A                     ;
       Ld A , TSR                             ;Clear ICAP interrupt Flag
       Ld A , TIC1LR                          ;
       Clr FirstEdge                          ;
       Iret                                   ;
.FirstRisingEdge                              ;
       Ld PeriodStartHR , A                   ; Load PriodStart Values
       Ld A , TIC1LR                          ;
       Ld PeriodStartLR , A                   ;
       Ld A , TSR                             ;Clear ICAP interrupt Flag
       Ld A , TIC1LR                          ;
       Inc FirstEdge                          ;
       Iret                                   ;
```

1.3 PERIOD MEASUREMENT CAPABILITY

This section describes a comparison based on period measurement capability (rising edge to rising edge measurement) between the ST7262 16-bit timer (ART + TBU) and ST7263B 16-bit timer. All comparisons and measurement range values are calculated using the software provided with this application note and at f_{CPU} = 8 MHz.

1.3.1 Using the ST7262 ART & TBU in cascading mode

Analyzing the ST7262 period measurement capability in the cascading mode can be done by focusing on two important points: the maximum period and the minimum period that can be measured using this mode.

1.3.1.1 Maximum period

In cascading mode, the timer acts as a 16-bit free running counter. At f_{CPU} = 8 MHz, the maximum period can reach t_{max}

$$t_{max} = FFFFh * Timer_{clk} = 65535\ Timer_{clk}$$

Table 2 shows the time measurement range covered by the ST7262 ART & TBU timer.

Table 1. Time Measurement Range using ART & TBU

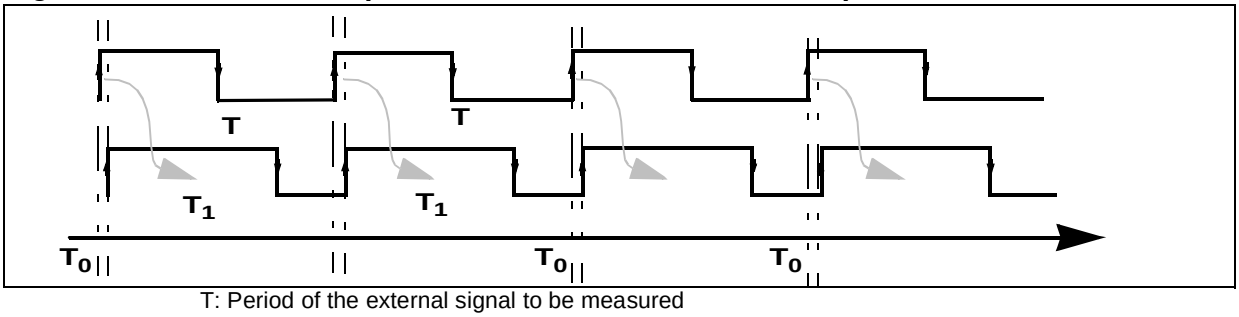
Prescaler	Timer Frequency	Maximum Period	Resolution
1	8 MHz	8,192 ms	0.125 µs
2	4 MHz	16,384 ms	0.250 µs
4	2 MHz	32,767 ms	0.500 µs
8	1 MHz	65,535 ms	1 µs
16	500 kHz	131,07 ms	2 µs
32	250 kHz	262,14ms	4 µs
64	125 kHz	524,28 ms	8 µs
128	62.5 kHz	1048,56 ms	16 µs

1.3.1.2 Minimum period

The minimum period that can be measured using the ST7262 16-bit timer depends on several parameters such as the software delay introduced by interrupt routine execution time and the hardware delay required to enter and exit the interrupt subroutine.

Figure 11 shows the normal behaviour of the input capture routine: each rising edge is detected and the input capture interrupt routine is executed after the T0 delay. In this case the period measurement operation can be easily done.

