Project Report
On

Implementation of Hill assist Control using CAN and RTOS

Submitted in partial fulfillment for the award of

Post Graduate Diploma in Embedded System Design

PG-DESD

From

CDAC ACTS (Pune)

Guided by:

Mr. Tarun Bharani

Presented by:

Ashwin Malpe	180840130017
Harshit	180840130042
Jitesh Sharma	180840130048
Nitish Srivastava	180840130065
Pragyan Seth	180840130071

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ashwin Malpe
Harshit
Jitesh Sharma
Nitish Kumar
Pragyan Seth

have successfully completed their project on

Implementation of Hill assist Control using CAN and RTOS

under the guidance of Mr. Tarun Bharani.

Project Guide **Project Supervisor**

HOD ACTS
Ms. Mita Karajagi

Acknowledgement

This project “Implementation of Hill Assist Control System using CAN bus and RTOS” was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of Mr. Tarun Bharani for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to Ms. Risha P.R. (Manager ACTS training Centre), C-DAC, for her guidance and support whenever necessary while doing this course Post Graduate Diploma in Embedded Systems Design (PG-DESD) through C-DAC ACTS, Pune.

Our most heartfelt thanks goes to Ms. Purvi (Course Coordinator, PG-DESD) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

TABLE OF INDEX

List of figures

Abstract

The objective of this topic is to study hill assist control used in vehicles. Hill assist is an automatic system that operates brakes to stop rolling back when it is starting on steep hill.

When hill assist system senses vehicle is starting from rest on slope, it automatically keeps footbrake even after you release the pedal by accelerated vehicle using parking brake / hand brake. The hill start assist control helps to increase control on steep grades and prevents from locking.

This hill stop mechanism is also described to prevent car from rolling back. Another Function is Hill Hold function which is a highly desirable feature in manual transmission vehicles; it also enhances the driving experience in automatic transmission vehicles equipped with hybrid power trains.

The Hill Hold feature supports the Stop and Go performance associated by holding the vehicle on an incline and preventing undesired motion.

Introduction

In Cars there has been development in technology in last few decades. Day by day numbers of vehicles are increasing. The major cause is while driving vehicle on off roads or terrain roads there are more no of accidents. Many times breakdown of vehicles takes place on hill roads while driving.

So hill assist control is technology which helps in driving vehicle on hill roads safely. It helps vehicle to stop rolling back from its position.

Hill Assist Control (HAC) is the new mechatronic technology as a safety features that detect the backward motion of the vehicle on hill or slope, that electronically controls and increases brake pressure at each wheel.

A typical problem that is encountered by motor vehicle when they are themselves stopped on an incline and want to begin moving again, is that the vehicle begins to roll in the unwanted direction when the brake pedal is released. So hill assist control is new feature which helps in starting vehicle smoothly and helps to stop rolling back on inclined roads.

Fig1: Comparison in hill assist car and normal car

1. Objectives:

The main objective of this topic is to study:

- To prevent car rolling back on slope roads
- To start vehicle smoothly on slope roads
- To have control on speed

2. Scope:

This system can be implemented in modern cars with manual transmission and automatic transmission. By devising a mechanism of this sort we could make driving more easier. It is the new mechatronic technology in brake control system as a safety feature & improves their off-road capability.

3. History:

An object of the invention is therefore to provide a method for facilitating hill-starting of a stationary motor vehicle in as simple and intuitive way as possible, and also a device for implementing the method of the invention. In at least one embodiment, the method comprises (includes, but is not necessarily limited to) applying at least one braking device using a brake pedal.

A control unit is used to estimate the travelling resistance of the vehicle, as well as determine a starting gear, a minimum engine torque and a minimum engine speed in order to overcome the estimated travelling resistance. The driver lets the brake pedal up completely and the control unit keeps the braking device applied so that the vehicle remains stationary.

The driver operates an accelerator pedal to a position at least corresponding to the minimum engine speed. A clutch device for the gearbox of the vehicle is activated by the control unit .When the engine

speed has reached the above mentioned minimum engine speed. The control unit releases the braking device in parallel with the clutch device being coupled together so that the brake torque of the braking device decreases as a function of the increase in the engine torque transmitted by the clutch device. In this connection, it is important that the braking device be fully released when the driving torque is sufficient to hold the vehicle stationary on the concerned uphill slope. The vehicle can then be started without rolling backward at the same time as the brakes are not kept applied for an unnecessarily long time.

System Requirements

- Hardware Requirements
  - STM32F4 – Discovery board
  - 2 CAN transceivers (CJMCU230)
  - USB to serial prolific cable
  - Computers with Linux and Windows OS installed
  - Accelerometer
  - Switch

- Software requirements
  - STM32CubeMX (Windows)
  - System Workbench for STM 32(Windows)
  - CMSIS library support (Windows)
  - Jdk support(Linux)
  - STM Studio

- Miscellaneous
  - Jumper wires
  - Twisted pair cable

STM32F4 – Discovery Board features

- STM32F407VGTX microcontroller featuring 1 MB of Flash memory, 192 KB of RAM in an LQFP100 package
- On-board ST-LINK/V2 with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging)
- Board power supply: through USB bus or from an external 5V supply voltage
- External application power supply: 3V and 5V
- LIS302DL or LIS3DSH, ST MEMS motion sensor, 3-axis digital output accelerometer
- MP45DT02, ST MEMS audio sensor, omni-directional digital microphone
- CS43L22, audio DAC with integrated class D speaker driver
- Eight LEDs: – LD1 (red/green) for USB communication – LD2 (red) for 3.3V power on – Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue) – 2 USB OTG LEDs LD7 (green) V-Bus and LD8 (red) over-current
- Two pushbuttons (user and reset)
- USB OTG with micro-AB connector
- Extension header for LQFP100 I/Os for quick connection to prototyping board and easy probing

