|  |  |
| --- | --- |
| Multi-sensor can-based vehicle dashboard with real time monitoring system | Abstract  The ARM7-TDMI (LPC2129) based vehicle dashboard monitors fuel level, engine temperature, and speed in real-time. It displays critical readings and warning messages when fuel is low, temperature is high, or speed exceeds safe limits.  santhosh R  ARM Project |

INDEX

|  |  |  |
| --- | --- | --- |
| S.NO | Title | Page no |
| 1 | ABSTRCAT | 2 |
| 2 | COMPONENT USED | 2 |
| 3 | BLOCK DIAGRAM | 3 |
| 4 | EXPLANATION | 4 |
| 5 | CODE | 5 |
| 6 | OUTPUT | 22 |
| 7 | CONCLUSION | 27 |
| 8 | COMPONENT DETAILS |  |

Multi-sensor can-based vehicle dashboard with real time monitoring system

**Abstract:**

* The ARM-Vehicle Dashboard Real-Time Monitoring System is developed using the ARM7-TDMI (LPC2129) microcontroller to provide drivers with accurate, real-time information about their vehicle’s critical parameters.
* The system monitors important vehicle parameter like fuel level, engine temperature, and vehicle speed using appropriate sensors. Analog signals from these sensors are converted to digital values using an ADC via SPI communication for measurement. Based on the processed data, the dashboard displays not only the current readings but also warning messages when the fuel is low, the engine temperature exceeds safe limits, or the vehicle speed goes beyond a set threshold. Communication between different modules is managed using the CAN protocol, ensuring reliable and timely data exchange.
* This project demonstrates the integration of embedded systems, sensor interfacing, real-time processing, and communication protocols, providing a compact and efficient solution for safe and informed driving.

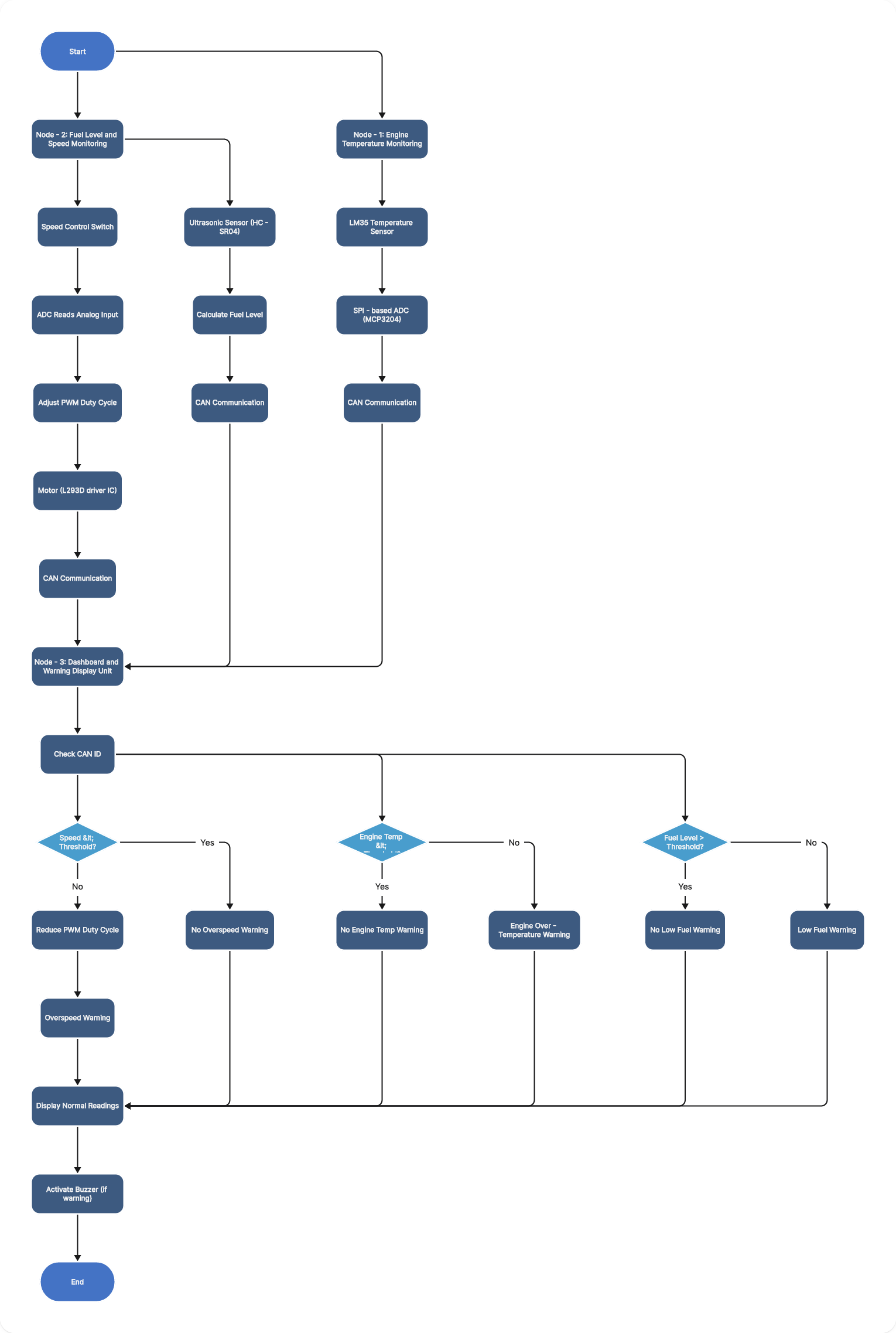
**Software Used:**

* Keil µVision IDE - Code writing, compiling, and debugging
* Flash Magic - Program uploading to controller
* Proteus - Circuit simulation

**Hardware Used:**

* ARM7-TDMI LPC2129 Microcontroller
* Ultrasonic sensor (HC-SR04)-Fuel Level Sensor
* LM-35 Temperature Sensor.
* Motor (L293D motor driver IC) – used to control the speed based on PWM.
* CAN Transceiver (MCP2551)
* LCD Display (20×4)
* Connecting Wires

**Block Diagram**

****

**Figure 1 flow chart -vehicle dashboard monitoring**

**Explanation:**

The ARM7-TDMI based LPC2129 microcontroller acts as the central processing unit of the vehicle dashboard monitoring system. The system is designed using a multi-ECU architecture, where different vehicle parameters are monitored by separate nodes and communicated to the dashboard node using the CAN protocol.

