

## Timer Function

### 1 Objectives

The objective of this laboratory exercise is to explore the timer system of the MRK board.

### 2 Theoretical Background

To time the duration of an event, you need a periodic signal of a known frequency and a way to keep track of the number of periods that elapsed during the event. On a wristwatch, the second hand moves at a frequency of 1 Hz, and the marks around the watch face count between 0 and 59 seconds. At the beginning and end of an event, you would note the count indicated by the second hand and then subtract the end count from the beginning count to find the elapsed time. As long as the event lasts less than 60 seconds, a second hand is sufficient. But for events longer than 60 seconds, you would also need to keep track of how many times the second hand rolled over from 59 seconds to 0 seconds.

In similar fashion, a microcontroller can time the duration of an event by using a clock signal of a known frequency in conjunction with a memory location which stores the number of clock ticks. The MRK board uses a clock frequency of 24 MHz, which can be divided down using a *prescaler* according to the following formula

$$\text{timer frequency} = \frac{24 \times 10^6}{2^{\text{prescaler}}} \text{ Hz} \quad (1)$$

The running count of clock ticks is stored as a 16-bit number. With 16 bits, it can count between 0 and 65535 ( $2^{16}-1$ ). When the counter is at 65535 and another clock tick causes it to increment, the counter rolls over to 0. We call this a *timer overflow*. As long as an event lasts less than one overflow period (less than 65535 counts), then the duration is simply the end count subtracted from the initial count. Even if an event started at 64536 and ended at 500, the duration can be calculated as simply  $\text{end} - \text{start} = 1500$ , if we remember these are 16-bit *unsigned numbers*. Here is the calculation done by hand:

$$\begin{array}{r} 0x01F4 \quad (500) \\ - 0xFC18 \quad (64536) \\ \hline \end{array} \quad \Rightarrow \quad \begin{array}{r} 0x01F4 \\ + 0x03E8 \quad (\text{two's complement}) \\ \hline 0x05DC \quad (1500) \end{array}$$

The timer system has eight channels, and each of these channels has two modes of operation: input capture and output compare. In addition, the timer system has two pulse accumulators. Input capture allows you to capture the time of an event. Given a digital input on one of the timer channels, you can configure the channel so that an event is a rising edge (0 to 1), a falling edge (1 to 0), or either. When the event occurs, the current value of the timer is saved and an interrupt is generated. The output compare feature allows you to schedule an event to occur in the future. You specify a timer value, and when the timer equals that value, an interrupt is generated. With output compare, you also have the option of generating an output signal on the timer channel.

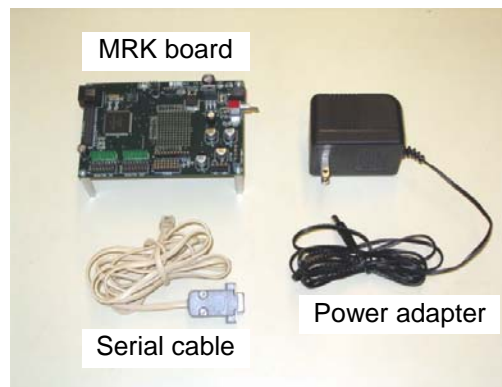
If you wanted to count pulses on an input signal, you could configure the input capture system to increment a variable in the interrupt service routine. However, this sort of task can be done more efficiently in hardware by using the pulse accumulators. There are two pulse accumulators, PACA and PACB. Each accumulator holds a 16-bit number indicating the number of pulses that have been counted. PACA is connected to timer channel 7 and PACB is connected to timer channel 0.

The real-time interrupt is a periodic interrupt which is separate from the timer system. It is ideal for any application where the rate is important, such as data acquisition and digital control. The period is controlled by two parameters according to Equation (2).

$$T = 0.256 \times 2^{\text{power}} \times \text{multiplier} \quad \text{milliseconds} \quad (2)$$

A table of all possible periods is given in the MRK User Manual, Table 11.

### 3 Equipment



- MRK board
- Serial communication cable
- Power adapter
- Function generator

## 4 Procedures

### 4.1 Enabling the Timer

You must enable the timer and set the prescaler before using it. The command is `setup_timer(ENABLE, prescaler)` where *prescaler* is an integer between 0 and 7. The prescaler sets the timer frequency according to Equation (1). The command to disable the timer is `setup_timer(DISABLE)`. The value of the timer can be obtained by using the symbol `TIMER`.

Let's use the timer to determine the accuracy of the `delay()` function. Enter the following program and complete Exercise 1.

```
#include "MRK.h"

int main() {
    unsigned int start, end;
    setup_timer(ENABLE, 2);

    while (1) {
        start = TIMER;
        delay(1);
        end = TIMER;
        fprintf(SCI0, "start = %u, end = %u,", start, end);
        fprintf(SCI0, " end - start = %u\r\n", end-start);
    }
}
```

#### Exercise 1:

What is the timer frequency?

What is the period?

What is the typical value of the difference `end-start`?

How much time did the command `delay(1)` take?

Look for a case where `end < start`. Is the timer difference the same?

