COMP551 – Interfacing

Spring 2017

Lab 4

Programming PIC18 Timers

Instructor: T. Fragomeli

Students:				
Student ID's:				
Section:	**01**	**02**	**03**	**04**

NOTE: Labs are due at the start of the next lab period. Only submit one lab per group of two students.

Lab 4 – Programming PIC18 Timers

4. Introduction:

The Microchip PIC18 has two to five timers depending on the PIC family member. They are referred to as Timers 0, 1, 2, 3 and 4. They can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller.

Every timer needs some sort of clock pulse to tick. The clock source can be internal or external. If the internal clock source is used, then 1/4th of the frequency of the crystal oscillator on the OSC1 and OSC2 pins (Fosc/4) is fed into the timer. Therefore, it is used for time delay generation and it is called a timer. By choosing the external clock option, pulses are fed through one of the PIC 18's pins. This is called a counter.

4.1 Registers and Programming Timers 0, 1, 2 and 3

4.1.1 Basic Registers of the Timers

Many of the PIC 18 timers are 16 bits wide. Because the PIC 18 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte (TMRxL) and high byte (TMRxH). Each timer also has the TCON (Timer control) register for setting modes of operation.

4.1.2 Timer0 Programming

Timer0 can be used as an 8-bit or 16 bit timer. The 16-bit register of Timer0 is accessed as low byte and high byte, as shown in Figure 4-1. The low byte register is called TMR0L (Timer0 Low byte) and the high byte register is referred to as TMR0H (Timer0 high byte). See Figure 4-2 for T0CON register options.

FIGURE 4-1: Timer0 High and Low Register

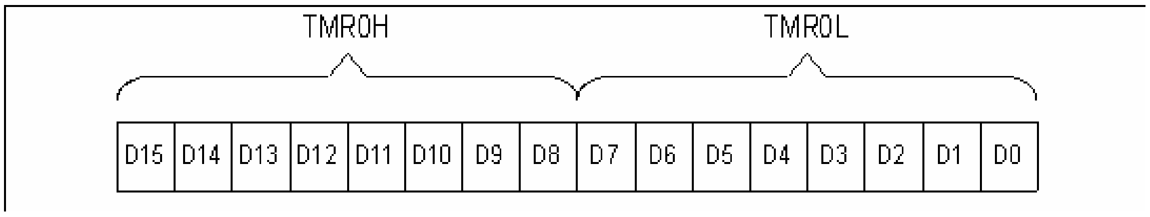
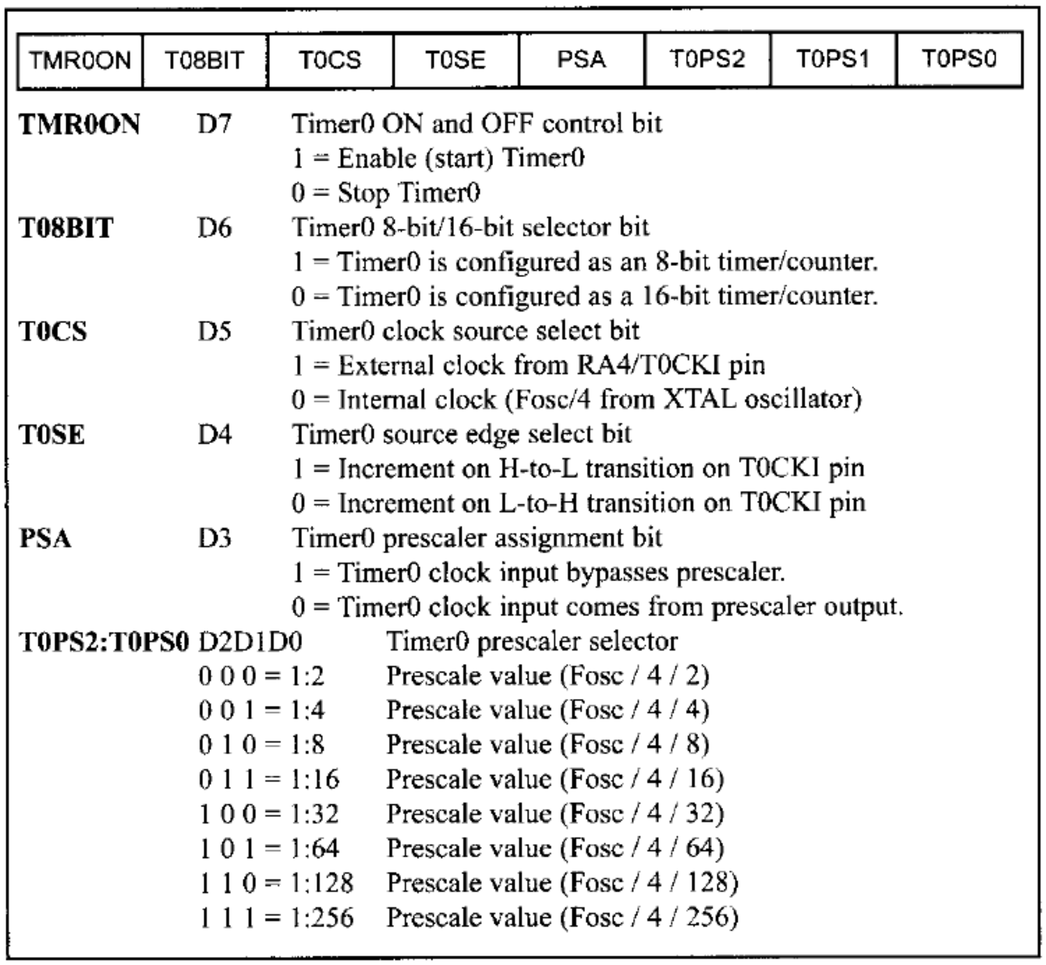