Figure 11. Case 1: Interrupt routine execution shorter than period to be measured

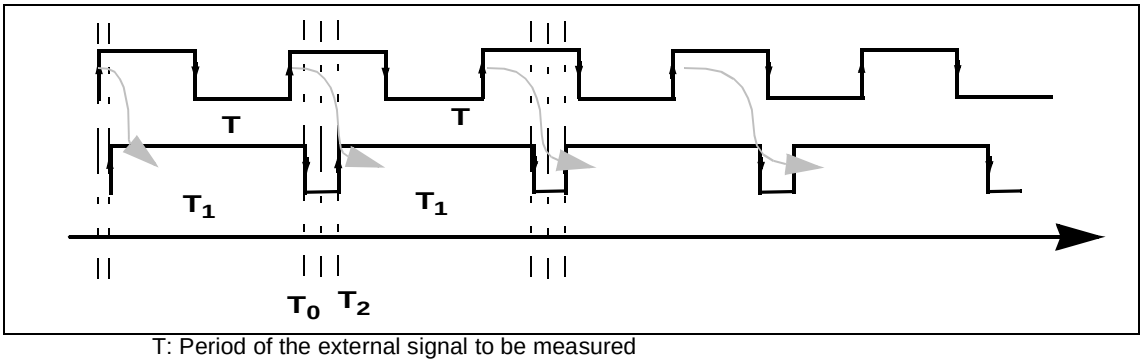


T: Period of the external signal to be measured
T_1: Interrupt routine execution time
T_0: Delay needed to activate the interrupt routine

The minimum period that can measured depends strongly on the number of rising edges during the interrupt routine execution.

A single rising edge during the execution of the interrupt routine does not corrupt the measurement value and the period measurement operation can be achieved successfully (Figure 12).

Figure 12. Case 2: Interrupt routine execution longer than period to be measured

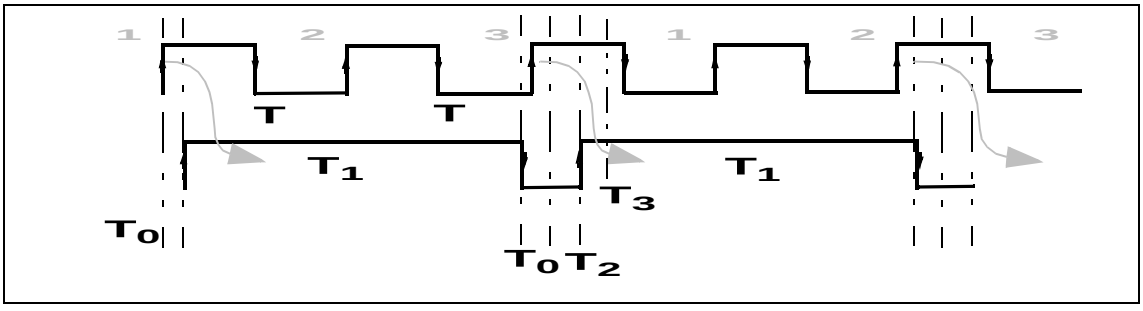


T: Period of the external signal to be measured
T_1: Interrupt routine execution time
T_0: Delay needed to activate the interrupt routine
T_2: Delay needed to exit from interrupt routine

However, any additional rising edge leads to a corrupted measurement. As shown in Figure 12, the second rising edge is ignored, in fact before executing the input capture interrupt triggered by the second rising edge, a third one occurs and overwrites the previous ICxR value. For this reason the period measurement operation result is corrupted
(result = 2 * T).

So the minimum period (the maximum frequency) that can be measured using the ST7262 16-bits cascaded timer can be calculated using the following formula.

Figure 13. Minimum measurable period (max. frequency)



$$2T > T_0+T_1+T_0+T_2+T_3$$

T_0: Time needed to activate the interrupt routine (10 t_{CPU}), and the time need to wait for the internal clock rising edge (about 2 t_{CPU}).

T_1: Time needed to execute the interrupt routine. Using the example software supplied in this application note, the input capture interrupt routine needs about 62 t_{CPU}.

T_2: The time needed to exit from the interrupt routine (9 t_{CPU}).

T_3: This represents the period of time from the start of the interrupt routine until the ICxR register is read (about 3 t_{CPU} in the example software).

Using the above values based on the software example:

$T_0 = 10$, $T_1 = 62\ t_{CPU}$, $T_2 = 9\ t_{CPU}$, $T_3 = 3\ t_{CPU}$

$$2T > T_0+T_1+T_0+T_2+T_3$$

$$2T > (12 + 62 + 12 + 9 + 3) * t_{CPU}$$

$$> 98 * 0.125\ \mu s$$

$$> 12.25$$

So we can calculate the minimum period Tmin = 6.125 µs.

