An introduction to PIC32 MX110F016B Microcontroller Part II

Sina Akbari

Timers – 1

- Timer/counter hardware is a crucial component of most embedded systems. In some cases a timer is needed to measure elapsed time; in others we want to count or time some external events.
- MX110F016B has 6 timers: Timers 1,2,3,4,5 and a Watchdog Timer.

Timers – 2: Overview

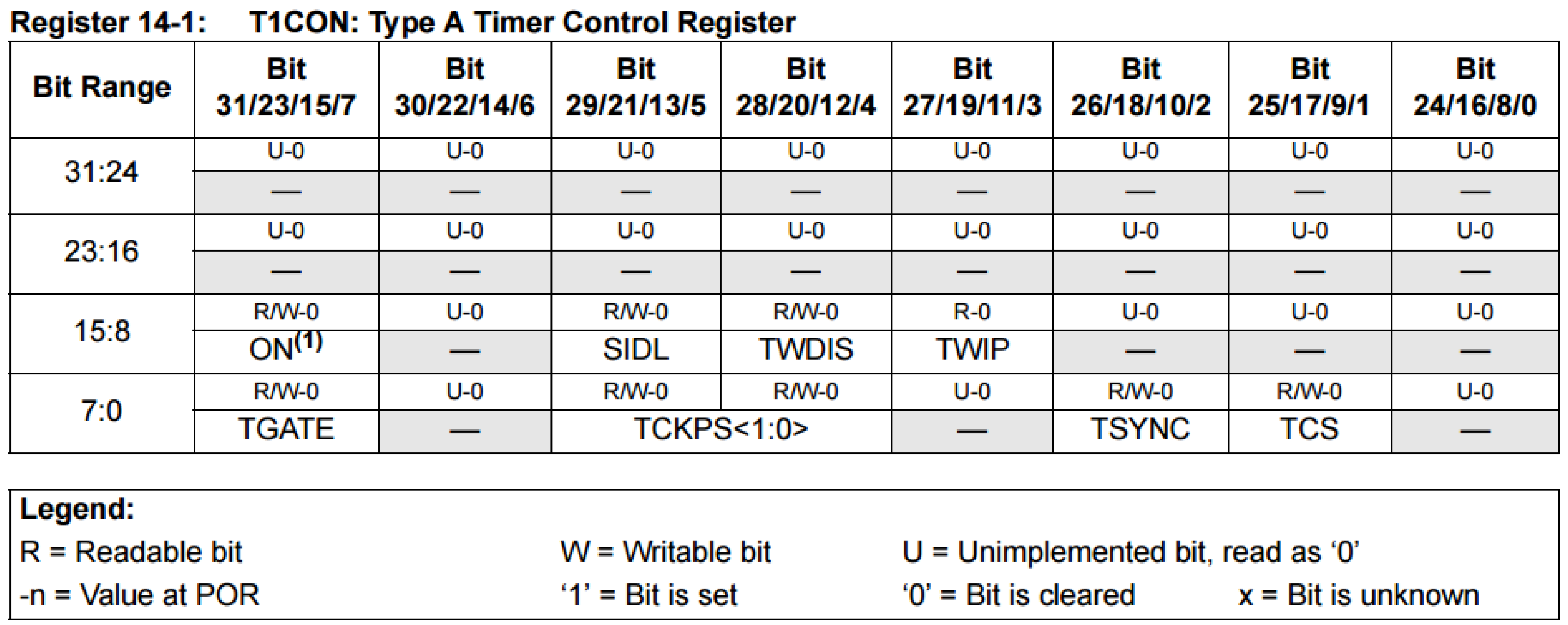
- Timer1:
  - 16bit type A timer
  - Synchronous/Asynchronous
- Timer2,3,4,5:
  - 16bit default (you will see we can change it) type B timer
  - Synchronous
- WDT:
  - 25bit Counter
  - Used to detect software malfunctions, or to wake the micro-controller up from its idle state.