The system consists of three ECU nodes:

* Node-1: Engine temperature monitoring
* Node-2: Fuel level and speed monitoring
* Node-3: Dashboard and warning display unit

**Engine Temperature Monitoring (Node-1)**

* The main purpose of this node is monitoring the engine temperature continuously.
* The LM35 temperature sensor is used to monitor engine temperature. The LM35 provides an analog output voltage proportional to the temperature. This analog signal is converted into a digital value using a SPI-based external ADC (MCP3204).
* The converted temperature data is transmitted to Node-3 using CAN communication. Node-3 compares the received temperature value with the preset temperature threshold. If the temperature is within the safe limit, no warning is displayed. If the temperature exceeds the threshold, an engine over-temperature warning is shown on the LCD in the ECU-3.

**Fuel Level Monitoring (Node-2)**

* The main purpose of this node is monitoring the fuel level continuously.
* An ultrasonic sensor is used to monitor the fuel level in the tank. The sensor measures the distance between the fuel surface and the sensor. Based on this measured distance, the fuel level is calculated.
* The fuel level data is transmitted from Node-2 to the dashboard Node-3 through CAN communication. Node-3 compares the received value with predefined fuel threshold levels. If the fuel level is normal, no indication is shown. When the fuel level falls below the threshold, a low fuel warning message is displayed on the dashboard LCD.

**Speed Monitoring (Node-2)**

* Vehicle speed in the system is represented using PWM control generated by the **LPC2129** microcontroller. A switch is used to enable the speed control mechanism. When the switch is pressed, the ADC reads the analog input value, and based on this value, the PWM duty cycle is adjusted accordingly. The variation in duty cycle directly represents the vehicle speed using motor driver (L293D motor driver IC)— depends on ADC value, duty cycle will increase or decreased the speed.
* If the vehicle speed exceeds the predefined safe limit, the system automatically reduces the PWM duty cycle and brings the motor back to normal operating condition. A warning message is also displayed to indicate that the speed limit has been exceeded.

**Dashboard and Warning Unit (Node-3)**

* Node-3 acts as the central dashboard unit. It continuously receives messages from Node-1 and Node-2 through CAN communication. Each message is identified using a unique CAN ID, and based on the ID, the corresponding parameter is processed.
* If any abnormal condition is detected—such as low fuel level, high engine temperature, or overspeed—Node-3 displays the appropriate warning message on a 20×4 LCD and activates a buzzer to provide an audible alert to the driver.
* All sensor data is processed periodically using the internal timers of the LPC2129 to ensure accurate and real-time monitoring. By integrating sensing, processing, display, and CAN communication, the system enhances vehicle safety and demonstrates reliable real-time vehicle dashboard monitoring.

**Code**

**------------------------------------- 1. File name: LCD.h --------------------------------------**

10

/\* LCD header file \*/

#include <LPC21xx.h>

#include <stdio.h>

void delay\_ms (unsigned int);

#define LCD\_D (0xF << 14) // LCD pins P0.14–P0.17 (D4–D7)

#define RS (1 << 8) //p0.8 -RS

#define E (1 << 9) //p0.9 -EN

/\* LCD COMMAND BYTE \*/

void lcd\_cmd (unsigned char cmd)

{

IOCLR0 = LCD\_D;

IOSET0 = ((cmd >> 4) & 0x0F) << 14; // Upper nibble

IOCLR0 = RS;

IOSET0 = E;

delay\_ms (3);

IOCLR0 = E;

IOCLR0 = LCD\_D;

IOSET0 = (cmd & 0x0F) << 14; // Lower nibble

IOCLR0 = RS;

IOSET0 = E;

delay\_ms (3);

IOCLR0 = E;

}

/\* LCD DATA COMMAND \*/

void lcd\_data (unsigned char d)

{

IOCLR0 = LCD\_D;

IOSET0 = ((d >> 4) & 0x0F) << 14; // Upper nibble

IOSET0 = RS;

IOSET0 = E;

delay\_ms (3);

IOCLR0 = E;

IOCLR0 = LCD\_D;

IOSET0 = (d & 0x0F) << 14; // Lower nibble

IOSET0 = RS;

IOSET0 = E;

delay\_ms (3);

IOCLR0 = E;

}

/\* LCD INITIALIZATION FUNCTION \*/

void lcd\_init(void)

{

IODIR0 |= LCD\_D | RS | E;

lcd\_cmd(0x01); //Clear the lc display

lcd\_cmd(0x02); //Move the cursor to the home position

lcd\_cmd(0x0c); //display on cursor off

lcd\_cmd(0x28); //two row display

lcd\_cmd(0x80); // Clear

}

/\* INTEGER DATA TRANSFER TO THE LCD \*/

void lcd\_int (int n)

{

unsigned char arr [10];

int i = 0, j;

if (n == 0) {lcd\_data ('0'); return;

if (n < 0) {lcd\_data ('-'); n = -n;}

while (n > 0)

{

arr[i++] = (n % 10);

n /= 10;

}

for (j = i - 1; j >= 0; j--)

lcd\_data(arr[j] + '0');

}

/\*string data transfer to the lcd\*/

void lcd\_str (char \*s)

{

while (\*s)

lcd\_data(\*s++);

}

/\*FLOAT DATA TRANSFER TO THE LCD\*/

void lcd\_float (float n)

{

int i=0, f,j=0;

int cnt;

int arr [12];

float temp;

unsigned char flag=1;

if(n>0&&n<10)

{

temp=1000.0;

cnt=1;

}

else if(n>=10.0&&n<100.0)

{

temp=100.0;

cnt=2;

}

else if(n>=100.0&&n<1000.0)

{

temp=10.0;

cnt=3;

}

f=n\*temp;

while(f>0)

{

arr[i++] =f%10;

f/=10;

}

for (i=i-1; i>=0; i--)

{

if(j==cnt&&flag)

{

lcd\_data ('.');

i++; flag=0;

}

else

{

lcd\_data(arr[i]+'0');

}

j++;

}

}

void lcd\_inti (int n)

{

char a [10];

sprintf (a,"%d”, n);

lcd\_str(a);

}

void floate (float f)

{

int x=f;

lcd\_data(((x/10) %10) +48);

lcd\_data((x%10) +48);

x=f\*100;

lcd\_data ('.');

lcd\_data(((x/10) %10) +48);

lcd\_data((x%10) +48);

lcd\_data ('0');

lcd\_data ('0');

}

/\* delay for microseconds \*/

void delay (unsigned int s)

{

T0PR = 60 - 1; // Prescaler for 1µs delay (15MHz PCLK)

T0TC = 0;

T0TCR = 0x01;

while (T0TC < s);

T0TCR = 0x00;

}

/\*DELAY for milliseconds\*/

void delay\_ms (unsigned int s)

{

T0PR = 60000 - 1; // Prescaler for 1µs delay (15MHz PCLK)

T0TC = 0;

T0TCR = 0x01;

while (T0TC < s);

T0TCR = 0x00;

}

---------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------- 2. File name: header.h -------------------------------------------------

