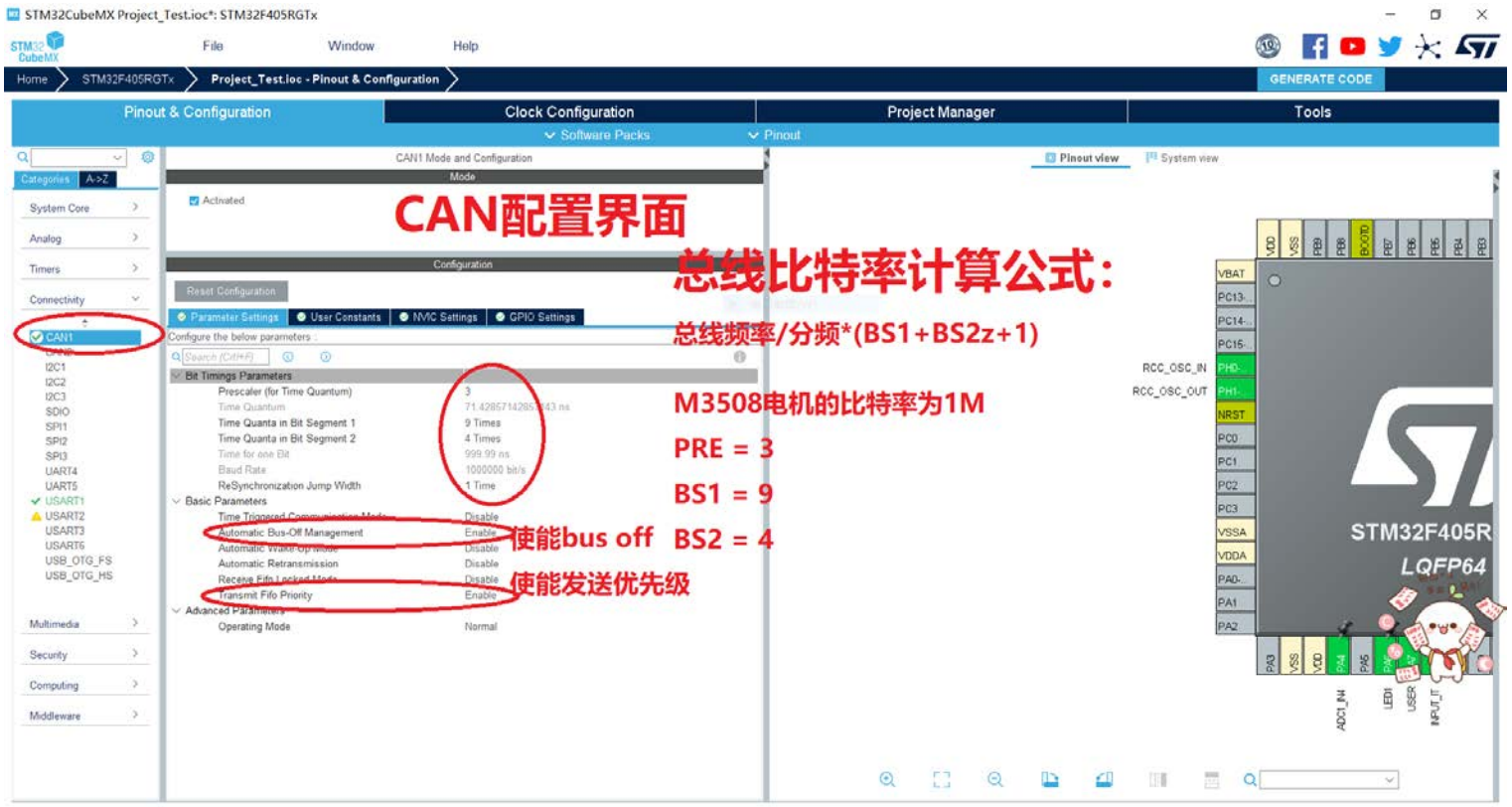
