ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

General Purpose Timer Modules

Experiment 4 - General Purpose Timer Modules

Objectives

In Experiment 3, you used SysTick to address Real-Time problems. SysTick is a single system clock whose only function is to count up or down for a pre-defined number of cycles. If you need multiple clocks, with more sophisticated capabilities such as reading an input signal or generating an output signal you can use the Timer Modules of TM4C. While doing this lab, you will:

1. Learn the different functions and modes of Timer modules
2. Learn how to configure the registers to use Timer modules
3. Write an interrupt subroutine to generate a pulse train
4. Write a program to read an input signal in Edge Time mode

1 Background Information

1.1 TM4C Timer Modules

The board is equipped with six 16/32 bit General Purpose Timer Modules named TIMERn (They are called as GPTM in the datasheet, which refer to the same thing). Each timer module consists of two 16 bit timers (A and B). They can be used seperately as two 16-bit counters or together as a 32-bit counter. In this lab, we will use them seperately as 16-bit counters.

TM4C Timer Modules can be used for various purposes. In One shot/Periodic modes the timer simply counts up or down, once or in a periodic way. You can use these modes to generate a pulse or pulse wave. In Edge Count mode, the timer can be used to count the number of falling/rising edges of an input signal and in Edge Time mode it can be used to measure the time between falling/rising edges of an input signal. In other words, Edge Time and Edge Count modes can be used to measure the duty cycle, period and frequency of a signal (You have to assign a timer to a GPIO port for these two modes). RTC, PWM, Wait-for-Trigger are other timer modes which will not be covered here. In this experiment, you will generate a square pulse train using Periodic Mode and read this as an input signal using Edge Time Mode.

1.1.1 One Shot / Periodic Mode

In this mode, the timer will start counting up from 0 to a specified value, or down from a specified value to 0. Counting starts once the timer is enabled from the timer's Control Register (TIMERn_CTL). If configured to count up, the timer starts at 0 and counts up to the value stored in the Interval Load Register (TIMERn_ILR). When that value is reached, a "Time-Out" event is triggered which sets the timeout flag in the Raw Interrupt Status Register (TIMERn_RIS). In One Shot mode, the timer is disabled at this moment. In Periodic Mode, timer starts counting again from 0. But the interrupt flag needs to be cleared after timeout event by using the Interrupt Clear Register (TIMERn_ICR), so that the next timeout event can trigger the interrupt flag.

The value of the timer count can be read any time from Timer Register (TIMERn_TAR for Timer A and TIMERn_TBR for Timer B).

1.1.2 Input Edge Time / Edge Count Mode

These are called as Capture Modes.

Edge Time Mode: Similar to the previous mode, the timer again starts counting once the timer is enabled, and counts up to/down from the specified value. However, it stores the current value of the timer to TIMERn_TAR **only when** an edge of a pulse is detected in the timer's port. When a rising and/or a falling edge is detected, the capture flag in the TIMERn_RIS is set. The timer keeps on counting, but the capture interrupt flag has to be reset from TIMERn_ICR so that the timer value can be written into TIMERn_TAR in the next edge.

Edge Count Mode: The timer counts only when an edge is detected. Every time an edge is detected, the counter is increased and the value is stored in TIMERn_TAR. (Again the interrupt flag is set, has to be reset etc.).

Timeout interrupt flag works in the same way as in One-Shot and Periodic modes.

REMARK: Please note that in all modes the interrupt flags in the Raw Interrupt Status Register are set even though the interrupt is masked, i.e. when no ISR is used. If the interrupt is enabled by setting the related bit in Interrupt Mask Register (TIMERn_IMR), then an ISR can be employed.

1.2 GPIO Setup for Timers

Each timer can be physically accessed through one of the GPIO pins, in which case, the GPIO ports will have to be configured accordingly. Figure 2 on the next page shows the ports assigned to each Timer module. TIMER0A for example, can be directly accessed through PB6 and PF0. To access TIMER0A from, let's say PB6, the alternate function of PB6 has to be chosen as 7, which is the timer function. Section 6 (Pin 6) of GPIO_PORTB_PCTL is set to 7 to choose this alternate function.