/\*header.h\*/

#ifndef \_\_HEADER\_H\_

#define \_\_HEADER\_H\_

/\*typedef for alias name of data types \*/

typedef unsigned int u32;

typedef signed int s32;

typedef unsigned char u8;

typedef signed char s8;

typedef unsigned short int u16;

typedef signed short int s16;

/\* structure for can message transfer\*/

typedef struct CAN2

{ u32 id;

u32 rtr;

u32 dlc;

u32 byteA;

u32 byteB;

} CAN2\_MSG;

/\* delay function declaration\*/

void delay\_sec (u32 sec);

void delay\_ms (unsigned int ms);

/\*SPI function declaration\*/

u32 adc0\_read (u32 channel);

void adc0\_init(void);

void spi0\_init(void);

u8 spi0\_read (u8 data);

/\* CAN function declaration\*/

void can2\_init(void);

void can2\_tx (CAN2\_MSG m1);

void can2\_rx (CAN2\_MSG \*m2);

#endif

---------------------------------------------------------------------------------------------------------------------------------------------------

**------------------------------------------------ 3. ECU1.c --------------------------------------**

**ECU-1 – Engine temperature monitoring**

/\* Temperature Monitoring \*/

#include <lpc21xx.h>

#include "lcd.h"

/\* typedef using for creating alias name of data types \*/

typedef unsigned char u8;

typedef signed char s8;

typedef unsigned short int u16;

typedef signed short int s16;

typedef unsigned int u32;

typedef signed int s32;

typedef float f32;

typedef double f64;

u8 stat; // variable

#define CS 7 /\* SPI chip selection pin \*/

/\*structure for transfer the data through CAN \*/

struct st

{ u32 id;

u32 dlc;

u32 rtr;

u32 temp\_val;

u32 byteB;

}M1;

/\* spi initialization \*/

void spi\_init(void)

{

PINSEL0 |=0X00001500;

S0SPCCR = 150; // to set 100kbps (The SPI rate)

S0SPCR = 1<<5|1<<4|1<<3; //SPI module in master mode and selecting mode 3.

IODIR0 |= 1<<7; // pin no 7 as output pin

IOSET0 |=1<<7; // set the pin no 7

}

u8 SPI0(u8 data)

{

stat = S0SPSR; //clear SPIF

S0SPDR = data; // load SPI Tx reg

while(((S0SPSR>>7) &1) ==0); // wait for transmission to complete

return S0SPDR; // read data from SPI data reg, place into buffer

}

// function covert the analog output voltage to digital output voltage.

f32 read\_volt ()

{

u8 h, l;

f32 adc=0;

IOCLR0=1<<7;

SPI0(0x06);

h = SPI0(0X00);

l = SPI0(0x00);

IOSET0 = (1<<7);

adc=((h&0x0f) <<8) |l; //upper 4-byte data only transfer

adc=((adc\*3.3)/4096); //analog value convert into digital value

return adc;

}

/\*CAN Initialization function \*/

void can\_init(void)

{

PINSEL1|=0X14000; // setting the pins 0.23 & 0.24 as RD2 And TD2.

VPBDIV = 1;

C2MOD=0X01; //Reset mode.

AFMR=0X02; // Accepting all msg without acceptance filtering.

C2BTR=0X001C001D; // setting the BRP Value.

C2MOD=0X00; //Normal mode...

}

/\* CAN Transmission function \*/

void can\_tx (struct st M1)

{

C2TID1=M1.id;

C2TFI1 &= ~ (0xF << 16);

C2TFI1|=(M1.dlc<<16);

C2TFI1&=~ (1<<30);

C2TDA1=M1.temp\_val;

C2CMR= (1<<0) | (1<<5); // Start the transmission and selecting the transmit buffer 1.

while (! (C2GSR& (1<<3))); // waiting for transmit buffer to complete.

}

int main ()

{

lcd\_init (); //lcd initialization

spi\_init (); //SPI initialization

can\_init (); // CAN initialization

lcd\_str("Temperature");

M1.id=0x11; // id for SPI node.

M1.rtr=0;

M1.dlc=4;

while (1){

f32 temp = read\_volt ();

temp=temp\*100; //that digital value convert into temperature.

M1. temp\_val = temp;

lcd\_cmd(0xc0);

lcd\_str ("temp: ");

lcd\_float(temp);

lcd\_str("c");

can\_tx(M1);

delay\_ms (200);

}

}

**---------------------------------------------------------------------------------------------------------**

**-----------------------------------------------**  4. File Name: can.c ---------------------------------------------

/\* can2\_driver \*/

#include <LPC21xx.H>

#include "header.h"

/\*CAN initialization\*/

void can2\_init(void)

{

PINSEL1|=0x00014000; //P0.23-->RD2 & P0.24-->TD2

VPBDIV=1; //PCLK=60MHz

C2MOD=0x1; //CAN2 into Reset Mode

AFMR=0x2; //accept all receiving messages(data/remote)

C2BTR=0x001C001D; //B125Kbps @ PLCK=60MHz (BRP= (Pclk/bit rate\*16)-1)

C2MOD=0x0; //CAN2 into Normal Mode

}

/\*CAN transmission \*/

void can2\_tx (CAN2\_MSG m1)

{

C2TID1=m1.id;

C2TFI1=(m1.dlc<<16);

if(m1.rtr==0) //if data frame

{

C2TFI1&=~ (1<<30) ;//RTR=0

C2TDA1=m1.byteA; //lower 4bytes of data

C2TDB1=m1.byteB; //upper 4bytes of data

}

else

{

C2TFI1|= (1<<30); //RTR=1

}

C2CMR= (1<<0) | (1<<5); //Start TXmission & select Tx Buf1

while ((C2GSR& (1<<3)) ==0); //wait for data TXmission.

