



## **ECE 612: REAL TIME EMBEDDED SYSTEMS**

A project report on

# **TEMPERATURE MONITORING ON DETECTION OF PERSON USING CAN PROTOCOL ON CMIS RTOS**

Under the guidance of **Dr.A.C.Sabzevari**

# INDEX

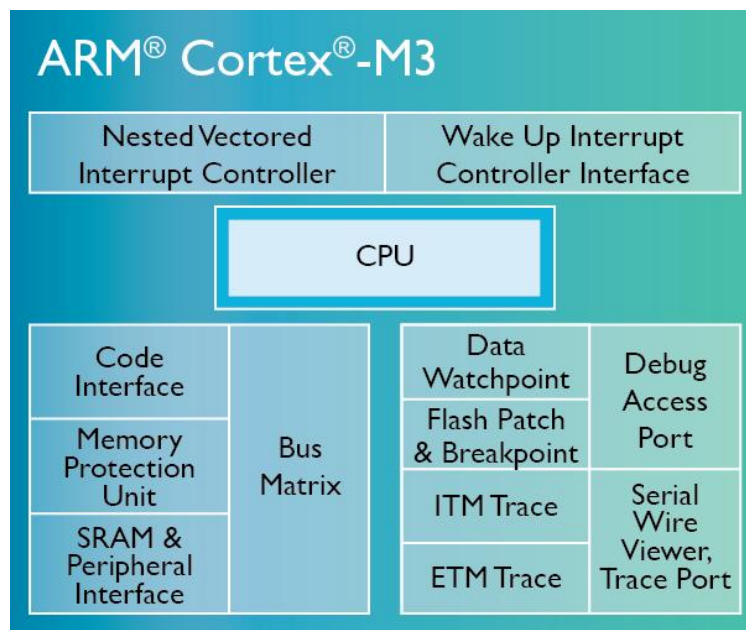
• Abstract .....	3
• Components .....	3
• Cortex M3 Processor .....	3
• Mbed NXP LPC 1768.....	4
• LM35 Temperature Sensor .....	4
• MCP 2551.....	4
• CMIS RTOS.....	5
• MBED Compiler .....	6
• IR Transceiver .....	6
• Serial Communication .....	7
• CAN protocol .....	7
• Hardware Connection .....	9
• Software Flow .....	11
• Code .....	12
• Output .....	14

**ABSTRACT:**

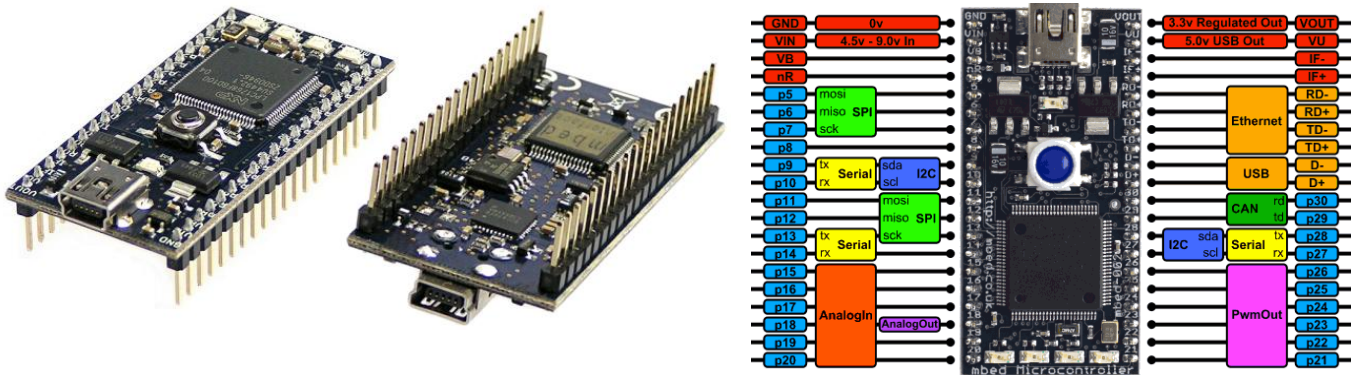
This project aims to understand the basics of real time operating systems. This is implemented by using a temperature sensor which takes reading on detecting the presence of a person on one node and sends these readings to other node via CAN. On the second node the reading is collected it is displayed on the terminal via serial port.

