Timers

ECE 362
https://engineering.purdue.edu/ee362/
Reading Assignment

- Family Reference Manual, Chapter 17, "General purpose timers (TIM2 and TIM3)", pages 377 – 443.
GPIO Alternate Function

• Recall that fields in the GPIOx_MODER register can take on four values to define a pin mode:
  - 00: Pin is an input.
  - 01: Pin is an output.
  - 10: Pin uses an alternate function.
  - 11: Pin is an analog input.

• Today, we’ll start using the alternate function codes.
  - Configured by four-bit fields in the GPIOx_AFRL and GPIOx_ARFH.
  - Datasheet, page 37, table 14, Alternate functions for Port A.
  - Datasheet, page 38, table 15, Alternate functions for Port B.

• Configuring a pin for AF means you cannot use that pin for general purpose I/O.
  - See FRM, Figure 20, page 163.
To set GPIO alternate function

9.4.9  GPIO alternate function low register (GPIOx_AFRL)
(x = A..F)

Address offset: 0x20
Reset value: 0x0000 0000

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFR2[3:0]</th>
<th>AFR1[3:0]</th>
<th>AFR0[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 31:0  AFRy[3:0]: Alternate function selection for port x pin y (y = 0..7)
These bits are written by software to configure alternate function I/Os

AFRx selection:
- 0000: AF0
- 0001: AF1
- 0010: AF2
- 0011: AF3
- 0100: AF4
- 0101: AF5
- 0110: AF6
- 0111: AF7
- 1000: Reserved
- 1001: Reserved
- 1010: Reserved
- 1011: Reserved
- 1100: Reserved
- 1101: Reserved
- 1110: Reserved
- 1111: Reserved

- Two AFR registers per port: AFRL/AFRH.
- Four bits per pin.
  - e.g., to set PA0 to use timer 2, channel 1:
    - GPIOA->AFRL_AFR0[3:0] = 0010
  - e.g., to set PB14 to use timer 15, channel 1:
    - GPIOB->AFRH_AFR6[3:0] = 0001
    - (14 – 8 = 6)
- Where did these numbers come from?
  - Table 14 and Table 15
### Table 14. Alternate functions selected through GPIOA_AFR registers for port A