}

/\*CAN receiver\*/

void can2\_rx(CAN2\_MSG \*m2)

{

while((C2GSR&0x1)==0);

m2->id = C2RID;

m2->dlc=(C2RFS>>16) &0xF;

m2->rtr=(C2RFS>>30) &0x1;

if(m2->rtr==0) // data frame

{

m2->byteA=C2RDA;

m2->byteB=C2RDB;

}

C2CMR= (1<<2); //free receiver buffer(imp)

}

---------------------------------------------------------------------------------------------------------------------------------------------------

**---------------------------------------------- 5. File Name: ecu2.c --------------------------------------------------------**

**Node 2: Ultrasonic Sensor for Fuel Measurement.**

/\* ECU-2 Fuel Monitoring node \*/

#include <lpc21xx.h>

#include "lcd.h"

#include "header.h"

#define TRIG (1<<10)

#define ECHO (1<<11)

unsigned int count, distance;

int main ()

{

CAN2\_MSG m1;

can2\_init (); // CAN Initialization

lcd\_init (); // LCD Initialization

/\*sending data frame\*/

m1.id=0x1EF;

m1.rtr=0;//data frame

m1.dlc=4;

IODIR0 |= TRIG;

IODIR0 &= ~ECHO;

while (1) {

/\* Trigger pulse \*/

IOSET0 = TRIG;

delay (10);

IOCLR0 = TRIG;

/\* Wait for echo \*/

while (! (IOPIN0 & ECHO));

count = 0;

/\* Measure Echo Duration \*/

while (IOPIN0 & ECHO)

count++;

distance = count / 58; //Convert to Distance

/\* CAN Frame \*/

m1. byteA=distance;

m1. byteB=0;

can2\_tx(m1); //Transmit CAN Message

/\* Fuel Level Logic \*/

if (m1. byteA <=8)

{

lcd\_cmd(0x80);

lcd\_str ("Fuel Level :95% ");

}

else if (m1. byteA > 8 && m1. byteA <=15)

{

lcd\_cmd(0x80);

lcd\_str ("Fuel Level :70% ");

}

else if (m1. byteA > 15 && m1. byteA <=22)

{

lcd\_cmd(0x80);

lcd\_str ("Fuel Level :50% ");

}

else if (m1. byteA > 22 && m1. byteA <=30)

{

lcd\_cmd(0x80)

lcd\_str ("Fuel Level :30% ");

}

else if (m1. byteA > 30 && m1. byteA <=35)

{

lcd\_cmd(0x80);

lcd\_str ("Fuel Level :20% ");

}

else if (m1. byteA > 35)

{

lcd\_cmd(0x80);

lcd\_str ("Fuel Level: NO FUEL ");

}

delay\_ms (500);

}

}

--------------------------------------------------------------------------------------------------------------------------------------------------

**------------------------------------- 6. File name: ecu3.c ----------------------------------**

Node:3 - Main dashboard display warning message and real time values

/\* Main node and Speed Control\*/

#include <lpc21xx.h>

#include "header.h"

#include "lcd.h"

#define sw1 1<<12 // switch assign to PWM -ADC value

int speed, A=0;

void pwm\_init(void)

{

PINSEL1 |=1<<10;

PWMPR =60-1;

PWMMR0=0xfff;

PWMMCR=(0X02);

PWMPCR= (1<<13);

PWMTCR=(0X09);

}

/\* PWM UPDATE\*/

void pwm\_update(void)

{

PWMMR5=speed;

PWMLER=1<<5;

}

/\*main function\*/

int main ()

{

CAN2\_MSG m1;

lcd\_init ();

can2\_init ();

lcd\_str ("Vehicle Dashboard");

pwm\_init ();

speed = 0;

pwm\_update ();

while (1)

{

can2\_rx(&m1);

if (! (IOPIN0 & sw1)); // Switch pressed (active LOW)

{

lcd\_cmd(0x80);

delay\_ms (100); // Debounce

if (! (IOPIN0 & sw1))

{

A++;

IOSET0=1<<21;

lcd\_cmd(0xC0);

switch(A)

{

case 1:

lcd\_str ("Vehicle Started");

speed = 600;

break;

case 2:

lcd\_str ("Low Speed ");

speed = 1500;

break;

case 3:

lcd\_str ("Medium Speed ");

speed = 2500;

break;

default:

lcd\_str ("High Speed ");

speed = 4000;

A = 0;

break;

}

}

pwm\_update ();

while (! (IOPIN0 & sw1)); // Wait until button release

}

}

if (m1.id == 0x1EF)

{

if (m1. byteA <=8)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level :100% ");

}

else if (m1. byteA > 8 && m1. byteA <=15)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level :80% ");

}

else if (m1. byteA > 15 && m1. byteA <=22)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level :60% ");

}

else if (m1. byteA > 22 && m1. byteA <=30)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level :40% ");

}

else if (m1. byteA > 30 && m1. byteA <=35)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level :20% ");

}

else if (m1. byteA > 35)

{

lcd\_cmd(0x94);

lcd\_str ("Fuel Level: NO FUEL ");

}

}

if (m1.id == 0x11)

{

if (m1. byteA > 0)

{

lcd\_cmd(0xD4);

lcd\_str ("Low Temp");

}

}

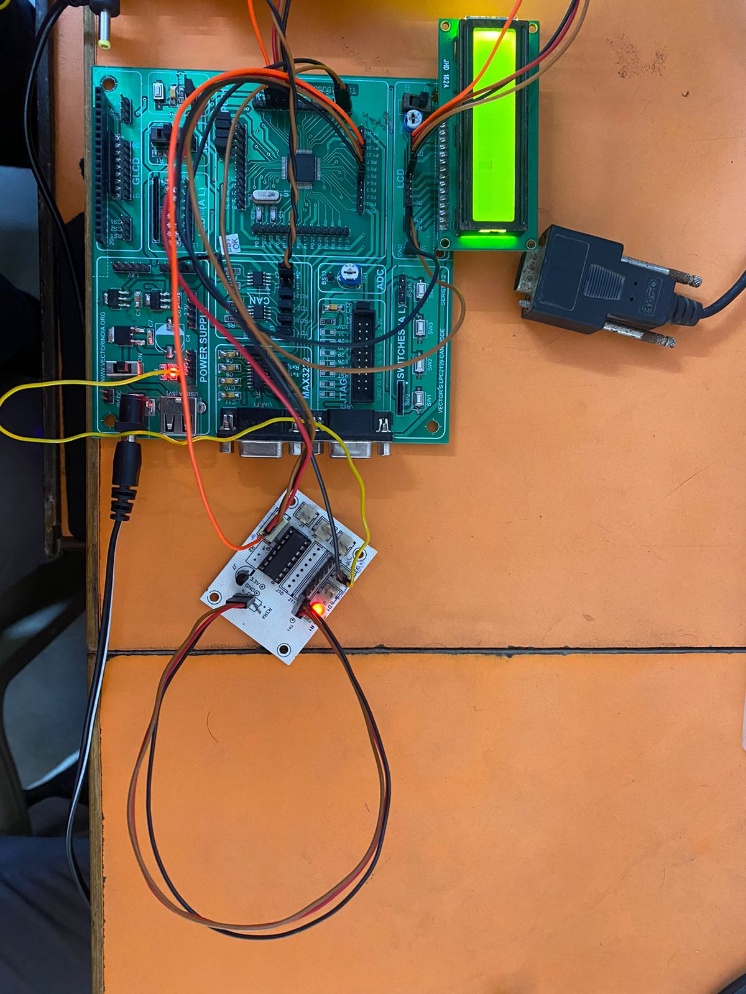
delay\_ms (100);

}

**--------------------------------------------------------------------------------------------------------**

**Output**

**Node 1 – Engine Temperature Monitoring Setup**

****

**Figure 2. Node 1 – Engine Temperature Monitoring Setup**

* In Node 1, the engine temperature is monitored using a temperature sensor connected to the **LPC2129** development board.
* First, the LPC2129 main board is powered using an external adapter supply. The temperature sensor is connected as follows:

**VCC** → 5V supply

**GND** → Ground

**Analog Output** → Connected to Channel 1 of the external ADC

* Since the system uses an external ADC (MCP3204), the sensor’s analog signal is converted into digital data through SPI communication.

