Programmable timing functions
Part 1: Timer-generated interrupts

STM32F4xx Technical Reference Manual:

Chapter 17 – Basic timers (ex. TIM6)
Chapter 15 – General-purpose timers (ex. TIM4)
Chapter 10 - Interrupt vectors (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
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

- **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

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

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

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

- **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></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></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

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
\[ F_{CK_CNT} = \frac{F_{CK_PSC}}{\text{Prescale}} \]

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 frequency = 2 x APBx bus frequency*
APB1 = 42MHz, TIMxCLK = 84MHz
APB2 = 84MHz, TIMxCLK = 168MHz
* as programmed in default system_stm32f4xx.c

TIMxCLK from RCC → Internal clock (CK_INT) → Trigger controller
\[ \text{ARR} \]
\[ \text{Update Event} \]
\[ \text{Update Event Interrupt} \]
\[ \text{CK_PSC} = \text{CK_INT} \text{ when count enabled} \]

APB1 = 42MHz, TIMxCLK = 84MHz
APB2 = 84MHz, TIMxCLK = 168MHz
* as programmed in default system_stm32f4xx.c
General-purpose timers TIM2 – TIM5

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

- Count-up mode overflow if TIMx_CNT reaches TIMx_ARR
  - On “overflow event”, UIF flag is set and TIMx_CNT resets to 0.
  - If UIE = 1 (update interrupt enabled), interrupt signal is sent to NVIC
- TIMx_PSC prescale value multiplies the input clock period (1/ Fclk) to produce counter clock period: $T_{cnt} = 1/F_{cnt} = (PSC+1)\times(1/F_{clk})$
- Periodic time interval is the ARR (Auto-Reload Register) value times the counter clock period: $T_{out} = (ARR+1)\times T_{cnt} = (ARR+1)\times(PSC+1)\times(1/F_{clk})$

**Example:** For 1 second time period, given $F_{clk} = 16MHz$:

$$T_{out} = (10000 \times 1600) \div 16000000 = 1 \text{ second}$$

Set $ARR = 9999$ and $PSC = 1599$ (other combinations can also be used)
$T_{EVENT} = \text{Prescale} \times \text{Count} \times T_{\text{CK\_INT}} = (\text{PSC}+1) \times (\text{ARR}+1) \times T_{\text{CK\_INT}}$

### Counter timing: Prescale = 1
ARR = 36

### Counter timing: Prescale = 4
ARR = 36
Counter timing (prescale changes 1->4)

- CK_PSC
- CNT_EN
- Timer clock = CK_CNT
- Counter register
- Update event (UEV)
- Prescaler control register
- Write a new value in TIMx_PSC
- Prescaler buffer
- Prescaler counter
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)
  - \( f_{\text{CK\_CNT}} = f_{\text{CK\_INT}} \div \text{prescale} \) (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
  - Can be “buffered” (new value held in buffer until update event)
# 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

**Counter Enable**

1 = enable, 0 = disable

CEN=1 to begin counting
(apply CK_INT to CK_PSC)

**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)

7 6 5 4 3 2 1 0

- **CC4F, CC3F, CC2F, CC1F**
  - Capture/Compare Channel n Interrupt Flags
  - (to be discussed later)

- **UIF**
  - Update Interrupt Flag
    - 1 = update interrupt pending
    - 0 = no update occurred
  - **Set** by hardware on update event
  - **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 DMA/Interrupt Control Register

TIMx_DIER address offset 0x0C  (default = all 0’s)

8  7  6  5  4  3  2  1  0

| UDE | | | | | | UIE |

Update DMA request enable
1 = enable
0 = disable

Update interrupt enable
1 = enable
0 = disable
Initialize the TIM4 with CMSIS

- Enable clock to Timer4
  \[ RCC->APB1ENR \ |= \ RCC\_APB1ENR\_TIM4EN; \]
- Set the auto-reload
  \[ TIM4->ARR = arr\_value; \]
- Set the prescaler
  \[ TIM4->PSC = psc\_value; \]
- Enable the update interrupt
  \[ TIM4->DIER \ |= \ TIM\_DIER\_UIE; \]
- Enable counting
  \[ TIM4->CR1 \ |= \ TIM\_CR1\_CEN; \]

Assembly:

```
RCC_TIM4EN  EQU  0x04
arr_value   EQU   4999
psc_value   EQU   9999
DIER_UIE    EQU   1
CR1_CEN     EQU   1

ldr r0,=RCC
ldr r1,[r0,#APB1ENR]
or r1,#RCC\_TIM4EN
str r1,[r0,#APB1ENR]

ldr r0,=TIM4
mov r1,#arr_value
str r1,[r0,#ARR]

mov r1,#psc_value
str r1,[r0,#PSC]

ldr r1,[r0,#DIER]
or r1,#DIER\_UIE
str r1,[r0,#DIER]

ldr r1,[r0,#CR1]
or r1,#CR1\_CEN
str r1,[r0,#CR1]
```
Interrupt Handler

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

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

- ISR activities
  
  - Clear pending IRQ
    ```c
    NVIC_ClearPendingIRQ(TIM2_IRQn);
    
    Assembly: write 1 to bit 30 of NVIC_ICPR0
    ```
  
  - Do the ISR’s work
  
  - Clear pending flag for timer
    ```c
    TIM2->SR &= ~TIM_SR_UIF;
    
    Assembly: write 0 to UIF (bit 0) of TIM4_SR
    ```
Interrupt handler format

```c
void TIM2_IRQHandler () {  // handler for TIM2

    // Perform the desired actions
    • update time/display
    • sample external sensor data
    • perform a control action
    • etc.

    // Clear timer’s update interrupt flag (cancel interrupt request)
    TIM2->SR &= ~0x01;  // UIF is bit 0 of SR
    or TIM2->SR &= ~TIM_SR_UIF;

    // (Should be unnecessary): clear pending flag in NVIC
    NVIC_ClearPendingIRQ(TIM2_IRQn);
}
```
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!)
SysTick Timer (Cortex function)