<table>
<thead>
<tr>
<th>Pin name</th>
<th>AF0</th>
<th>AF1</th>
<th>AF2</th>
<th>AF3</th>
<th>AF4</th>
<th>AF5</th>
<th>AF6</th>
<th>AF7</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>-</td>
<td>USART2_CTS</td>
<td>TIM2_CH1_ETR</td>
<td>TSC_G1_IO1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>COMP1_OUT</td>
</tr>
<tr>
<td>PA1</td>
<td>EVENTOUT</td>
<td>USART2_RTS</td>
<td>TIM2_CH2</td>
<td>TSC_G1_IO2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>TIM15_CH1</td>
<td>USART2_TX</td>
<td>TIM2_CH3</td>
<td>TSC_G1_IO3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>COMP2_OUT</td>
</tr>
<tr>
<td>PA3</td>
<td>TIM15_CH2</td>
<td>USART2_RX</td>
<td>TIM2_CH4</td>
<td>TSC_G1_IO4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA4</td>
<td>SPI1_NSS, I2S1_WS</td>
<td>USART2_CK</td>
<td>-</td>
<td>TSC_G2_IO1</td>
<td>TIM14_CH1</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA5</td>
<td>SPI1_SCK, I2S1_CK</td>
<td>CEC</td>
<td>TIM2_CH1_ETR</td>
<td>TSC_G2_IO2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA6</td>
<td>SPI1_MISO, I2S1_MCK</td>
<td>TIM3_CH1</td>
<td>TIM1_BKIN</td>
<td>TSC_G2_IO3</td>
<td>TIM16_CH1</td>
<td>EVENTOUT</td>
<td>-</td>
<td>COMP1_OUT</td>
</tr>
<tr>
<td>PA7</td>
<td>SPI1_MOSI, I2S1_SD</td>
<td>TIM3_CH2</td>
<td>TIM1_CH1N</td>
<td>TSC_G2_IO4</td>
<td>TIM14_CH1</td>
<td>TIM17_CH1</td>
<td>EVENTOUT</td>
<td>COMP2_OUT</td>
</tr>
<tr>
<td>PA8</td>
<td>MCO</td>
<td>USART1_CK</td>
<td>TIM1_CH1</td>
<td>EVENTOUT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA9</td>
<td>TIM15_BKIN</td>
<td>USART1_TX</td>
<td>TIM1_CH2</td>
<td>TSC_G4_IO1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA10</td>
<td>TIM17_BKIN</td>
<td>USART1_RX</td>
<td>TIM1_CH3</td>
<td>TSC_G4_IO2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA11</td>
<td>EVENTOUT</td>
<td>USART1_CTS</td>
<td>TIM1_CH4</td>
<td>TSC_G4_IO3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>COMP1_OUT</td>
</tr>
<tr>
<td>PA12</td>
<td>EVENTOUT</td>
<td>USART1_RTS</td>
<td>TIM1_ETR</td>
<td>TSC_G4_IO4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>COMP2_OUT</td>
</tr>
<tr>
<td>PA13</td>
<td>SWDIO</td>
<td>IR_OUT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA14</td>
<td>SWCLK</td>
<td>USART2_TX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PA15</td>
<td>SPI1_NSS, I2S1_WS</td>
<td>USART2_RX</td>
<td>TIM2_CH1_ETR</td>
<td>EVENTOUT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Pin name</td>
<td>AF0</td>
<td>AF1</td>
<td>AF2</td>
<td>AF3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB0</td>
<td>EVENTOUT</td>
<td>TIM3_CH3</td>
<td>TIM1_CH2N</td>
<td>TSC_G3_IO2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>TIM14_CH1</td>
<td>TIM3_CH4</td>
<td>TIM1_CH3N</td>
<td>TSC_G3_IO3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB2</td>
<td></td>
<td></td>
<td>TSC_G3_IO4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB3</td>
<td>SPI1_SCK, I2S1_CK</td>
<td>EVENTOUT</td>
<td>TIM2_CH2</td>
<td>TSC_G5_IO1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB4</td>
<td>SPI1_MISO, I2S1_MCK</td>
<td>TIM3_CH1</td>
<td>EVENTOUT</td>
<td>TSC_G5_IO2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB5</td>
<td>SPI1_MOSI, I2S1_SD</td>
<td>TIM3_CH2</td>
<td>TIM16_BK1N</td>
<td>I2C1_SMBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB6</td>
<td>USART1_TX</td>
<td>I2C1_SCL</td>
<td>TIM16_CH1N</td>
<td>TSC_G5_IO3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB7</td>
<td>USART1_RX</td>
<td>I2C1_SDA</td>
<td>TIM17_CH1N</td>
<td>TSC_G5_IO4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB8</td>
<td>CEC</td>
<td>I2C1_SCL</td>
<td>TIM16_CH1</td>
<td>TSC_SYNC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB9</td>
<td>IR_OUT</td>
<td>I2C1_SDA</td>
<td>TIM17_CH1</td>
<td>EVENTOUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB10</td>
<td>CEC</td>
<td>I2C2_SCL</td>
<td>TIM2_CH3</td>
<td>TSC_SYNC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB11</td>
<td>EVENTOUT</td>
<td>I2C2_SDA</td>
<td>TIM2_CH4</td>
<td>TSC_G6_IO1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB12</td>
<td>SPI2_NSS</td>
<td>EVENTOUT</td>
<td>TIM1_BK1N</td>
<td>TSC_G6_IO2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB13</td>
<td>SPI2_SCK</td>
<td></td>
<td>TIM1_CH1N</td>
<td>TSC_G6_IO3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB14</td>
<td>SPI2_MISO</td>
<td>TIM15_CH1</td>
<td>TIM1_CH2N</td>
<td>TSC_G6_IO4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB15</td>
<td>SPI2_MOSI</td>
<td>TIM15_CH2</td>
<td>TIM1_CH3N</td>
<td>TIM15_CH1N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Accessing AFRx registers with C

• It’s complicated.

• You can refer to values with symbols like this:
  
  \[
  \begin{align*}
  \text{GPIO AFRL AFRL0} & \equiv 0x0000000F \\
  \text{GPIO AFRH AFRH5} & \equiv 0x00F00000 \\
  \end{align*}
  \]

• It’s often easier to do it manually (PB11: AF2):
  
  \[
  \begin{align*}
  \text{GPIOB->AFR}[1] & \&= \sim (0xf << (4*(11-8))) \\
  \text{GPIOB->AFR}[1] & | = \quad 0x2 << (4*(11-8)) \\
  \end{align*}
  \]
Timers: “Where do we even begin?”