The SPI connections are:

**P0.4 → SCK (Clock)**

**P0.5 → MISO**

**P0.6 → MOSI**

**P0.7 → CS (Chip Select)**

* These pins allow the LPC2129 to communicate with the ADC and read temperature values.
* The CAN transmission cable is connected to all nodes to enable communication between different ECUs in the system.

An LCD is connected to the board for monitoring and testing purposes. The LCD displays temperature readings and warning messages. This LCD connection is optional and mainly used for debugging and demonstration.

**Motor driver connection:**

****

**Figure 3. Node2-speed control using PWM- motor driver (L293D) setup**

In this setup, the motor driver section is used to control the DC motor based on the PWM signal generated by the **LPC2129**.

The ADC reads the analog input value. This ADC value is then used to adjust the PWM duty cycle. Based on the duty cycle:

* **Low ADC value → Low duty cycle → Motor rotates at low speed**
* **Medium ADC value → Medium duty cycle → Motor rotates at normal speed**
* **High ADC value → High duty cycle → Motor rotates at high speed**

In this setup, the motor is controlled using the motor driver section (L293D) on the LPC2129 development board.

* **P0.21 (PWM5)** of the **LPC2129** is connected to the **1A (Input 1)** pin of the motor driver.
* The PWM5 signal from P0.21 controls the motor speed.
* The motor is connected to the driver output pins:
  + **1Y (Red wire)**
  + **2Y (Black wire)**
* The PWM signal is given to the motor driver IC (L293D section on the board). The motor driver amplifies the signal because the microcontroller cannot directly drive the motor due to current limitations or cannot directly supply enough current to drive the motor.
* So, the motor speed is completely dependent on the PWM value, which is controlled by the ADC reading

**Node 2 – Ultrasonic Sensor (HC-SR04) Connection (Fuel Level Monitoring)**

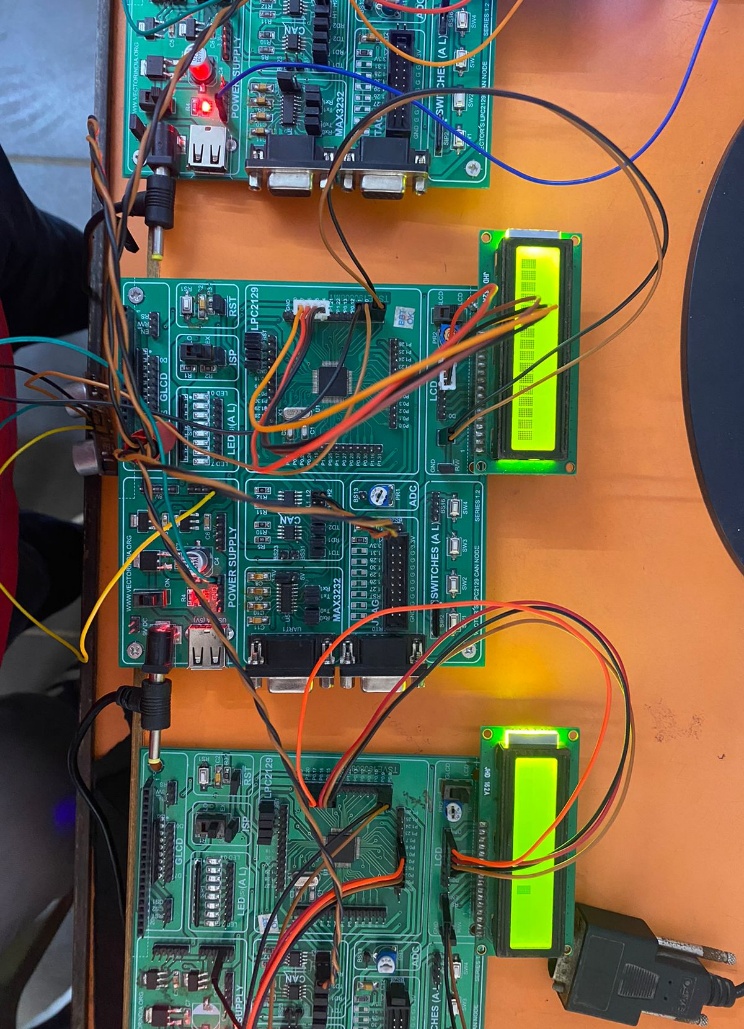
****

Figure 4.Node-2 : ultrasonic sensor for fuel monitoring -node2 connection

* In Node 2, the ultrasonic sensor is used to monitor the fuel level inside the tank. The system is built using the **LPC2129**.
* First, the LPC2129 development board is powered using an external adapter supply.

**🔹 Ultrasonic Sensor Connections:**

* **VCC → 5V supply**
* **GND → Ground**
* **TRIG → P0.10**
* **ECHO → P0.11**

The TRIG pin is configured as output, and the ECHO pin is configured as input.

The microcontroller sends a 10µs pulse to the TRIG pin. The ultrasonic sensor transmits sound waves, and when the echo returns, the ECHO pin becomes HIGH. The LPC2129 measures the time duration of the echo signal to calculate the distance. Based on this distance, the fuel level is determined.

If the fuel level falls below a predefined limit, a warning message is generated and transmitted through the CAN network.

