Real-time Monitoring and Display of Potentiometer Reading via CAN Bus Communication and I2C LCD Display

1) AIM OF THE PROJECT

In this project, we will be utilizing two STM32F303 microcontroller boards, one MCP2515 CANBUS-SPI module, and one 2x16 LCD display with I2C connection. The first STM32F303 board will be responsible for reading the ADC value of a potentiometer using DMA and transmitting this value to the second STM32F303 board via the CANBUS module. The second STM32F303 board will then receive this value and display it on the connected 2x16 LCD display using I2C communication. This project demonstrates the use of various communication protocols and showcases the versatility of the STM32F303 microcontroller in handling multiple tasks simultaneously.

2)BOARD PINOUT & CONFIGURATION

STM32F303RE board #2

PB8-> CAN_RX

PB9-> CAN_TX

PB6-> I2C1_SCL

PB7-> I2C1_SDA

STM32F303RE board #1

PB8-> CAN_RX

PB9-> CAN_TX

PA0-> ADC1_IN1

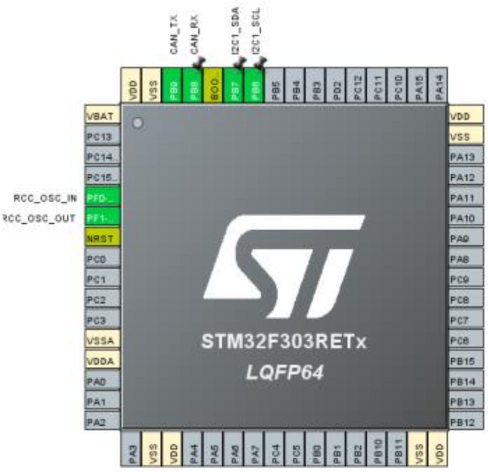

Figure 1. STM32303RE board #2 Pinouts

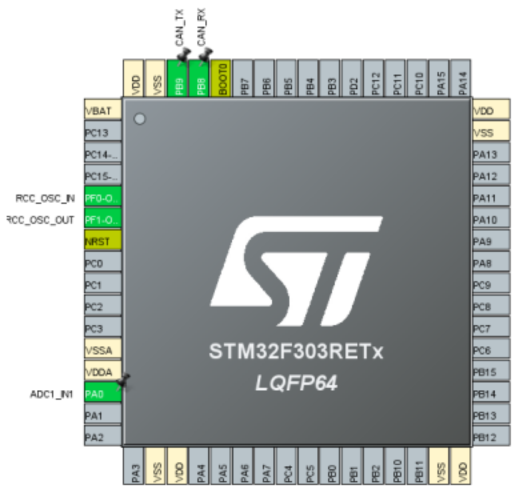

Figure 2. STM32303RE board #1 Pinouts

In order to properly receive data via CAN communication on the STM32F303RE board, the CAN_RX0 interrupt must be enabled in the NVIC settings. Additionally, as the data being transmitted only consists of 1 byte, the resolution for the ADC reading must be set to 8 bits. Furthermore, as the data will be read using DMA (Direct Memory Access), the DMA request must also be enabled in the configuration.

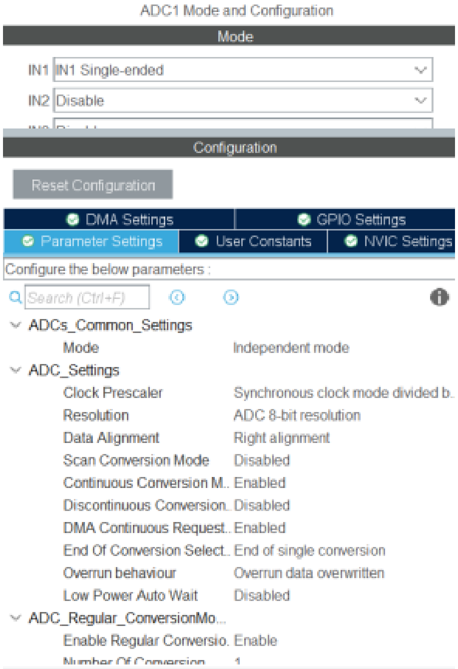

Figure 3. ADC1 Mode and Configuration

2.1 Calculation of Bit Rate

$$\text{Bit Rate} = \frac{P_{Clock}}{P_{Scale}\ x\ (1+T_{seg_1}+T_{seg_2})}$$

I want to bit rate value is 500 KHz.

