ELEC 3040/3050
Lab 6

Time-based interrupts (Digital stopwatch design)

Timer usage in computer systems

- Periodically interrupt CPU to perform tasks
  - Sample temperature/pressure readings
  - Generate music samples
- Provide accurate time delays
  - Instead of software loops
- Generate pulses or periodic waveforms
  - PWM signal for motor control
  - Strobe pulse for an external device
- Determine time/duration of an external event
  - Measure a tachometer signal from a motor
Performing periodic operations

- Operations to be performed every T seconds
  - Timer module interrupts main thread every T seconds
    - Timer period usually programmable
  - Interrupt handler performs required operations
    - Operations usually include clearing a flag in the timer
Based on pre-settable binary counter

- Count value can be read (and written on some uCs)
- Count direction (up/down) may be fixed or selectable
- Counter’s clock source may be fixed or selectable
  - Counter mode: count pulses/events (e.g. odometer pulses)
  - Timer mode: periodic clock source;
    count value proportional to elapsed time (e.g. stopwatch)

- Count overflow/underflow action configurable
  - Set a flag (testable by software)
  - Generate an interrupt when flag set (if enabled)
  - Reload counter with a designated value and continue counting
  - Activate/toggle a hardware output signal
## STM32L1xx programmable timers

- **8 programmable timers (all use 16-bit counters)**

<table>
<thead>
<tr>
<th>Timers</th>
<th>Count direction</th>
<th>Periodic interrupts</th>
<th>OC/IC/PWM* channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM2-3-4</td>
<td>up/down</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>TIM9</td>
<td>up/down</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>TIM10-11</td>
<td>up</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>TIM6-7</td>
<td>up</td>
<td>yes</td>
<td>0</td>
</tr>
</tbody>
</table>

*OC = Output Compare  
IC = Input Capture  
PWM = Pulse Width Modulated Waveforms

### Other available timers:

- **Watchdog (WWDG):** 7-bit down-counter
  - **Interrupt CPU if count reaches 0**
  - **Correctly-operating S/W periodically resets count to prevent this**

- **Real-Time Clock (RTC):**
  - **Maintains time of day and calendar (with battery backup)**

- **SysTick timer**
  - **24-bit down-counter in all Cortex-M processors**
  - **Triggers periodic interrupts (“clock ticks”)**
STM32L1xx Clock Options

- **MSI (Multi-Speed Internal) DEFAULT AT POWER ON**
  - Internal oscillator: 2.097 MHz (0x20_0000 Hz)
  - Default clock selected by startup code

- **HSI (High-Speed Internal)**
  - Internal oscillator: 16 MHz
  - Select at startup by executing:
    ```c
    RCC->CR |= RCC_CR_HSION; // Turn on 16MHz HSI oscillator
    while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready
    RCC->CFGR |= RCC_CFGR_SW_HSI; // Select HSI as system clock
    ```

- **HSE (High-Speed External)**
  - Supplied via pins PH0/PH1: 8 MHz on STM32L100-Discovery

- **PLL (Phase-Lock Loop)**
  - Generate up to 32MHz clock from HSI or HSE source
  - More precise than internal oscillators
Basic timing function

All TIMx clocks = 2.097MHz*
TIM2-3-4-6-7 on APB1
TIM9-10-11 on APB2
* Unless reprogrammed

Scaled clock triggers
up-counter/down-counter
$F_{\text{CK}_\text{CNT}} = F_{\text{CK}_\text{PSC}} \div \text{Prescale}$

Event: CNT=ARR (up-count) or CNT=0 (down-count)
- CNT resets to 0 (if count up) or reloads ARR (if count down)
- UIF flag is set in the status register

$\text{CK}_\text{PSC} = \text{CK}_\text{INT}$ when count enabled

Update Event Interrupt
Signaled when UIF sets, if enabled (UIE=1)
Timer as a periodic interrupt source

- **Auto-Reload Value**
- **TIMx_ARR**
  - 16 bits
- **TIMx_PSC**
  - 16 bits
- **TIMx_CNT**
  - 16 bits

- **Count-up “overflow event”** if **TIMx_CNT** reaches **TIMx_ARR**
  - \(1 \rightarrow \text{UIF}\) (update interrupt flag) and **TIMx_CNT** resets to 0.
  - If **UIE** = 1 (update interrupt enabled), interrupt signal sent to NVIC

- **Prescale** value (set by **TIMx_PSC**) multiplies input clock period (1/ Fclk) to produce counter clock period:
  \[
  T_{cnt} = \frac{1}{F_{cnt}} = (PSC+1) \times \left(\frac{1}{F_{clk}}\right)
  \]