**Node3: Main Dashboard Node – LCD and CAN Connection**

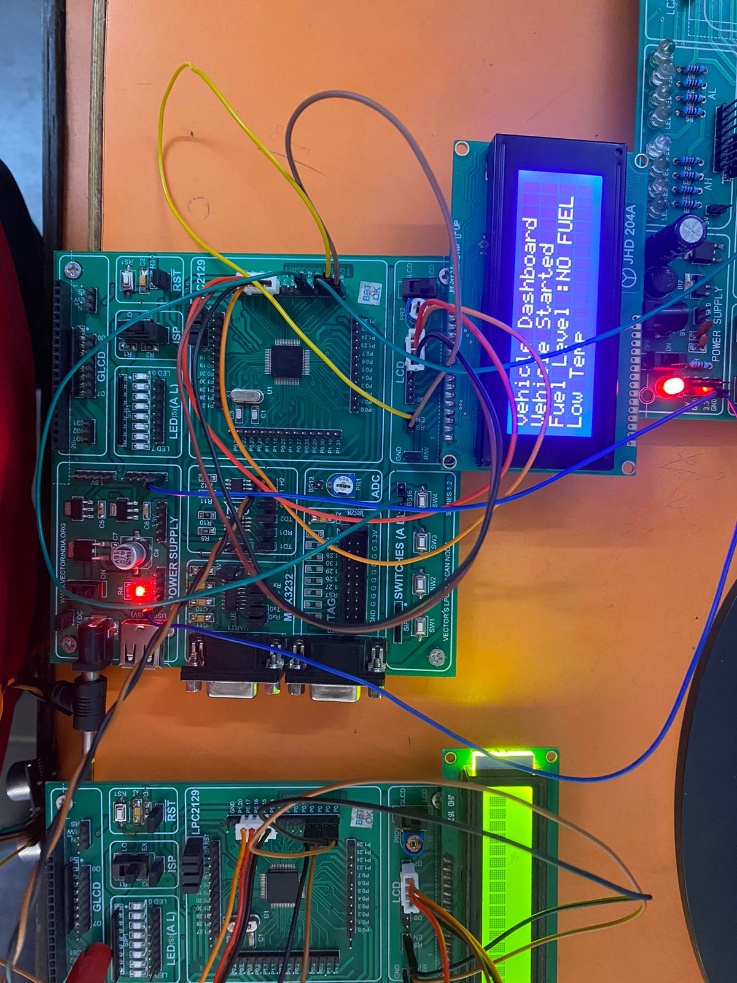
****

Figure 5.node-3 vehicle main dashboard

* In this section, the main board acts as the central dashboard unit. It receives data from all other nodes through CAN communication and displays the information on a 20x4 LCD.
* The system is built using the LPC2129 microcontroller.

**LCD Connection (20x4 Display – 4-bit Mode)**

* The LCD is connected in 4-bit mode to reduce pin usage.

Pin Connections:

* **P0.4 – P0.7 → LCD Data Pins (D4–D7)**
* **P0.8 → RS (Register Select)**
* **P0.9 → EN (Enable)**

In 4-bit mode:

* Data is sent in two parts (higher nibble and lower nibble).This saves microcontroller pins.

The LCD displays:

* Warning messages
* Vehicle status
* Fuel level
* Engine temperature

**CAN Communication**

The CAN transmission cable is connected between all nodes in the system. It allows communication between:

* **Node 1 → Temperature Monitoring**
* **Node 2 → Fuel Level Monitoring**
* **Node 3 → Speed Control**
* This ensures reliable data transfer between ECUs.

Each node sends its data to the main dashboard node through CAN. The main LPC2129 receives the CAN messages and updates the LCD accordingly.

This ensures:

* Reliable communication
* Real-time data transfer
* Proper synchronization between ECUs

**Videos:**

Main board and all the node monitoring









**Conclusion:**

* This project successfully implements a CAN-based vehicle dashboard real-time monitoring system using the ARM7-TDMI LPC2129 microcontroller. The system effectively monitors critical vehicle parameters such as fuel level, engine temperature, and vehicle speed using a multi-ECU architecture. Sensor data from different nodes is reliably transmitted to the dashboard node through the CAN protocol.
* The dashboard unit accurately processes the received data and provides real-time visual warnings on a 20×4
* LCD along with audible alerts using a buzzer whenever abnormal conditions such as low fuel level, engine overheating, or overspeed are detected. The use of CAN communication ensures robust, noise-free, and reliable data transfer, making the system suitable for automotive environments.
* Overall, this project demonstrates the practical application of ARM7 microcontrollers, sensor interfacing, SPI-based ADC, PWM control, and CAN communication in automotive embedded systems, thereby enhancing vehicle safety and driver awareness**.**

**Ultrasonic Sensor – HC-SR04**

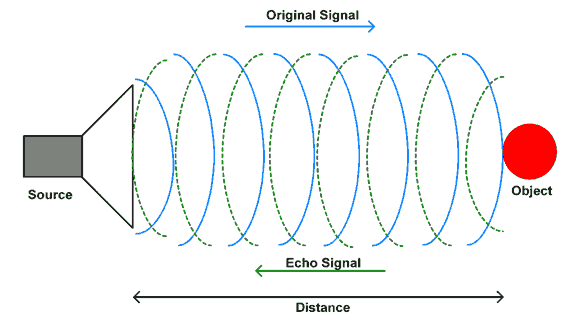
**HC-SR04 Pin Description**



* **VCC:** +5 V supply
* **TRIG:** Trigger input of sensor. Microcontroller applies 10 us trigger pulse to the HC-SR04 ultrasonic module.
* **ECHO:** Echo output of sensor. Microcontroller reads/monitors this pin to detect the obstacle or to find the distance.
* **GND:** Ground

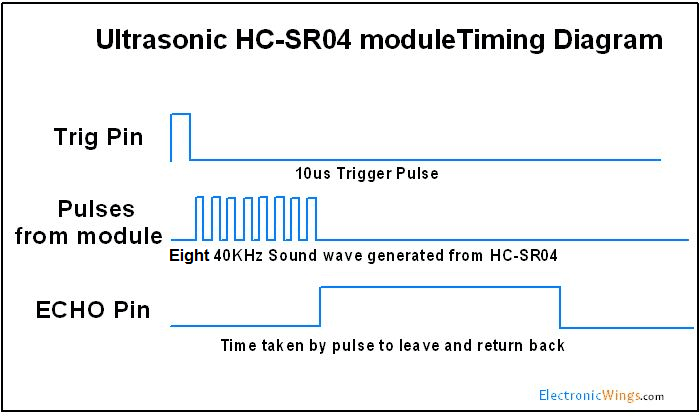
**Overview of ultrasonic sensor:**

* The ultrasonic sensor works on the principle of **SONAR** and **RADAR** system which is used to determine the distance to an object.
* An ultrasonic sensor generates high-frequency sound (ultrasound) waves. When this ultrasound hits the object, it reflects as echo which is sensed by the receiver as shown in below figure.



* By measuring the time required for the echo to reach to the receiver, we can calculate the distance.
* This is the basic working principle of Ultrasonic module to measure distance  
  HC-SR-04 has an ultrasonic transmitter, receiver and control circuit.
* In the ultrasonic module HCSR04, we must give trigger pulse, so that it will generate ultrasound of frequency 40 kHz. After generating ultrasound i.e. 8 pulses of 40 kHz, it makes echo pin high. Echo pin remains high until it does not get the echo sound back. So, the width of echo pin will be the time for sound to travel to the object and return. Once we get the time we can calculate distance, as we know the speed of sound.
* Ultrasonic Sensor HC-SR04 can measure up to **range from 2 cm - 400 cm**.