FIGURE 4-2: T0CON (Timer 0 Control) Register



TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON	D7	Timer0 ON and OFF control bit
		1 = Enable (start) Timer0
		0 = Stop Timer0
T08BIT	D6	Timer0 8-bit/16-bit selector bit
		1 = Timer0 is configured as an 8-bit timer/counter.
		0 = Timer0 is configured as a 16-bit timer/counter.
T0CS	D5	Timer0 clock source select bit
		1 = External clock from RA4/T0CKI pin
		0 = Internal clock (Fosc/4 from XTAL oscillator)
T0SE	D4	Timer0 source edge select bit
		1 = Increment on H-to-L transition on T0CKI pin
		0 = Increment on L-to-H transition on T0CKI pin
PSA	D3	Timer0 prescaler assignment bit
		1 = Timer0 clock input bypasses prescaler.
		0 = Timer0 clock input comes from prescaler output.
T0PS2:T0PS0	D2D1D0	Timer0 prescaler selector
	0 0 0 = 1:2	Prescale value (Fosc / 4 / 2)
	0 0 1 = 1:4	Prescale value (Fosc / 4 / 4)
	0 1 0 = 1:8	Prescale value (Fosc / 4 / 8)
	0 1 1 = 1:16	Prescale value (Fosc / 4 / 16)
	1 0 0 = 1:32	Prescale value (Fosc / 4 / 32)
	1 0 1 = 1:64	Prescale value (Fosc / 4 / 64)
	1 1 0 = 1:128	Prescale value (Fosc / 4 / 128)
	1 1 1 = 1:256	Prescale value (Fosc / 4 / 256)

4.1.3 16-bit Timer Programming of Timer0

The following are the characteristics and operations of 16-bit mode:

