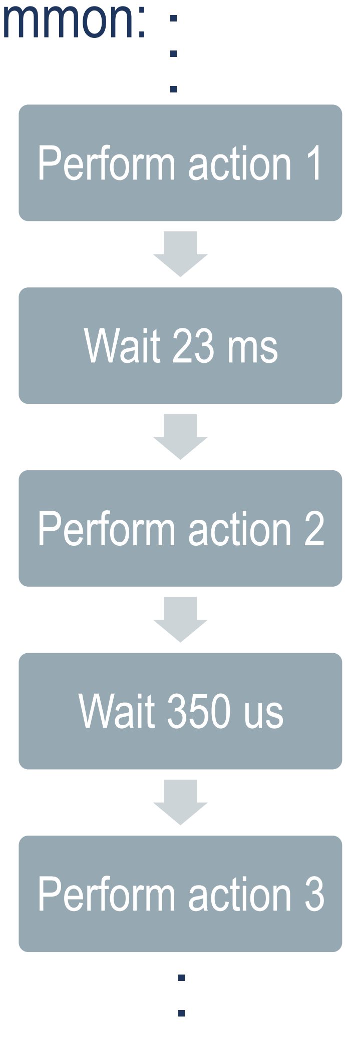
ECE 2534

# A Gentle Introduction to Timers

# Many embedded applications require the measuring of *elapsed time*

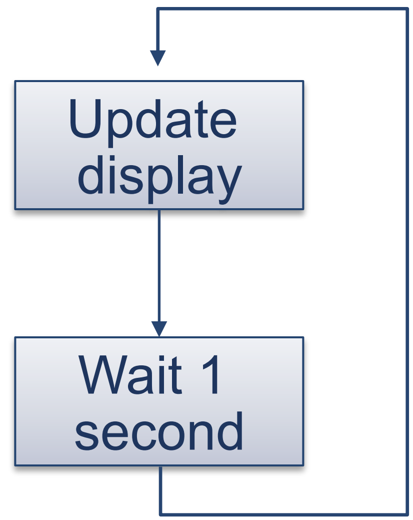
- Very common:



# Many embedded applications require the measuring of *elapsed time*

- Very common:


- Also common:



# So, how does the computer wait X microseconds?

- In the early days, delay loops were often used

```
int i, tmp;
. . .
for (i = 0; i < desired_num_of_microseconds; i++)
{
      tmp = tmp + 0; // do nothing
      tmp = tmp * 1; // do nothing
      . . .
}
. . .
```

Programmers would insert instructions that accomplished nothing, and carefully figure out how much time the CPU spent executing them

*Problem*: modern compilers and architectures make it VERY difficult to estimate these execution times with desired accuracy

# So, how does the computer wait X microseconds?

*Idea*

- Interface a **hardware counter** to the processor
- Design the hardware so that the counter updates at known, regular intervals
- Provide a mechanism for software to read the counter's contents
- Possibly provide a mechanism for software to control the counter: e.g., initialize the counter, start, stop
- Call this thing a **timer module** or **timer subsystem**

❑ What are some advantages of a hardware-based approach?

❑ What are some disadvantages of a hardware-based approach?

# So, how does the computer wait X microseconds?

- Possible pseudocode for the hardware-based approach

```
Clear the counter (load 0 into it)
Direct it to begin counting up

Repeat until counter has reached desired value
{
      Read value from counter
}
```

The programmer can figure out, in advance,
what value the counter will contain after X microseconds have elapsed.

# So, how does the computer wait X microseconds?

- Actual PIC32 code for the hardware-based approach (partial)

```
. . .
OpenTimer1( … );

. . .

WriteTimer1(0);
while (ReadTimer1() < cntMsDelay);

. . .
```

This is an excerpt from Lab 1's delay.c

- So far, so good.

- Problem: What useful things does the CPU accomplish during this loop?

```
while (ReadTimer1() < cntMsDelay);
```

- So far, so good.

- Problem: Designers are human.

  Humans can’t resist making things “better”.

- So far, so good.

- Problem: Designers are human.

  Humans can't resist making things "better".

- So, instead of a simple, elegant subsystem that does one thing well, nearly all timer subsystems provide multiple capabilities. E.g.,
  - Provide several other modes of operation!
  - Allow the counter to update at different rates!
  - Provide an option for an external clock to drive the counter!
  - Provide options for both 16-bit and 32-bit operation!
  - Allow an external gating signal to control when the counter updates!
  - Introduce versions called Type A and Type B!
  - Provide "hooks" into other subsystems on the processor, such as the ADC (analog-to-digital converter)!