- Start with the STM32F051R8T6 datasheet page 1.
  - “Too many options.”
  - “I am stymied, once again, by the paradox of choice.”

- Up to 11 timers
  - One 16-bit 7-channel advanced-control timer for 6 channels PWM output, with deadtime generation and emergency stop
  - One 32-bit and one 16-bit timer, with up to 4 IC/OC, usable for IR control decoding
  - One 16-bit timer, with 2 IC/OC, 1 OCN, deadtime generation and emergency stop
  - Two 16-bit timers, each with IC/OC and OCN, deadtime generation, emergency stop and modulator gate for IR control
  - One 16-bit timer with 1 IC/OC
  - Independent and system watchdog timers
  - SysTick timer: 24-bit downcounter
  - One 16-bit basic timer to drive the DAC
“How should I choose?”

- Start with the STM32F051R8T6 datasheet...
  - page 31, table 13, pin definitions.
    - See the "Alternate functions" column.
  - page 37, table 14, Alternate functions for Port A.
  - page 38, table 15, Alternate functions for Port B.
  - "What about Ports C, D, E, and F?"
    - Port C pins have, at most, one alternate function.
      - You don’t have to set the AFRL/AFRH registers.
    - Port D pin can only have an AF of TIM3_ETR.
    - Port F pins are only for I2C or EVENTOUT.
“What pins should we start with?”

• From datasheet, table 13, pin definitions...
  – User pushbutton (PA0) can be connected to TIM2_CH1_ETR [Timer 2, channel 1 (and external trigger)]
  – Blue LED (PC8) can be connected to TIM3_CH3 [Timer 3, channel 3]
  – Green LED (PC9) can be connected to TIM3_CH4 [Timer 3, channel 4]

• So, we’ll start with Timers 2 and 3.
“If I want a certain function, how do I chose a pin for it?”

Table 7. Timer feature comparison

<table>
<thead>
<tr>
<th>Timer type</th>
<th>Timer</th>
<th>Counter resolution</th>
<th>Counter type</th>
<th>Prescaler factor</th>
<th>DMA request generation</th>
<th>Capture/compare channels</th>
<th>Complementary outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced control</td>
<td>TIM1</td>
<td>16-bit</td>
<td>Up, down, up/down</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>TIM2</td>
<td>32-bit</td>
<td>Up, down, up/down</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TIM3</td>
<td>16-bit</td>
<td>Up, down, up/down</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TIM4</td>
<td>16-bit</td>
<td>Up</td>
<td>integer from 1 to 65536</td>
<td>No</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TIM5</td>
<td>16-bit</td>
<td>Up</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TIM6</td>
<td>16-bit</td>
<td>Up</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>TIM1</td>
<td>16-bit</td>
<td>Up</td>
<td>integer from 1 to 65536</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Start with the STM32F051R8T6 datasheet again.
- page 21, table 7, timer feature comparison
- Eh. Not really helpful, and we already decided on Timer groups 2 and 3, right?
Timer groups 2 and 3

- This is what either timer 2 or timer 3 look like.
  - Each one contains four timer channels.
  - Each can:
    - measure input
    - count events
    - pulse length
    - generate output
    - toggle
    - PWM: Pulse Width Modulation
    - generate interrupts
How do TIM2/TIM3 work?

- Each has a 16-bit (TIM3) or 32-bit (TIM2) free-running counter (CTR).
  - 16-bit prescaler for input clock (PSC).
    - e.g. divide input clock by N.
    - System clock is default clk.
  - Auto-reload register (ARR)
  - Can count up from 0 to ARR
  - Can count down from ARR to 0
  - Can count up, down, up, down… 0 – ARR – 0 …
Let’s look at output first. In this mode we:
- configure a GPIO pin for an alternate function that connects it to a timer rather than its ODR.
- set up the prescaler
- set up the auto-reload register
- set the direction/mode of the counter
- set the output compare mode
- enable the "channel" the pin is connected to
- enable the timer’s counter (with the CEN bit)
Counter modes

UP

UP/DOWN

DOWN
Counter mode configuration

- Set with the TIMx_CR1 register:
  - CMS: Center-aligned (up/down) mode selection.
    - Non-zero value enables up/down mode.
  - DIR: Direction: 0: counts up, 1: counts down
  - CEN: Counter enable: 1: enable timer.
    - Set this one last.

```
+------------------+
| 9 8 7 6 5 4 3 2 1 0 |
| CKD[1:0] | ARPE | CMS | DIR | OPM | URS | UDIS | CEN |
| rw      | rw   | rw  | rw  | rw  | rw  | rw   | rw  |
+------------------+
```

SET CEN LAST!
How do channels work?