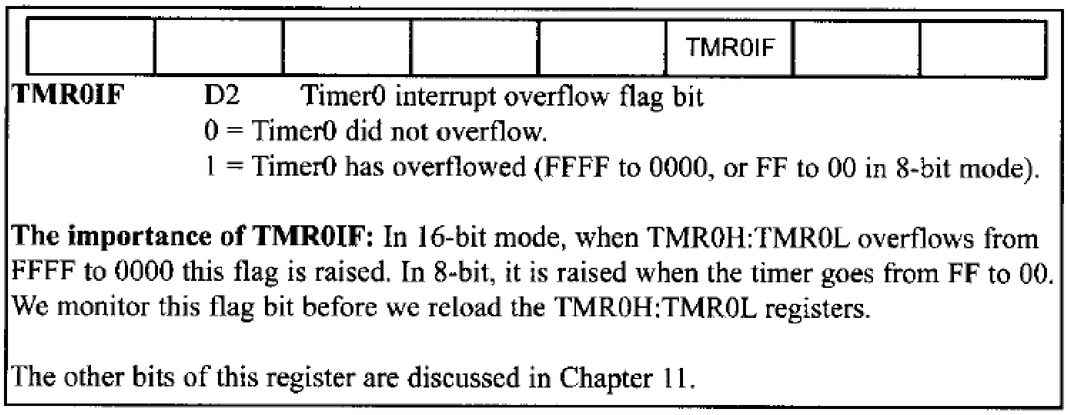
1. It is a 16-bit timer; therefore, it allows values of 0000H to FFFFH to be loaded into the register.

2. After TMR0H and TMR0L are loaded with a 16-bit initial value, the timer must be started. This is done by setting the "TMR0ON" for Timer0.

3. After the timer is started, it rolls over from FFFFH to 0000H, it sets a flag bit called TMR0IF (timer interrupt flag, which is part of the INTCON register, see Figure 4-3) HIGH. This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer.

4. After the timer reaches its limit and rolls over, in order to repeat the process, the registers TMR0H and TMR0L must be reloaded with the original value, and the TMR0IF flag must be reset to 0 for the next round.

FIGURE 4-3: INTCON (Interrupt Control Register) has the TMR0IF Flag



					TMR0IF		

TMR0IF D2 Timer0 interrupt overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (FFFF to 0000, or FF to 00 in 8-bit mode).

The importance of TMR0IF: In 16-bit mode, when TMR0H:TMR0L overflows from FFFF to 0000 this flag is raised. In 8-bit, it is raised when the timer goes from FF to 00. We monitor this flag bit before we reload the TMR0H:TMR0L registers.

The other bits of this register are discussed in Chapter 11.

4.1.4 8-bit Timer Programming of Timer0

Timer0 can also be used in 8-bit mode. The 8-bit mode allows only values of 00H to FFH to be loaded into the timers register TMRL0. After the timer is started, it starts to count up by incrementing the TMR0L register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00H, it sets TMR0IF HIGH.

4.1.5 Timer1 Programming

Timer1 is a 16-bit timer, and its 16-bit register is split into two bytes, referred to as TMR1L (Timer 1 low byte) and TMR1H (Timer1 high byte), see Figure 4-4. Timer1 can be programmed in 16-bit mode only and unlike Timer0, it does not support 8-bit mode. Timer1 also has the T1CON (Timer 1 control) register in addition to the TMR1IF (Timer1 interrupt flag). The TMR1IF flag bit goes HIGH when TMR1H:TMR1L overflows from FFFF to 0000. Timer 1 also has the prescaler option, but it only supports factors of 1:1, 1:2, 1:4 and 1:8. See Figure 4-5 for T1CON register options. The PIR1 register contains the TMR1IF flags, see Figure 4-6.