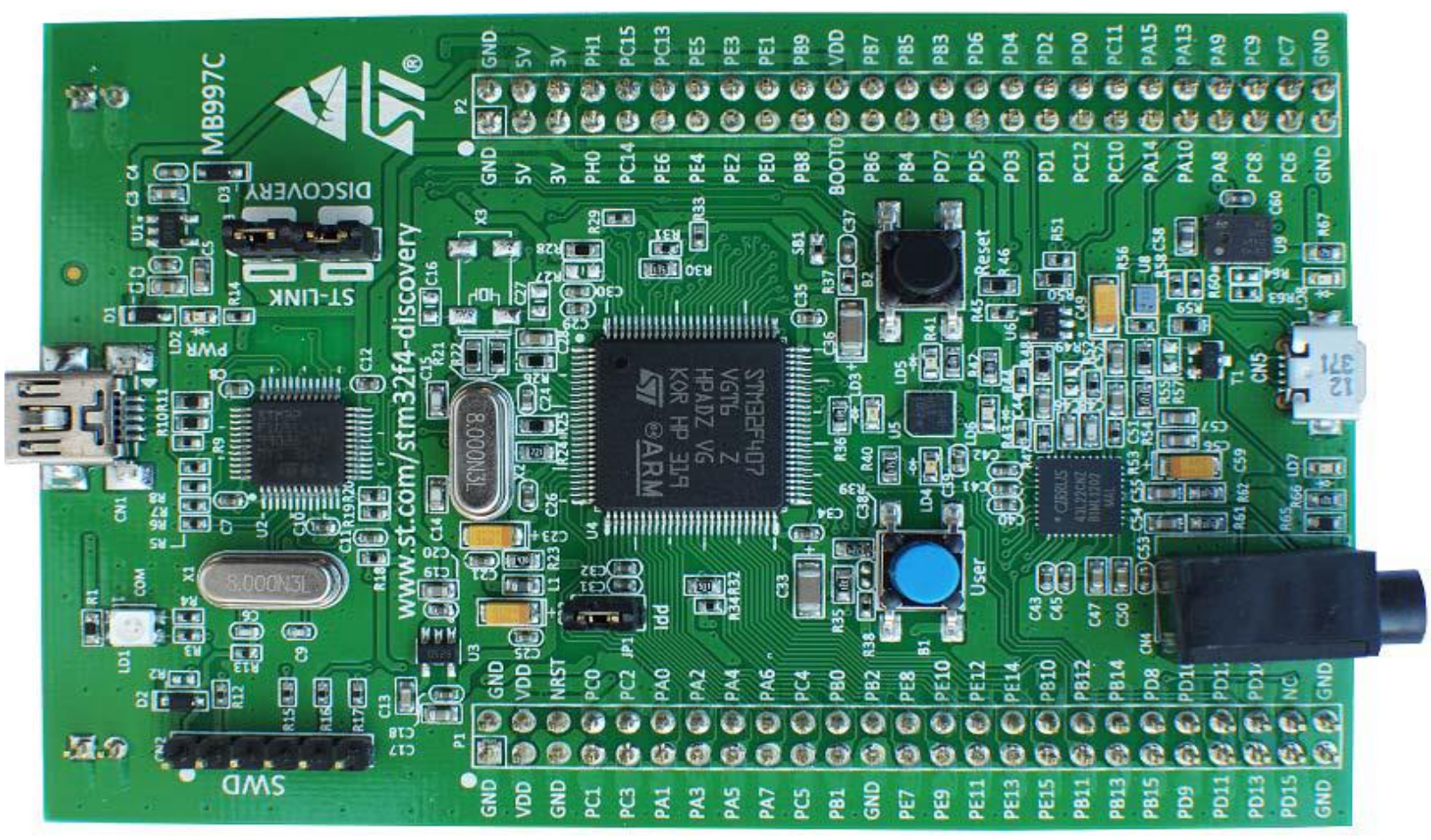

Fig2: STM32F303 Discovery

CAN Bus Protocol

- CAN has been standardized since 1993 and is available as an ISO standard (International Standardization Organization): ISO 11898.
- While it initially consisted of three parts, today it has five parts.
- The first part describes the event-driven communication protocol.
- A time-triggered extension can be found in the fourth part.
- The second and third parts cover information on the bus interface and physical data transmission: A distinction is made here between the High-Speed variant (data rates up to 1 MBit/s) and Low-Speed variant (data rates up to 125 KBit/s).
- The fifth part describes the behavior of a CAN node in the High-Speed network in Low Power Mode.
- A maximum network extension of about 40 meters is allowed. At the ends of the CAN network, bus termination resistors contribute to preventing transient phenomena (reflections).
- In a CAN network, the transmitted data frames and their sequence are not a function of the progression of time, rather they depend on the occurrence of special events.
- In principle, each CAN node is authorized to access the CAN bus immediately after an event occurs. In conjunction with the relatively short message length of a maximum of 130 bits in standard format, and the high data transmission rate of up to 1 MBit/s, the method enables quick reactions to asynchronous processes.
- This is an important prerequisite for real-time data transmission capability in the low milliseconds range, which is primarily demanded by applications in the power-train and chassis areas.
- To guarantee real-time communication despite random bus access, bus access is based on the CSMA/CA method (Carrier Sense Multiple Access/Collision Avoidance).
- First, the CSMA/CA method ensures that CAN nodes wishing to send do not access the CAN bus until it is available.

- Second, in simultaneous bus accesses occur, the CSMA/CA method ensures that the CAN node with the highest priority data frame prevails.

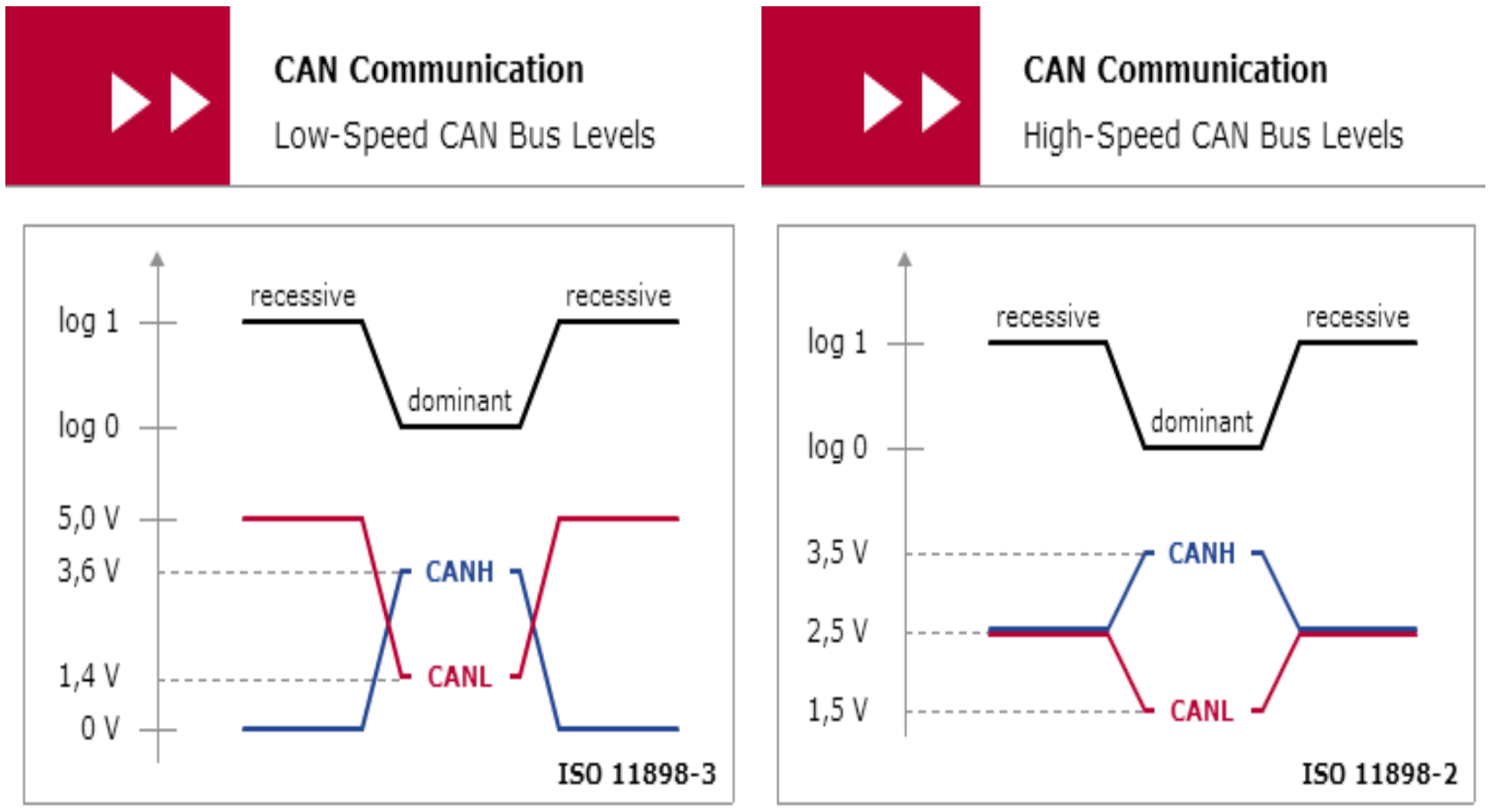

Fig3: CAN Bus level

- **Types and Role of each frame:-**

Frame	Roles of frame	User settings
Data frame	This frame is used by the transmit unit to send a message to the receive unit.	Necessary
Remote frame	This frame is used by the receive unit to request transmission of a message that has the same ID from the transmit unit.	Necessary
Error frame	When an error is detected, this frame is used to notify other units of the detected error.	Unnecessary
Overload frame	This frame is used by the receive unit to notify that it has not been prepared to receive frames yet.	Unnecessary
Interframe space	This frame is used to separate a data or remote frame from a preceding frame.	Unnecessary