1.3.2 Using the ST7263B 16-bit timer

1.3.2.1 Maximum Period

Using the ST7263B 16-bit timer, the maximum measurable period t_{max} can be expressed as follows:

t_{max} = FFFFh * $Timer_{clk}$ = 65535 $Timer_{clk}$

So the maximum period that can measured using the ST7263B 16-bit timer and the ST7262 cascaded timer is the same, however the minimum period is different.

Table 2. Time measurement range using ST7263B

Prescalers	Timer Frequency	Maxi Period	Resolution = 1 t_{CLK}
2	4 MHz	16,384 ms	0.250 µs
4	2 MHz	32,767 ms	0.500 µs
8	1 MHz	65,535 ms	1 µs

1.3.2.2 Minimum Period

In the case of ST7263B, on each valid input capture, the 16-bit counter value is latched automatically into the 16-bit input capture registers (TICxHR & TICxLR) without any additional software routine. So compared to the ST7262, the ST7263B needs less time to store the input capture parameters into temporary variables.

Using the same method as for the ST7262 (see Figure 13):

$$2T > T_0+T_1+T_0+T_2+T_3$$

And with the following parameters:

T_0= 12 t_{CPU} (10 t_{CPU} for the interrupt activation and 2 t_{CPU} the maximum time needed to synchronise the interrupt flag with the internal CPU frequency).

T_1= 26 t_{CPU}

T_2=9 t_{CPU}

T_3 = 23 t_{CPU} (from the start of the interrupt routine until the interrupt flag is cleared).

Using the above values based on the software example:

T_0 = 12, T_1 = 26 t_{CPU}, T_2 = 9 t_{CPU}, T_3 = 23 t_{CPU}

$$2 T > T_0+T_1+T_0+T_2+T_3$$

$$2 T > (12 + 26 + 12 + 9 + 23) * t_{CPU}$$

$$> 82 * 0.125 \mu s$$

$$> 10.25 \mu s$$

So we can calculate the minimum period Tmin = 5.125 µs.

1.3.2.3 Period Measurement Comparison Chart

The following table can be used to select the best prescaler value for a precise time or frequency measurement with the appropriate resolution.

Table 3. Period Measurement Capability Comparison Chart

	8-Bit Timer ART (ST7262)			16-Bit Timer ART & TBU (ST7262)			16-Bit Timer (ST7263B)		
Pre-scaler	Max. Measurement Period	Re-solution	Freq.	Max. Measurement Period	Re-solution	Freq.	Max. Measurement Period	Re-solution	Freq.
1	31,875 µs	0.125 µs	31,372 kHz	8,192 ms	0.125 µs	122,1 Hz			
2	63,750 µs	0.25 µs	15,686 kHz	16,384 ms	0.250 µs	61.05 Hz	16,384 ms	0.25 µs	61.05 Hz
4	127,5 µs	0.5 µs	7,843 kHz	32,767 ms	0.5 µs	30.52 Hz	32,767 ms	0.5 µs	30.52 Hz
8	255 µs	1 µs	3,921 kHz	65,535 ms	1 µs	15.26 Hz	65,535 ms	1 µs	15.26 Hz
16	510 µs	2 µs	1,996 kHz	131,07 ms	2 µs	7.63 Hz			
32	1.02 ms	4 µs	998 Hz	262,14 ms	4 µs	3,815 Hz			
64	2.04 ms	8 µs	499 Hz	524,28 ms	8 µs	1.907 Hz			
128	4.08 ms	16 µs	249,5 Hz	1048,56 ms	16 µs	0.953 Hz			

For example, to measure an external signal pulse with a period not exceeding 50 ms, we have to choose a prescaler greater than or equal to 8 (8,16, 32, 64,128) with the ART and TBU in cascading mode. However, using a prescaler of 8 gives a more precise measurement (the error is about 1 µs)

2 PWM MODE

Definition of terms

– PWM Period: The PWM period is the time interval needed by the PWM pattern to repeat itself.