FIGURE 4-4: Timer1 High and LOW Register

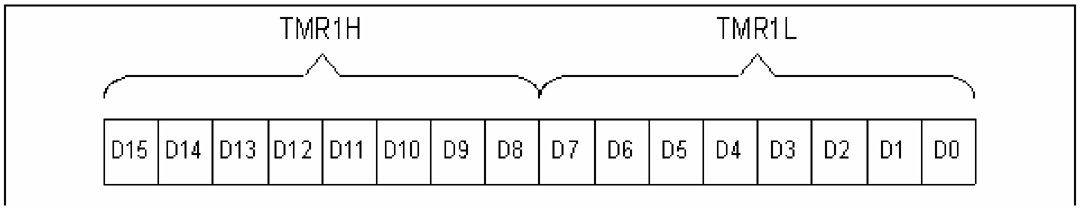
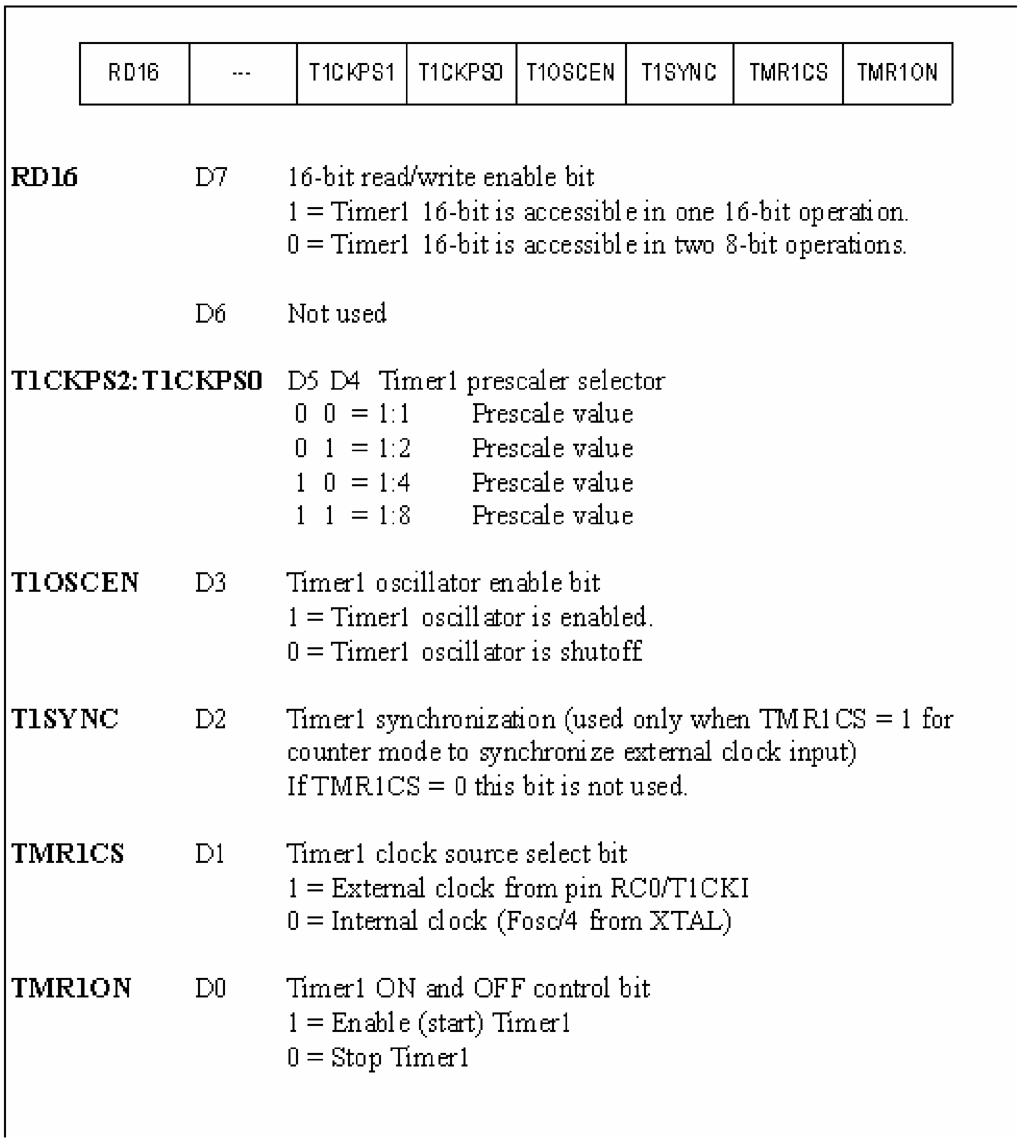