- **CAN data frame:-**

  - Transmission of a data frame begins with a start symbol (Start of Frame: SOF). It is used by the CAN nodes to synchronize to the sender.
  - During transmission, the CAN nodes maintain accurate timing by evaluating each edge transition and adjusting their clocks as necessary.
  - The bit stuffing method guarantees that an edge transition occurs after five homogeneous bits at most.
  - An end symbol (End of Frame: EOF) marks the end of a data frame.

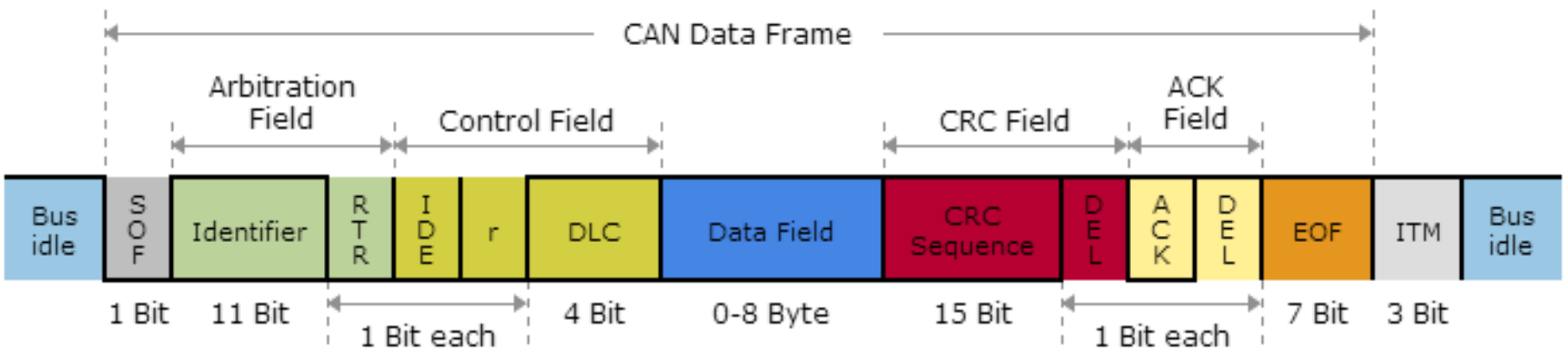

Fig4: CAN data frame

- **BxCAN Test Modes:-**

  1. Loopback mode
  2. Silent mode
  3. Normal mode
  4. Silent + Loopback mode

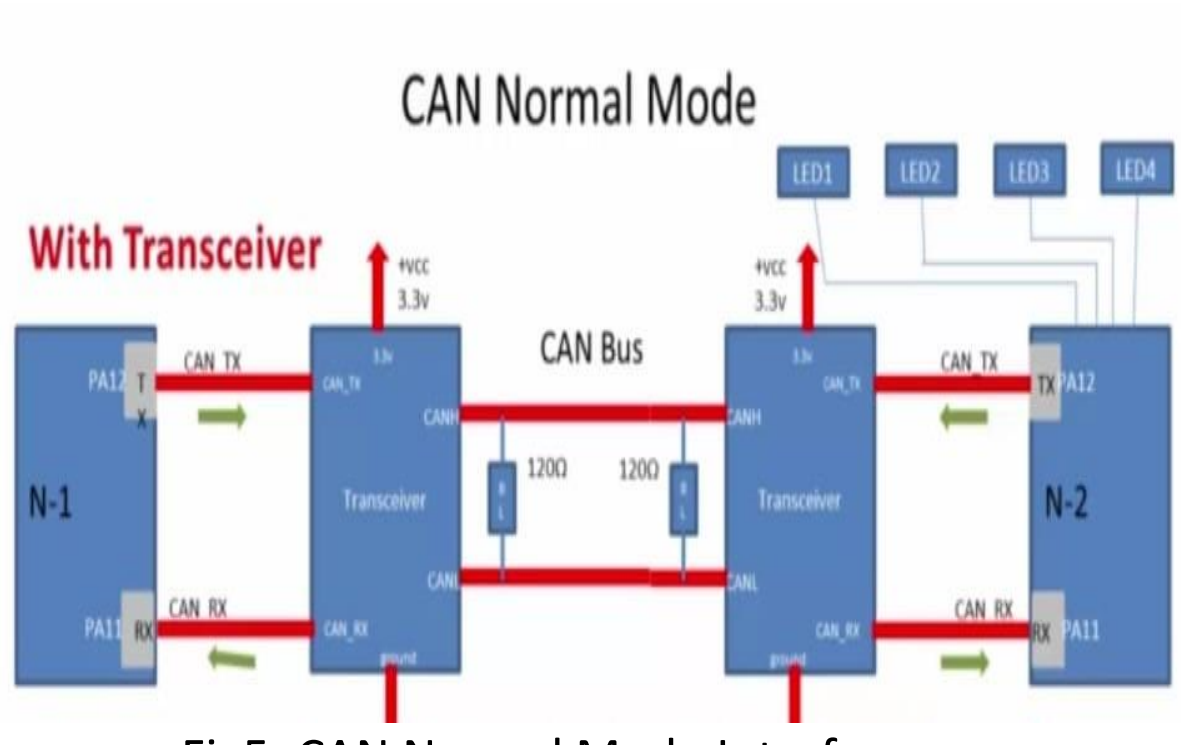

Fig5: CAN Normal Mode Interface

Hill Start Assist Working

The hill start was achieved by the brake systems. When the vehicle is prepared to start up on a slope road, the driver push the hill-start pushbutton to change the vehicle into hill-start mode.

The control strategy changes the gear to a proper gear number by hydraulic actuator. When the driver releases the hand brake, the fluid is prepared to pass through the ABS valve. However, the abs valve is controlled by the VECU module, which controlled the ABS valve by engine torque, gearbox, and the hill-start pushbutton.

If the output torque of the engine is not achieved the threshold value, the ABS valve does not open to let the fluid pass through to release the brake chamber. The brake system locks the wheel to avoid roll back. So the vehicle remains stationary on the slope road.

When the driver press the throttle pedal, the output torque of the engine will increase according to the throttle position.

When the engine torque is above the threshold value, the VECU controls the ABS valve which opens and allows high pressure fluid to pass through brake chamber and gets released through the brake chamber. So the vehicle will move on the road.

The increase of the engine torque is proportional to the decrease of brake force.

The advantage of the control strategy is that, firstly, if the engine torque is not achieved the threshold, the vehicle will remain in the initial position, second, if the driver press the throttle pedal deeply, the engine torque will increase very fast, then the vehicle will start quickly.