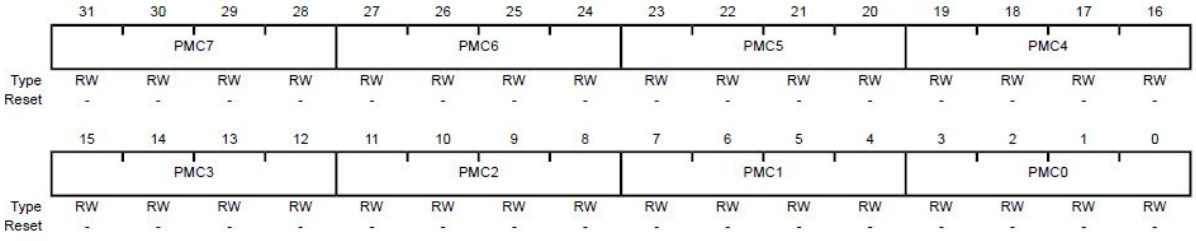

Figure 1: GPIO Port Control Register

The configuration of PB6 to access timer can be done with the following piece of code. Keep in mind that in addition to the given code, you also have to configure additional registers to power up the GPIO clock for port, set the direction to choose the pin as input and digitally enable the pin, as you have learned in Experiment 4.

```
;GPIO Registers
;Port B base 0x40005000
GPIO_PORTB_AFSEL        EQU 0x40005420 ; Alt function enable
GPIO_PORTB_AMSEL        EQU 0x40005528 ; Analog enable
GPIO_PORTB_PCTL         EQU 0x4000552C ; Choose alternate functions

; Set bit6 for alternate function on PB6
        LDR     R1, =GPIO_PORTB_AFSEL
        LDR     R0, [R1]
        ORR     R0, R0, #0x40
        STR     R0, [R1]

; Set bits 27:24 of PCTL to 7 to enable TIMER0A on PB6
        LDR     R1, =GPIO_PORTB_PCTL
        LDR     R0, [R1]
        ORR     R0, R0, #0x07000000
        STR     R0, [R1]

; clear AMSEL to diable analog
        LDR     R1, =GPIO_PORTB_AMSEL
        MOV     R0, #0
        STR     R0, [R1]
```

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
T0CCP0	1 28	PB6 (7) PF0 (7)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 0.
T0CCP1	4 29	PB7 (7) PF1 (7)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 1.
T1CCP0	30 58	PF2 (7) PB4 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 0.
T1CCP1	31 57	PF3 (7) PB5 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 1.
T2CCP0	5 45	PF4 (7) PB0 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 0.
T2CCP1	46	PB1 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 1.
T3CCP0	47	PB2 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 0.
T3CCP1	48	PB3 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 1.
T4CCP0	52	PC0 (7)	I/O	TTL	16/32-Bit Timer 4 Capture/Compare/PWM 0.
T4CCP1	51	PC1 (7)	I/O	TTL	16/32-Bit Timer 4 Capture/Compare/PWM 1.
T5CCP0	50	PC2 (7)	I/O	TTL	16/32-Bit Timer 5 Capture/Compare/PWM 0.
T5CCP1	49	PC3 (7)	I/O	TTL	16/32-Bit Timer 5 Capture/Compare/PWM 1.

Figure 2: List of associated pins for 16/32 bit timers

1.3 Timer Setup

Similar to GPIO, in order to use the timer peripheral you must first "power it up" by starting its clock using the GPTM Run Mode Clock Gating Control (RCGCTIMER). Setting bits [5:0] enables the corresponding timer module. After you have powered up a timer, you have to wait for a few cycles for the clock to settle. The following list shows the RCGCTIMER register address, the base register address of each timer and the offsets of timer registers.

```
;Timer Clock Register Address
SYSCTL_RCGCTIMER        EQU     0x400FE604

;Base Addresses of 16/32 Bit Timers
TIMER0  EQU     0x40030000
TIMER1  EQU     0x40031000
TIMER2  EQU     0x40032000
TIMER3  EQU     0x40033000
TIMER4  EQU     0x40034000
TIMER5  EQU     0x40035000

;16/32 Bit Timer Register Offsets
_CFG    EQU     0x000 ; Timer Config
_TAMR   EQU     0x004 ; TimerA Mode
_CTL    EQU     0x00C ; Timer Control
_IMR    EQU     0x018 ; Timer Interrupt Mask
_RIS    EQU     0x01C ; Timer Raw Interrupt Status
_ICR    EQU     0x024 ; Timer Interrupt Clear
_TAILR  EQU     0x028 ; TimerA Interval Load
_TAPR   EQU     0x038 ; TimerA Prescale
_TAR    EQU     0x048 ; TimerA
```

Control Register (_CTL): This register is used to disable/enable the timer and to set which edges (falling, rising or both) will be detected from input signal. Both timer A and B can be controlled with this register.