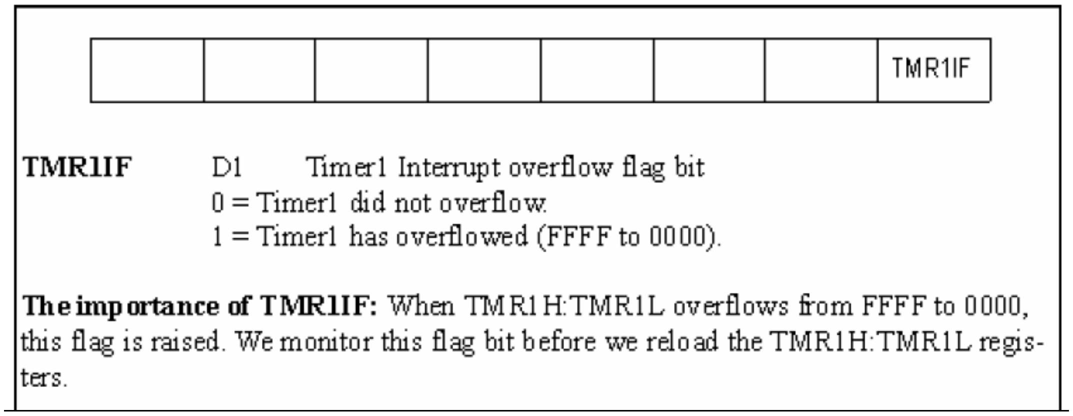
Figure 4-5: T1CON (Timer 1 Control) Register



RD16	---	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

RD16	D7	16-bit read/write enable bit 1 = Timer1 16-bit is accessible in one 16-bit operation. 0 = Timer1 16-bit is accessible in two 8-bit operations.
	D6	Not used
T1CKPS2:T1CKPS0		D5 D4 Timer1 prescaler selector 0 0 = 1:1 Prescale value 0 1 = 1:2 Prescale value 1 0 = 1:4 Prescale value 1 1 = 1:8 Prescale value
T1OSCEN	D3	Timer1 oscillator enable bit 1 = Timer1 oscillator is enabled. 0 = Timer1 oscillator is shutoff
T1SYNC	D2	Timer1 synchronization (used only when TMR1CS = 1 for counter mode to synchronize external clock input) If TMR1CS = 0 this bit is not used.
TMR1CS	D1	Timer1 clock source select bit 1 = External clock from pin RC0/T1CKI 0 = Internal clock (Fosc/4 from XTAL)
TMR1ON	D0	Timer1 ON and OFF control bit 1 = Enable (start) Timer1 0 = Stop Timer1

Figure 4-6: PIR1 (Interrupt Control Register 1) Contains the TMR1IF Flag



TMR1IF D1 Timer1 Interrupt overflow flag bit
0 = Timer1 did not overflow.
1 = Timer1 has overflowed (FFFF to 0000).

The importance of TMR1IF: When TMR1H:TMR1L overflows from FFFF to 0000, this flag is raised. We monitor this flag bit before we reload the TMR1H:TMR1L registers.

4.2 - Procedure for Programming Timers

4.2.1 Steps to program Timer0 in 16-bit mode

To generate a time delay using the Timer0 in 16-bit mode, use the following steps:

1. Load the value into the T0CON register, indicating which mode (8-bit or 16 bit) is to be used and the selected prescaler option.
2. Load register TMR0H followed by register TMR0L with initial count values.
3. Start the timer with the instruction.
4. Keep monitoring the timer flag (TMR0IF) to see if it is raised. Get out of the loop when TMR0IF becomes high.
5. Stop the timer with the proper instruction.
6. Clear the TMR0IF flag for the next round.
7. Go back to step 2 to load TMR0H and TMR0L again.

4.2.2 Steps to program Timer0 in 8-bit mode

To generate a time delay using Timer0 in 8-bit mode, use the following steps:

1. Load the T0CON value register indicating 8-bit mode is selected.
2. Load the TMR0L register with the initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TMR0IF) to see if it is raised. Get out of the loop when TMR0IF goes HIGH.
5. Stop the timer with the proper instruction.
6. Clear the TMR0IF flag for the next round.
7. Go back to step 2 to load TMR0L.

Exercises:

1. Assuming the processor frequency is 10 MHz, find the lowest square wave frequency that can be generated using Timer0 in 16 bit mode. (Calculations only, no code needed).

2. Assuming the processor frequency is 10 MHz, find the highest square wave frequency that can be generated using Timer0 in 16 bit mode. (Calculations only, no code needed).

3. Write a program using Timer0 to generate a square wave with a frequency of 500 Hz on PORTB RB0. Use the MPLAB Logic Analyzer and Stopwatch to examine the frequency of the square wave and modify the count value, if necessary, to make sure that the frequency is as close to 500 Hz as possible.