**COMPONENTS:**

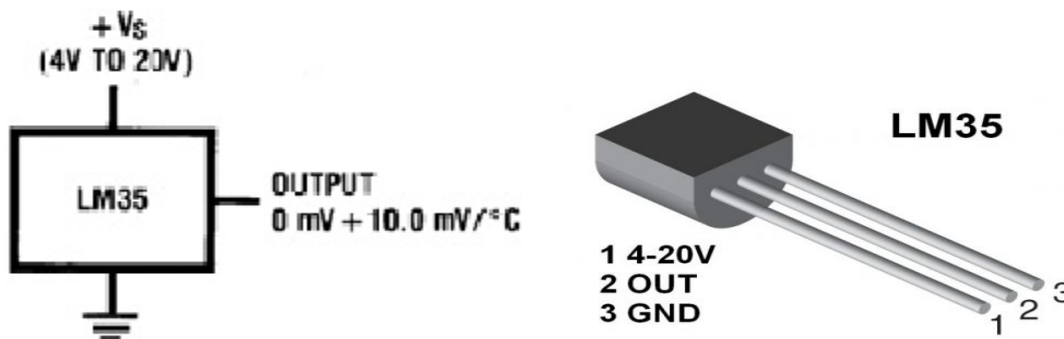
- Cortex M3 processor** - The ARM Cortex-M3 processor is the industry-leading 32-bit processor for highly deterministic real-time applications, specifically developed to enable partners to develop high-performance low-cost platforms for a broad range of devices including micro-controllers, automotive body systems, industrial control systems and wireless networking and sensors. Introduced in 2004 and recently updated with new technologies and configurability, the Cortex-M3 is the mainstream ARM processor developed specifically with micro-controller applications in mind. With high performance and low dynamic power consumption the Cortex-M3 processor delivers leading power efficiency. Coupled with integrated sleep modes and optional state retention capabilities the Cortex-M3 processor ensures there is no compromise for applications requiring low power and excellent performance. The processor executes Thumb-2 instruction set for optimal performance and code size, including hardware division, single cycle multiply, and bit-field manipulation. The ARM Cortex-M processors utilize the ARM Thumb-2 technology which provides excellent code density. With Thumb-2 technology, the Cortex-M processors support a fundamental base of 16-bit Thumb instructions, extended to include more powerful 32-bit instructions. In many cases a C compiler will use the 16-bit version of the instruction unless the operation can be carried out more efficiently using a 32-bit version. The Cortex-M3 NVIC is highly configurable at design time to deliver up to 240 system interrupts with individual priorities, dynamic reprioritization and integrated system clock. The combination of features and performance enables Cortex-M3 based devices efficiently to handle multiple I/O channels and protocol standards such as USB OTG (On-The-Go).



- **mbed NXP LPC1768** - It is a rapid prototyping board with 40 pin DIP, Cortex M3 hardware. It features a 100 MHz ARM processor with 64 KB of SRAM and 512 KB of flash. It has facilities for Ethernet, USB OTG, SPI, I2C, UART, CAN. The board has 12 bit ADC with eight channels and 10 bit DAC. Other peripherals include ultra low power (< 1uA) RTC, general purpose DMA controller with eight channels, up to 70 GPIO and four 32 bit general purpose timers/counters. It also has a motor control PWM and Quadrature Encoder interface to support 3 phase motors.



- **LM35 Temperature sensor** - LM35 is a precision IC temperature device with an output voltage linearly proportional to the centigrade temperature, for example, if the temperature is 33°C, then the output voltage shall be 330mV. It is rated to operate for full temperature range of (-55) deg. C to 150 deg.C. The operating voltage for LM35 is 4V to 30V with less than 60 uA current drain.



- **MCP 2551** - It is a 8 pin DIP, high speed CAN transceiver fully compatible with ISO- 11898 requirements which supports 1Mb/s operation. It is suitable for suitable for 12V and 24 V systems. The CAN bus has two states: Dominant and Recessive. A Dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g., 1.2V). A Recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). The Dominant and Recessive states correspond to the Low and High state of the TXD input pin, respectively. However, a Dominant state initiated by another CAN node will override a Recessive state on the CAN bus. The MCP2551 CAN outputs will drive a minimum load of 45Ω, allowing a maximum of 112 nodes to be connected (given a minimum differential input resistance of 20 kΩ and a nominal termination resistor value of 120Ω). The RXD output pin reflects the differential bus voltage between CANH and CANL. The Low and High states of the RXD output pin correspond to the Dominant and Recessive states of the

CAN bus, respectively. It has 3 modes of operation i.e. high speed, slope control and standby which is allowed by controlling the Rs pin.

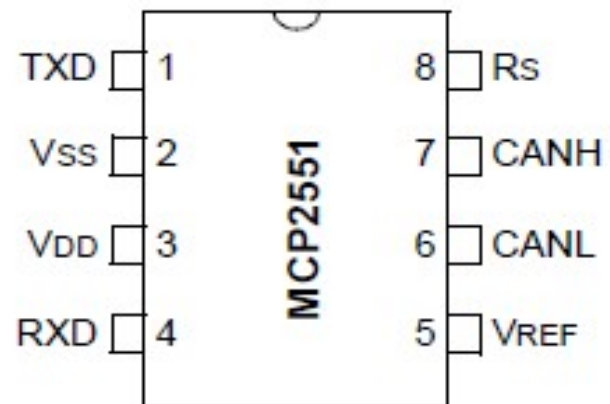
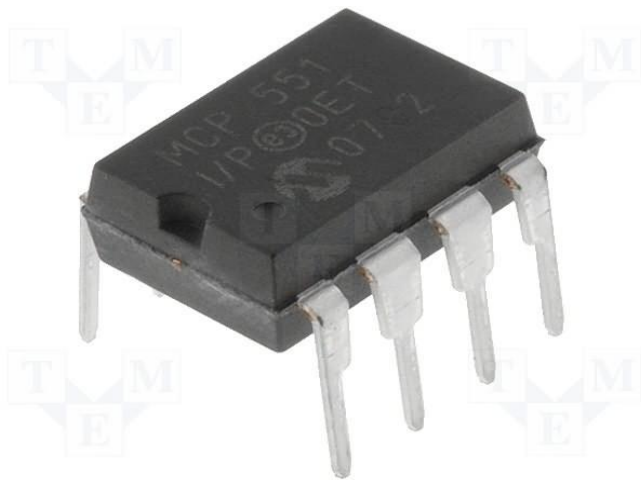


TABLE 1-3: MCP2551 PINOUT

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	Vss	Ground
3	VDD	Supply Voltage
4	RXD	Receive Data Output
5	VREF	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	Rs	Slope-Control Input

- CMSIS RTOS** - The ARM Cortex Micro-controller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. Creation of software is a major cost factor in the embedded industry. By standardizing the software interfaces across all Cortex-M silicon vendor products, especially when creating new projects or migrating existing software to a new device, means significant cost reductions. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new micro-controller developers and cutting the time-to-market for devices.