- TAEN [0]: Set this bit to enable timer A
- TASTALL [1]: Set this bit to make timer A freeze counting when the processor is halted by the debugger.
- TAEVENT [3:2]: The edge detection condition bits for timer A.

0	Positive edge
1	Negative edge
3	Both edges

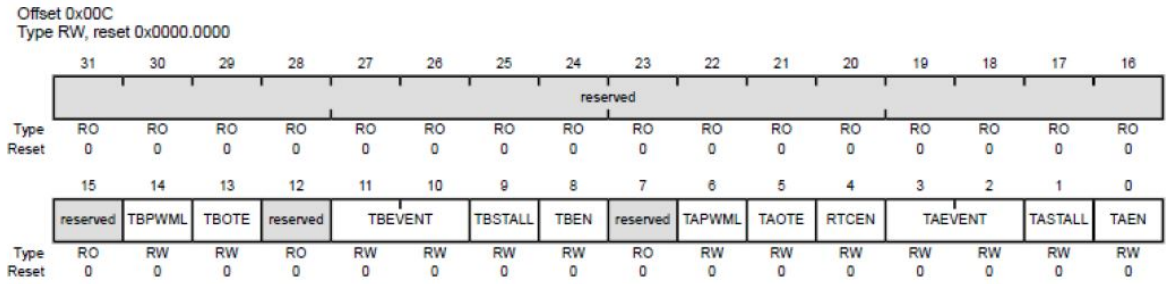

Figure 3: Control Register

REMARK: You should disable the timer before doing any configuration with the timer registers.

Configuration Register (_CFG): Selects the configuration of the timers A and B (16 bit or 32 bit mode). Set bits [2:0] to 0x4 for 16 bit mode.

Interrupt Mask Register (_IMR): Used to enable timer interrupts. If the related bits are set, an interrupt from the timer will make the processor enter an ISR. You don't have to write anything yourself to this register for this lab.

Raw Interrupt Status Register (_RIS): When an interrupt occurs in the timer, the related bits of _RIS register are set to 1 to. These flags are set even though the interrupt is not enabled from _IMR.

- TATORIS [0]: This bit being 1 means a Timer A timeout interrupt has occured (Timer A counted up to the specified value or counted down to 0).
- CAERIS [2]: This bit being 1 means a capture mode event has occured for Timer A (An edge is detected).

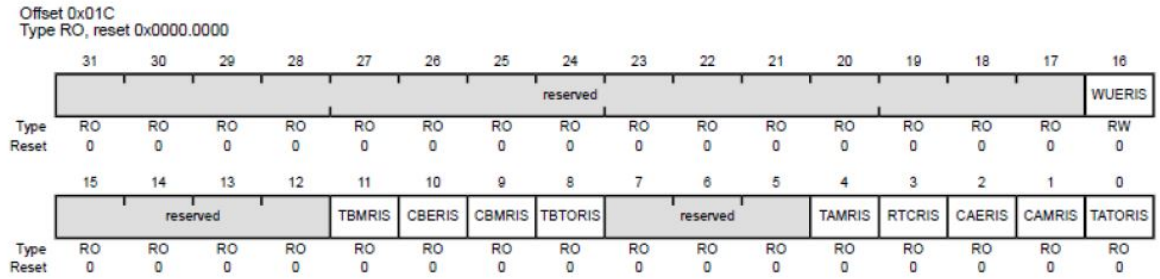

Figure 4: Raw Interrupt Status Register

Interrupt Clear Register (_ICR): Used to clean the interrupt flags in _RIS.

- TATOCINT [0]: Writing 1 to this bit clears the TATORIS bit in the _RIS register
- CAECINT [2]: Writing 1 to this bit clears the CAERIS bit in the _RIS register.