4. Write a program using Timer1 and the largest prescaler possible to generate a square wave with a frequency of 3 kHz on PORTC RC7. Use the MPLAB Logic Analyzer and Stopwatch to examine the frequency of the square wave and modify the count value, if necessary, to make sure that the frequency is as close to 3 kHz as possible.

5. Write a program using Timer0 to generate a square wave with a frequency of 1 kHz and 2 kHz on PORTB RB0 and PORTB RB4 respectively. PORTB RB7 will be used as a switch to select the frequencies. If RB7 = 0 (Low), the 1 kHz square wave will be generated and if RB7 = 1 (High), a 2 kHz square wave will be generated. Examine the frequencies using the MPLAB Logic Analyzer and Stopwatch. Modify the count value to make sure that the frequencies are as close as possible to 1 kHz and 2 kHz.

Note: Use the Asynchronous Stimulus feature of the MPLAB Simulator to simulate the toggling of the switch connected to RB7. See the next section for information on how to use this feature. The options you need to use have been highlighted.

Tip: To make your analysis easier, use a breakpoint to stop the program after each half cycle of the square wave. Also, when doing your calculations, be sure to note the processor frequency of the PIC you are using, usually 20 MHz by default.

Submit a hardcopy of the source code for each program, calculations, as well as a Logic Analyzer screenshot demonstrating the proper output of each program.

Using Stimulus

During simulation, the program being executed by the simulator may require stimuli from the outside. Stimulus is the simulation of hardware signals/data into the device. This stimulus could be a level change or a pulse to an I/O pin of a port. It could be a change to the values in an SFR (Special Function Register) or other data memory.

In addition, stimulus may need to happen at a certain instruction cycle or time during the simulation. Alternately, stimulus may need to occur when a condition is satisfied; for example, when the execution of program has reached a certain instruction address during simulation.

Basically, there are two types of stimulus:

- **Asynchronous** - A one-time change to the I/O pin or RCREG triggered by a firing button on the stimulus GUI within the MPLAB IDE.
- **Synchronous** - A predefined series of signal/data changes to an I/O pin, SFR or GPR (e.g., a clock cycle).

To define when, what and how external stimuli are to happen to a program, you would use the Stimulus Dialog (see below) tabs to create both asynchronous and synchronous stimulus on a stimulus workbook.

Stimulus Dialog

Use the Stimulus dialog to create asynchronous or synchronous stimuli. The Stimulus dialog allows you to enter stimulus information which is saved in a file called a workbook.