- As a result, “all” you need to do is figure out the purpose of these bits
- … and you need to figure out the diagram on the next page

## Section 14. Timers

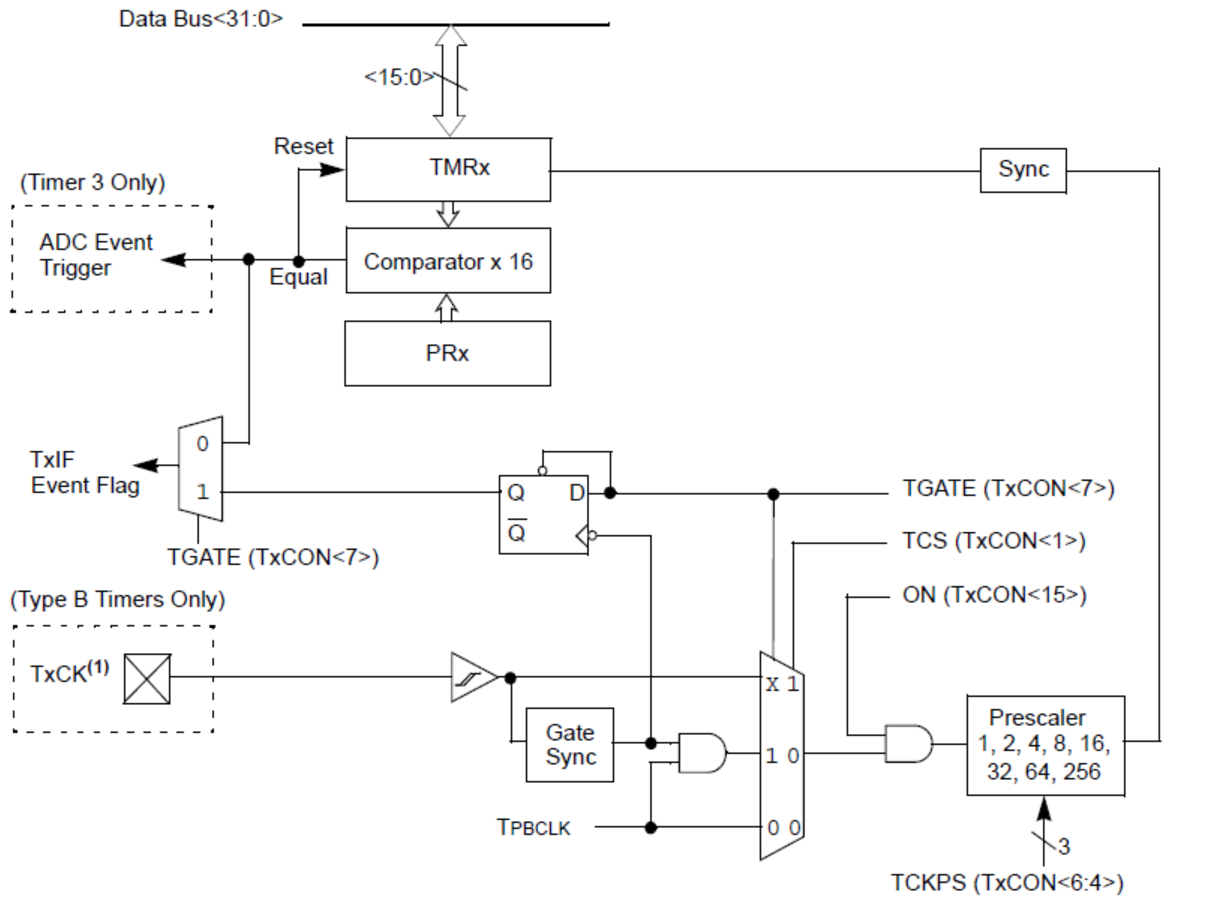
**Register 14-2: TxCON: Type B Timer Control Register**

| 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] | — |

| 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 |

# Timer 2 / 3 / 4 / 5 (16-bit operation)

**Figure 14-2: Type B Timer Block Diagram (16-Bit)**




**Note 1:** The TxCK pin is not available in certain variants of the PIC32MX device family. In such cases, the timer must use the peripheral clock as its input clock. Refer to the specific device data sheet for the I/O pin details.

# Summary

- All modern microcontrollers incorporate timer modules
  - A timer module is essentially a hardware counter that is updated at a known rate
  - The counter an be configured/accessed through software
- Why are timers so popular?
  - Many applications of microcontrollers involve time-related actions
  - A timer can be used to measure time intervals much more accurately than a software delay loop
- Timer modules usually offer several modes of operation

- More details are coming soon!