- Timer/Counter operation
  - 24-bit counter *decrements* at bus clock frequency
    - With 80 MHz bus clock, decrements every 12.5 ns
  - Counting is from $n \rightarrow 0$
    - Setting $n$ appropriately will make the counter a modulo $n+1$ counter. That is:
      - \[ \text{next\_value} = (\text{current\_value}-1) \mod (n+1) \]
      - Sequence: \( n, n-1, n-2, n-3 \ldots 2, 1, 0, n, n-1 \ldots \)
SysTick Timer

 Initialization (4 steps)
- **Step1**: Clear ENABLE to stop counter
- **Step2**: Specify the RELOAD value
- **Step3**: Clear the counter via NVIC_ST_CURRENT_R
- **Step4**: Set CLK_SRC=1 and specify interrupt action via INTEN in NVIC_ST_CTRL_R

<table>
<thead>
<tr>
<th>Address</th>
<th>31-24</th>
<th>23-17</th>
<th>16</th>
<th>15-3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E000E010</td>
<td>0</td>
<td>0</td>
<td>COUNT</td>
<td>0</td>
<td>CLK_SRC</td>
<td>INTEN</td>
<td>ENABLE</td>
<td>NVIC_ST_CTRL_R</td>
</tr>
<tr>
<td>$E000E014</td>
<td>0</td>
<td></td>
<td></td>
<td>24-bit RELOAD value</td>
<td>NVIC_ST_RELOAD_R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E000E018</td>
<td>0</td>
<td></td>
<td></td>
<td>24-bit CURRENT value of SysTick counter</td>
<td>NVIC_ST_CURRENT_R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SysTick_Init
; disable SysTick during setup
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0          ; Clear Enable
    STR R0, [R1]
; set reload to maximum reload value
    LDR R1, =NVIC_ST_RELOAD_R
    LDR R0, =0x00FFFFFF;  ; Specify RELOAD value
    STR R0, [R1]         ; reload at maximum
; writing any value to CURRENT clears it
    LDR R1, =NVIC_ST_CURRENT_R
    MOV R0, #0
    STR R0, [R1]         ; clear counter
; enable SysTick with core clock
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0x0005      ; Enable but no interrupts (later)
    STR R0, [R1]         ; ENABLE and CLK_SRC bits set
    BX  LR

24-bit Countdown Timer
SysTick Timer

;-----------------SysTick_Wait-----------------
; Time delay using busy wait.
; Input: R0  delay parameter in units of the core clock
; 80 MHz(12.5 nsec each tick)
; Output: none
; Modifies: R1

SysTick_Wait
SUB R0, R0, #1  ; delay-1
LDR R1, =NVIC_ST_RELOAD_R
STR R0, [R1]  ; time to wait
LDR R1, =NVIC_ST_CURRENT_R
STR R0, [R1]  ; any value written to CURRENT clears
LDR R1, =NVIC_ST_CTRL_R

SysTick_Wait_loop
LDR R0, [R1]  ; read status
ANDS R0, R0, #0x00010000  ; bit 16 is COUNT flag
BEQ SysTick_Wait_loop  ; repeat until flag set
BX   LR
SysTick Timer

;-------------------SysTick_Wait10ms-------------------
; Call this routine to wait for R0*10 ms
; Time delay using busy wait. This assumes 80 MHz clock
; Input: R0  number of times to wait 10 ms before returning
; Output: none
; Modifies: R0
DELAY10MS  EQU 800000     ; clock cycles in 10 ms
SysTick_Wait10ms
    PUSH {R4, LR}           ; save R4 and LR
    MOVS R4, R0            ; R4 = R0 = remainingWaits
    BEQ SysTick_Wait10ms_done ; R4 == 0, done
SysTick_Wait10ms_loop
    LDR R0, =DELAY10MS      ; R0 = DELAY10MS
    BL  SysTick_Wait       ; wait 10 ms
    SUBS R4, R4, #1        ; remainingWaits--
    BHI SysTick_Wait10ms_loop ; if(R4>0), wait another 10 ms
SysTick_Wait10ms_done
    POP {R4, PC}
SysTick Timer in C

```c
#define NVIC_ST_CTRL_R (*((volatile uint32_t *)0xE000E010))
#define NVIC_ST_RELOAD_R (*((volatile uint32_t *)0xE000E014))
#define NVIC_ST_CURRENT_R (*((volatile uint32_t *)0xE000E018))

void SysTick_Init(void){
    NVIC_ST_CTRL_R = 0; // 1) disable SysTick during setup
    NVIC_ST_RELOAD_R = 0x00FFFFFF; // 2) maximum reload value
    NVIC_ST_CURRENT_R = 0; // 3) any write to CURRENT clears it
    NVIC_ST_CTRL_R = 0x00000005; // 4) enable SysTick with core clock
}

// The delay parameter is in units of the 80 MHz core clock (12.5 ns)
void SysTick_Wait(uint32_t delay){
    NVIC_ST_RELOAD_R = delay-1; // number of counts
    NVIC_ST_CURRENT_R = 0; // any value written to CURRENT clears it
    while((NVIC_ST_CTRL_R&0x00010000)==0){ // wait for flag
    }
}

// Call this routine to wait for delay*10ms
void SysTick_Wait10ms(uint32_t delay){
    unsigned long i;
    for(i=0; i<delay; i++){
        SysTick_Wait(800000); // wait 10ms
    }
}
```

Bard, Gerstlauer, Valvano, Yerraballi