Timers – 3: TxCON, PRx, TMRx Registers

- 3 SPFs are assigned to each timer:
- TxCON:
  - Timer CONtrol register. Used to start/stop the counter, select the input clock source and clock prescaler, etc.
- PRx:
  - Period Register. Used to determine the period of the timer, i.e. if you set it to 10, the timer counts up to 10, then it returns to 0 and goes on like this.
- TMRx:
  - TiMeR value. Represents the complete 16-bit timer count.

Timers – 4: T1CON

- Upper 16bits are unused.
- Bit #15 determines if the timer is on. Set this bit to start the timer.
- Other control bits include Gated/Ungated timer, Clock Source, Clock Prescaler,… but you won't need to use them – at least for now.

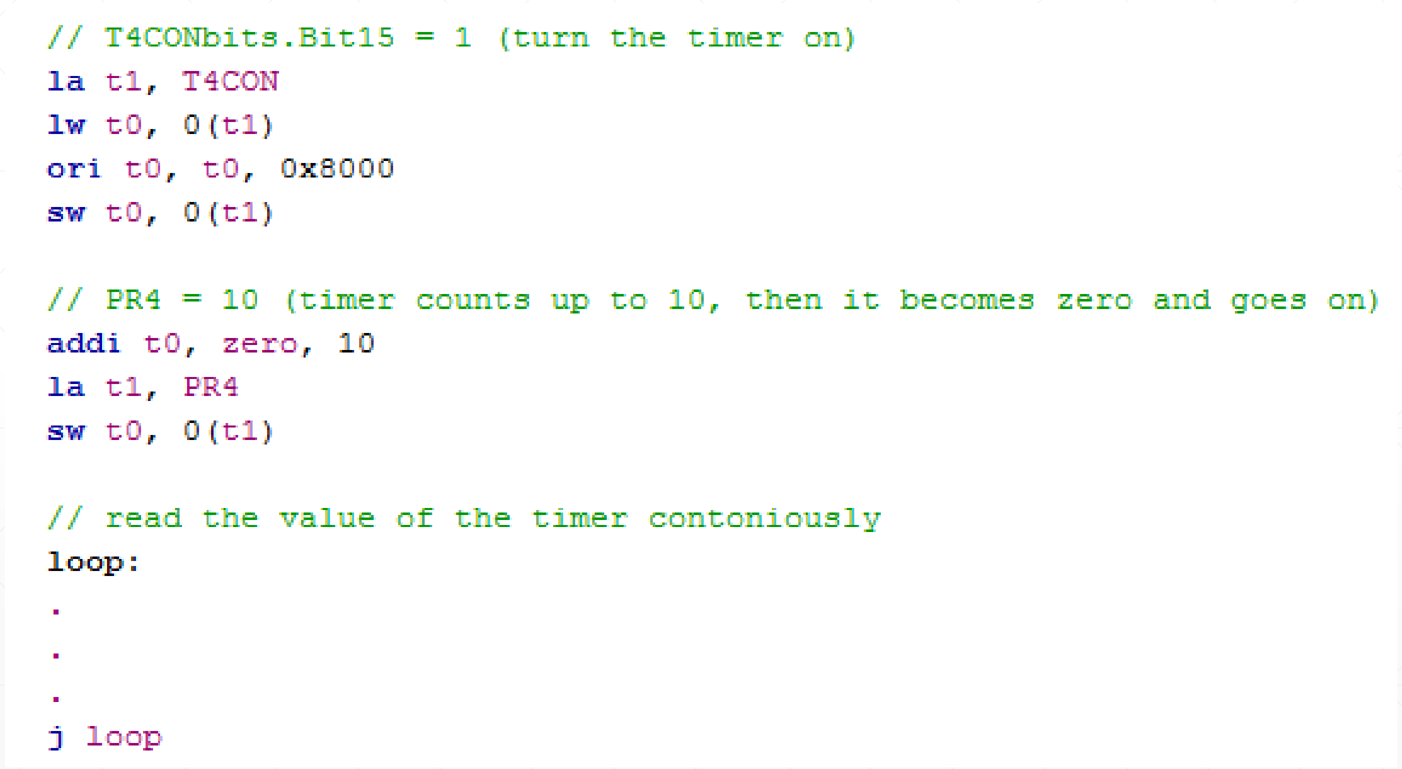


Register 14-1: T1CON: Type A Timer Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	R/W-0	R-0	U-0	U-0	U-0
	ON[1]	—	SIDL	TWDIS	TWIP	—	—	—
7:0	R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

