Advanced Microcontrollers

Agenda

- Timer Fundamentals
- STM32 RTC
- SysTick Timer
- STM32 Timers
    - Polling vs Interrupt
    - Output Compare
    - Input Capture
- Watchdog Timer

Time Manangement

- Timing requirements in any system
    - Absolute time (Wall time)
    - Relative time
- Absolute time
    - Get calendar date + clock time.
    - Display date & time, alarm set, etc.
- Relative time
    - To get duration between two events (e.g. two ADC readings, two external interrupts, ...)
    - Schedule a task after certain duration
    - Delay generation
    - Periodic interrupts

STM32 Time Management

- RTC
- SysTick

- Timers (Basic, General purpose and Advanced)
- Watchdog Timer
- Debug Watch Timer

Timer Concepts

- Timer is peripheral.
- Timer vs Counter
  - Counter is typically used to count pulses (at variable rate).
  - Counter types: Up counter (0 to MAX) and Down counter (MAX to 0).
  - Timer is a counter (circuit) that counts pulses/edges with fixed frequency.
  - Usually timer hardware produces interrupt on certain event compare, overflow, etc.
  - Timer is used measure time or frequency of input signal.
- Clock (1 Hz) --> Timer/Counter --> cnt
  - Incremented per second.
  - If count = 100, then time = 100 sec.
- Clock (1000 Hz) --> Timer/Counter --> cnt
  - 1000 pulses per second.
  - If count = 5000, then time = 5 sec.
- F = pulses / sec
- Period of one clock T = 1 / F
- Number of clocks "N" : time (t) = N * T = N / F
- To measure "t" seconds, how many clocks are required?
  - N = t * F
- N = (t / 1000) * F / PR = (F / 1000) * t / PR
  - t is time to be measured in ms.
  - F is timer peripheral clock.

RTC

- Provides Absolute time in the system (calendar + clock time).
- Typically RTC is connected to CMOS battery and keep functioning even if processor is off.

STM32 RTC

- The real-time clock (RTC) is an independent "BCD" timer/counter.
  - 04:15:30
    - Binary : 0000 0100 : 0000 1111 : 0001 1110
    - BCD (0-9) : 0000 0100 : 0001 0101 : 0011 0000
- Time + Calendar Date
- Two alarms (Max 1 month) with Interrupt
- Periodic wakeup Interrupt
- Two 32-bit registers for date and time for time-keeping.
  - RTC_TR -- Time Register
  - RTC_DR -- Date Register
- Auto management of Leap year and Daylight saving.
- Year range is 2000 to next 100 years.
- Additional registers for Alarm setting.
- As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low-power mode or under reset).

RTC Clock Settings

- RTC clock source -- Need stable clock 1 Hz.
  - LSI -- 32 KHz (RC-osc -- less stable)
  - LSE -- 32768 Hz (Xtal -- stable) -- Not fitted on DISC1 board.
  - HSE / Prescalar -- Xtal
- RTC clock = LSE (32768 Hz) --> Async Prescalar (128) --> 256 Hz --> Sync Prescalar (256) --> 1 Hz.
- RTC clock = LSI (32000 Hz) --> Async Prescalar (125) --> 256 Hz --> Sync Prescalar (256) --> 1 Hz.
- RTC clock = HSE (8 MHz) --> / 20 --> 400000 Hz --> Async Prescalar (100) --> 4000 Hz --> Sync Prescalar (4000) --> 1 Hz.
- RTC Config Registers / RTC HAL
- RTCCLK --> Async Prescalar --> Sync Prescalar --> 1 Hz --> RTC Counters
  - Async Prescalar -- 0 to 127
  - Sync Prescalar -- 0 to 32767
- RTCCLK should be stable clock -- XTAL is preferred.

- Option 1 -- LSE = 32.768 KHz (ideal but not fitted on DISC1 board)
  - 32768 Hz --> 32768 / 128 --> 256 --> 256 / 256 --> 1 Hz
- Option 2 -- HSE = 8 MHz / Prescalar (Prescalar can be 0 to 31)
  - RTCCLK = 8 MHz / 20 = 400000 Hz
  - 400000 Hz --> 400000 / 100 --> 4000 --> 4000 / 4000 --> 1 Hz

STM32 RTC

```
// Initialize RTC
```

```
RTC_DateTypeDef date;
RTC_TimeTypeDef time;
HAL_RTC_GetDate(&hrtc, &date, RTC_FORMAT_BIN);
HAL_RTC_GetTime(&hrtc, &time, RTC_FORMAT_BIN);
```

SysTick

- Periodic timer interrupt -- Tick interrupt
- Usually 1 ms
- SysTick is part of Cortex-M3/M4 core.
- Works on Core Clock (CCLK).
- Internally 24-bit downcounter -- When count is zero, will generate interrupt.
- Example 1:
  - Time to measure: t = 1 ms.
  - Given:

        - CCLK = 25 MHz
        - PR = 1 (No prescalar)
    - N = (CCLK / 1000) x 1 / 1
    - N = 25000000 / 1000 = 25000.