- Each has a **counter compare register (CCRx)**.
  - Each is continually compared to the counter (CTR).
  - When $CCRx = CTR$, something happens to $OCxREF$ (output channel reference).
Timer Output Modes

- Configured with the CCMRs (Capture/Compare Mode Registers)

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC2CE</td>
<td>OC2M[2:0]</td>
<td>OC2PE</td>
<td>OC2FE</td>
<td>CC2S[1:0]</td>
<td>OC1CE</td>
<td>OC1M[2:0]</td>
<td>OC1PE</td>
<td>OC1FE</td>
<td>CC1S[1:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>

- **CC1S**: direction of channel:
  - 00: output, 01: input mapped on TI1, 10: input mapped on TI2, 11: input mapped on TRC.

- **OC1M**: output compare mode:
  - 001: Logic high if CNT = CCR
  - 010: Logic low if CNT = CCR
  - 011: Toggle if CNT = CCR
  - 110: PWM output mode 1: Logic high if CNT < CCR, else logic low
Remember to enable the channels

- Use the TIMx_CCER (capture/compare enable register)

- CCxE: enable the channel
Each CCR is continually comparing its value with the free-running counter. When it matches, the operation (go high) happens.
Each CCR is continually comparing its value with the free-running counter. When it matches, the operation (toggle) happens.
Example: count up, toggle mode

Changing the CCRx value changes the offset into the count when the output toggles.
Example: up/down, toggle mode

Up/down mode is referred to as "Centered Mode"
Example: up/down, toggle mode

Changing the CCRx value widens the active cycle of the wave, but it stays "centered"
PWM output mode is an easy way of configuring a variable "duty cycle" output.
Example: count up, PWM mode 1

As CCRx value is decreased, the "duty cycle" is decreased proportionally.
```c
#include "stm32f0xx.h"
#include "stm32f0_discovery.h"

int main(void)
{
    // Enable Port C
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

    // Set the mode for PC8 for "alternate function"
    GPIOC->MODER &= ~(3<<16);
    GPIOC->MODER |= 2<<16;

    // Set the alternate function for PC8
    // PC0-7 are on AFR[0], 8-15 are on AFR[1]
    GPIOC->AFR[1] &= ~0xf;
    GPIOC->AFR[1] |= 1;

    // Enable the system clock for timer 3
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    // Counting direction: 0=up, 1=down
    TIM3->CR1 &= ~TIM_CR1_DIR; // clear it to count up

    // Set prescaler output to 4kHz (48MHz/12000)
    TIM3->PSC = 12000 - 1;

    // Auto-reload 4000
    TIM3->ARR = 4000 - 1;

    // Any value between 0 and 4000.
    TIM3->CCR3 = 3456;

    // Channel 3 of the timer is configured in CCMR2.
    // Set the bits to select toggle mode (011)
    TIM3->CCMR2 &= ~TIM_CCMR2_OC3M_2;
    TIM3->CCMR2 |= TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_0;

    // Enable output for channel 3 active-high output
    TIM3->CCER |= TIM_CCER_CC3E;

    // Enable timer 3
    TIM3->CR1 |= TIM_CR1_CEN;

    for(;;) asm("wfi");

    return 0;
}
```

This will blink the blue LED. Adapted from Textbook, page 380.
Register summary

- See Family Reference Manual, table 60, page 442...
- TIMx_CR1: Control Reg
- TIMx_CCMR1/2: Capture/Compare Mode Reg
- TIMx_CCER: Capture/Compare Enable Reg
- TIMx_CNT: 16- or 32-bit Free-running counter
- TIMx_PSC: 16-bit Prescaler Reg
- TIMx_ARR: Auto-Reload Reg
- TIMx_CCRx: Capture/Compare Reg

- RCC_APB1ENR: Location of the TIM3EN bit.
Full view of Timer 2/3 again

Output Section

Just Channels 1 & 2
Output config register arrangement

```
RCC→APB1ENR
TIM3EN

Prescaler ÷<65536
CK_INT

TIM3→PSC
CK_PSC

Auto-reload
TIM3→ARR

COUNTER
TIM3→CTR

CMS, DIR, CEN

TIM3 CR1
→CR1

TIM3 CCER
→CR1

CC1E

OC1REF

Output Control
0C1
TIM3_CH1

TIM3→CCMR1
CC1S, OC1M

TIM3→CCR1

Compare
CC1S, OC1M

CC2S, OC2M

TIM3→CCMR1

TIM3→CCR2

Compare
CC2S, OC2M

Everything is simple once you understand it.
```

"Timer 3"

"Channel 1"

"Channel 2"
What do timer inputs do?