Slope calculations:

The longitudinal vehicle road load includes the rolling resistance, aerodynamic resistance, and uphill resistance. The vehicle model can be described as follows:

$$ma = F_r + F_t$$

$$F = \mu mg\cos\Theta + 0.5C_dAV^2 + mg\sin\Theta$$

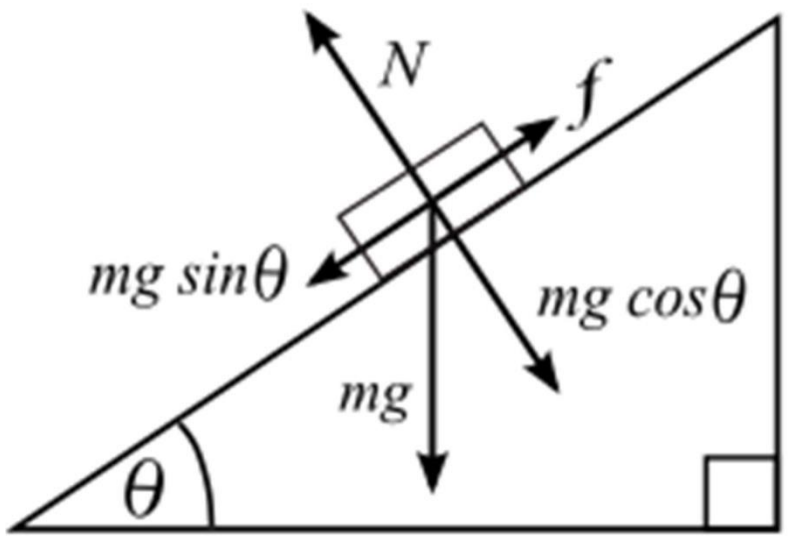

Fig6: Force Analysis of vehicle on inclined plane

Where F_t is the driving force on the vehicle, F_r is the total resistance, m is the mass of the vehicle, a is the acceleration, α is the grade angle, μ is the coefficient of friction of the wheel, C_d is the air drag coefficient that depends on the body style and dimension and A is the frontal area of the vehicle.

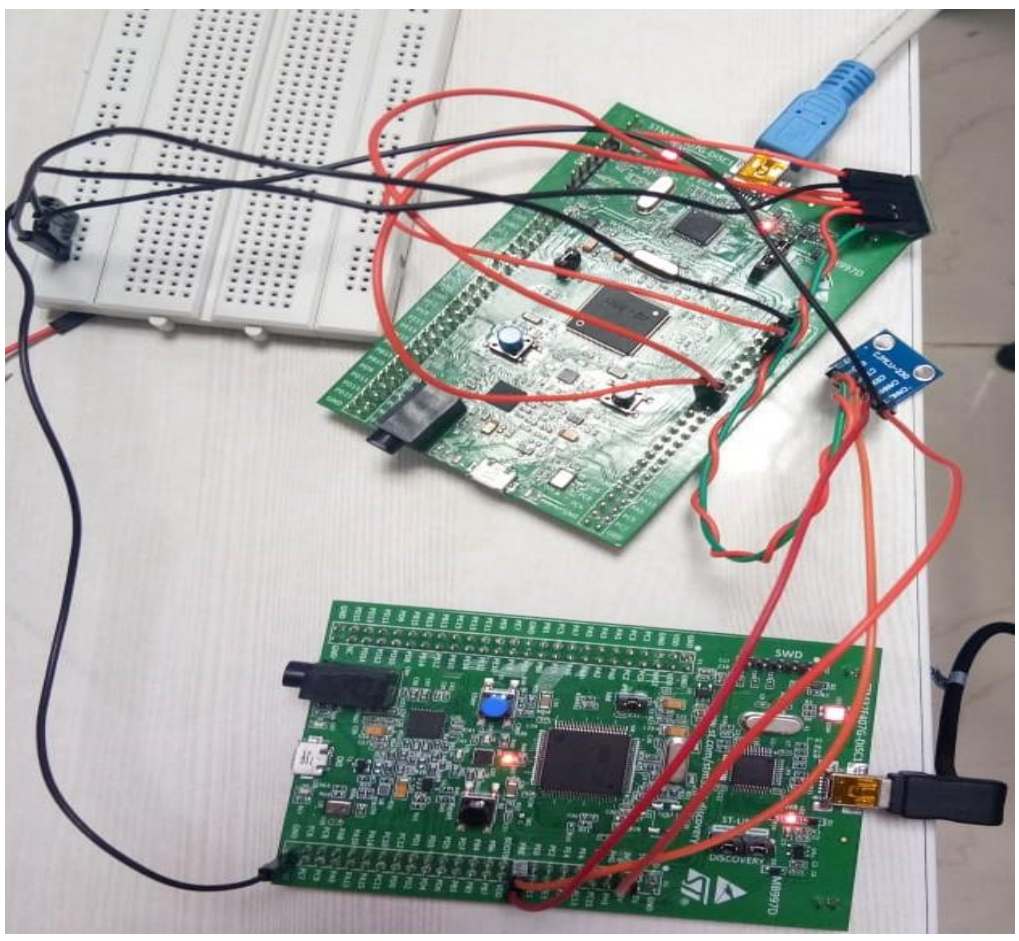

Fig7: CAN interfacing with STM32f4 board

- **Advantages:**
  1. Prevents accidents while starting the vehicle on slopes.
  2. Stress free driving.
  3. Longer clutch life.

- **Disadvantages:**
  1. Holding of brake may lead to warping of disc.
  2. Maintenance cost is high.

- **Applications:**

These are cars given below having hill assist control feature in existence:-

1. Honda CR-V
2. Honda CR-Z
3. Toyota Fortuner
4. Toyota Prius
5. Toyota Land Cruiser
6. Toyota RAV4
7. Chevrolet Trax
8. Volkswagen Touareg
9. Opel Mokka