P_{Clock} = 72 MHz

P_{Scale} = 16

So, $$500000 = \frac{72000000}{16\ x\ (1+T_{seg_1}+T_{seg_2})}$$

$$(1 + T_{seg_1} + T_{seg_2}) = 9$$

Then, T_{seg_1} = 5 Times T_{seg_2} = 3 Times.

3. Programming the STM32F303RE #1

STM32F303RE #1 responsibles for initializing the CAN bus communication and setting the necessary parameters for transmission and filtering. It continuously reads the value from the potentiometer and sends it via the CAN bus to another device.

```c
/* USER CODE BEGIN PV */

      uint8_t adc_value;  /* Reading 8bit ADC Data*/
      uint32_t pTxMailbox;
      CAN_TxHeaderTypeDef pHeader;
      CAN_FilterTypeDef sFilterConfig;

/* USER CODE END PV */

/* USER CODE BEGIN 2 */
        //Set Transmit Parameters
        pHeader.DLC = 1;
        pHeader.IDE = CAN_ID_STD;
        pHeader.RTR = CAN_RTR_DATA;
        pHeader.StdId = 0x0112;
        //Set Filter Parameters
        sFilterConfig.FilterActivation = ENABLE;
        sFilterConfig.FilterBank = 0;
        sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
        sFilterConfig.FilterIdHigh = 0x000;
        sFilterConfig.FilterIdLow = 0x000;
        sFilterConfig.FilterMaskIdHigh = 0x000;
        sFilterConfig.FilterMaskIdLow = 0x000;
        sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
        sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT;

        HAL_CAN_ConfigFilter(&hcan, &sFilterConfig);

        //Start CAN Communication
        HAL_CAN_Start(&hcan);
        //Start ADC with DMA
        HAL_ADC_Start_DMA(&hadc1, &adc_value, 1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        HAL_CAN_AddTxMessage(&hcan, &pHeader, &adc_value, &pTxMailbox);
  }
  /* USER CODE END 3 */
}
```

The code initializes and configures CAN bus communication using the MCP2515 module and the STM32F303RE board. The CAN peripheral on the board is controlled using the CAN_HandleTypeDef structure. The value read from the ADC on the potentiometer is stored in the adc_value variable. The CAN_TxHeaderTypeDef structure is used to set the parameters for the message to be transmitted on the CAN bus. The CAN_FilterTypeDef structure is used to set the parameters for the filter configuration of the CAN bus. The filter is configured using the HAL_CAN_ConfigFilter function and the CAN communication is started using the HAL_CAN_Start function. The ADC is started with DMA using the HAL_ADC_Start_DMA function. In the infinite loop, the value read from the ADC is continuously sent via the CAN bus using the HAL_CAN_AddTxMessage function. The parameters for the transmit header and filter configuration are set during initialization.