The CMSIS consists of the following components:

**CMSIS-CORE:** API for the Cortex-M processor core and peripherals. It provides a standardized interface for Cortex-M0, Cortex-M3, Cortex-M4, SC000, and SC300. Included are also SIMD intrinsic functions for Cortex-M4 SIMD instructions.

**CMSIS-Driver:** defines generic peripheral driver interfaces for middleware making it reusable across supported devices. The API is RTOS independent and connects micro-controller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces.

**CMSIS-DSP:** DSP Library Collection with over 60 Functions for various data types: fix-point (fractional q7, q15, q31) and single precision floating-point (32-bit). The library is available for Cortex-M0, Cortex-M3, and Cortex-M4. The Cortex-M4 implementation is optimized for the SIMD instruction set.

**CMSIS-RTOS API:** Common API for Real-Time operating systems. It provides a standardized programming interface that is portable to many RTOS and enables therefore software templates, middleware, libraries, and other components that can work across supported the RTOS systems.

**CMSIS-Pack:** describes with a XML based package description (PDSC) file the user and device relevant parts of a file collection (called software pack) that includes source, header, and library files, documentation, Flash programming algorithms, source code templates, and example projects. Development tools and web infrastructures use the PDSC file to extract device parameters, software components, and evaluation board configurations.

**CMSIS-SVD:** System View Description for Peripherals. Describes the peripherals of a device in an XML file and can be used to create peripheral awareness in debuggers or header files with peripheral register and interrupt definitions.

**CMSIS-DAP:** Debug Access Port. Standardized firmware for a Debug Unit that connects to the Core Sight Debug Access Port. CMSIS-DAP is distributed as separate package and well suited for integration on evaluation boards. This component is provided as separate download.

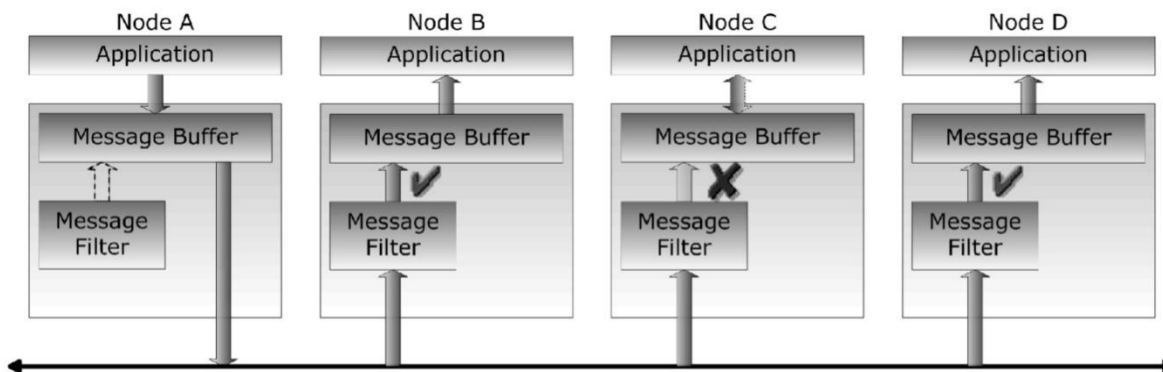
- **MBED COMPILER :** The mbed Compiler delivers full online editor, standard keyboard shortcuts, flexible workspace management and easy to use version control via intuitive interface design and innovative workflow, optimized for the mbed eco-system. List of the Editor features:

1. Standard C/C++ syntax highlighting
2. Python, Lua, JavaScript, XML, HTML, CSS syntax highlighting
3. Unlimited undo/redo operations buffer
4. Compliant clipboard text operations - cut/copy/paste
5. "Artistic Style" code formatting for text selections or whole file
6. Find text with case, word and regular expression matching
7. Find in Files with filtering capabilities
8. Mouse, keyboard, touch/tap navigation
9. Text selection block comment and code indention manipulation
10. UTF-8/Unicode base encoding
11. Print-friendly code preview

- **IR TRANSCEIVER :** The model is 276-0142. This is used for edge detection system. IR LED having a reverse breakdown voltage of 5V, continuous forward current of 100mA and wavelength of 850nm, is connected to 5VDC supply coming directly from NXP LPC1768 board. IR receiver which is placed head to head with IR LED, and its output is taken at the cathode which is connected to the GPIO pin(pin 19) of NXP LPC1768 board which receives the input signal from the receiver. IR LED is connected to 5VDC supply, which is continuously transmitting IR rays in the wavelength of 850 nm (peak wavelength), received by the IR receiver, which is head-to-head to the IR LED. Whenever an object is detected i.e. it cuts the IR rays from the trans receiver pair, it sends a low pulse to the NXP LPC1768 board, which is detected by polling control structure i.e. continuous polling of the signals from IR transmitter receiver

pair. With the active low pulse detection, in the polling loop the LED glows. We have also implemented a counter in it that keeps a count of the number of times the loop has been activated, which is used for our application of people counter.