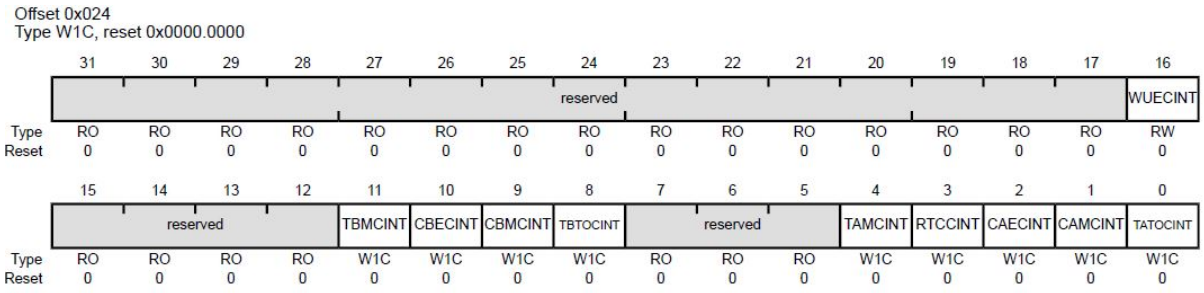

Figure 5: Interrupt Clear Register

Timer A Interval Load Register (_TAILR): In 16 bit mode, the value written to lower [15:0] bits of this register specifies the upper bound for count up and the starting value for count down modes.

Timer A Mode Register (_TAMR): Used to set the function of the timer.

- TAMR [1:0]: One-shot, Periodic or Capture Mode

1	One-Shot
2	Periodic
3	Capture

- TACMR [2]: Type of Capture mode

0	Edge-Count
1	Edge-Time

- TACDIR [4]:

0	Count-Down
1	Count-Up

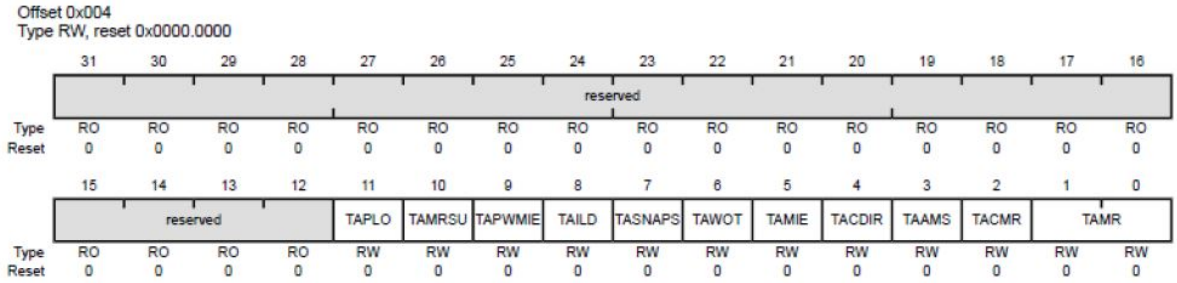

Figure 6: Timer A Mode Register

Timer A Register (_TAR): Shows the current value of the timer in One-Shot and Periodic modes. The lower bits [15:0] has the timer value and the upper bits are zero.

In Edge-Time Mode, _TAR shows the timer value from the last edge-detection event.

In Edge-Count Mode, _TAR shows the number of edge-detection events.

REMARK: In capture modes, the timer becomes a 24 bit timer, i.e. the timer value will be equal to [23:0] of _TAR. This will be further explained in the next section.

1.3.1 Prescaling

The timer modules in TM4C use 16 MHz clock. In other words, the timer is increased by 1 every 62.5 nanoseconds. This means that a timeout will occur after 4.096 ms, assuming that the counting range is set to maximum (0xFFFF). Sometimes this number can be too small when you deal with events or signals with lower frequencies. For this purpose, the prescale register _TAPR can be used to decrease the timer frequency. The new timer frequency is 16 MHz divided by the value written to _TAPR +1.

$$f_{timer} = \frac{16 \text{ MHz}}{\text{Prescale} + 1}$$

If you want to have a timer with a frequency of 4 MHz, for example, you will need to write 0x03 to lower [7:0] bits of _TAPR.