### 4.2 Output Compare

To use output compare, you must first enable the timer. Then, you configure a specific channel for output compare by using the command `setup_timer(OUTPUT_COMPARE, channel, action, &isr)`. The parameters are defined below.

*channel* – The timer channel (0-7) being configured

*action* – Action taken on the timer channel output

action	description
NOTHING	no action taken on timer channel output
TOGGLE	change to opposite logic level
CLEAR	change to logical 0
SET	change to logical 1
DISABLE	disable output compare for this channel

*&isr* – Function name of the interrupt service routine. Setting to 0 disables the interrupt.

The following program produces a periodic signal with three 100 ms pulses.

```
#include "MRK.h"

void toc2_isr(void) {
    static int n = 0;
    static char state = LOW;
    if (n == 5) {
        TIME2 = TIMER + 56250; // next OC in 300ms
        n = 0;
    }
    else {
        TIME2 = TIMER + 18750; // next OC in 100ms
        n++; // increment
    }
    state = ~state; // toggle
    digital_out(OUT,0,state);
}

int main() {
    digital_out(OUTPUT,ALL);
    setup_timer(ENABLE, 7); // 187.5 kHz timer
    setup_timer(OUTPUT_COMPARE, 2, TOGGLE, &toc2_isr);

    TIME2 = TIMER + 18750; // next OC will occur in 100ms
    while (1); // loop forever
}
```

The keyword `static` in a variable declaration causes the variable to maintain its value between function calls. The same output will appear on both the digital output port and on timer channel 2 (the timer port connections are next to the digital output port).

### 4.3 Input Capture

To use input capture, you must first enable the timer. Then, you configure a specific channel for input capture by using the command `setup_timer(INPUT_CAPTURE, channel, event, &isr)`. The parameters are defined below.

*channel* – The timer channel (0-7) being configured

*event* – Type of event to watch for

event	description
RISING	rising edge (0 to 1 transition)
FALLING	falling edge (1 to 0 transition)
ANY	all edges
DISABLE	disable input capture for this channel

`&isr` – Function name of the interrupt service routine. Setting to 0 disables the interrupt.

The following program illustrates input capture.

```
#include "MRK.h"

unsigned int event, print_flag;

void tic2_isr(void) {
    event = TIME2;
    print_flag = 1;
}

int main() {
    setup_timer(ENABLE, 0); // 24.0 MHz timer
    setup_timer(INPUT_CAPTURE, 2, RISING, &tic2_isr);

    print_flag = 0;
    while (1) {
        if (print_flag == 1) {
            fprintf(SCI0, "Event occurred at %u\n\r", event);
            print_flag = 0;
        }
    }
}
```

**Exercise 2:** Modify the previous input capture example so that it will calculate the frequency of a square wave input (leave the prescaler at 0). Use a signal generator to test it. Note: the MRK board is not configured for floating point math, so an operation like  $2 / 5$  will equal 0. You will need to avoid intermediate results that are less than zero or have fractional parts. Include the program in your lab report and indicate the usable frequency range. If you wanted to measure low frequencies (less than 100 Hz) how would you change your program? If you wanted to measure high frequencies (greater than 100 kHz) how would you change your program?

#### 4.4 Timer Overflow Interrupt

The timer system can be configured so that an interrupt occurs whenever the timer overflows. To configure this interrupt, use the command `setup_timer(OVERFLOW, &isr)`. The parameter `isr` is the function name of the interrupt service routine.

**Exercise 3:** Write a program which uses the timer overflow interrupt. Have the interrupt service return increment a variable and display the variable on the digital output port. Don't

forget to enable the timer and configure `digital_out`. When you enable the timer, choose the frequency so that you are able to see each increment. Include the program in your lab report.

#### 4.5 Pulse Accumulators

To configure a pulse accumulator, use the command `setup_pac(accumulator, edge, &isr)`. The parameters are defined below.

*accumulator* – either A or B

*edge* – specify which edge to count: RISING, FALLING, ANY

*isr* – function name of interrupt service routine. Set to 0 to disable interrupt.

The symbols PACA and PACB hold the count value.

#### 4.6 Real-Time Interrupt

To configure the real-time interrupt, use the command `setup_rti(power, multiplier, &isr)`, where *power* and *multiplier* are defined in Equation (2) and *isr* is the function name of the interrupt service routine.

Exercise 4: Write a program which uses a pulse accumulator and the real-time interrupt to calculate the frequency of a square wave input. Choose *power* and *multiplier* so that the usable frequency range is similar to what you obtained in Exercise 2. Use a signal generator to test it. Include the program in your lab report and indicate the usable frequency range.

## 5 Report and analysis requirements

### 5.1 Theory

On the MC19S12 block diagram, identify components enabling operation of the timer. Describe how input capture and output compare routines work.

### 5.2 Results and analysis

Present your results.

Explain how the output compare program example works.

List all programs you created during laboratory exercises. Explain how your programs work. Provide line-by-line comments for each program.

Answer all questions in previous sections of these laboratory instructions.

Compare your two programs which calculate the frequency of a square wave. Which one had the better frequency resolution?

*Last updated  
September 27 2006.*