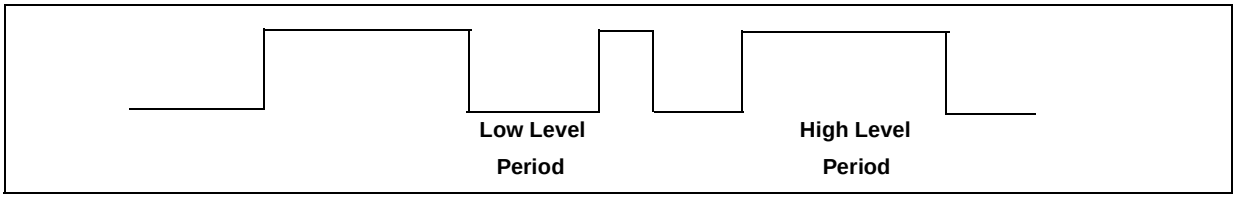
– PWM Frequency: The PWM frequency is 1/PWM period.

– PWM Resolution: The PWM resolution is the step with which the duty cycle is modulated

– High Level period: period of time when the PWM signal is at high Level

– Low Level period: period of time when the PWM signal is at Low Level

Figure 14. PWM Level definition



2.1 USING ST7262

2.1.1 16-Bit PWM generation

2.1.1.1 Principles

The ST7262 can be used to generate of PWM signals through the PWMx pins. This function can be selected using the OEx bit in the PWM Control register (PWMCR). All the PWM signals have the same frequency which is controlled by the counter period and the ARTARR register value.

$$f_{PWM} = f_{COUNTER} / (256 - ARTARR)$$

The registers used to perform the PWM signal generation are all 8-bit wide registers, so they cannot generate 16-bit PWM signals. To by-pass this hardware limitation the following method can be used.

The main idea of the method is to split all the 16-bit PWM parameters (Duty Cycle, Low level period, full signal period) into 8-bit parts .

For example if we consider that we have to generate a PWM signal with the following characteristics:

– PWM Full period = 700 µs

– PWM Duty Cycle = 5/7 (PWM high level = 500 µs)

with T_{Timer} = 0.5 µs

The High level period will be divided into 8-bit parts as described below .

High Level period = 500 µs

= 1000 * T_{Timer}

= 3E8h * T_{Timer} Or 3E8h = (3 * FFh + EBh)

So to generate the high level part of the PWM signal we have to:

– Generate three elementary PWM waves all at high level , each wave has a period length of FFh .

– Then generate an additional wave at high level but with a period length of EBh

Once the high level of the wave is split up we will use the same technique to generate the low level.

Low Level period = 200 µs

$= 400 * T_{Timer}$

$= 190h * T_{Timer}$

$= 1* FFh + 91h$

So to generate the low level part of the PWM signal we have to:

– Generate an elementary PWM wave at low level

– Then generate the last wave at low level with a period of 91h

These different waves are computed and then cascaded by software using the Interrupt routine.

For this reason, a minimum PWM period has to be respected to allow enough time to complete the interrupt routine task. In the software provided with this application note the minimum time to be respected is the following :

The minimum DutyCycle = 55 Tcpu with LowLevel period >= 85 Tcpu.

with Ttimer = 0.125 µs we were able to generate a PWM signal with high level period=10.6 µs and Lowlevel period = 16.63 µs .

This limitation applies only when working with values lower than FFh.

