Programmable timing functions

Part 1: Timer-generated interrupts

Textbook: Chapter 15, General-Purpose Timers and Timer Interrupts
Chapter 12.4, Cortex SysTick Timer and Interrupts

STM32F4xx Technical Reference Manual:
Chapter 17 – Basic timers (TIM6)
Chapter 15 – General-purpose timers (TIM4)
Chapter 10 - Interrupt vectors (for TIM4/TIM6 interrupts)
Timing functions in computer systems

- Periodically interrupt CPU to perform tasks
  - Sample sensor readings (temperature, pressure, etc.)
  - 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
- Measure duration of an external event
  - Tachometer signal period to measure motor speed
Performing periodic operations

- Certain operations are to be performed every $T$ seconds
  - Timer module interrupts the main thread every $T$ seconds
    - Timer period is usually programmable
  - Interrupt handler performs required operations
    - Operations usually include clearing a flag in the timer

Diagram:

- Interrupt handler
- Main thread
- Timer events
- Time points: $T$, $2T$, $3T$, $4T$, $5T$, ...
Timer Peripheral Modules

- Based on pre-settable binary counter
  - Count value can be read and written by MCU
  - Count direction might be fixed or selectable (up or down)
  - Counter’s clock source might be fixed or selectable
    - Counter mode: count pulses which indicate events (e.g. odometer pulses)
    - Timer mode: periodic clock source, so count value proportional to elapsed time (e.g. stopwatch)
- Counter’s overflow/underflow action can be configured
  - Set a flag (testable by software)
  - Generate an interrupt (if enabled)
  - Reload counter with a designated value and continue counting
  - Activate/toggle a hardware output signal
STM32F4 Timer Peripherals

- **Basic Timer (Simple timer)**
  - TIM6 and TIM7
  - Can be generic counter and internally connected to DAC
  - 16-bit auto reload register

- **General Purpose Timer**
  - TIM9 to TIM14
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit auto-reload register

- **General Purpose Timer**
  - TIM2, TIM3, TIM4, TIM5
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit or 32-bit auto-reload register

- **Advanced Control Timer**
  - TIM1 and TIM8
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit auto-reload register
  - Additional control for driving motor or other devices

- **24 bit system timer (SysTick) – standard in all Cortex-M CPUs**
STM32F407 programmable timers

14 timer modules – vary in counter width, max clock, and functionality

<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 output</th>
<th>Max interface clock (MHz)</th>
<th>Max timer clock (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced-control</td>
<td>TIM1, TIM8</td>
<td>16-bit</td>
<td>Up, Down, Up/down</td>
<td>Any integer between 1 and 65536</td>
<td>Yes</td>
<td>4</td>
<td>Yes</td>
<td>84</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>TIM2, TIM5</td>
<td>32-bit</td>
<td>Up, Down, Up/down</td>
<td>Any integer between 1 and 65536</td>
<td>Yes</td>
<td>4</td>
<td>No</td>
<td>42</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>TIM3, TIM4</td>
<td>16-bit</td>
<td>Up, Down, Up/down</td>
<td>Any integer between 1 and 65536</td>
<td>Yes</td>
<td>4</td>
<td>No</td>
<td>42</td>
<td>84</td>
</tr>
<tr>
<td>General purpose</td>
<td>TIM9</td>
<td>16-bit</td>
<td>Up</td>
<td>Any integer between 1 and 65536</td>
<td>No</td>
<td>2</td>
<td>No</td>
<td>84</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>TIM10, TIM11</td>
<td>16-bit</td>
<td>Up</td>
<td>Any integer between 1 and 65536</td>
<td>No</td>
<td>1</td>
<td>No</td>
<td>84</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>TIM12</td>
<td>16-bit</td>
<td>Up</td>
<td>Any integer between 1 and 65536</td>
<td>No</td>
<td>2</td>
<td>No</td>
<td>42</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>TIM13, TIM14</td>
<td>16-bit</td>
<td>Up</td>
<td>Any integer between 1 and 65536</td>
<td>No</td>
<td>1</td>
<td>No</td>
<td>42</td>
<td>84</td>
</tr>
<tr>
<td>Basic</td>
<td>TIM6, TIM7</td>
<td>16-bit</td>
<td>Up</td>
<td>Any integer between 1 and 65536</td>
<td>Yes</td>
<td>0</td>
<td>No</td>
<td>42</td>
<td>84</td>
</tr>
</tbody>
</table>
Alternate functions for pins PD12-13-14-15
From STM32F407 Data Sheet – Table 6