Timers – 5: Getting started

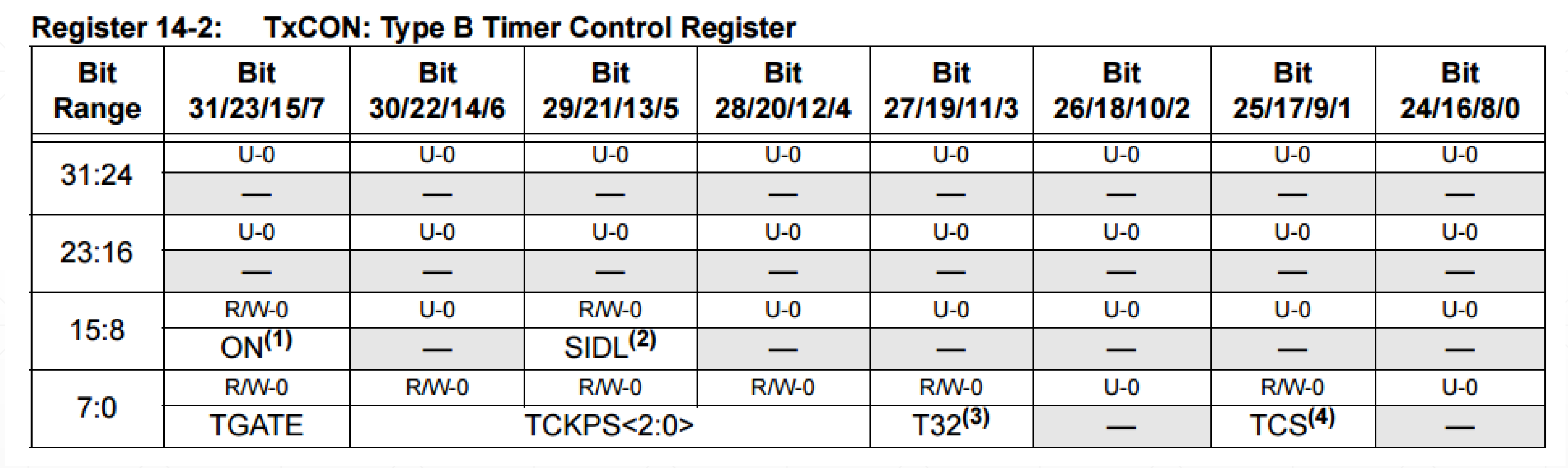


```
// T4CONbits.Bit15 = 1 (turn the timer on)
la t1, T4CON
lw t0, 0(t1)
ori t0, t0, 0x8000
sw t0, 0(t1)

// PR4 = 10 (timer counts up to 10, then it becomes zero and goes on)
addi t0, zero, 10
la t1, PR4
sw t0, 0(t1)

// read the value of the timer contoniously
loop:
.
.
.
j loop
```

What if we need a larger timer?

Timers – 6: Combining two timers together

- We can obtain a 32 bit timer by combining Timer2 withTimer3, or Timer4 with Timer5.
- Note the bit #3 of TxCON register. If it is set, timers will function in 32bit mode.



