stackoverflow.com

Using STM32 HAL Timer and Adjusting the Duty Cycle of a PWM signal

NadimNadim 11311 gold badge22 silver badges77 bronze badges

5-6 minutos

I used the STM32Cube initialization code generator to generate an initialized Timer function. To generate a fixed duty cycle PWM signal I added `HAL_TIM_Base_Start(&htim1); // Starts the TIM Base generation` and `HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1)//Starts the PWM signal generation` to the Timer initialization function as shown below.

```
/* Private variables
---------------------------------------------------
*/
int pulse_width=0;

/* TIM1 init function */
static void MX_TIM1_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;
  TIM_BreakDeadTimeConfigTypeDef
```

```
sBreakDeadTimeConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 0;//we want a max
frequency for timer, so we set prescaller to 0
  //And our timer will have tick frequency
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 1066;//max value for timer is
16bit = 65535, TIM_Period = timer_tick_frequency /
PWM_frequency - 1
  //In our case, for 15Khz PWM_frequency, set
Period to TIM_Period = 16MHz / 15KHz - 1 = 1066
  htim1.Init.ClockDivision =
TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)/* to use
the Timer to generate a simple time base for TIM1
*/
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;//the default clock is the
internal clock from the APBx, using this function
  if (HAL_TIM_ConfigClockSource(&htim1,
&sClockSourceConfig) != HAL_OK)//Initializes the
TIM PWM Time Base according to the specified
//parameters in the TIM_HandleTypeDef and create
the associated handle.
  {
    Error_Handler();
```

```
  }

  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger =
TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  //sConfig: TIM PWM configuration structure
  //set duty cycle: pulse_length = ((1066 + 1) *
duty_cycle) / (100 - 1)
    sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = pulse_width;/* 50% duty cycle
is 538, set to 0 initially*///
  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(&htim1, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
```

```
  }

  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }

  sBreakDeadTimeConfig.OffStateRunMode =
TIM_OSSR_ENABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode =
TIM_OSSI_ENABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_1;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState =
TIM_BREAK_ENABLE;
  sBreakDeadTimeConfig.BreakPolarity =
TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput =
TIM_AUTOMATICOUTPUT_ENABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1,
&sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_TIM_MspPostInit(&htim1);//output pin
assignment
    HAL_TIM_Base_Start(&htim1); //Starts the TIM
Base generation
  if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) !=
HAL_OK)//Starts the PWM signal generation
```

```
  {
    /* PWM Generation Error */
    Error_Handler();
  }

  /* Start channel 2 */
  if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2) !=
HAL_OK)
  {
    /* PWM Generation Error */
    Error_Handler();
  }

}
```

This is enough to run the PWM at a fixed duty cycle specified in the comments above when I hard code the right value to replace `pulse_width` value in`sConfigOC.Pulse = pulse_width`. In another function, I have an algorithm that would update the `pulse_width` global variable. The function is called: `adjust_PWM();`. The algorithm calculate values measured from the ADC and stored as global variables. That function is called: `Data_Update();`. In `main()`, after all functions are initialized. I call these three functions endlessly

```
Data_Update();
adjust_PWM();
MX_TIM1_Init();
```

I tried that and obtained weird waveforms on the oscilloscope, but that might be because The ADC pins where floating, causing floating measurements to interfere with the duty cycle by the algorithm. Also recalling the initialization of the timer continuously would interrupt the PWM signal. Is there a better

way to change the duty cycle while running the code without using global variables, or without initializing the timer every time I want to update the duty cycle. Any link would be appreciated.