4) Programming the STM32F303RE #2

The second board receives this data and displays it on a 2x16 LCD screen using an I2C connection. Firstly, we are handling the interrupt for the reception of data on the CAN bus. It should be written in “stm32f3xx.c” file. This interrupt is triggered when the first data is received on the CAN BUS.

```c
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f3xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

      #include "i2c-lcd.h"

/* USER CODE END Includes */

/* External variables --------------------------------------------------------*/
extern CAN_HandleTypeDef hcan;
/* USER CODE BEGIN EV */

extern CAN_RxHeaderTypeDef pRxHeader;
   extern uint8_t rAdc_value;

/* USER CODE END EV */

void USB_LP_CAN_RX0_IRQHandler(void)
{
  /* USER CODE BEGIN USB_LP_CAN_RX0_IRQn 0 */
      lcd_send_cmd(0x01);
  /* USER CODE END USB_LP_CAN_RX0_IRQn 0 */
  HAL_CAN_IRQHandler(&hcan);
  /* USER CODE BEGIN USB_LP_CAN_RX0_IRQn 1 */
HAL_CAN_GetRxMessage(&hcan, CAN_RX_FIFO0, &pRxHeader, &rAdc_value);
  /* USER CODE END USB_LP_CAN_RX0_IRQn 1 */
lcd_send_cmd(0x80);
lcd_send_string("RADC VALUE:  ");

      if(rAdc_value < 9)
      {
      lcd_send_cmd(0xc0) ;
      lcd_send_data(rAdc_value % 10 + 48);

      }
      else if(rAdc_value> 9 && rAdc_value < 100)
      {
        lcd_send_cmd(0xc0);
        lcd_send_data(rAdc_value / 10 + 48);
        lcd_send_data(rAdc_value % 10 + 48);
      }
      else
      {
        lcd_send_cmd(0xc0) ;
        lcd_send_data(rAdc_value / 100 + 48);
        lcd_send_data((rAdc_value / 10) % 10 + 48);
        lcd_send_data(rAdc_value % 10 + 48);
      }
}
```

In this function, we are first using the HAL_CAN_IRQHandler function to handle the interrupt and then using the HAL_CAN_GetRxMessage function to read the received data. This function takes the handle of the CAN peripheral, the FIFO that the data was received on and the pointers to the receive header and the received data. Once the data has been received and read, it is being sent to the LCD screen. The lcd_send_string function is used to print the text "RADC VALUE: " on the first line of the LCD screen. The value of the ADC is then printed on the second line of the LCD screen. The value is first checked to see if it is less than 9, between 9 and 100, or greater than 100. Depending on this value, the code will send the appropriate number of digits to the LCD screen.

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "i2c-lcd.h"

   /* USER CODE END Includes */

/* USER CODE BEGIN PV */
      CAN_TxHeaderTypeDef pHeader;
      CAN_RxHeaderTypeDef pRxHeader;
      CAN_FilterTypeDef sFilterConfig;
      uint8_t rAdc_value;

/* USER CODE END PV */

  /* USER CODE BEGIN 2 */

  pHeader.DLC = 1;
  pHeader.IDE = CAN_ID_STD;
  pHeader.RTR = CAN_RTR_DATA;
  pHeader.StdId = 0x0156;

  sFilterConfig.FilterActivation = ENABLE;
  sFilterConfig.FilterBank = 0;
  sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
  sFilterConfig.FilterIdHigh = 0;
  sFilterConfig.FilterIdLow = 0;
  sFilterConfig.FilterMaskIdHigh = 0;
  sFilterConfig.FilterMaskIdLow = 0;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_16BIT;

  HAL_CAN_ConfigFilter(&hcan, &sFilterConfig);


  HAL_CAN_Start(&hcan);
  HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING);

      lcd_init();
      lcd_send_string("CAN BUS ADC");
      HAL_Delay(2000);
```

The received value is then displayed on the LCD screen using the lcd_send_string and lcd_send_data functions. The code also includes initializing the LCD screen and displaying a message CAN BUS ADC before starting the infinite loop.

5. Conclusion

In conclusion, the project demonstrated the use of a CAN bus communication system in conjunction with two STM32F303RE boards and an MCP2515 CANBUS-SPI module. One board was used to read an analog value from a potentiometer using DMA and transmit this value to the other board via the CAN bus. The second board received this value and displayed it on a 2x16 LCD screen using an I2C connection. The project was successful in demonstrating the effective communication and data transfer capabilities of the CAN bus protocol. In addition, the use of DMA and I2C communication protocols was also highlighted in the project. Overall, the project highlighted the versatility and effectiveness of using multiple communication protocols to transfer data between different devices.