- Time interval = **TIMx_ARR** (Auto-Reload Register) value \(\times\) counter clock period:
  \[
  T_{out} = (ARR+1) \times T_{cnt} = (ARR+1) \times (PSC+1) \times \left(\frac{1}{F_{clk}}\right)
  \]

**Example:** For 1 second time period, given Fclk = 2.097MHz:
\[
T_{out} = (0x2000 \times 0x100) \div 0x2000000 = 1 \text{ second}
\]
\[
ARR = 0x1FFF \quad \& \quad PSC = 0xFF \quad (or \text{ equivalent \ combination})
\]
Counter timing:
PSC = 0
(prescale x1)
ARR = 36

Counter timing:
PSC = 3
(prescale x4)
ARR = 36

\[ T_{EVENT} = \text{Prescale} \times \text{Count} \times T_{CK\_INT} = (\text{PSC}+1) \times (\text{ARR}+1) \times T_{CK\_INT} \]
Timer registers

- **TIMx Counter (TIMx->CNT)** \((x = \text{timer} \#)\)
  - 16-bit binary counter, operating at \(f_{\text{CK_CNT}}\)
  - Up counter in TIM6, TIM7, TIM10, TIM11
  - Up/down counter in TIM2, TIM3, TIM4, TIM9

- **TIMx Prescale Register (TIMx->PSC)**
  - 16-bit clock prescale value
  - \(f_{\text{CK_CNT}} = f_{\text{CK_INT}} \div \text{prescale} \quad (\text{CK_INT is the clock source})\)

- **TIMx Auto-Reload Register (TIMx->ARR)**
  - 16-bit auto-reload value
  - End value for up count / initial value for down count
  - New ARR value can be written while the timer is running
    - Takes effect immediately if ARPE=0 in TIMx_CR1
    - Held in buffer until next update event if ARPE=1 in TIMx_CR1
Timer System Control Register 1

TIMx_CR1 (default = all 0’s)

7 6 5 4 3 2 1 0
| ARPE | | | URS | UDIS | CEN |

Examples:
TIM4->CR1 |= 0x01; //Enable counting
TIM4->CR1 &~= ~0x01; //Disable counting

Other Options:
UDIS = 0 enables update event to be generated (default)
URS = 0 allows different events to generate update interrupt (default)
1 restricts update interrupt to counter overflow/underflow
ARPE = 0 allows new ARR value to take effect immediately (default)
1 enables ARR buffer (new value held in buffer until next update event)
TIM2-4 and TIM9 include up/down direction and center-alignment controls

Counter Enable
1 = enable, 0 = disable
CEN=1 to begin counting
(apply CK_INT to CK_PSC)
Timer Status Register

$$TIMx_{-}SR \ (\text{reset value = all 0’s})$$

<table>
<thead>
<tr>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CC4F</td>
<td>CC3F</td>
<td>CC2F</td>
</tr>
</tbody>
</table>

Capture/Compare Channel n Interrupt Flags
(to be discussed later)

Update Interrupt Flag
1 = update interrupt pending
0 = no update occurred

Set by hardware on update event
($CNT$ overflow)

Cleared by software
(write 0 to UIF bit)

Example: do actions if UIF=1
if ($TIM4->{SR} \& 0x01 == 0x01$) { //test UIF
  .. do some actions
  $TIM4->{SR} \&= \neg0x01; \ //\text{clear UIF}$
}
Timer Interrupt Control Register

TIMx_DIER (default = all 0’s)

Capture/Compare n Interrupt Enable
(To be discussed later)

Examples:
TIM4->DIER |= 0x01;  //Enable interrupt
TIM4->DIER &= ~0x01; //Disable interrupt

Update interrupt enable
1 = enable, 0 = disable
(interrupt if UIF=1 when UIE=1)
Timer clock source

- Clock TIMx_CLK to each timer module TIMx must be **enabled** in the RCC (reset and clock control) module
  - TIMx_CLK is derived from a peripheral bus clock
    - TIM2-3-4-6-7 on APB1 (*peripheral bus 1*), enabled in RCC->APB1ENR
    - TIM9-10-11 on APB2 (*peripheral bus 2*), enabled in RCC->APB2ENR
  - **Example:** enable clocks to TIM2 and TIM9:
    ```
    RCC->APB1ENR |= 0x00000001;  //TIM2EN is bit 0 of APB1ENR
    RCC->APB2ENR |= 0x00000004;  //TIM9EN is bit 2 of APB2ENR
    ```
  - *STM32L1xx.h* defines symbols for bit patterns
    ```
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;  //same as 0x00000001
    RCC->APB2ENR |= RCC_APB2ENR_TIM9EN;  //same as 0x00000004
    ```
- **Default STM32L1xx startup code sets all bus/timer clocks to 2.097MHz on the Discovery board**
  - Reprogram if higher frequency desired
Timer interrupt vectors