However, you have to consider that the prescaling works in two ways. In One-Shot and Periodic modes with Count-Down, it works as a true prescaler. The prescaler will count for 3 more cycles before the timer counts down one tick. In this case, for a timer frequency of 4 MHz, after 250 ns you will read 0x0001 from _TAR . In other modes it functions as a time extender. For Edge-Time mode, for instance, with the same prescaling factor of 4, you wil read 0x0004 after 250 ns (Assuming an edge is detected

at this point). Seems like the prescaling didn't work? No, it works but in another way. This time, the upper range of the timer will be 0x03.FFFF (Upper 8 bit part comes from the prescale value _TAPR), which means that the timeout will occur four times later. The value read by the _TAR hasn't changed but the maximum number of timer counts has increased, which is as well useful for the same purpose of extending the duration of the timer. Figure 7 below shows the method of prescaling for different modes.

Mode	Timer Use	Count Direction	Counter Size		Prescaler Size[a]		Prescaler Behavior (Count Direction)
			16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
One-shot	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
Periodic	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
RTC	Concatenated	Up	32-bit	64-bit	-	-	N/A
Edge Count	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
Edge Time	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
PWM	Individual	Down	16-bit	32-bit	8-bit	16-bit	Timer Extension

Figure 7: Timer Capabilities

2 Preliminary Work

1. (30%) *Pulse.s* (You can find it on ODTUCLASS) is a program to generate a square pulse train with 20 kHz frequency. But the code is not complete, so you have to write the missing parts.

   You are given the PULSE_INIT subroutine, which carries out all the neccessary configurations, enables PF2 to output a digital value and enables TIMER0A to start counting. Read this code carefully. In which mode is TIMER0A used?

   Please note that the TIMER0A interrupt is enabled from _IMR, which means that every time an interrupt is generated from TIMER0A, it won't be masked by the interrupt system and your program should enter an ISR. **Your task is to write this My_Timer0A_Handler ISR**. In other words, you will write what has to be done after each interrupt. Make sure that you IMPORT your interrupt subroutine to *Startup.s* as you did for My_SysTick_Handler in Experiment 5.

   You are not supposed to change anything in PULSE_INIT subroutine.

   You will need to change HIGH and LOW values, representing HIGH and LOW durations of your signal, respectively. You are supposed to pick the duty cycle as 60%.

2. (70%) Now you will write a program which reads an input signal from an external pin and measures its pulse width, period and duty cycle. The measurements are to be printed out on Terminal.

   The input will be the very same signal you generated in Step 1. Thus, in your program, you will have to call this subroutine to start generating a a pulse train from PF2.

   You are free to chose any TIMER and GPIO module, but you must be careful not to cause any conflicts.

   This time, you won't enable the interrupt generated by the TIMER you use to read input signal. You will rather poll the related flags in the corresponding registers.

3 Experimental Work

1. (30%) Build and run the pulse train generator program. You can observe the generated signal from the scope. Show your work to TA and answer the questions.
2. (70%) Build and run the program in Step 2. Show your work to TA and answer the questions.