2.1.1.2 Generation Sequence

In general cases, the following variables are used to define the PWM signal in 16-bit mode:

– DutyCycleHR: 8-bit variable that defines the PWM signal duty cycle High register

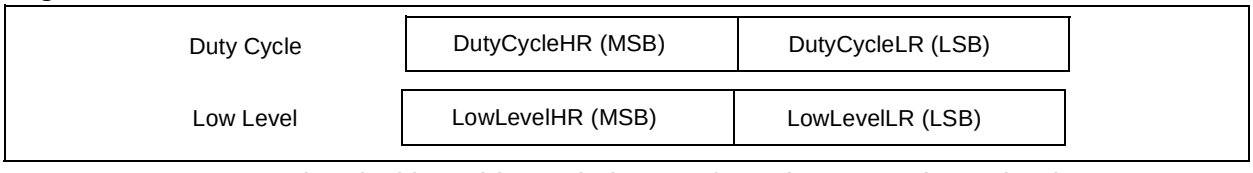
– DutyCycleLR: 8-bit variable that defines the PWM signal duty cycle low register

– LowLevelHR: 8-bit variable that defines the PWM signal low level period High Register

– LowLevelLR: 8-bit variable that defines the PWM signal low level period Low Register

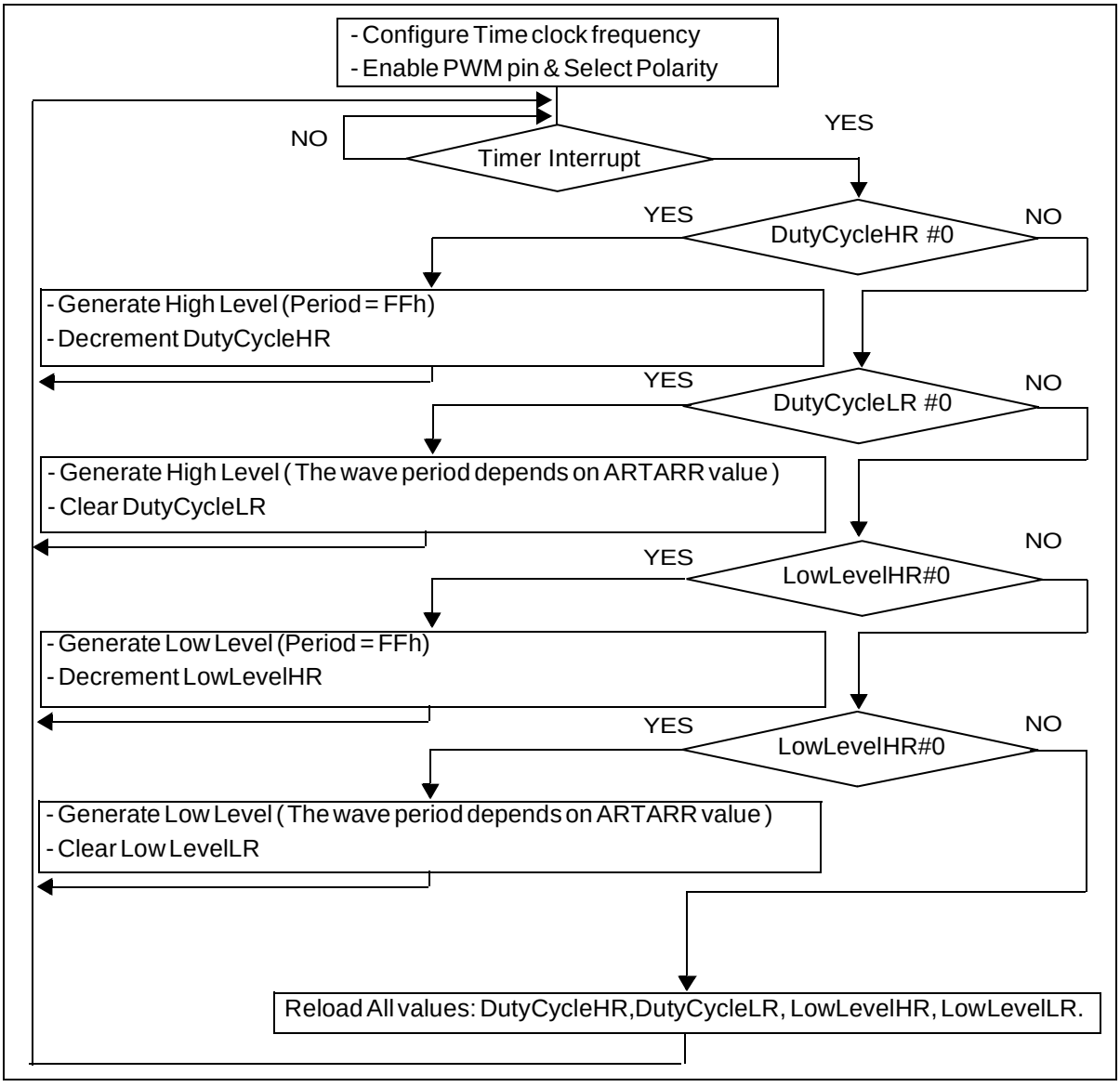
The pair of 8-bit variables (DutyCycleHR, DutyCycleLR) are used to hold the 16-bit value of the PWM high level. The second pair of registers used to hold the low level period are LowlevelHR and LowLevelLR.