- **Serial Communication with PC :** The mbed Microcontroller can communicate with a host PC through a "USB Virtual Serial Port" over the same USB cable that is used for programming. Your mbed Microcontroller can appear on your computer as a serial port. It is common to use a *terminal application* on the host PC to communicate with the mbed Microcontroller.
- **CAN protocol :** A controller area network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for automotive applications, but is also used in many other contexts. CAN data transmission uses a lossless bit-wise arbitration method of contention resolution. This arbitration method requires all nodes on the CAN network to be synchronized to sample every bit on the CAN network at the same time. The CAN specifications use the terms "dominant" bits and "recessive" bits where dominant is a logical 0 (actively driven to a voltage by the transmitter) and recessive is a logical 1 (passively returned to a voltage by a resistor). The idle state is represented by the recessive level (Logical 1). If one node transmits a dominant bit and another node transmits a recessive bit then there is a collision and the dominant bit "wins". This means there is no delay to the higher-priority message, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message. This makes CAN very suitable as a real time prioritized communications system. General transmission of message using CAN :



- Node A transmits a message
- Nodes B, C and D receive the message
- Nodes B and D accept the message, Node C declines

CAN has four frame types :

1. *Data frame:* a frame containing node data for transmission
2. *Remote frame:* a frame requesting the transmission of a specific identifier
3. *Error frame:* a frame transmitted by any node detecting an error
4. *Overload frame:* a frame to inject a delay between data and/or remote frame

#### Data Frame :

The data frame is the only frame for actual data transmission. There are two message formats:



- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires the implementation must accept the base frame format and may accept the extended frame format, but must tolerate the extended frame format.

Components of the data frame :

*Start of frame* – 1 dominant bit. A frame can only start when the bus is IDLE. All stations synchronize to the leading edge of the SOF bit

*Identifier* – 11 (or 29 in version 2.0) bits. In order from most significant to least significant. The 7 most significant bits cannot be all recessive

*RTR* – remote transmission request, dominant for REQUEST frames, recessive for DATA frames

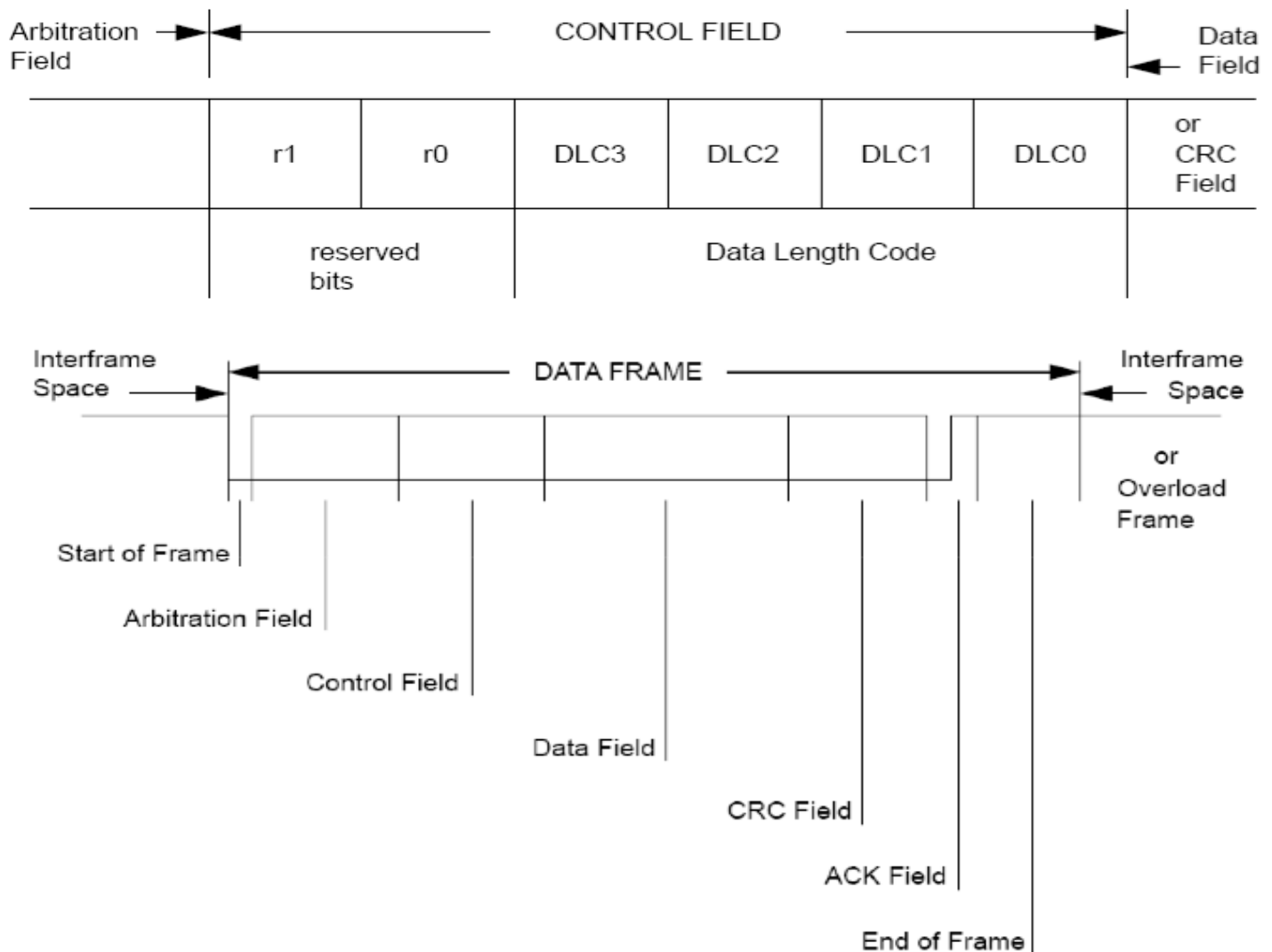
*CONTROL* – (see figure) maximum data length is 8 (bytes) other values are not used