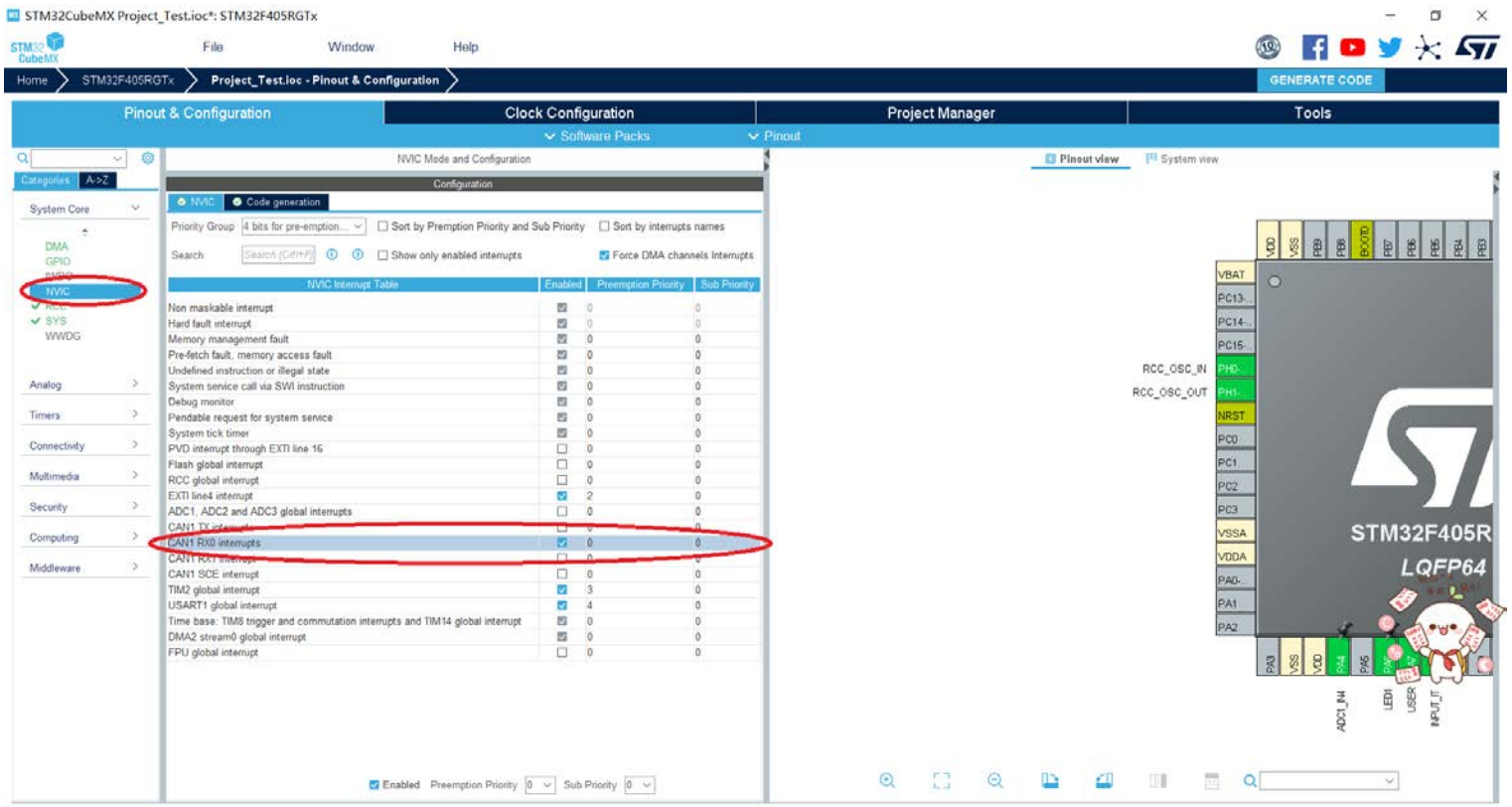
目录