**Ultrasonic Sensor HC-SR04 Working Principle**



1. We need to transmit trigger pulse of at least 10 us to the HC-SR04 Trig Pin.
2. Then the HC-SR04 automatically sends Eight 40 kHz sound wave and wait for rising edge output at Echo pin.
3. When the rising edge capture occurs at Echo pin, start the Timer and wait for falling edge on Echo pin.
4. As soon as the falling edge is captured at the Echo pin, read the count of the Timer. This time count is the time required by the sensor to detect an object and return from an object.

**Basic ultrasonic distance formula:**

**But ultrasonic sensor :**

Why **divide by 2**?

When the sensor sends sound:

* Sound travels from sensor → object
* Sound reflects and comes back → object → sensor

So, the measured time is for **going + returning**

**Why 343**?

The speed of sound waves is=n air at room temperature(25°c): **343 m/s.**

But be careful:

343 is in **meters per second**(m/s). Echo time is usually in **microseconds(**µs**).**

So, we convert units properly.

**Converting to cm**

Speed of sound: 343 m/s = 0.0343 cm/µs

So:

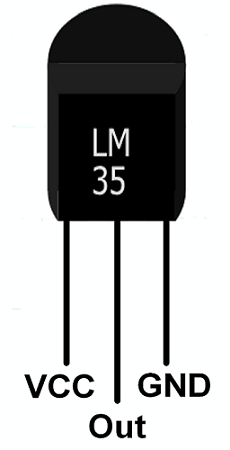
Now,

So, we simplify as:

**Alternate options for HC-SR04 Ultrasonic Sensor**

* **Maxbotix Ultrasonic Sensors**: These sensors offer a wider range of detection and higher accuracy compared to the HC-SR04. They are available in different models with varying ranges and beam patterns.
* **Sharp Infrared Sensors**: These sensors use infrared light to measure distances and are available in a range of detection distances. They are suitable for use in low light conditions and have a faster response time compared to ultrasonic sensors.
* **Lidar Sensors**: These sensors use laser light to measure distances and provide high accuracy and precision. They are commonly used in robotics and automation applications and are available in different models with varying ranges and resolutions.
* **Time-of-Flight (ToF) Sensors**: These sensors use infrared light to measure distances and are suitable for use in applications that require high accuracy and fast response times. They are available in different models with varying ranges and resolutions.
* **Ultrasonic Ranging Module HC-SR05**: This is a similar module to the HC-SR04 with minor differences, it has a wider detection range of up to 4.5 meters and a higher accuracy.

**LM35-temprature sensor:**



**VCC:**Supply Voltage (4V – 30V)

**GND:**Ground

**Out:**It gives analog output voltage which is proportional to the temperature (in degree Celsius).

**Introduction**

* LM35 is a precision analog temperature sensor.
* LM35 is a temperature measuring device having an analog output voltage proportional to the temperature.
* It provides output voltage in Centigrade (Celsius). It does not require any external calibration circuitry. So output is linear.
* The sensitivity of LM35 is 10 mV/degree Celsius. As temperature increases, output voltage also increases.

          E.g. 250 mV means 25°C.

* It is a 3-terminal sensor used to measure surrounding temperature ranging from -55 °C to 150 °C.
* LM35 gives temperature output which is more precise than thermistor output.

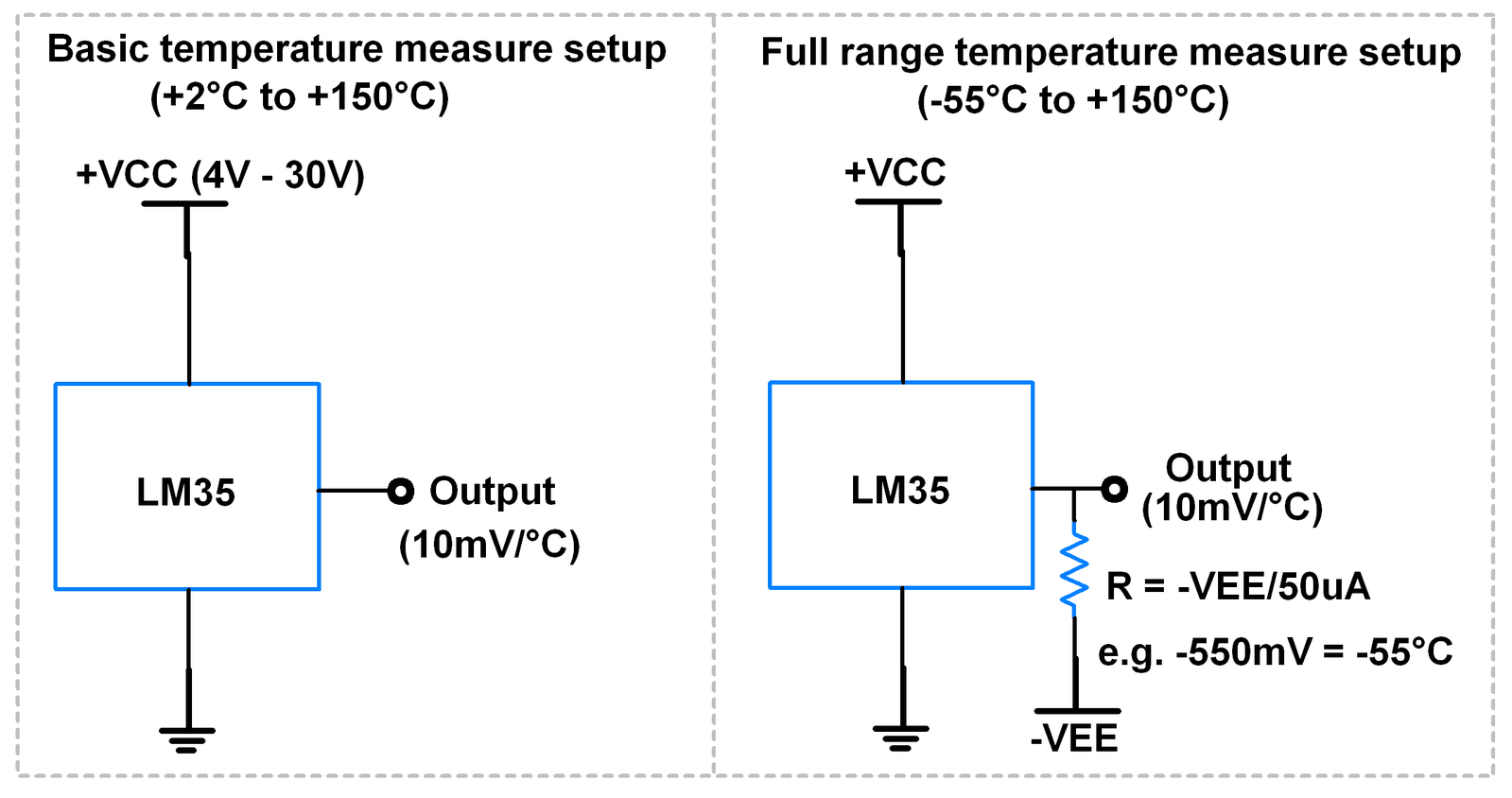
**Specification of LM35 Temperature Sensor**