*Data* – 0 to 8 bytes of data

*CRC* – 15 CRC bits plus one CRC delimiter bit (recessive)

*ACK* – two bits (SLOT + DELIMITER) all stations receiving the message correctly (CRC check) set the SLOT to dominant (the transmitter transmits a recessive). The DELIMITER is recessive

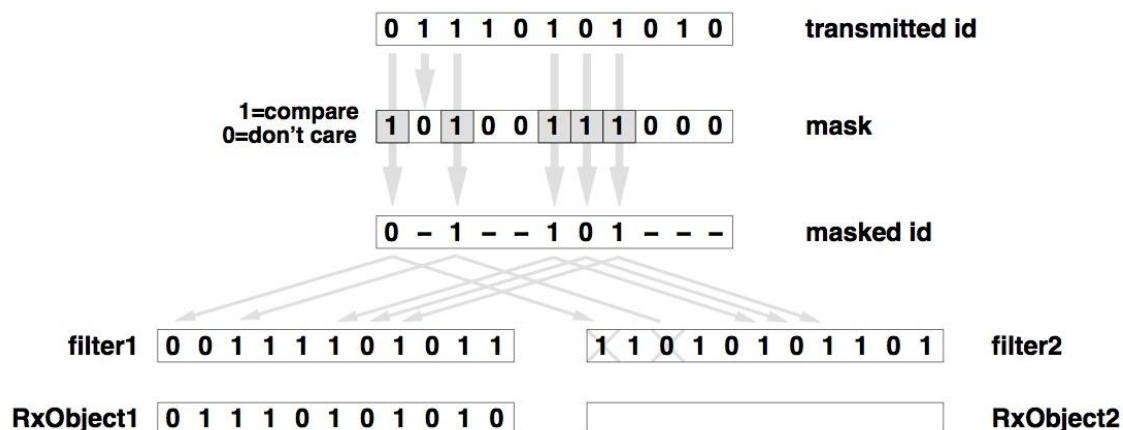
*END OF FRAME* – seven recessive bits





### Message Reception and Filtering:

Controllers have one or more registers (commonly defined as RxObjects) for the reception of CAN messages. Nodes can define one or more message *Filters* (typically one associated to each RxObject) and one or more reception *Masks* to declare the messages they are interested in receiving.