- Each timer has its own interrupt vector in the vector table
  (refer to the startup file and Table 48 of STM32L100xx Reference Manual)
  - IRQ# determines vector position in the vector table
    - Timer#: TIM9 - 10 - 11 - 2 - 3 - 4 - 6 - 7
  - Symbols for IRQ#’s for NVIC functions defined in stm32l1xx.h
    - NVIC_EnableIRQ(TIM9_IRQn); // TIM9 = IRQ25
    - NVIC_ClearPendingIRQ(TIM7_IRQn); // TIM7 = IRQ44
  - Default interrupt handler names* in the startup file:
    - TIM9_IRQHandler(); //handler for TIM9 interrupts
    - TIM7_IRQHandler(); //handler for TIM7 interrupts

*Either use this name for your interrupt handler, or modify the startup file to change the default to your own function name.
Enabling timer interrupts

Timer interrupts must be enabled in three places

1. **In the timer**: UIE bit in register TIMx DIER
   
   ```
   TIM4->DIER |= 1; // UIE is bit 0
   ```
   or:
   
   ```
   TIM4->DIER |= TIM_DIER_UIE;
   ```
   
   - Interrupt triggered if UIF is set while UIE = 1
   - Interrupt handler must reset UIF (**write 0 to it**)

2. **In the NVIC** – set the enable bit for each IRQn source:
   
   ```
   NVIC_EnableIRQ(TIM9_IRQn); // enable TIM9 interrupts
   ```
   
   - NVIC “Pending Flag” resets **automatically** when interrupt handler entered

3. **In the CPU** – enable CPU to respond to any configurable interrupt
   
   ```
   __enable_irq();
   ```
Example: Initialize TIM4 for periodic interrupts

- Enable the clock to timer TIM4
  \[
  \text{RCC->APB1ENR} |\!=\!\text{RCC_APB1ENR\_TIM4EN};
  \]

- Calculate prescale and auto-reload values for desired period and write to PSC and ARR registers:
  \[
  // T_{\text{INT}} = T_{\text{CLK}} \times (PSC + 1) \times (ARR + 1)
  \]
  \[
  \text{TIM4->PSC} = \text{psc\_value}; //\text{prescaler value}
  \]
  \[
  \text{TIM4->ARR} = \text{arr\_value}; //\text{auto-reload value}
  \]

- Enable interrupts on timer update events
  \[
  \text{TIM4->DIER} |\!=\!\text{TIM\_DIER\_UIE}; //\text{Enable TIM4 to signal an interrupt}
  \]
  \[
  \text{NVIC\_EnableIRQ(TIM4\_IRQn)}; //\text{Enable TIM4 interrupt in NVIC}
  \]
  \[
  \_\_\_\text{enable\_irq()}; //\text{Enable interrupts in CPU}
  \]

- Enable the timer to start counting
  \[
  \text{TIM4->CR1} |\!=\!\text{TIM\_CR1\_CEN}; //\text{Bit CEN=1 to enable counting}
  \]
  \[
  //\text{Bit CEN=0 to stop counting}
  \]
Interrupt handler format

```c
void TIM4_IRQHandler () { // handler for TIM4
    // Perform the desired actions
    // - Update a time/display
    // - Sample external sensor data
    // - Perform a control action
    // - Etc.

    // Clear timer’s update interrupt flag (cancel interrupt request)
    TIM4->SR &= ~0x01;  // UIF is bit 0 of SR
    or TIM4->SR &= ~TIM_SR_UIF;

    // (Should be unnecessary): clear pending flag in NVIC
    NVIC_ClearPendingIRQ(TIM4_IRQn);
}
```
Stopwatch design

- Control watch with two keypad buttons
  - Button 0: start and stop the timer
    - Freeze the displayed time when stopped
    - Continue from displayed value if restarted
  - Button 1: clear time to 0.0, but only if watch is stopped

- While watch is running, display elapsed time from 0.0 to 9.9 seconds, and repeat until stopped
  - Display two BCD digits on two sets of four LEDs.
    - Use previous port configuration (PC7-PC0) to output two 4-bit values
    - Use previous Static IO window setup to display the values on LEDs
    - Verify count sequence with the logic analyzer
  - Timing must be precise, requiring timer interrupts.
    - Verify timing precision with the oscilloscope