Advanced Microcontrollers

Agenda

- PWM
- Watchdog Timer
- Brownout Detector
- Event vs Interrupt
- PendSV exception

Timers

STM32 Timer - PWM

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

htim8.Instance = TIM8;
htim8.Init.Prescaler = 24;
htim8.Init.CounterMode = TIM_COUNTERMODE_UP;
htim8.Init.Period = 100;
htim8.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim8.Init.RepetitionCounter = 0;
htim8.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim8) != HAL_OK) {
Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim8, &sClockSourceConfig) != HAL_OK) {
Error_Handler();
```

```
}
if (HAL_TIM_PWM_Init(&htim8) != HAL_OK) {
Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim8, &sMasterConfig) != HAL_OK) {
Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim8, &sConfigOC, TIM_CHANNEL_1) != HAL_OK) {
Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim8, &sBreakDeadTimeConfig) != HAL_OK) {
Error_Handler();
}

HAL_TIM_MspPostInit(&htim8);
```

```
HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);
```

```
while (1)
{
    htim8.Instance->CCR1 = dutyCycle;
    dutyCycle = dutyCycle + delta;
    if(dutyCycle >= 100 || dutyCycle <= 0)
        delta = -delta;
    HAL_Delay(200);
}
```

Watchdog Timer

- Downcounter -- If clears, reset the processor.
- Used to ensure that tasks are completed in well-defined time -- otherwise system reset (so that error can be reported).
- STM32 WDT
    - Independent WDT --> Only set max time
        - Reset on timeout (Beyond the max time)
        - Separate clock (LSI)
        - Less timing accuracy
    - Window WDT --> Set the time window i.e. min time to max time.
        - Reset on timeout (Beyond the max time OR Before the min time)
        - Can produce interrupt on timeout
        - APB1 -- prescaled (Hence cannot detect failure on CPU clock)
        - More timing accuracy
- Independent WDT
    - 12-bit timer (MAX = 4096-1)
    - Input frequency = 32 KHz
    - Max Prescalar = 256
    - Max Time (ms) = N * PR * 1000 / F
        - = 4096 * 256 * 1000 / 32000 = 32768 ms
    - 125 us to 32.8 secs -- time period
    - Programming sequence
        - Initialization

        - step 1. Enable IWDG by writing 0xCCCC into IWDG_KR.
        - step 2. Enable register access by writing 0x5555 into IWDG_KR.
        - step 3. Set Prescalar value in IWDG_PR.
        - step 4. Set Reload value in IWDG_RLR.
        - step 5. Wait for registers to be updated i.e. IWDG_SR == 0x00000000;
    - Feed/Refresh WDT
        - step 6. Refresh counter by writing 0xAAAA into IWDG_KR.
    - Check Reset reason
        - step 7. If reset due to IWDG, Bit 29 in RCC_CSR is set. Check the bit immediately after initialization and report the error.

```
int Uart_Gets(char str[]) {
    uint8_t ch;
    int i = 0;
    do {
        HAL_UART_Receive(&huart2, &ch, 1, HAL_MAX_DELAY);
        str[i++] = ch;
    } while(ch != '\r');
    str[i++] = '\n';
    str[i] = '\0';
    return i;
}
```

```
while (1) {
    count = Uart_Gets(str);
    strupr(str);
    HAL_UART_Transmit(&huart2, (uint8_t*)str, count, HAL_MAX_DELAY);
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);

    // feed/refresh WDT
    HAL_IWDG_Refresh(&hiwdg);
}
```

```
// check if reset is due to wdt
if(__HAL_RCC_GET_FLAG(RCC_FLAG_IWDGRST)) {
	iwdgReset = 1;
	__HAL_RCC_CLEAR_RESET_FLAGS();
}

while(iwdgReset) {
	HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
	HAL_Delay(200);
}
```

Brown Out Detector

- BOD is used to detect drop in Vdd level.
- Voltage level is reduced due to
  - Drop in supply voltage (power supply).
  - Browning of the components.

Interrupt vs Event

- Interrupt
  - Interrupt is an exception. It should be handled.
    - Task is under execution.
    - Peripheral generate interrupt and send to NVIC.
    - NVIC send interrupt to ARM core.
    - ARM core pause current task.
    - Follow exception handling sequence -- 1. Stacking, 2. Vector fetch, 3. Register updates
    - Exception/Interrupt handler is executed
    - Return from exception --> 1. Unstacking, 2. Register updates
    - Paused task is resumed.
  - Interrupt has priority config, NVIC config, Peripheral ack.

    - Interrupt must have Interrupt handler (code/program).
    - WFI --> CM3 Assembly instruction --> Wait For Interrupt.
        - Put CPU in low power state.
        - Will continue further execution when any interrupt is raised.
- Event
    - Event represent some event like Ext event, ADC conversion complete, Timer event, ...
    - On event some action can be configured.
    - The action is executed/taken at hardware level.
    - There is NO code running for event handling.
    - WFE --> CM3 Assembly instruction --> Wait For Event.
        - Put CPU in low power state.
        - Will continue further execution when any event is raised.

PendSV exception

- Applications
    - Context switch (Dispatcher)
    - Long running ISR
- Embedded OS
    - Kernel + Interrupt/Exception handler
    - User task
- Entry to kernel code
    - SVC (System call from user task)
    - Exception/Interrupt -- SysTick
- Simple Round-Robin scheduler
    - Task1 --> SysTick Handler (sched) --> Task2 --> SysTick Handler (sched) --> Task1 --> ...
    - May cause usage fault (if some ISR is interrupted due to SysTick)
- PendSV is set to lowest priority interrupt.
- PendSV is pended (triggered) by software into ICSR (interrupt status & control regr) (by scheduler/ISR).
    - ICSR |= BV(PENDSVSET); // PENDSVSET -- bit 28
- Typical Round-Robin scheduler
    - Task1 --> SysTick (sched) --> PendSV --> Task2 --> SysTick (sched) --> PendSV --> Task1 --> ...

- Typical ISR execution
  - Thread --> Handler --> Thread
  - Thread --> Handler1 --> Handler2 --> Handler1 --> Thread (Intr2 > Intr1)
- ISR should be as short as possible, so that other interrupt latency will not increase.

```
// this is not good ISR example -- delay
void Eint_Handler(void) {
    // ack interrupt
    // led on
    // delay 1 sec
    // led off
}
```

- PendSV can be used to execute time-consuming portion of the ISR, so that other interrupt latency is not affected.

```
// top-half of ISR --> time-sensitive part of ISR
void Eint_Handler(void) {
    // ack interrupt
    // trigger PendSV (bottom-half)
}
// bottom-half of ISR --> time-consuming part of ISR
void PendSV_Handler(void) {
    // led on
    // delay 1 sec
    // led off
}
```