- SysTick_Config(25000); // CMSIS function

```
// global var to count number of tick interrupts
volatile uint32_t ticks = 0;
// executed after every 1 ms
void SysTick_Handler(void) {
    ticks++;
}
// delay of desired ms
void SysTick_Delay(uint32_t ms) {
    uint32_t now = ticks;   // e.g. 5000
    uint32_t waitUntil = now + ms;  // e.g. 5000 + 2000 = 7000
    while(ticks < waitUntil)
        ;           // do nothing -- until ticks is equal to 7000
}
```

```
// main()
// HAL_SysTick_Config(25000); --> SysTick_Config()
// "SystemCoreClock" global variable in CMSIS that represent Core CLK.
SysTick_Config(SystemCoreClock / 1000);

while (1)
{
    SysTick_Delay(1000);
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
}
```

STM32 Timers

- STM32 have variety of timer modes/functionalities.
  - Time base generation (Delay/Periodic Interrupts)
  - External clock counter
  - Input capture
  - Output compare
  - Pulse Width Modulation (PWM)
  - Encoder mode
  - Infrared mode
  - ...
- STM32 has different timers.
  - Basic timers (TIM6 and TIM7)
    - 16-bit timers (0-65535)
    - Only time base generation
    - No capture/compare channels, PWM, ...
  - General purpose timers
    - 16-bit timers (TIM3 and TIM4)
    - 32-bit timers (TIM2 and TIM5)
    - Time base generation
    - Capture/compare channels
    - PWM mode, Infrared mode, Encoder mode, One pulse mode, ...
  - Advanced timers (TIM1 and TIM8)
    - 32-bit timers
    - Time base generation
    - Capture/compare channels
    - PWM mode, Infrared mode, Encoder mode, One pulse mode, ...
    - Advanced controls: PWM brake (stop), ...
- https://deepbluembedded.com/stm32-timers-tutorial-hardware-timers-explained/
- https://embedded-lab.com/blog/stm32-timers/6/

Timer

- N = (TCLK / 1000) x t / PR
  - N = Number of clock/pulses to count
  - TCLK = Timer input frequency
  - PR = Prescalar
  - t = Time in milliseconds
- Example 1:
  - Time to measure: t = 7000 ms.
  - Given:
    - TCLK = 12.5 MHz
    - PR = 12500
  - N = (12500000 / 1000) x 7000 / 12500 = 7000
- Example 1:
  - Time to measure: t = 3500 ms.
  - Given:
    - TCLK = 12.5 MHz
    - PR = 12500
  - N = (12500000 / 1000) x 3500 / 12500 = 3500

Time-base generation

- Basic timer -- TIM6
- Desired delay t = 10000 ms
- F = 12.5 MHz
- Prescalar = 12500 (-1)
- Number of clocks N = (F / 1000) * t / PR = 10000 (-1)

STM32 Timer - Polling

```
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM6_Init 1 */
```

```
/* USER CODE END TIM6_Init 1 */
htim6.Instance = TIM6;
htim6.Init.Prescaler = 12499;
htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
htim6.Init.Period = 9999;
htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
{
Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
{
Error_Handler();
}
```

```
// main
HAL_TIM_Base_Start(&htim6);

while (1)
{
    // wait for update flag
    while( (htim6.Instance->SR & TIM_SR_UIF) == 0 )
    ;
    // clear update flag
    htim6.Instance->SR = 0;
    // toggle led
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
}
```

STM32 Timer - Interrupt

```
TIM_MasterConfigTypeDef sMasterConfig = {0};
htim6.Instance = TIM6;
htim6.Init.Prescaler = 12499;
htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
htim6.Init.Period = 9999;
htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim6) != HAL_OK) {
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK) {
    Error_Handler();
}
```

```
// main
HAL_TIM_Base_Start_IT(&htim6);
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
}
```

STM32 Timer - Output Compare

```
TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};
```

```
htim4.Instance = TIM4;
htim4.Init.Prescaler = 12499;
htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
htim4.Init.Period = 500;
htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim4) != HAL_OK) {
Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK) {
Error_Handler();
}
if (HAL_TIM_OC_Init(&htim4) != HAL_OK) {
Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK) {
Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_TOGGLE;
sConfigOC.Pulse = 100;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_OC_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK) {
Error_Handler();
}
sConfigOC.Pulse = 200;
if (HAL_TIM_OC_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2) != HAL_OK) {
Error_Handler();
}
sConfigOC.Pulse = 300;
if (HAL_TIM_OC_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_3) != HAL_OK) {
Error_Handler();
```

```
}
sConfigOC.Pulse = 400;
if (HAL_TIM_OC_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK) {
Error_Handler();
}

HAL_TIM_MspPostInit(&htim4);
```

```
// main
HAL_TIM_OC_Start(&htim4, TIM_CHANNEL_1);
HAL_TIM_OC_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_OC_Start(&htim4, TIM_CHANNEL_3);
HAL_TIM_OC_Start(&htim4, TIM_CHANNEL_4);
```