<table>
<thead>
<tr>
<th>Pin number</th>
<th>Pin name (function after reset)</th>
<th>Pin type</th>
<th>I/O structure</th>
<th>Notes</th>
<th>Alternate functions</th>
<th>Additional functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQFP64</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WL CSP90</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LOFP100</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LOFP144</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LQFP176</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>60  82 M15 101</td>
<td>PD13</td>
<td>I/O</td>
<td>FT</td>
<td>FSMC_A18/TIM4_CH2/</td>
<td>EVENTOUT</td>
</tr>
<tr>
<td>-</td>
<td>83  - 102</td>
<td>VSS</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>84  J13 103</td>
<td>VDD</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>61  85 M14 104</td>
<td>PD14</td>
<td>I/O</td>
<td>FT</td>
<td>FSMC_D0/TIM4_CH3/</td>
<td>EVENTOUT/ EVENTOUT</td>
</tr>
<tr>
<td>-</td>
<td>62  86 L14 105</td>
<td>PD15</td>
<td>I/O</td>
<td>FT</td>
<td>FSMC_D1/TIM4_CH4/</td>
<td>EVENTOUT</td>
</tr>
<tr>
<td>-</td>
<td>59  81 N13 100</td>
<td>PD12</td>
<td>I/O</td>
<td>FT</td>
<td>FSMC_A17/TIM4_CH1/</td>
<td>USART3_RTS/ EVENTOUT</td>
</tr>
</tbody>
</table>

TIM4 can drive LEDs connected to
PD12 with TIM4_CH1
PD13 with TIM4_CH2
PD14 with TIM4_CH3
PD15 with TIM4_CH4

(So, we will examine TIM4)
Basic timing function

Scaled clock triggers up-counter/down-counter
\[ F_{\text{CK_CNT}} = F_{\text{CK_PSC}} \div \text{Prescale} \]

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

TIMxCLK from RCC | Internal clock (CK_INT)\[ \rightarrow \]
Trigger controller\[ \rightarrow \]
TRGO \[ \rightarrow \]
to DAC\[ \rightarrow \]

16 MHz default freq. (programmable to higher freq’s)

TIM4, TIM6 on APB1

CK_PSC = CK_INT when count enabled

Signaled when UIF sets, if enabled (UIE=1)

Update Event Interrupt

Update Event

Auto-reload Register

\[ \text{ARR} \]

Stop, Clear or up

\[ \pm \]

Controller

Reset, Enable, Count,
General-purpose timers TIM2 – TIM5

Basic timer, plus:
- Capture/compare support,
- PWM generation,
- Triggering options,
Timer as a periodic interrupt source

- **Auto-Reload Value**
  - TIMx_ARR: 16 bits
  - TIMx_CNT: 16 bits
  - Reload
  - Event
  - UIF
  - TIMx_SR
  - TIMx_DIER

- **Count-up “overflow event” if TIMx_CNT reaches TIMx_ARR**
  - UIF (update interrupt flag) sets 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:
  \[ T_{cnt} = \frac{1}{F_{cnt}} = \frac{(PSC+1)}{F_{clk}} \]

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

**Example:** For 1 second time period, given Fclk = 16MHz:
\[
T_{out} = \left(10000 \times 1600\right) \div 16000000 = 1 \text{ second}
\]
Set ARR = 9999 and PSC = 1599 (other combinations can also be used)
$T_{EVENT} = \text{Prescale} \times \text{Count} \times T_{CK\_INT} = (\text{PSC}+1) \times (\text{ARR}+1) \times T_{CK\_INT}$

**Counter timing:**
- **Prescale** = 1
- **ARR** = 36

**Counter timing:**
- **Prescale** = 4
- **ARR** = 36
Counter timing (prescale changes 1->4)
Basic timer function registers
(present in all 14 timers)

- **TIMx Counter** (TIMx_CNT, address offset 0x24)
  - 16-bit binary counter (32 in TIM2, TIM5)
  - Up counter in TIM6-TIM7, TIM9-TIM14
  - Up/down in TIM1-TIM5, TIM8

- **TIMx Prescale Register** (TIMx_PSC, address offset 0x28)
  - Clock prescale value (16 bits)
  - \( f_{CK\_CNT} = f_{CK\_INT} \div \text{prescale} \) (assuming \(CK\_INT\) is clock source)

- **TIMx Auto-Reload Register** (TIMx_ARR, addr. offset 0x2C)
  - 16-bit register (32 in TIM2, TIM5)
  - 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)

<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>ARPE</td>
<td></td>
<td></td>
<td></td>
<td>URS</td>
<td>UDIS</td>
<td>CEN</td>
<td></td>
</tr>
</tbody>
</table>

**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
Timer Status Register

TIMx_SR (reset value = all 0’s)

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

```c
if (TIM4->SR & 0x01 == 0x01) { //test UIF
 .. do some actions
 TIM4->SR &= ~0x01; //clear UIF
}
```
Timer Interrupt Control Register

TIMx_DIER  (default = all 0’s)

<table>
<thead>
<tr>
<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></td>
<td></td>
<td></td>
<td></td>
<td>CC4IE</td>
<td>CC3IE</td>
<td>CC2IE</td>
<td>CC1IE</td>
<td>UIE</td>
</tr>
</tbody>
</table>

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

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

Examples:
TIM4->DIER |= 0x01;      //Enable interrupt
TIM4->DIER &= ~0x01;  //Disable interrupt
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
    - TIM2-3-4-5-6-7 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:
    
    ```c
    RCC->APB1ENR |= 0x00000001;  // TIM2EN is bit 0 of APB1ENR
    RCC->APB2ENR |= 0x00000004;  // TIM9EN is bit 2 of APB2ENR
    ```