Program and Clock configuration

- Program for Master :-

```
/**
  ******************************************************************************
  * @file           : Mastermain.c
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "cmsis_os.h"

/* USER CODE BEGIN Includes */

/*CAN Message IDs to be used*/
#define MSG_TILT_ANGLE 0x65D
#define MSG_GEAR   0x651
#define MSG_WHEEL  0x652
/* For Wheel and Gear Position(Forward and Reverse)*/
#define WFWD 1
#define WREV 0
#define GFWD 1
#define GREV 0

/* signals used in Task Notification*/

#define ACTIVATE 0x01
#define DEACTIVATE 0x02
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;

osThreadId defaultTaskHandle;
osThreadId vHeartBeatHandle;
osThreadId vDataProcessHandle;
osThreadId vTaskActionHandle;
osMutexId dataMutexHandle;
osSemaphoreId dataSemaphoreHandle;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
CAN_RxHeaderTypeDef RxHeader;
uint8_t rcvd_msg[1],recv_Data;
uint8_t dataTiltAngle=0,dataGear=0,dataWheel=0;
int32_t sigNotify=0x00;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/

/* Clock and Peripheral Initialization Functions*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
/* user defined functions declaration*/
void StartDefaultTask(void const * argument);
void tHeartBeat(void const * argument);
void tDataProcess(void const * argument);
void tTaskAction(void const * argument);
void CAN_START();

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  /* USER CODE BEGIN 2 */
  CAN_START();           // Start the CAN  Module
  /* USER CODE END 2 */

  /* Create the mutex(es) */
  /* definition and creation of dataMutex */
  osMutexDef(dataMutex);
  dataMutexHandle = osMutexCreate(osMutex(dataMutex));

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* Create the semaphores(s) */
  /* definition and creation of dataSemaphore */
 /* osSemaphoreDef(dataSemaphore);
  dataSemaphoreHandle = osSemaphoreCreate(osSemaphore(dataSemaphore), 1);*/

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of vHeartBeat */
  osThreadDef(vHeartBeat, tHeartBeat, osPriorityAboveNormal, 0, 128);
  vHeartBeatHandle = osThreadCreate(osThread(vHeartBeat), NULL);

  /* definition and creation of vDataProcess */
  osThreadDef(vDataProcess, tDataProcess, osPriorityAboveNormal, 0, 128);
  vDataProcessHandle = osThreadCreate(osThread(vDataProcess), NULL);

  /* definition and creation of vTaskAction */
  osThreadDef(vTaskAction, tTaskAction, osPriorityAboveNormal, 0, 128);
  vTaskActionHandle = osThreadCreate(osThread(vTaskAction), NULL);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */


  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

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

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    }
    /* USER CODE END 3 */

}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
```

```
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 84;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
RCC_OscInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

  /**Configure the Systick interrupt time
  */
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  /**Configure the Systick
  */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
}

/* CAN1 init function */
static void MX_CAN1_Init(void)
{

  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 3;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_11TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = ENABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                          |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pins : PD9 PD10 PD11 PD12
                           PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                          |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
/**
 * @brief CAN functions
 *Functions used --->
 *CAN1_Tx()     --        To transmit data over CAN BUS
```

```
* @param1      data to be transimitted
*@param2       CAN Message ID
 */

void CAN1_Tx(uint8_t data,uint32_t id)
{
       CAN_TxHeaderTypeDef TxHeader;

              uint32_t TxMailbox;

              uint8_t message;

              TxHeader.DLC = 1;
              TxHeader.StdId = id;
              TxHeader.IDE   = CAN_ID_STD;
              TxHeader.RTR = CAN_RTR_DATA;

          message=data;

              HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);

              if( HAL_CAN_AddTxMessage(&hcan1,&TxHeader,&message,&TxMailbox) != HAL_OK)
              {
                     Error_Handler();
              }
}
/*@brief    CAN Message Filter Configuration
Config used: No Filter apllied(accept all CAN Messages)
                   Select Filter Bank 0,Select CAN Receiver FIFO0,FilterScale 32 Bit
*/
void CAN_Filter_Config(void)
{
       CAN_FilterTypeDef can1_filter_init;

       can1_filter_init.FilterActivation = ENABLE;
       can1_filter_init.FilterBank  = 0;
       can1_filter_init.FilterFIFOAssignment = CAN_RX_FIFO0;
       can1_filter_init.FilterIdHigh = 0x0000;
       can1_filter_init.FilterIdLow = 0x0000;
       can1_filter_init.FilterMaskIdHigh = 0X0000;
       can1_filter_init.FilterMaskIdLow = 0x0000;
       can1_filter_init.FilterMode = CAN_FILTERMODE_IDMASK;
       can1_filter_init.FilterScale = CAN_FILTERSCALE_32BIT;

       if( HAL_CAN_ConfigFilter(&hcan1,&can1_filter_init) != HAL_OK)              //HAL layer API to config CAN Filter
       {
              Error_Handler();
       }

}
/*@brief   Function used to enablethe CAN Module
must be called before using CAN functionality
*/
void CAN_START(){
         CAN_Filter_Config();                     //call filter config function

       if(HAL_CAN_ActivateNotification(&hcan1,CAN_IT_TX_MAILBOX_EMPTY|CAN_IT_RX_FIFO0_MSG_PENDING|CAN_IT_BUSOFF)!=
HAL_OK) //HAL Layer API to start CAN Module
              {
                           Error_Handler();
              }
              if( HAL_CAN_Start(&hcan1) != HAL_OK)
              {
                     Error_Handler();
              }
}
/*
 * @brief Interrupt CallBacks
 *   3 Transmit Mailboxes interrupt callbacks
 *    1 Receive interrupt FIFO0 callback
 * */
void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *hcan)
{
       HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}

void HAL_CAN_TxMailbox1CompleteCallback(CAN_HandleTypeDef *hcan)
{
       HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}

void HAL_CAN_TxMailbox2CompleteCallback(CAN_HandleTypeDef *hcan)
{
       HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}
/* Messgae is filter based on message ID and  data is copied to respective sensor variables */

void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{

    uint32_t ID;
       if(HAL_CAN_GetRxMessage(hcan,CAN_RX_FIFO0,&RxHeader,rcvd_msg) != HAL_OK)
       {
              Error_Handler();
       }
      // ID=RxHeader.StdId;

       if(RxHeader.StdId == 0x65D && RxHeader.RTR == 0 )
              {
                     //This is data frame sent by n1 to n2
```

```
                dataTiltAngle=rcvd_msg[0];

              }
        if(RxHeader.StdId == 0x651 && RxHeader.RTR == 0 )
              {
                 dataGear=rcvd_msg[0];
              }
        if(RxHeader.StdId == 0x652 && RxHeader.RTR == 0 )
              {
                    //This is data frame sent by n1 to n2
                dataWheel=rcvd_msg[0];

              }
               /*if(RxHeader.RTR == 0 && RxHeader.StdId>0){
                 switch(ID){

                 case MSG_TILT_ANGLE:
                        dataTiltAngle=rcvd_msg[0];
                        break;
                 case MSG_GEAR:
                        dataGear=rcvd_msg[0];
                        break;
                 case MSG_WHEEL:
                        dataWheel=rcvd_msg[0];
                        break;
                 default:
                        break;
                 }
                }*/


}
/* CAN Functions End*/

/* USER CODE END 4 */

/* StartDefaultTask function */
void StartDefaultTask(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}

/* tHeartBeat function */
/* Task Handle used to indicate whether our systen is running or not
It is periodic task having period of 250 ms*/
void tHeartBeat(void const * argument)
{
  /* USER CODE BEGIN tHeartBeat */
                  TickType_t xPreviousWakeTime=xTaskGetTickCount();
                          TickType_t ticksToIncrement=pdMS_TO_TICKS(250);
                      for(;;)
                      {
                                    HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_15);
                                              vTaskDelayUntil(&xPreviousWakeTime,ticksToIncrement);

                      }
}

/* tDataProcess function */
/*@brief    Data received from slave node is processed in this function to check whether critical occurs or not
*periodic task of 100ms
*Here Task Notification mechanism is used to notify ACTION task if Critical action occurs notify the Action Task
 */
void tDataProcess(void const * argument)
{
  /* USER CODE BEGIN tDataProcess */
  /* Infinite loop */
                                TickType_t xPreviousWakeTime=xTaskGetTickCount();
                                TickType_t ticksToIncrement=pdMS_TO_TICKS(100);
                             for(;;)
                             {
                                          if(dataTiltAngle<150)
                                          {
                                                if(dataGear==GFWD && dataWheel==WREV){
                                                       osSignalSet(vTaskActionHandle,(int32_t)ACTIVATE);         //
Notify the Action Task to Activate the System


                                                }
                                                else if(dataGear==GFWD && dataWheel==WFWD){              // Notify the
Action to task to deacitivate the system
                                                       osSignalSet(vTaskActionHandle,(int32_t)DEACTIVATE);


                                                }
                                          }
                                              HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
                                                       vTaskDelayUntil(&xPreviousWakeTime,ticksToIncrement);

                             }

  /* USER CODE END tDataProcess */
}
```

```c
/* tTaskAction function */
/*If Critical action occurs activate the System
*/
void tTaskAction(void const * argument)
{
  /* USER CODE BEGIN tTaskAction */
  /* Infinite loop */
       osEvent event;
  for(;;)
  {
               HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);


                     event=osSignalWait(sigNotify,osWaitForever);     //wait for the Signal event to occurred
                     if(event.status==osEventSignal){
                            if(event.value.signals==ACTIVATE){

                                 HAL_GPIO_WritePin(GPIOD,GPIO_PIN_10,GPIO_PIN_SET);       //Indicates System is
activated
                    }
                            else if(event.value.signals==DEACTIVATE){
                                 HAL_GPIO_WritePin(GPIOD,GPIO_PIN_10,GPIO_PIN_RESET);  // indicates system is
deactivated

                            }
                     }
    osDelay(1);
  }
  /* USER CODE END tTaskAction */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**
  * @}
  */

/**
  * @}
  */

/*************************END OF FILE*************************/
```