- When a timer is configured as an input, it can
  - count events.
  - measure the length of a pulse.
- Best example is pushbutton input.
  - Very important thing to get right.
  - Everything bounces.
  - Dealing with bounce is a common challenge.
Differences from Output

- The CCRx registers are used for “capture” rather than “compare”.
  - Read-only
  - When an event occurs, an interrupt is generated.
    - Which can read the CCRx register to find out “what time” at which it happened.
    - Reading CCRx acknowledges the interrupt and clears the interrupt flag.
- Complex input filtering, routing, and prescaling.

- Timer input interrupts: just like GPIO interrupts, except for the timing analysis and complex filtering/routing.
Full view again

Input Section

Just Channels 1 & 2
Input config register arrangement

RCC->APB1ENR
TIM2EN

TIM2->CR1
CKD[1:0]

Prescaler ÷1,2,4
TIM2->PSC

Prescaler ÷<65536
CK_INT

Auto-reload
TIM2->ARR

Counter
TIM2->CTR

CEN, CMS, DIR

TIM2 CR1
TIM2->CR1

TIM2 DIER
TIM2->CC2IE
TIM2->CC1IE

Everything is simple once you understand it.

TI1
TIM2_CH1

Filter

Edge Detect

TI2
TIM2_CH2

Filter

Edge Detect

“Timer 2”

“Channel 1”

“Channel 2”
Typical ISR for pushbutton (e.g. textbook)

- Enable interrupt on rising and falling edges of pushbutton.

- If TIM2->SR has CC1IF (interrupt flag) set.
  - Save the timestamp from TIM2->CCR1
    - This clears the interrupt flag.
  - Check the previous captured timestamp.
    - Pulse width = current – previous
      - If long enough, then button has stopped bouncing.
volatile uint32_t pulse_width = 0;
volatile uint32_t last_captured = 0;
volatile uint32_t signal_polarity = 0;        // Assume input is low initially.

void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_CC1IF != 0) {            // Check interrupt flag is set
        uint32_t current_captured = TIM2->CCR1;     // Clears interrupt flag.
        signal_polarity ^= 1;                      // Toggle polarity.
        if (signal_polarity == 0)                  // Calculate only when input is low.
            pulse_width = current_captured - last_captured;

        last_captured = current_captured;
    }

    if ((TIM2->SR & TIM_SR_UIF) != 0) {         // Check for overflow.
        // Clear the UIF to prevent endless IRQ.
        TIM2->SR &= ~TIM_SR_UIF;
    }
}

Difficulties with ISRs

• Hard to debug an ISR if you only want to look at it in the debugger after the Nth invocation.
  – If you stop every time, you will interfere with button timing.
  – Really want only one interrupt when we’re sure that the button press has stopped bouncing.
Consider the following code

```c
// Very simplified ISR

void TIM2_IRQHandler() {
  increment();

  // Reading CCRx clears CCxIF int. flag.
  int discard __attribute__((unused));
  discard = TIM2->CCR1;
  discard = TIM2->CCR2;
}
```