Register 14-2: TxCON: Type B Timer Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
	ON[1]	—	SIDL[2]	—	—	—	—	—
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	U-0
	TGATE		TCKPS<2:0>		T32[3]	—	TCS[4]	—

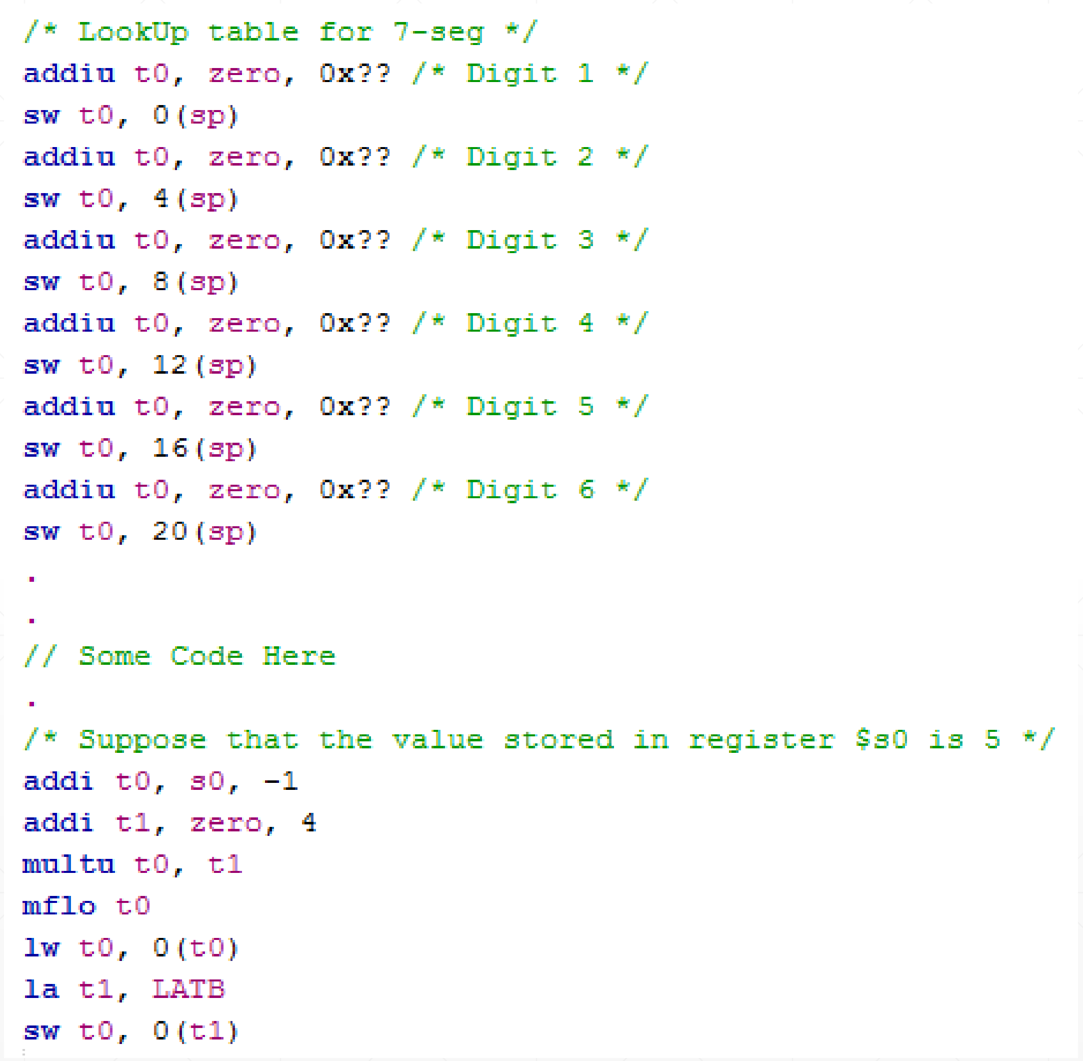
PIC Lab – Session2

Statement) We want to design a dice.
So we need to generate a random digit.

Question) How can we generate a random digit from 1 to 6, using just a push button and one of the timers?