使用 STM32CubeMX 配





初始化代码

在 can.c 源文件中，有如下配置代码

```
1.  void MX_CAN1_Init(void)
2.  {
3.
4.    /* USER CODE BEGIN CAN1_Init 0 */
5.
6.    /* USER CODE END CAN1_Init 0 */
7.
8.    /* USER CODE BEGIN CAN1_Init 1 */
9.
10.   /* USER CODE END CAN1_Init 1 */
11.   hcan1.Instance = CAN1;
12.   hcan1.Init.Prescaler = 3;
13.   hcan1.Init.Mode = CAN_MODE_NORMAL;
14.   hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
15.   hcan1.Init.TimeSeg1 = CAN_BS1_9TQ;
16.   hcan1.Init.TimeSeg2 = CAN_BS2_4TQ;
17.   hcan1.Init.TimeTriggeredMode = DISABLE;
18.   hcan1.Init.AutoBusOff = ENABLE;
19.   hcan1.Init.AutoWakeUp = DISABLE;
20.   hcan1.Init.AutoRetransmission = DISABLE;
21.   hcan1.Init.ReceiveFifoLocked = DISABLE;
22.   hcan1.Init.TransmitFifoPriority = ENABLE;
23.   if (HAL_CAN_Init(&hcan1) != HAL_OK)
24.   {
25.     Error_Handler();
26.   }
27.   /* USER CODE BEGIN CAN1_Init 2 */
28.
29.   /* USER CODE END CAN1_Init 2 */
30.
31. }
```

初始化 CAN 总线过滤器

在 HAL 库中，我们需要自己初始化 CAN 总线的过滤器。在配置 CAN1 时，过滤器可设置为 0-13，在配置 CAN2 时，需配置 Slave 的起始过滤器为 14，再设置 CAN2 的过滤器为 14。用户初始化代码如下：

```
1.  void CAN1_FilterInit_And_Start(void)
```

```
{
  CAN_FilterTypeDef can1_filter;
  can1_filter.FilterBank = 0;
  can1_filter.FilterMode = CAN_FILTERMODE_IDMASK;
  can1_filter.FilterScale = CAN_FILTERSCALE_32BIT;
  can1_filter.FilterIdHigh = 0x0000;
  can1_filter.FilterIdLow = 0x0000;
  can1_filter.FilterMaskIdHigh = 0x0000;
  can1_filter.FilterMaskIdLow = 0x0000;
  can1_filter.FilterFIFOAssignment = 0;
  can1_filter.FilterActivation = ENABLE;
  can1_filter.SlaveStartFilterBank = 14;
  HAL_CAN_ConfigFilter(&hcan1,&can1_filter);//初始化 CAN1 过滤器
  HAL_CAN_Start(&hcan1);//启动 CAN1
  HAL_CAN_ActivateNotification(&hcan1,CAN_IT_RX_FIFO0_MSG_PENDING);//激活
CAN1 FIFO0 接收
}
```

在 main 函数中调用滤波器初始化函数完成 CAN1 总线的开启

常用 CAN 相关操作函数 stm32f4xx_hal_can.c

1.1 开启 CAN 总线函数 HAL_CAN_Start

```
HAL_StatusTypeDef HAL_CAN_Start(CAN_HandleTypeDef *hcan)
{
uint32_t tickstart;

if (hcan->State == HAL_CAN_STATE_READY)
{
  /* Change CAN peripheral state */
  hcan->State = HAL_CAN_STATE_LISTENING;

  /* Request leave initialisation */
  CLEAR_BIT(hcan->Instance->MCR, CAN_MCR_INRQ);

  /* Get tick */
  tickstart = HAL_GetTick();

  /* Wait the acknowledge */
  while ((hcan->Instance->MSR & CAN_MSR_INAK) != 0U)
  {
    /* Check for the Timeout */
```

```
    if ((HAL_GetTick() - tickstart) > CAN_TIMEOUT_VALUE)
    {
      /* Update error code */
      hcan->ErrorCode |= HAL_CAN_ERROR_TIMEOUT;

      /* Change CAN state */
      hcan->State = HAL_CAN_STATE_ERROR;

      return HAL_ERROR;
    }
  }

  /* Reset the CAN ErrorCode */
  hcan->ErrorCode = HAL_CAN_ERROR_NONE;

  /* Return function status */
  return HAL_OK;
}
else
{
  /* Update error code */
  hcan->ErrorCode |= HAL_CAN_ERROR_NOT_READY;

  return HAL_ERROR;
}
}
```