Figure 15. PWM Parameters



To generate a PWM signal with 16-bit resolution, perform the steps shown in Figure 16.

Figure 16. 16-bit PWM generation

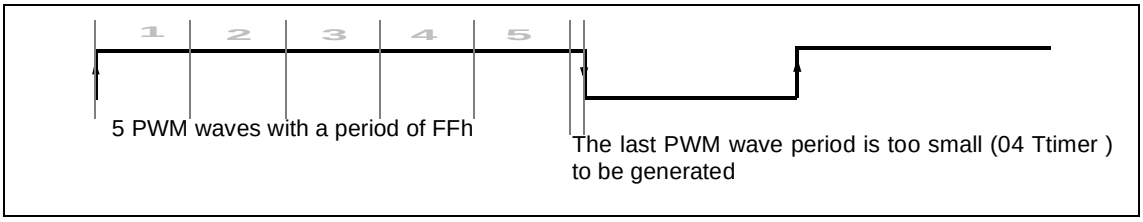


2.1.1.3 Optimization method

To get around the limitation imposed by the minimum PWM period, an optimization method has been implemented in the software.

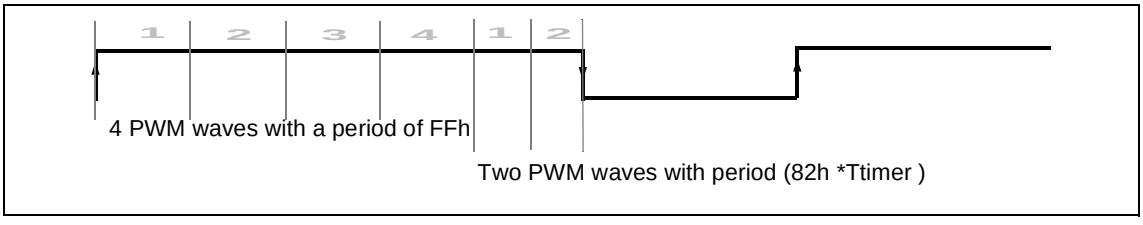
To generate a PWM signal with a duty cycle of 0504h , we have to generate 05 waves at high level (period = FFh) then the last wave with a period of 04 (see Figure 17) which is impossible (generation of a signal with a period equal to 04) due to the time needed by the interrupt routine .

Figure 17. Without Optimization



Using the optimization method we will generate only 04 waves (period = FFh) then two additional waves with a period of 82h (Figure 18) which provide the necessary delay for the interrupt routine.

Figure 18. With Optimization



2.2 USING ST7263B

2.2.1 Principles

The Pulse Width Modulation (PWM) mode in the ST7263B, enables the generation of a PWM signal with a frequency and pulse length controlled by the value of the OC1R and OC2R registers. To use Pulse Width Modulation mode we have to follow the steps described below:

Step 1. Load the OC2R register with the value corresponding to the period of the signal using the following formula

$$OCiR = (t \times Fcpu)/(Prescaler) - 5$$

With:

t = Signal or pulse period (in seconds)

F_{CPU} = CPU clock frequency (in hertz)

Prescaler = Timer prescaler factor

Step 2. Load the OC1R register with the value corresponding to the period of the pulse using the same formula as above.

Step 3. Configure CR1 register:

– Using the OLVL1 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC1R register.

– Using the OLVL2 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC2R register.

Step 4. Configure the CR2 register:

– Set the PWM bit.

– Select the timer clock (CC[1:0]).

2.2.2 Example

For example, to generate a PWM signal with the same characteristics as the example for the ST7262 (see Section 2.1.1):

F_{cpu} = 8 Mhz, So T_{Timer} = 0.5 µs

High Level period = 500 µs

= 1000 * T_{Timer}

= (3E8h) * T_{Timer}

Low Level period = 200 µs

= 400 * T_{Timer}

= (190h) * T_{Timer}

PWM signal full period = 700 µs

= (578h) * T_{Timer}

So we can calculate the following values:

OC2R = 578h - 5h = 573h

OC1R = 3E8h - 5h = 3E3h

OLVL1 = 1 & OLVL2 = 0

“THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.”