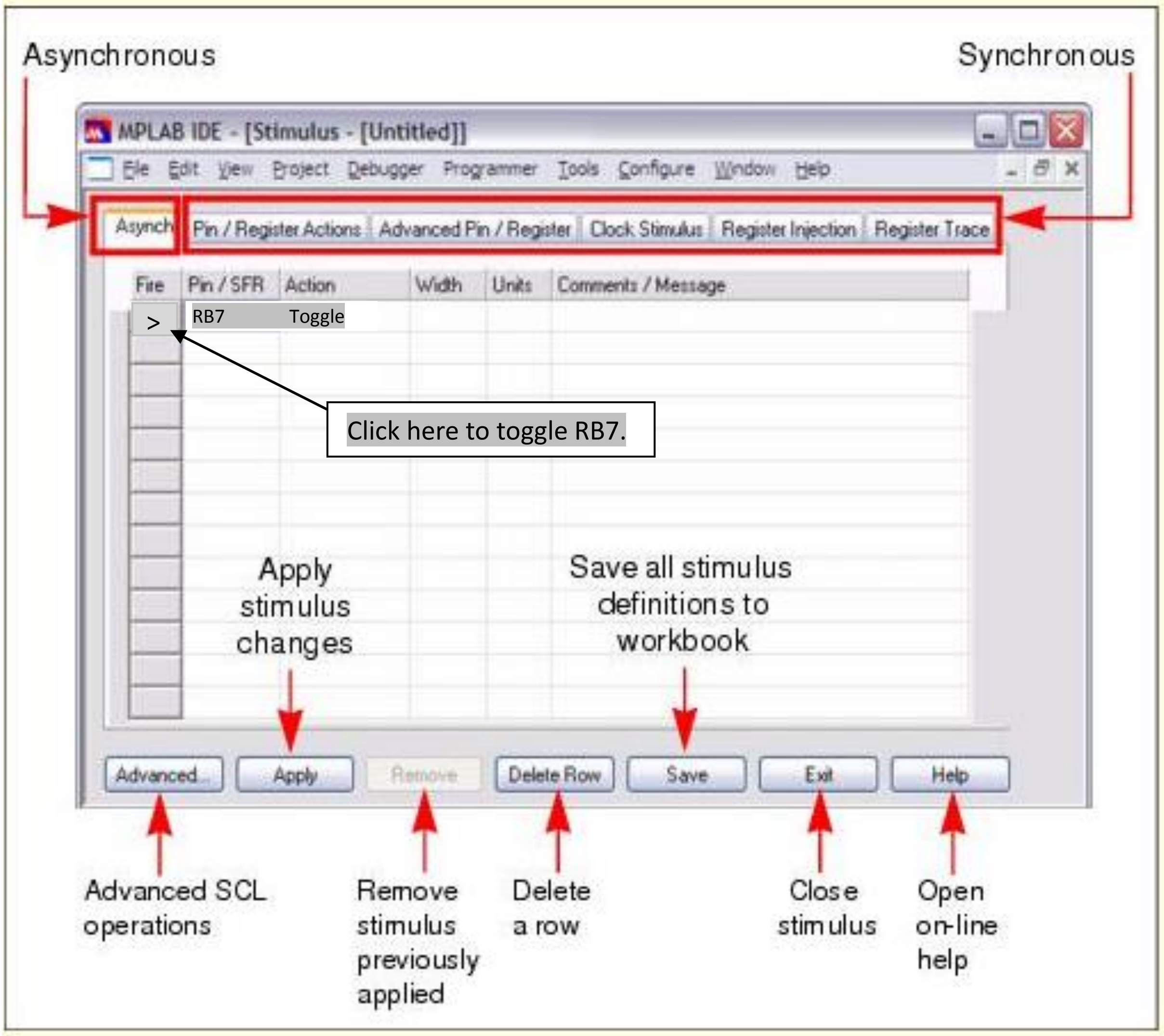
To open a new workbook, select ***Debugger->Stimulus->New Workbook***.

The Stimulus dialog (see Figure: Stimulus Dialog) has these tabs.

- Asynch Tab
- Pin/Register Actions Tab
- Advanced Pin/Register Tab
- Clock Stimulus Tab
- Register Injection Tab
- Register Trace Tab

Note: The Stimulus dialog must be open for stimulus to be active during simulation.