* Operating Voltage: 4 V to 30 V
* Output Voltage: 10mV/°C
* Sensitivity: 10mV/°C
* Linearity Error: ±1°C (for 0°C to +100°C)
* Operating Temperature: -55°C to +150°C
* Output Impedance: 100 Ω
* Power Consumption: 60 μA (typical)
* Package Type: TO-92, TO-220, SOIC
* Output Type: Analog
* Accuracy: ±1°C (typical)

**Alternate options for LM35 Sensor**

* TMP36
* DHT11
* DS18B20
* LM34
* RTD PT100

**Application Setup**



**LPC2129 Reads Temperature**

Since LM35 gives **analog output**, the microcontroller must convert it to digital using ADC.

Steps:

1. LM35 gives analog voltage.
2. ADC converts voltage to digital value.
3. Microcontroller calculates temperature.

If ADC reference value =3.3v for LPC2129

Ex ADC value =155

Step 1: Convert ADC to voltage

Step 2: Convert to temperature

**PWM-Pulse width modulation**

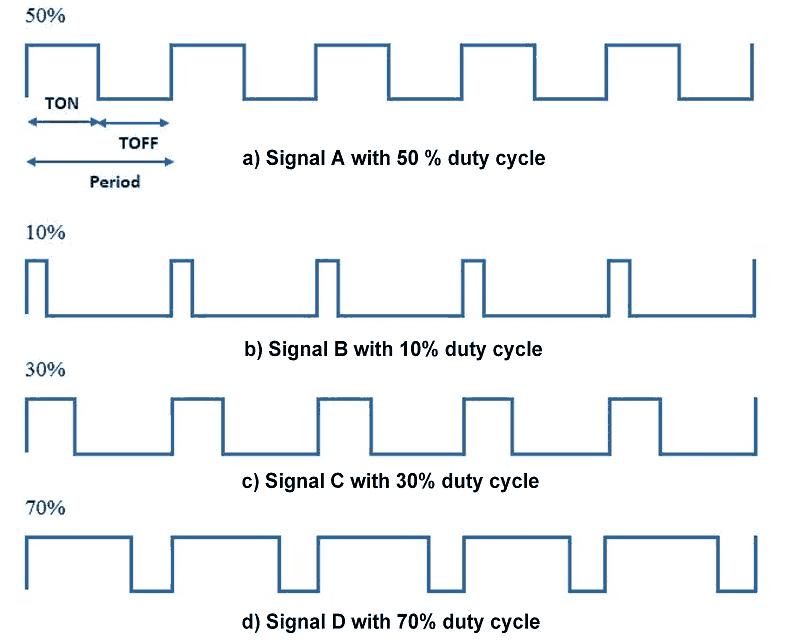
**Pulse Width Modulation (PWM)** is a technique by which width of a pulse is varied while keeping the frequency of the wave constant. It is a method for generating an analog signal using a digital source.



A PWM signal consists of two main components that define its behaviour: a **duty cycle** and a **frequency**.

A period of a pulse consists of an **ON** cycle (5V) and an **OFF** cycle (0V). The fraction for which the signal is ON over a period is known as a **duty cycle**.

Through PWM technique, we can control the power delivered to the load by using ON-OFF signal. The PWM signals can be used to control the speed of DC motors.



**Frequency of Signal**

The frequency of a signal determines how fast the PWM completes a cycle (i.e. 1000 Hz would be 1000 cycles per second) which means how fast it switches between ON (high) and OFF (low) states. By repeating this ON-OFF pattern at a fast-enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices.

**Overview of DC Motor**



* DC motor converts electrical energy in the form of Direct Current into mechanical energy in the form of rotational motion of the motor shaft.
* The DC motor speed can be controlled by applying varying DC voltage; whereas the direction of rotation of the motor can be changed by reversing the direction of current through it.
* For applying varying voltage, we can make use of PWM technique.
* L293D motor driver IC is used for controlling the direction of the motor.
* PWM wave generated on the LCP2129 is used to provide a variable voltage to the motor through L293D.

**ADVANTAGES AND DISADVANTAGES:**

**1️⃣ Ultrasonic Sensor (HC-SR04)**

**Advantages**

* Non-contact measurement
* Good accuracy
* Low cost
* Easy to interface
* Safe (no radiation)

**Dis-advantages**

* Affected by temperature
* Affected by soft/irregular surfaces
* Limited range (~2cm to 400cm)
* Not suitable for vacuum

**2️⃣ Temperature Sensor (LM35)**

**Advantages**

* Linear output (10mV/°C)
* No calibration required
* Direct Celsius output
* Low power consumption
* Low cost

**Disadvantages**

* Analog output (needs ADC)
* Noise sensitive
* Limited long-distance transmission
* Less accurate than digital sensors

**3️⃣ PWM (Pulse Width Modulation)**

**Advantages**

* Efficient motor speed control
* Low power loss
* Smooth speed variation
* Easy implementation in LPC2129

**Disadvantages**

* Generates switching noise
* Requires filtering in some applications
* Not suitable for very high precision analog control

**4️⃣ DC Motor Driver (Example: L293D)**

**Advantages**

* Protects microcontroller
* Provides higher current
* Supports bidirectional control
* Simple interfacing

**Disadvantages**

* Voltage drop across driver
* Heat generation
* Limited current capacity

**5️⃣ SPI Protocol**

**Advantages**

* High speed communication
* Full duplex
* Simple hardware
* Low overhead

**Disadvantages**

* More wires required
* No built-in error checking
* Not suitable for long distance
* No addressing system

**6️⃣ CAN Protocol**

**Advantages**

* High reliability
* Error detection & correction
* Multi-master system
* Reduces wiring
* Suitable for automotive
* Real-time communication

**Disadvantages**

* Complex compared to SPI
* Limited data length (8 bytes per frame – classical CAN)
* Cost of CAN transceiver
* Requires proper termination

**8️⃣ LPC2129 Microcontroller**

**Advantages**

* 32-bit ARM7 processor
* High speed (up to 60 MHz)
* Built-in ADC
* Built-in CAN controller
* PWM module available
* Multiple communication protocols (SPI, UART, I2C)
* Low power consumption

**Disadvantages**

* ARM7 is older architecture
* Limited RAM compared to modern MCUs
* Programming is slightly complex
* Requires external debugging tools