References

[1] TI, "Tiva tm4c123gh6pm microcontroller data sheet." `http://www.ti.com/lit/ds/spms376e/spms376e.pdf`.

```
; Pulse.s
; Routine for creating a pulse train using interrupts
; This uses Channel 0, and a 1MHz Timer Clock (_TAPR = 15 )
; Uses Timer0A to create pulse train on PF2

;Nested Vector Interrupt Controller registers
NVIC_EN0_INT19          EQU 0x00080000 ; Interrupt 19 enable
NVIC_EN0                EQU 0xE000E100 ; IRQ 0 to 31 Set Enable Register
NVIC_PRI4               EQU 0xE000E410 ; IRQ 16 to 19 Priority Register

; 16/32 Timer Registers
TIMER0_CFG              EQU 0x40030000
TIMER0_TAMR             EQU 0x40030004
TIMER0_CTL              EQU 0x4003000C
TIMER0_IMR              EQU 0x40030018
TIMER0_RIS              EQU 0x4003001C ; Timer Interrupt Status
TIMER0_ICR              EQU 0x40030024 ; Timer Interrupt Clear
TIMER0_TAILR            EQU 0x40030028 ; Timer interval
TIMER0_TAPR             EQU 0x40030038
TIMER0_TAR              EQU 0x40030048 ; Timer register

;GPIO Registers
GPIO_PORTF_DATA         EQU 0x40025010 ; Access BIT2
GPIO_PORTF_DIR          EQU 0x40025400 ; Port Direction
GPIO_PORTF_AFSEL        EQU 0x40025420 ; Alt Function enable
GPIO_PORTF_DEN          EQU 0x4002551C ; Digital Enable
GPIO_PORTF_AMSEL        EQU 0x40025528 ; Analog enable
GPIO_PORTF_PCTL         EQU 0x4002552C ; Alternate Functions

;System Registers
SYSCTL_RCGCGPIO         EQU 0x400FE608 ; GPIO Gate Control
SYSCTL_RCGCTIMER        EQU 0x400FE604 ; GPTM Gate Control

;-------------------------------------------------------
LOW                     EQU     0x00000100
HIGH                    EQU     0x00000100
;-------------------------------------------------------

                        AREA    routines, CODE, READONLY
                        THUMB
                        EXPORT  My_Timer0A_Handler
                        EXPORT  PULSE_INIT

;-------------------------------------------------------
My_Timer0A_Handler      PROC
                        ;...
                        BX      LR
                        ENDP
;-------------------------------------------------------

PULSE_INIT      PROC
                LDR R1, =SYSCTL_RCGCGPIO ; start GPIO clock
                LDR R0, [R1]
                ORR R0, R0, #0x20 ; set bit 5 for port F
                STR R0, [R1]
                NOP ; allow clock to settle
```

```
                NOP
                NOP
                LDR R1, =GPIO_PORTF_DIR ; set direction of PF2
                LDR R0, [R1]
                ORR R0, R0, #0x04 ; set bit2 for output
                STR R0, [R1]
                LDR R1, =GPIO_PORTF_AFSEL ; regular port function
                LDR R0, [R1]
                BIC R0, R0, #0x04
                STR R0, [R1]
                LDR R1, =GPIO_PORTF_PCTL ; no alternate function
                LDR R0, [R1]
                BIC R0, R0, #0x00000F00
                STR R0, [R1]
                LDR R1, =GPIO_PORTF_AMSEL ; disable analog
                MOV R0, #0
                STR R0, [R1]
                LDR R1, =GPIO_PORTF_DEN ; enable port digital
                LDR R0, [R1]
                ORR R0, R0, #0x04
                STR R0, [R1]

                LDR R1, =SYSCTL_RCGCTIMER ; Start Timer0
                LDR R2, [R1]
                ORR R2, R2, #0x01
                STR R2, [R1]
                NOP ; allow clock to settle
                NOP
                NOP
                LDR R1, =TIMER0_CTL ; disable timer during setup LDR R2, [R1]
                BIC R2, R2, #0x01
                STR R2, [R1]
                LDR R1, =TIMER0_CFG ; set 16 bit mode
                MOV R2, #0x04
                STR R2, [R1]
                LDR R1, =TIMER0_TAMR
                MOV R2, #0x02 ; set to periodic, count down
                STR R2, [R1]
                LDR R1, =TIMER0_TAILR ; initialize match clocks
                LDR R2, =LOW
                STR R2, [R1]
                LDR R1, =TIMER0_TAPR
                MOV R2, #15 ; divide clock by 16 to
                STR R2, [R1] ; get 1us clocks
                LDR R1, =TIMER0_IMR ; enable timeout interrupt
                MOV R2, #0x01
                STR R2, [R1]
; Configure interrupt priorities
; Timer0A is interrupt #19.
; Interrupts 16-19 are handled by NVIC register PRI4.
; Interrupt 19 is controlled by bits 31:29 of PRI4.
; set NVIC interrupt 19 to priority 2
                LDR R1, =NVIC_PRI4
                LDR R2, [R1]
                AND R2, R2, #0x00FFFFF ; clear interrupt 19 priority
                ORR R2, R2, #0x40000000 ; set interrupt 19 priority to 2
```

```
                STR R2, [R1]
; NVIC has to be enabled
; Interrupts 0-31 are handled by NVIC register EN0
; Interrupt 19 is controlled by bit 19
; enable interrupt 19 in NVIC
                LDR R1, =NVIC_EN0
                MOVT R2, #0x08 ; set bit 19 to enable interrupt 19
                STR R2, [R1]
; Enable timer
                LDR R1, =TIMER0_CTL
                LDR R2, [R1]
                ORR R2, R2, #0x03 ; set bit0 to enable
                STR R2, [R1] ; and bit 1 to stall on debug
                BX LR ; return
                ENDP
                END
```