An Idea: Lookup Table

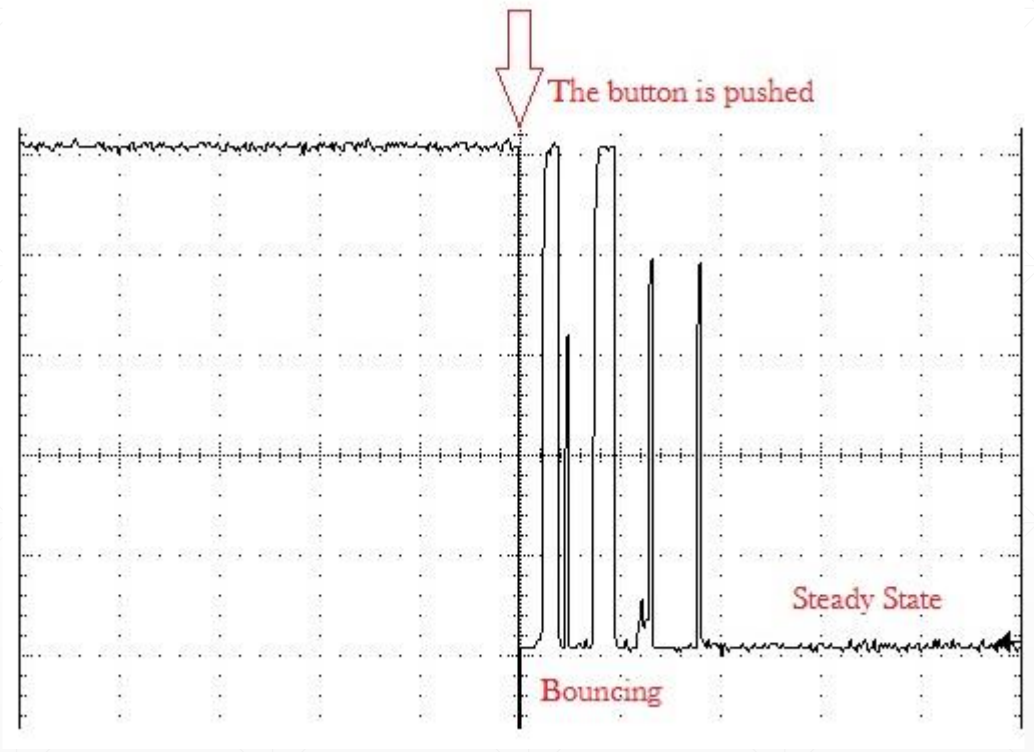
- You will need to set the output port so as to turn on a 7-seg. As no decoder ICs (such as BCD to 7-seg) will be provided, you yourself have to convert the output to the proper format inside your code. An idea to perform such a task is to store these outputs in a segment of the memory called a lookup table in order that you won't have to do the conversion every time.
- For instance, in the code besides, a lookup table is created in the stack.
- Persuade yourself that when this code is executed, the 7-seg connected to port B shows the digit 5!
- Use of this idea is totally optional!



```
/* LookUp table for 7-seg */
addiu t0, zero, 0x?? /* Digit 1 */
sw t0, 0(sp)
addiu t0, zero, 0x?? /* Digit 2 */
sw t0, 4(sp)
addiu t0, zero, 0x?? /* Digit 3 */
sw t0, 8(sp)
addiu t0, zero, 0x?? /* Digit 4 */
sw t0, 12(sp)
addiu t0, zero, 0x?? /* Digit 5 */
sw t0, 16(sp)
addiu t0, zero, 0x?? /* Digit 6 */
sw t0, 20(sp)
.
.
// Some Code Here
.
/* Suppose that the value stored in register $s0 is 5 */
addi t0, s0, -1
addi t1, zero, 4
multu t0, t1
mflo t0
lw t0, 0(t0)
la t1, LATB
sw t0, 0(t1)
```

A Physical Consideration

- When you push a button, it doesn’t get to its final level immediately. It “bounces” for a limited time period until it gets stable. This is due to the mechanical structure of these buttons.
- There are two solutions to overcome bouncing: either use a “Debouncing Hardware” or use a proper delay in your code until the bouncing period is over.
- A few milliseconds of delay would be appropriate.



That’s it! Enjoy.