GCC trick to tell it, “Yes, I know this variable is unused, and that is Okay.”
void setup_pa0_timer_interrupt(void) {
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOCEN;
    GPIOC->MODER &= 0x0000ffffff; // Set mode for Port C
    GPIOC->MODER |= 0x00055555;
    GPIOA->MODER &= ~(3<<2); // Set up PA0
    GPIOA->MODER |= 2<<2;
    GPIOA->AFR[0] &= ~(0xf<<4); // Set up AFR
    GPIOA->AFR[0] |= 0x2<<4;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // Enable the system clock for timer 2
    TIM2->PSC = 1 - 1; // Set prescaler output to maximum rate.
    TIM2->ARR = 0xffffffff; // Auto-reload the maximum value.
    TIM2->CCMR1 &= ~(TIM_CCMR1_CC1S); // Set direction of channel 1 as input, and set TI1 as input.
    TIM2->CCMR1 |= TIM_CCMR1_CC1S_0; // 01 = input, CC1 mapped to TI1
    TIM2->CCMR1 &= ~(TIM_CCMR1_IC1F); // No filtering
    TIM2->CCMR1 &= ~(TIM_CCMR1_IC1PSC); // Clear the prescaler.
    TIM2->CCER &= ~(TIM_CCER_CC1P|TIM_CCER_CC1NP); // Rising edge only.
    TIM2->CCER |= TIM_CCER_CC1E; // Enable capture for CC1
    TIM2->DIER |= TIM_DIER_CC1IE; // Allow Chan 1 to generate interrupts.
    TIM2->CR1 |= TIM_CR1_CEN; // Enable the timer counter.
    NVIC->ISER[0] = 1<<TIM2_IRQn; // Enable timer 2 interrupt in the interrupt controller (NVIC).
    for(;;)
        asm("wfi");
}
Problems with this setup...

• Every click (and bounce) of the button invokes the interrupt handler.
• It doesn’t check the CC1IF interrupt flag
  – but we don’t care much about that today.
• It doesn’t check the SR UIF flag
  – but we don’t care much about that today.
  • There will be other interrupt handlers in the future.
What does a bounce look like?

height

“press”

“release”

conductance

unstable

accepted as “on”

unstable

accepted as “off”

timescale exaggerated

unstable

accepted as “on”

unstable

accepted as “off”

press

release
How can input filtering help us?

- We can configure the filter (with IC1F) to:
  - Sample at every M ticks of CK_DTS
  - Require N positive (“on”) samples before accepting.
  - Generate an event only on rising edge.
  - No prescaler division.
TIM2->CCMR1 IC1F[3:0]  
(page 432 of FRM)

- Choose a configuration:
  - Slow clock rate
  - Lots of samples

- \( f_{\text{DTS}} \) can be CK_INT /1, /2, /4 as determined by TIM2_CR1 CKD[1:0] field.

- With \( f_{\text{CK_INT}}=48\text{MHz}, f_{\text{DTS}}=12\text{MHz}, f_{\text{SAMPLING}}=375\text{kHz} \), would require 21.3\( \mu \text{s} \) stability before recognizing as “on”.
Sampling the bounce

press

release

timescale exaggerated

searching for 8 positive reads in a row

Success. Positive edge registered and interrupt request raised.
Setting the sample clock

- Set TIM2->CR1 CKD[1:0] field to select $f_{DTS}$:
  - 00: $f_{DTS} = f_{CK\_INT}$
  - 01: $f_{DTS} = f_{CK\_INT} / 2$
  - 10: $f_{DTS} = f_{CK\_INT} / 4$  
    <= select slowest
  - 11: (reserved)

TIM2->CR1 &= ~TIM_CR1_CKD;
TIM2->CR1 |= TIM_CR1_CKD_1;
Turning on filtering for PA0

- Set TIM2->CCMR1 IC1F[3:0] field to ‘1111’
  - slowest sampling rate
  - most samples required

TIM2->CCMR1 &= ~TIM_CCMR1_IC1F;
TIM2->CCMR1 |= TIM_CCMR1_IC1F_3 | TIM_CCMR1_IC1F_2 | TIM_CCMR1_IC1F_1 | TIM_CCMR1_IC1F_0;

Everything is simple once you understand it.
Debouncing is still not perfect

- This works great when the system clock is 8MHz.
  - 8 sample intervals take about .1ms
- At 48MHz, it’s better, but not perfect.
  - 21.3μs is too short of a sample interval for many types of buttons.
  - There are no other options for fixing how to do this using built-in hardware and configuration.
  - We’ll do software solutions later.
  - See the textbook, pages 360 – 371, for more ideas.
Using a timer for a periodic interrupt

- Relatively simple!
  - Enable the clock to the timer.
  - Set the PSC and ARR.
  - Set the UIF bit in the DIER.
  - Enable the interrupt in the NVIC.
  - Set the CR1 CEN bit to enable the timer to run.
  - (No input/output configuration needed!)
Using timers to trigger other things

• Timers can “trigger” internal peripherals.
  – Primarily through the DMA mechanism
    • (Direct Memory Access)
  – The DIER register has multiple bits that can turn on periodic DMA triggers:
    • CCxDE: per-channel DMA request
    • UDE: per-timer DMA request on counter update
  – We’ll use this for several peripherals later.