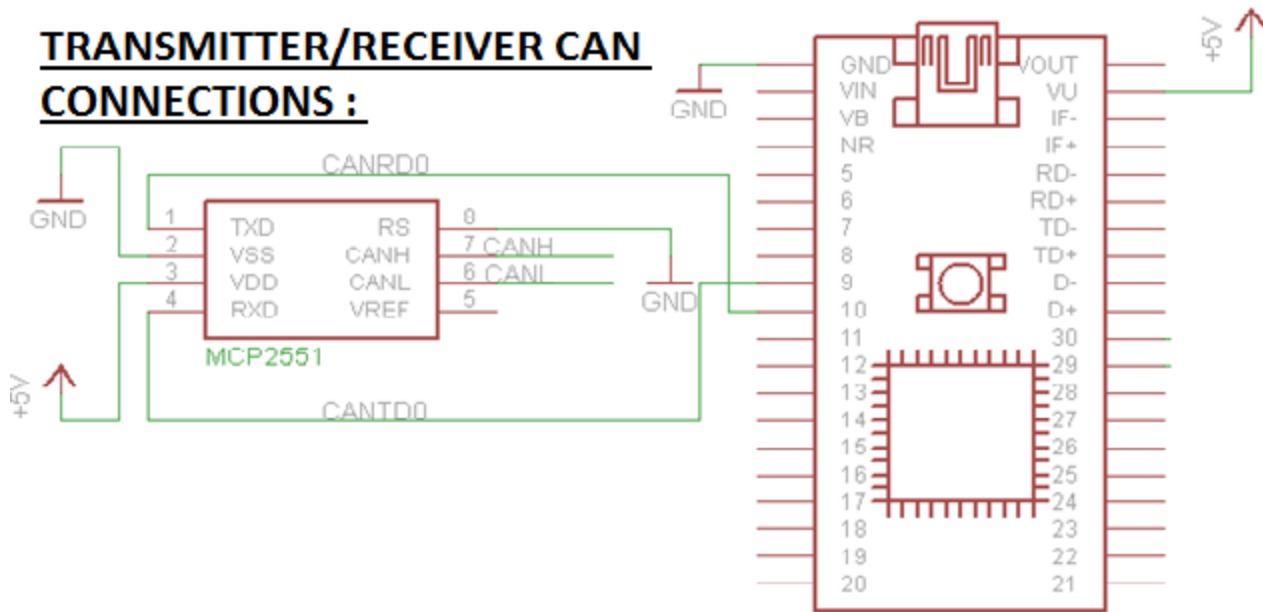


### Hardware Connections:

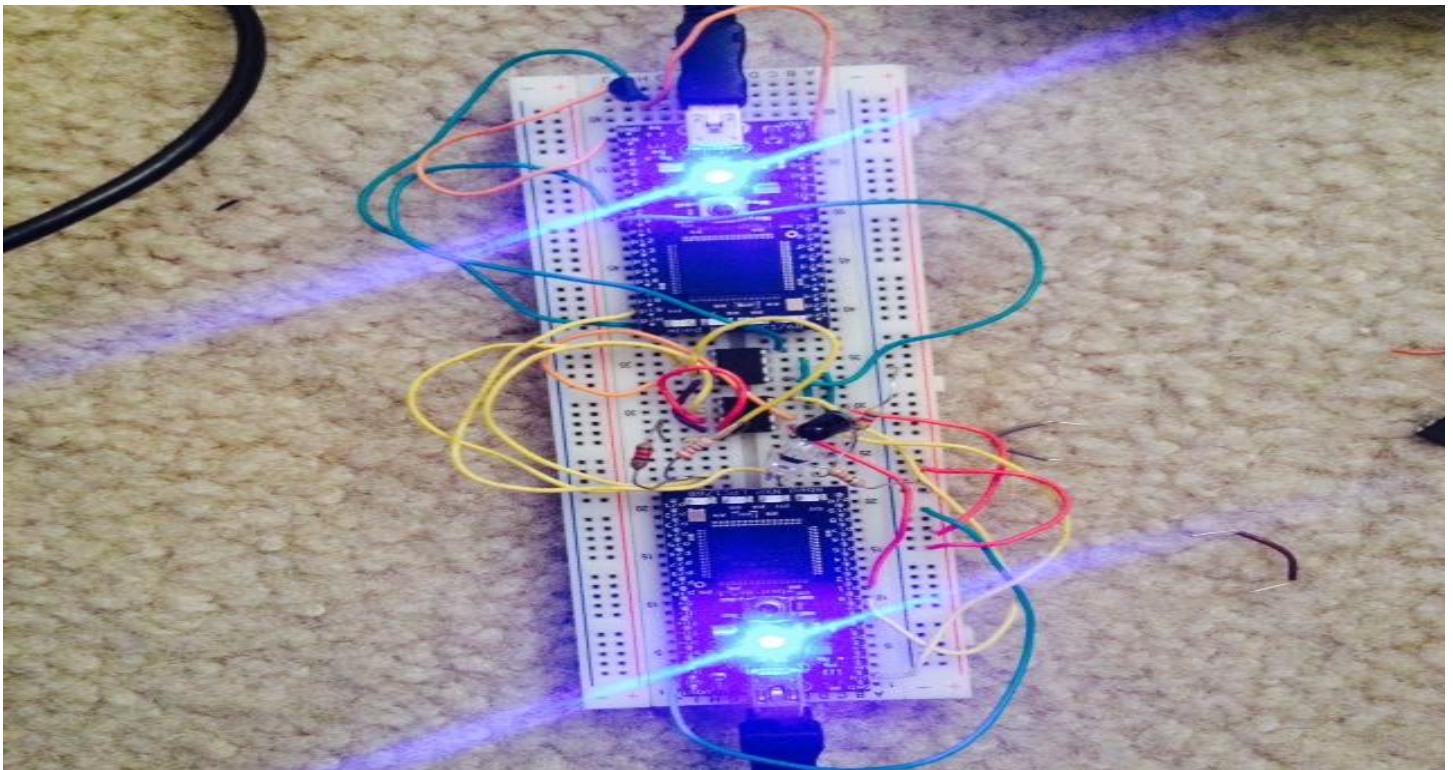
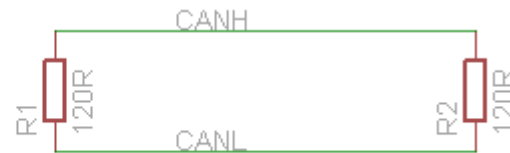
The complete hardware is mounted on a testing breadboard the image for the same is given Details description of the connection is as follows :