Figure: Stimulus Dialog

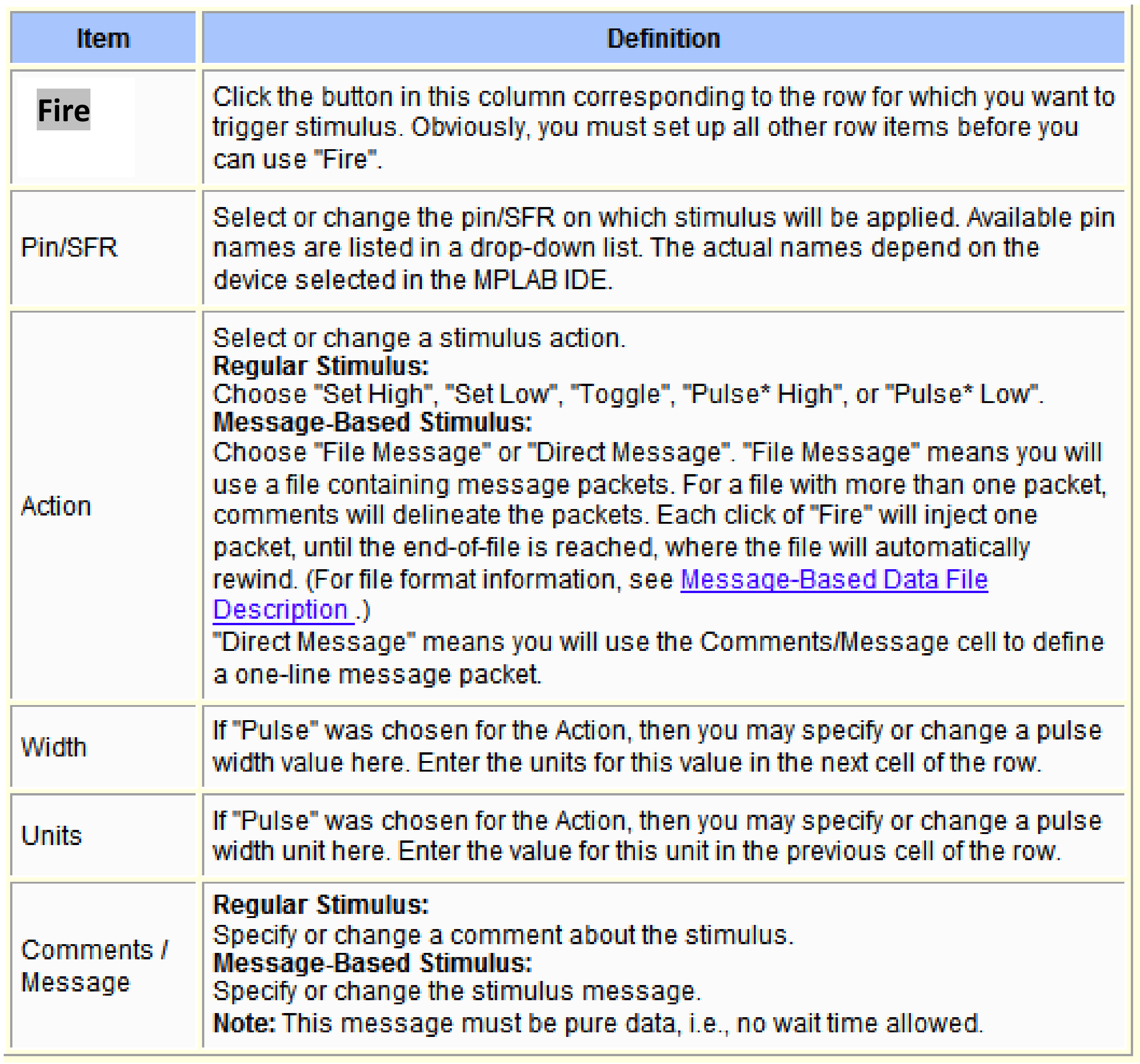


Asynch Tab

Use the **Asynch** tab to control asynchronous events generated by the user.

Enter any asynchronous, or user-fired, stimulus here, row by row. To remove a row, select the row and then click **Delete Row**.

There are two types of asynchronous stimulus: regular stimulus, for most pins/SFRs, and message-based stimulus, for USART/UART SFRs.



Item	Definition
Fire	Click the button in this column corresponding to the row for which you want to trigger stimulus. Obviously, you must set up all other row items before you can use "Fire".
Pin/SFR	Select or change the pin/SFR on which stimulus will be applied. Available pin names are listed in a drop-down list. The actual names depend on the device selected in the MPLAB IDE.
Action	Select or change a stimulus action. **Regular Stimulus:** Choose "Set High", "Set Low", "Toggle", "Pulse* High", or "Pulse* Low". **Message-Based Stimulus:** Choose "File Message" or "Direct Message". "File Message" means you will use a file containing message packets. For a file with more than one packet, comments will delineate the packets. Each click of "Fire" will inject one packet, until the end-of-file is reached, where the file will automatically rewind. (For file format information, see Message-Based Data File Description .) "Direct Message" means you will use the Comments/Message cell to define a one-line message packet.
Width	If "Pulse" was chosen for the Action, then you may specify or change a pulse width value here. Enter the units for this value in the next cell of the row.
Units	If "Pulse" was chosen for the Action, then you may specify or change a pulse width unit here. Enter the value for this unit in the previous cell of the row.
Comments / Message	**Regular Stimulus:** Specify or change a comment about the stimulus. **Message-Based Stimulus:** Specify or change the stimulus message. **Note:** This message must be pure data, i.e., no wait time allowed.