入口参数：CAN 句柄
返回值：HAL_OK(成功) or HAL_ERROR(失败)
使用示例：

```
HAL_CAN_Start(&hcan1);
```

1.2 关闭 CAN 总线函数 HAL_CAN_Stop

```
HAL_StatusTypeDef HAL_CAN_Stop(CAN_HandleTypeDef *hcan)
{
uint32_t tickstart;

if (hcan->State == HAL_CAN_STATE_LISTENING)
{
  /* Request initialisation */
  SET_BIT(hcan->Instance->MCR, CAN_MCR_INRQ);

```

```
  /* Get tick */
  tickstart = HAL_GetTick();

  /* Wait the acknowledge */
  while ((hcan->Instance->MSR & CAN_MSR_INAK) == 0U)
  {
    /* Check for the Timeout */
    if ((HAL_GetTick() - tickstart) > CAN_TIMEOUT_VALUE)
    {
      /* Update error code */
      hcan->ErrorCode |= HAL_CAN_ERROR_TIMEOUT;

      /* Change CAN state */
      hcan->State = HAL_CAN_STATE_ERROR;

      return HAL_ERROR;
    }
  }

  /* Exit from sleep mode */
  CLEAR_BIT(hcan->Instance->MCR, CAN_MCR_SLEEP);

  /* Change CAN peripheral state */
  hcan->State = HAL_CAN_STATE_READY;

  /* Return function status */
  return HAL_OK;
}
else
{
  /* Update error code */
  hcan->ErrorCode |= HAL_CAN_ERROR_NOT_STARTED;

  return HAL_ERROR;
}
}
```

入口参数：CAN 句柄
返回值：HAL_OK(成功) or HAL_ERROR(失败)
使用示例：

```
HAL_CAN_Stop(&hcan1);
```

1.4 获取 CAN 已接收数据函数 HAL_CAN_GetRxMessage

```
HAL_StatusTypeDef HAL_CAN_GetRxMessage(CAN_HandleTypeDef *hcan, uint32_t RxFifo, CAN_RxHeaderTypeDef *pHeader, uint8_t aData[])
```

入口参数：CAN 句柄、FIFO 邮箱、CAN 接收句柄、数据存放数组
返回值：HAL_OK(成功) or HAL_ERROR(失败)
使用示例：

```
uint8_t data[8];
CAN_RxHeaderTypeDef can1_rx;
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(hcan);

  /* NOTE : This function Should not be modified, when the callback is needed,
            the HAL_CAN_RxFifo0MsgPendingCallback could be implemented in the
            user file
   */
  if(hcan->Instance == CAN1)
  {
    HAL_CAN_GetRxMessage(&hcan1,CAN_RX_FIFO0,&can1_rx,data);
  }
}
```

在用户重写的 CANFIFO0 等待回调函数中
＞判断是否是 CAN1 发生了中断
＞调用函数 HAL_CAN_GetRxMessage
＞用户代码

1.5 添加数据发送函数 HAL_CAN_AddTxMessage

```
HAL_StatusTypeDef HAL_CAN_AddTxMessage(CAN_HandleTypeDef *hcan, CAN_TxHeaderTypeDef *pHeader, uint8_t aData[], uint32_t *pTxMailbox)
```

入口参数：CAN 句柄、CAN 发送句柄、需要发送的数组、FIFO 邮箱
返回值：HAL_OK(成功) or HAL_ERROR(失败)
使用示例：

```
void CAN1_tx_f(float * data)
{
```

```
3.    CAN_TxHeaderTypeDef can1_tx;
4.    can1_tx.StdId = 0x200;
5.    can1_tx.RTR = CAN_RTR_DATA;
6.    can1_tx.IDE = CAN_ID_STD;
7.    can1_tx.DLC = 0x08;
8.    uint8_t can1_v[8];
9.    //用户数据处理
10.   HAL_CAN_AddTxMessage(&hcan1,&can1_tx,can1_v,(uint32_t)CAN_TX_MAILBOX0);
11. }
```

注意事项

在比特率为 1M、发送频率 1000Hz 的总线带宽中，可挂载的电机数量上限为 7，切忌不可多挂。

26-APR-2021

厦大嘉庚 TCR 嵌入式