- Program for Slave :-

```
/**
  ******************************************************************************
  * @file           : main.c


  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "cmsis_os.h"

/* USER CODE BEGIN Includes */
#include "math.h"
/*  Macro Definitions */
/* for 3 axes data*/
#define Xaxis 1
#define Yaxis 2
#define Zaxis 3
/*CAN Message IDs to be used*/
#define MSG_TILT_ANGLE 0x65D
#define MSG_GEAR   0x651
#define MSG_WHEEL  0x652
/* For Wheel and Gear Position(Forward and Reverse)*/
#define WFWD 1
#define WREV 0
#define GFWD 1
#define GREV 0
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;

SPI_HandleTypeDef hspi1;

osThreadId defaultTaskHandle;
osThreadId vHeartBeatHandle;
osThreadId vAngleofTiltHandle;
osThreadId vTaskWheelPosHandle;
osMutexId mutexCANHandle;
osSemaphoreId semaCANHandle;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
uint8_t TxBuf[2],RxBuf_X[2],RxBuf_Y[2],RxBuf_Z[2],ReturnVal[2],readValX,readValY,readValZ,recv_Data;   // Accelerometer
data
CAN_RxHeaderTypeDef RxHeader;
uint8_t rcvd_msg[1];
float resultX,resultY;
uint8_t angleData[2];
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/*----Peripheral Initialization------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
static void MX_SPI1_Init(void);
/*----Task Function Declarations-------*/
void StartDefaultTask(void const * argument);
void tHeartBeat(void const * argument);
void tMeasureTilt(void const * argument);
void tWheelPosition(void const * argument);
void accln_tilt(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
void SPI_start();
void spiRead(int choice);
void CAN1_Tx(uint8_t data,uint32_t id);
void CAN_START();
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  MX_SPI1_Init();
  /* USER CODE BEGIN 2 */
  CAN_START();                  //start the CAN Module
  /* USER CODE END 2 */

  /* Create the mutex(es) */
  /* definition and creation of mutexCAN */
  osMutexDef(mutexCAN);
  mutexCANHandle = osMutexCreate(osMutex(mutexCAN));

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* Create the semaphores(s) */
  /* definition and creation of semaCAN */
  /*osSemaphoreDef(semaCAN);
  semaCANHandle = osSemaphoreCreate(osSemaphore(semaCAN), 1);*/

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of vHeartBeat */
  osThreadDef(vHeartBeat, tHeartBeat, osPriorityAboveNormal, 0, 128);
  vHeartBeatHandle = osThreadCreate(osThread(vHeartBeat), NULL);

  /* definition and creation of vAngleofTilt */
  osThreadDef(vAngleofTilt, tMeasureTilt, osPriorityAboveNormal, 0, 128);
  vAngleofTiltHandle = osThreadCreate(osThread(vAngleofTilt), NULL);

  /* definition and creation of vTaskWheelPos */
  osThreadDef(vTaskWheelPos, tWheelPosition, osPriorityAboveNormal, 0, 128);
  vTaskWheelPosHandle = osThreadCreate(osThread(vTaskWheelPos), NULL);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */


  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

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

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    }
    /* USER CODE END 3 */

}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
```

```
  */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 84;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
RCC_OscInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

  /**Configure the Systick interrupt time
  */
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
}

/* CAN1 init function */
static void MX_CAN1_Init(void)
{

  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 3;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_11TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_2TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = ENABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* SPI1 init function */
static void MX_SPI1_Init(void)
{

  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
```

```
__HAL_RCC_GPIOE_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

/*Configure GPIO pin : PE3 */
GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/*Configure GPIO pin : PA0 */
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PD11 */
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/*Configure GPIO pins : PD12 PD13 PD14 PD15 */
GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 5, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

}

/* USER CODE BEGIN 4 */
void SPI_start(){

        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
        TxBuf[0]=0x20;
        TxBuf[1]=0x037;
        HAL_SPI_Transmit(&hspi1,TxBuf,2,50);
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
}
void spiRead(int choice){


      switch(choice){

      case 1:
                     HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
                        TxBuf[0]=0x29|0x80;
                        HAL_SPI_Transmit(&hspi1,TxBuf,1,50);
                        HAL_SPI_Receive(&hspi1,RxBuf_X,1,50);  //Read X
                        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
                        readValX=RxBuf_X[0];
                     break;
      case 2:
                                        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
                                TxBuf[0]=0x2B|0x80;
                                HAL_SPI_Transmit(&hspi1,TxBuf,1,50);
                                HAL_SPI_Receive(&hspi1,RxBuf_Y,1,50);  //Read Y
                                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
                                readValY=RxBuf_Y[0];
                                break;
      case 3:
                                                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
                                               TxBuf[0]=0x2D|0x80;
                                               HAL_SPI_Transmit(&hspi1,TxBuf,1,50);
                                               HAL_SPI_Receive(&hspi1,RxBuf_Z,1,50);  //Read Z
                                               HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
                                               readValZ=RxBuf_Z[0];
                                               break;
      default:
             break;
      }


}
/* SPI functions END*/

/* @brief calculate angle of tilt and pitch
 *
 * **/
void accln_tilt(void){
      uint8_t x2,y2,z2,temp1,temp2,tempR;
      float accln_angleX,accln_angleY;
      x2=readValX*readValX;
      y2=readValY*readValY;
      z2=readValZ*readValZ;
      temp1=(uint8_t)(x2+y2);
```

```
    temp2=(uint8_t)(x2+z2);