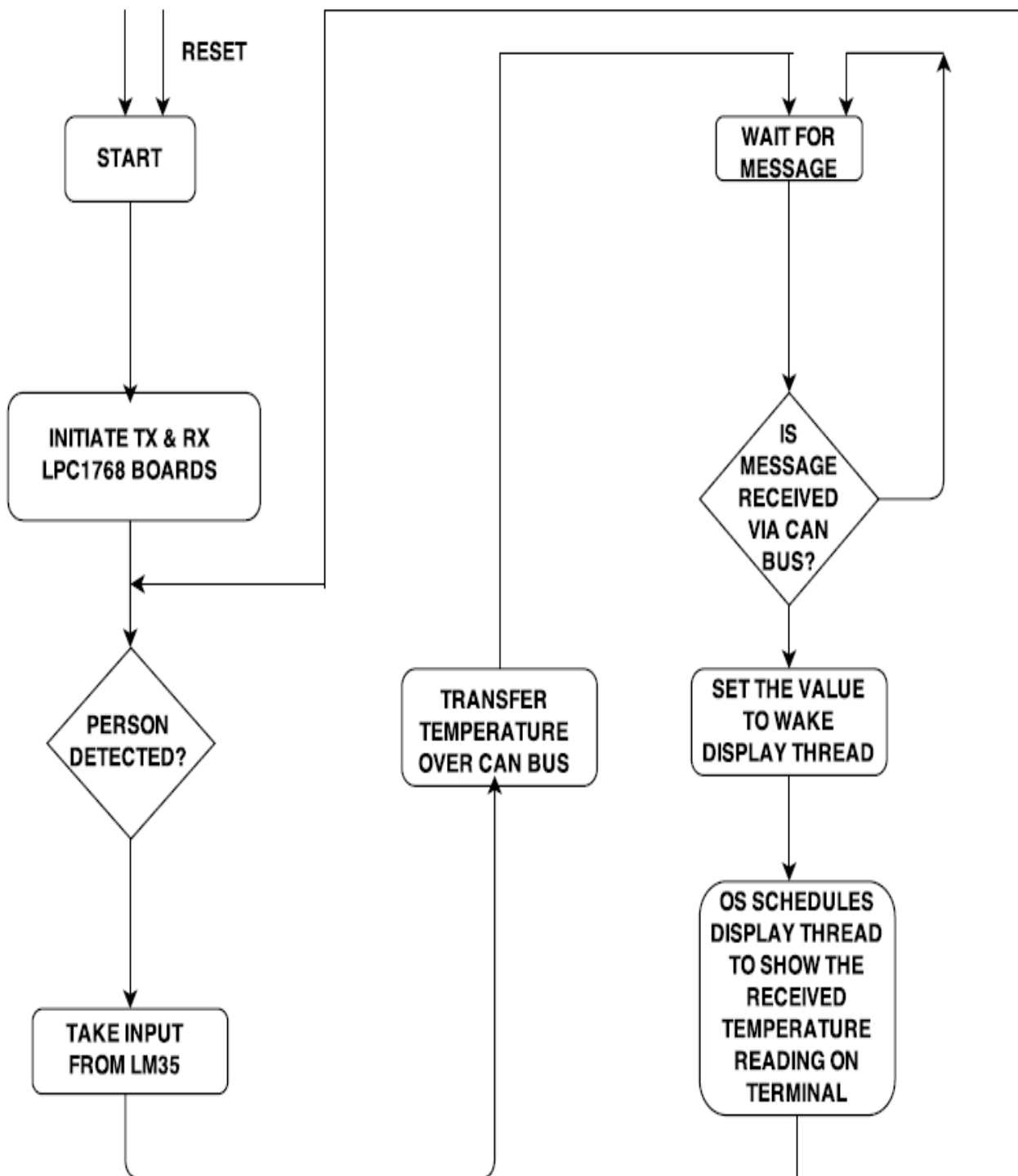
1. I have used one LPC1768 board as transmitter and one is used as receiver.
2. On the transmitter side, connect LM35 temperature sensor to GPIO pin 20 on LPC 1768 board.
3. To detect the person the IR sensor is connected on P18
4. For initiating the CAN bus we make the following connections :
  - A. *On transmitter side :*
    - Power up IC MCP2551 by connecting pin 2 on MCP2551 to "VU" pin on LPC 1768 which provides 5V DC
    - TXD (pin 1 on MCP2551) and RXD (pin 4 on MCP2551) is connected to GPIO 10 and GPIO 9 respectively on LPC 1768 board
    - Terminating resistors on CANH and CANL is 120 ohm. CANH is pin 7 and CANL is pin 6 on MCP2551
    - Pin 8 on MCP2551 is connected to ground
  - B. *On Receiver side :*
    - Power up IC MCP2551 by connecting pin 2 on MCP2551 to "VU" pin on LPC 1768 which provides 5V DC
    - TXD (pin 1 on MCP2551) and RXD (pin 4 on MCP2551) is connected to GPIO 10 and GPIO 9 respectively on LPC 1768 board
    - Terminating resistors on CANH and CANL is 120 ohm. CANH is pin 7 and CANL is pin 6 on MCP2551
    - Pin 8 on MCP2551 is connected to ground

## TRANSMITTER/RECEIVER CAN CONNECTIONS :



## CAN BUS :



**Software Flow :**

**Code :****Transmitter side:**

```

#include "mbed.h"

#include "cmsis_os.h"

// Initialize a pins to perform analog input and digital output functions
AnalogIn ain(p20); // Take input from temperature sensor
AnalogIn tin(p18); // Take input from IR led which detects the person
DigitalOut led(LED1);

CAN can1(p9, p10); // Initialise P9 as RXD and P10 as TXD of CAN
CANMessage msg;

Serial pc(USBTX, USBRX); // USB port used as serial port.

float ADCdata;

char temp;

void tempsend_thread(void const *args) {
    while (true) {
        // Signal flags that are reported as event are automatically cleared
        osSignalWait(0x1, osWaitForever); // Wait for signal 0x1 and oswaitforever is the time out value
        ADCdata = (ain*5) * 100;
        temp = ADCdata; // Convert the data to Character
        can1.write(CANMessage (1337, &temp)); // Send char a via can transmit buffer ,1337 is message ID to the Filter
        pc.printf("the temperature transmitted is : %d\n\r", msg);
        osDelay(1000); // 1000 msec delay
    }
}

osThreadDef(tempsend_thread, osPriorityHigh, DEFAULT_STACK_SIZE); // Thread definition

int main()
{
    while (1){
        osThreadId t2 = osThreadCreate(osThread(tempsend_thread), NULL); // Create thread t2
        if (tin > 1) // check the presence of a person, if detected the voltage will be more than 1 else it is less than 1