- **Default STM32F4xx startup code sets all bus/timer clocks to 16MHz on the Discovery board**
Initialize the TIM4 with CMSIS

- Enable clock to Timer4
  
  \[\text{RCC} \rightarrow \text{APB1ENR} |= \text{RCC\_APB1ENR\_TIM4EN};\]

- Set the auto-reload
  
  \[\text{TIM4} \rightarrow \text{ARR} = \text{arr\_value};\]

- Set the prescaler
  
  \[\text{TIM4} \rightarrow \text{PSC} = \text{psc\_value};\]

- Enable the update interrupt
  
  \[\text{TIM4} \rightarrow \text{DIER} |= \text{TIM\_DIER\_UIE};\]

- Enable counting
  
  \[\text{TIM4} \rightarrow \text{CR1} |= \text{TIM\_CR1\_CEN};\]

**Assembly:**

\[
\begin{align*}
\text{RCC\_TIM4EN} & \text{ EQU } 0x04 \\
\text{arr\_value} & \text{ EQU } 4999 \\
\text{psc\_value} & \text{ EQU } 9999 \\
\text{DIER\_UIE} & \text{ EQU } 1 \\
\text{CR1\_CEN} & \text{ EQU } 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{ldr} & \ r0,=\text{RCC} \\
\text{ldr} & \ r1,[r0,\#\text{APB1ENR}] \\
\text{orr} & \ r1,\#\text{RCC\_TIM4EN} \\
\text{str} & \ r1,[r0,\#\text{APB1ENR}] \\
\text{ldr} & \ r0,=\text{TIM4} \\
\text{mov} & \ r1,\#\text{arr\_value} \\
\text{str} & \ r1,[r0,\#\text{ARR}] \\
\text{mov} & \ r1,\#\text{psc\_value} \\
\text{str} & \ r1,[r0,\#\text{PSC}] \\
\text{ldr} & \ r1,[r0,\#\text{DIER}] \\
\text{orr} & \ r1,\#\text{DIER\_UIE} \\
\text{str} & \ r1,[r0,\#\text{DIER}] \\
\text{ldr} & \ r1,[r0,\#\text{CR1}] \\
\text{orr} & \ r1,\#\text{CR1\_CEN} \\
\text{str} & \ r1,[r0,\#\text{CR1}] \\
\end{align*}
\]
Timer interrupt vectors

• Each timer has its own interrupt vector in the vector table
  (refer to the startup file and Table 61 in the STM32F4xx Reference Manual)

• IRQ# determines vector position in the vector table
  • IRQ#:   IRQ28 – 29 – 30 – 54 - 55
  • Timer#:  TIM2 - 3 - 4 - 6 - 7

• Default interrupt handler names* in the startup file:

  TIM4_IRQHandler();       // handler for TIM4 interrupts
  TIM6_DAC_IRQHandler();   // handler for TIM6 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 (bit 0) in register TIMx_DIER
     
     ```c
     TIM4->DIER |= 1;                           // UIE is bit 0
     
     or: TIM4->DIER |= TIM_DIER_UIE;       // symbol from header file
     ```

     • Interrupt triggered if UIF is set while UIE = 1
     • Interrupt handler must reset UIF (**write 0 to it**) 

  2. In the NVIC – set enable bit in NVIC_ISERx for each IRQn source:
     
     ```c
     NVIC_EnableIRQ(TIM9_IRQn);    // enable TIM9 interrupts
     ```

     • NVIC “Pending Flag” should reset automatically when the interrupt handler is entered

  3. In the CPU – enable CPU to respond to any configurable interrupt
     
     ```c
     __enable_irq();
     ```
Interrupt Handler

- Interrupts should be enabled in the CPU
  ```
  __enable_irq();
  Assembly: CPSIE I
  ```

- CMSIS ISR name: `TIM4_IRQHandler`
  ```
  NVIC_EnableIRQ(TIM4_IRQn); // n = 30 for TIM4
  Assembly: Enable TIM4_IRQHandler (bit 30) in NVIC_ISER0
  ```

- ISR activities
  - Clear pending IRQ
    ```
    NVIC_ClearPendingIRQ(TIM4_IRQn);
    Assembly: write 1 to bit 30 of NVIC_ICPR0
    ```
  - Do the ISR’s work
  - Clear pending flag for timer
    ```
    TIM4->SR &= ~TIM_SR_UIF;
    Assembly: write 0 to UIF (bit 0) of TIM4_SR
    ```
Example: Stopwatch

- Measure time with 100 us resolution
- Display elapsed time, updating screen every 10 ms
- Use SysTick (basic time unit in the system)
  - Counter increment every 100 us
  - LCD Update every 10 ms
    - Update LCD every nth periodic interrupt
    - $n = \frac{10 \text{ ms}}{100 \text{ us}} = 100$
    - Don’t update LCD in ISR! Too slow.
    - Instead set flag LCD_Update in ISR, poll it in main loop
    - Usually the ISR is only for update the timer or for delaying (precise timing!)