   resultX=sqrtf(temp1);
   resultY=sqrtf(temp2);

   resultX=readValX/resultX;
   resultY=readValY/resultY;

   /*accln_angleX=tanf(resultX);
   accln_angleY=tanf(resultY);*/

   angleData[0]=100*tanf(resultX);   //roll angle
   //*(angleDataptr+1)=tanf(resultY);   //pitch angle

   //temp4=(float)angleData[0]


}
/*
 * @brief CAN functions
 *Functions used --->
 *CAN1_Tx()     --        To transmit data over CAN BUS
* @param1      data to be transimitted
*@param2       CAN Message ID
 */
void CAN1_Tx(uint8_t data,uint32_t id)
{
      CAN_TxHeaderTypeDef TxHeader;

            uint32_t TxMailbox;

            uint8_t message;
            osStatus status;
               TxHeader.DLC = 1;
               TxHeader.StdId = id;
               TxHeader.IDE   = CAN_ID_STD;
               TxHeader.RTR = CAN_RTR_DATA;

           message=data;

               HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
               status=osMutexWait(mutexCANHandle,50);
               if(status!=osOK){
                     Error_Handler();
               }

               if( HAL_CAN_AddTxMessage(&hcan1,&TxHeader,&message,&TxMailbox) != HAL_OK)
               {
                     Error_Handler();
               }
               status=osMutexRelease(mutexCANHandle);
               if(status!=osOK){
                                 Error_Handler();
                           }

}
/*@brief    CAN Message Filter Configuration
Config used: No Filter apllied(accept all CAN Messages)
                  Select Filter Bank 0,Select CAN Receiver FIFO0,FilterScale 32 Bit
*/
void CAN_Filter_Config(void)
{
      CAN_FilterTypeDef can1_filter_init;

      can1_filter_init.FilterActivation = ENABLE;
      can1_filter_init.FilterBank  = 0;
      can1_filter_init.FilterFIFOAssignment = CAN_RX_FIFO0;
      can1_filter_init.FilterIdHigh = 0x0000;
      can1_filter_init.FilterIdLow = 0x0000;
      can1_filter_init.FilterMaskIdHigh = 0X0000;
      can1_filter_init.FilterMaskIdLow = 0x0000;
      can1_filter_init.FilterMode = CAN_FILTERMODE_IDMASK;
      can1_filter_init.FilterScale = CAN_FILTERSCALE_32BIT;

      if( HAL_CAN_ConfigFilter(&hcan1,&can1_filter_init) != HAL_OK)
      {
            Error_Handler();
      }

}
/*@brief   Function used to enablethe CAN Module
must be called before using CAN functionality
*/
void CAN_START(){
         CAN_Filter_Config();                    //call filter config function

      if(HAL_CAN_ActivateNotification(&hcan1,CAN_IT_TX_MAILBOX_EMPTY|CAN_IT_RX_FIFO0_MSG_PENDING|CAN_IT_BUSOFF)!=
HAL_OK)
            {
                        Error_Handler();
            }
            if( HAL_CAN_Start(&hcan1) != HAL_OK)
            {
                  Error_Handler();
            }
}
/*
 * @brief Interrupt CallBacks
 *   3 Transmit Mailboxes interrupt callbacks
*    1 Receive interrupt FIFO0 callback
```

```
 * */void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *hcan)
{
	HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}

void HAL_CAN_TxMailbox1CompleteCallback(CAN_HandleTypeDef *hcan)
{
	HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}

void HAL_CAN_TxMailbox2CompleteCallback(CAN_HandleTypeDef *hcan)
{
	HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
}
/* Messgae is filter based on message ID and  data is copied to respective sensor variables */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{


	if(HAL_CAN_GetRxMessage(hcan,CAN_RX_FIFO0,&RxHeader,rcvd_msg) != HAL_OK)
	{
		Error_Handler();
	}

	if(RxHeader.StdId == 0x65D && RxHeader.RTR == 0 )
	{
		//This is data frame sent by n1 to n2
		//HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
		recv_Data=rcvd_msg[0];

	}

}
/* CAN Functions End*/

/**
 * @brief External Interrrupt for Gear Position:
 *   pin PA0 is used.
 *   logic 1=gear forward
 *   logic 0=gear Reverse
 *
 * */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

	static uint8_t dataGear=1;
	if(dataGear){
		CAN1_Tx(GREV,(uint32_t)MSG_GEAR);
		dataGear=0;
	}else{

		CAN1_Tx(GFWD,(uint32_t)MSG_GEAR);
	   dataGear=1;
	}

}
/* USER CODE END 4 */

/* StartDefaultTask function */
void StartDefaultTask(void const * argument)
{

  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}

/* tHeartBeat function */
/* Task Handle used to indicate whether our systen is running or not
It is periodic task having period of 250 ms*/
void tHeartBeat(void const * argument)
{
	/* USER CODE BEGIN tHeartBeat */
		/* Infinite loop */
			TickType_t xPreviousWakeTime=xTaskGetTickCount();
			TickType_t ticksToIncrement=pdMS_TO_TICKS(250);
		for(;;)
		{
				HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_15);
					vTaskDelayUntil(&xPreviousWakeTime,ticksToIncrement);

		}
		/* USER CODE END tHeartBeat */
}

/* tMeasureTilt function */
/*Task to measure the angle of tilt using Acclerometer based on data received by X,Y and Z axis*/
void tMeasureTilt(void const * argument)
{
  /* USER CODE BEGIN tMeasureTilt */
  /* Infinite loop */

	//uint8_t angleData[2];
		TickType_t xPreviousWakeTime=xTaskGetTickCount();
		TickType_t ticksToIncrement=pdMS_TO_TICKS(200);
		SPI_start();   //Initialize Accelerometer Registers
```

```
		for(;;)
		{

			spiRead(Xaxis);                                // function to read Axes data
			spiRead(Yaxis);
			spiRead(Zaxis);

			HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_14);
			accln_tilt();
			CAN1_Tx(angleData[0],(uint32_t)MSG_TILT_ANGLE); //Transmit the CAN message for angle of tilt

			     vTaskDelayUntil(&xPreviousWakeTime,ticksToIncrement);
		}

}

/* tWheelPosition function */
/*task to check gear position */
void tWheelPosition(void const * argument)
{
  /* USER CODE BEGIN tWheelPosition */
      uint8_t dataWheel=0;
      TickType_t xPreviousWakeTime=xTaskGetTickCount();
      TickType_t ticksToIncrement=pdMS_TO_TICKS(200);
  /* Infinite loop */
  for(;;)
  {

        if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_11)){
               dataWheel=WREV;       //if Switch is pressed--> Wheel Direction---Reverse
            HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
        }
        else {
               dataWheel=WFWD;     //if switch is not pressed--> Wheel Direction---Forward
         }

       CAN1_Tx(dataWheel,(uint32_t)MSG_WHEEL);

    vTaskDelayUntil(&xPreviousWakeTime,ticksToIncrement);
  }
  /* USER CODE END tWheelPosition */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {

  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************END OF FILE****************************/
```