```

```

{
    osSignalWait(0x1, osWaitForever);// signal 0x1 to thread t2 so it wakes the thread
    osDelay(500);// msec delay
}
}
}

```

### **Receiver side:**

```

#include "mbed.h"
#include "cmsis_os.h"

DigitalOut led1(LED1);
DigitalOut led2(LED2);
CAN can1(p9, p10); //Initialise P9 as RXD and P10 as TXD of CAN
CANMessage msg;
Serial pc(USBTX, USBRX);// USB port used as serial port.

void tempdisp_thread(void const *args) {
    while (true) {
        // Signal flags that are reported as event are automatically cleared
        osSignalWait(0x1, osWaitForever);//Wait for singal 0x1 and oswaitforever is the time out value
        pc.printf("thread wakeup\n");
        pc.printf("the temperature received is : %f\n\r",msg);//Displays the received temperature on Terminal
        osDelay(1000);//1000 msec delay
    }
}

osThreadDef(tempdisp_thread, osPriorityHigh, DEFAULT_STACK_SIZE);//Thread Defination

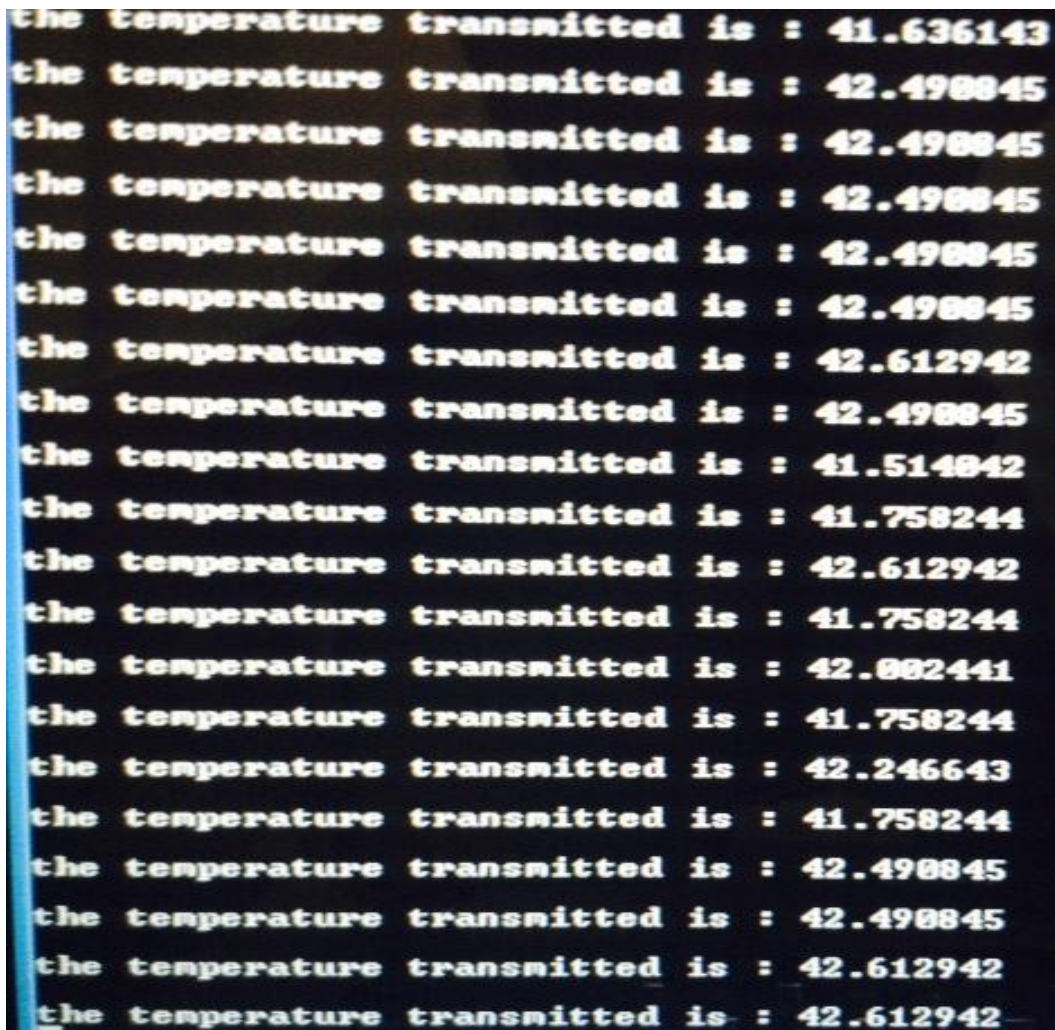
int main() {
    osThreadId t1 = osThreadCreate(osThread(tempdisp_thread), NULL);// Create thread t2
    while (true) {
        if (can1.read(msg)) //check if the message is received from the CAN bus
        {

```

```
osSignalSet(t1, 0x1); //signal the display thread to wake up and display the message
}
led1 = !led1;
osDelay(500);
}
}
```

### Output :

#### Transmitter Side :



the temperature transmitted is : 41.636143  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.612942  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 41.514842  
the temperature transmitted is : 41.758244  
the temperature transmitted is : 42.612942  
the temperature transmitted is : 41.758244  
the temperature transmitted is : 42.882441  
the temperature transmitted is : 41.758244  
the temperature transmitted is : 42.246643  
the temperature transmitted is : 41.758244  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.498845  
the temperature transmitted is : 42.612942  
the temperature transmitted is : 42.612942



Receiver Side :