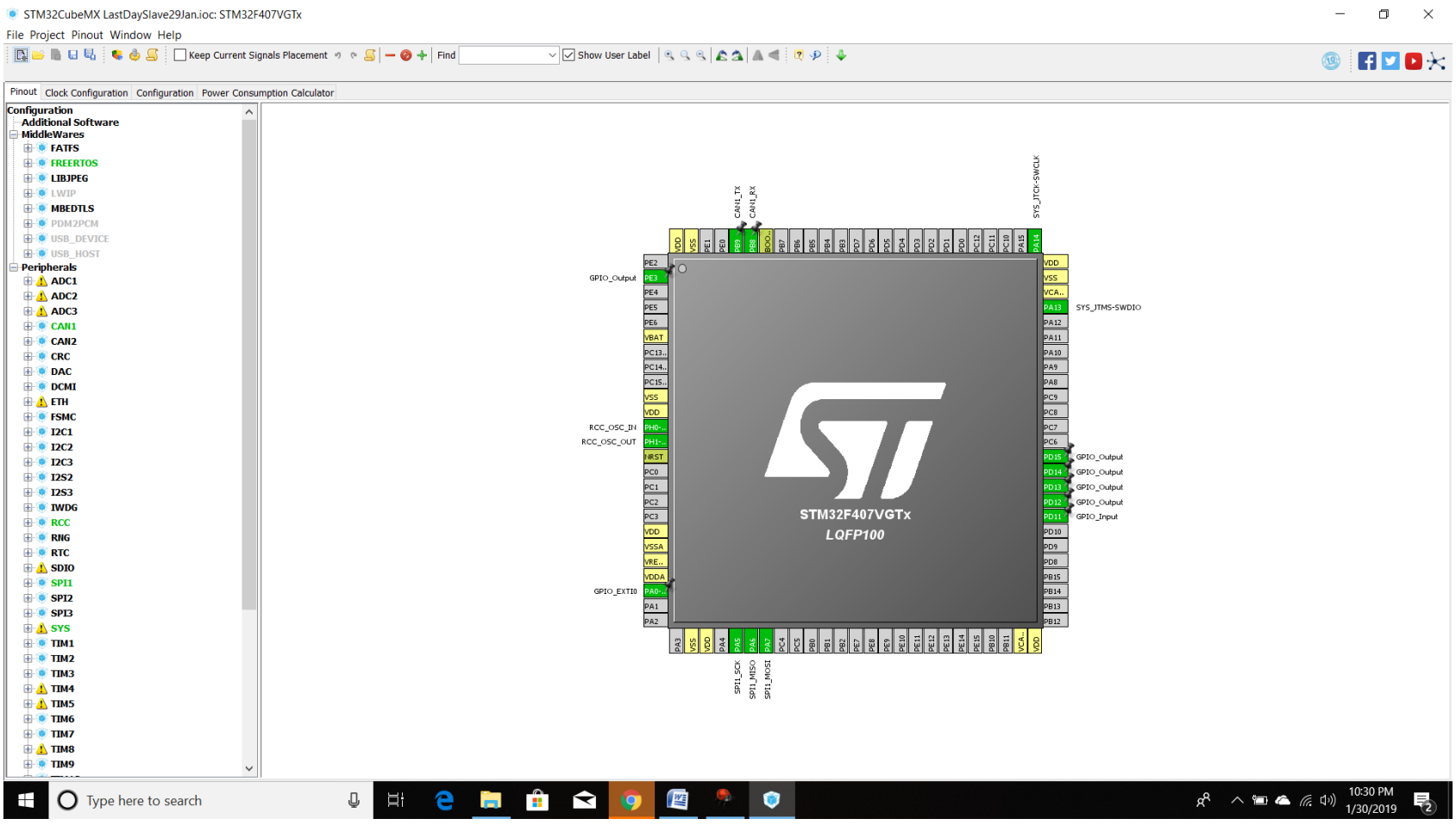

Fig 8: Pinout Configuration

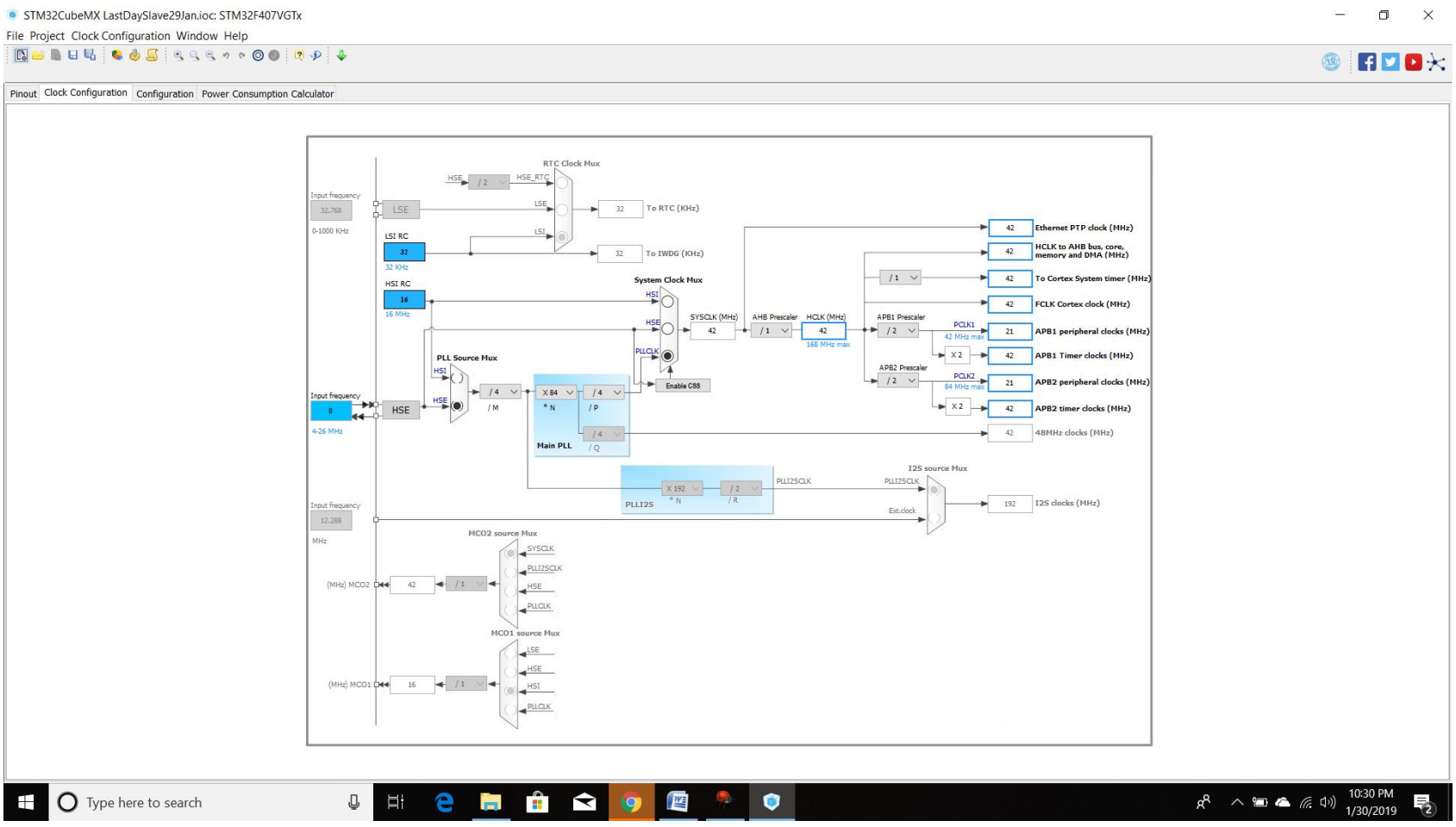

Fig 9: Clock Configuration

Conclusion

The Hill Assist feature itself has proven to be very beneficial to manual transmission drivers that find themselves on grades that would normally make driving difficult. In addition to manual transmission vehicles, the driving experience of vehicles with automatic transmissions can be enhanced. The choice of which method of Hill Hold to implement is influenced by many factors: desired performance, system cost and vehicle power train. When implemented correctly, the functionality of the Hill Hold feature seamlessly matches the driver's normal driving habits and instincts. Hill Hold function makes the driving experience as comfortable and natural as possible while keeping the system commercially feasible. By using this feature we can drive vehicle easily.

References

- Reference manual of STM32f4
- Embedded Hardware Design book
- Datasheet for STM32f4
- http://esd.cs.ucr.edu/webres/can20.pdf
- https://www.st.com/resource/en/datasheet/DM00040962.pdf
- https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf
- https://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/hill-start-control.htm
- www.freertos.org