TÉCNICO LISBOA

NETWORK EMBEDDED SYSTEMS

Development of an alarm system
Interconnecting Mbed nodes with CAN

Luís Rei, 78486
João Girão, 78761
João Belfo, 78913

May 28, 2017

Abstract

In this report we propose the development of an alarm system, resultant of the implementation of ”mbed” nodes, interconnected by a Controller Area Network (CAN), capable of sensing their surroundings and communicating with a central unit. We define a communications' protocol and make an analysis on the requirements and conditions needed to be met in order to achieve our goal. We also provide a video demonstration that exposes all of the different tasks our application can execute.

Keywords: Controller Area Network, Embedded Systems, Mbed, Sensors, Actuators.

Contents

1 Introduction **1**

2 Alarm System **2**
2.1 Node and Network Architecture 2
2.2 Energy Management 6
2.3 Sensors and Actuators 8
2.3.1 Humidity sensor DHT22 8
2.3.2 Light sensor GL5528 9
2.3.3 Other sensors and actuators 9
2.4 System Support and Special Requirements 11
2.5 Network Communications 11

3 Setup **14**

4 Conclusions **15**

A Mbed pin layout **17**

B Transceiver pin layout **17**

C Front of the application board **18**

D Back of the application board **19**

E Humidity sensor specifications **20**

F Light sensor specifications **21**

G 2-Pin push button **22**

H 5-Pin push button **22**

List of Figures

1 Master's main cycle . . . 3
2 Pass insertion cycle . . . 4
3 Mode selection insertion cycle . . . 4
4 Options selection cycle . . . 5
5 Slaves' main cycle . . . 6
6 Energy management . . . 7
7 Humidity sensor . . . 8
8 Light sensor . . . 9
9 Joystick . . . 9
10 Potentiometer . . . 9
11 Temperature sensor . . . 10
12 Two push buttons used . . . 10
13 LPC1768 LED . . . 10
14 Example routine of our program . . . 13
15 Mbed pin layout . . . 17
16 Transceiver used . . . 17
17 Application board layout . . . 18
18 Application board layout . . . 19

List of Tables

1 Communications protocol . . . 12
2 DHT22 . . . 20
3 GL5528 . . . 21
4 Push button characteristics . . . 22
5 Push button specifications . . . 22

1 Introduction

Involved in a Network Embedded Systems (NES) course we were given the opportunity to develop an application chosen from a variety of proposed projects, or born from our own thoughts and outside of the box thinking. In this regard, we came up with an idea: an implementation of an alarm system through the implementation of a CAN network system. This alarm system was composed of (but not limited to) three static nodes, each of them with a series of sensors (listed in 2.3), capable of communicating between them and handling concurrency problems.

This report resolves around five major areas: Node and Network Architecture (global architecture of the project), Energy Management (specific requirements and consumption management), Sensors and Actuators (requirements and characteristics), System Support and Special Requirements (real-time requirements, resource management and development environment), and Network Communications (network type and topology, and definition of real-time and consumption requirements).

Our main objective is to develop a functional demonstration application using ”mbed” nodes interconnected by a CAN network, highlighting CAN characteristics in what concerns real-time and reliability. We provide a solution to deal with concurrency issues and present our results and conclusions taken.

2 Alarm System

In this project we aimed at making a realistic and energy efficient implementation of an alarm system. In a relatively closed quarters environment we idealized a controller area network (CAN), composed of static nodes able to sense their surroundings, and able to talk to a more central unit that will be from now on referred to as main node, or the master. The master would work almost as a simple interface to the user, and would allow the redefinition of several system settings, and the checking of all the values being measured by the several (enslaved) nodes spread through the rooms.

Our goal was not only to take advantage of the collision free environment made possible through the use of a controller area network differential bus to make a consistent alarm system, but also to make use of the thousands of libraries available online from and for the mbed community to seamlessly optimize the energy management of the entire project.

To achieve this, we performed tests and experiments throughout the semester to validate what could be or couldn't be done, and to guarantee that all the knowledge gained as we advanced through the semester was applied and the objective was met.

In the next sections we will go into detail on the five major areas of development. Firstly we will describe the global architecture of our program, we will then touch on the energy management of our project and analyze paths that were taken and considered to make this a "greener" application. We will after analyze all the sensing and acting components used to bring our alarm system to life, followed by two sections describing the system support and network communications responsible for binding all of the hardware together into a reliable technology demonstrator.

2.1 Node and Network Architecture

Our project consists of a master/slave architecture, where the master acts as an information base station, gathering sensor measurements from secondary nodes, and serving as the primary interface between the system and the user. It is entirely scalable, but some concerns arise when the number of nodes start to become too much. We would probably like to look into incorporating several local master to reduce data centralization and prevent a certain node from wearing much earlier than the other nodes. A strategy we could apply would be the rotation of master through all the nodes, guaranteeing that everyone was stressed equally and delaying the failure of the first node due to to much stress on a single mbed.

In our application "slave" may not be precisely accurate as the auxiliary nodes do take initiative in transmission. this only occurs tho when an alarming value has been sensed - the threshold limit of a node has been surpassed - and the user must be forcibly informed. Here, and only in this case, does the "slave" take a more active part in our alarm system.

In order to make this project viable nodes need to have a way to communicate to each other via a CAN bus, and that requires the node to have:

- a central processing unit, microprocessor, or host processor. The host processor decides what the received messages mean and what messages it wants to transmit. Sensors, actuators and control devices can be connected to the host processor.

- a CAN controller; often an integral part of the micro-controller.
  - when receiving the CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor (usually by the CAN controller triggering an interrupt).
  - when sending the host processor sends the transmitted message(s) to a CAN controller, which transmits the bits serially onto the bus when the bus is free.
- a transceiver.
  - when receiving it converts the data stream from CAN bus levels to levels that the CAN controller uses. It usually has protective circuitry to protect the CAN controller.
  - when transmitting it converts the data stream from the CAN controller to CAN bus levels.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are connected to the bus through a host processor, a CAN controller, and a CAN transceiver.

Now that we raised some network requirements it is important to layout our application for a better understanding of what it can do and how it does it. The following diagrams aim to clarify the global architecture, and specific node architecture imposed by us. Let us dive in the master unit main cycle (figure 1).

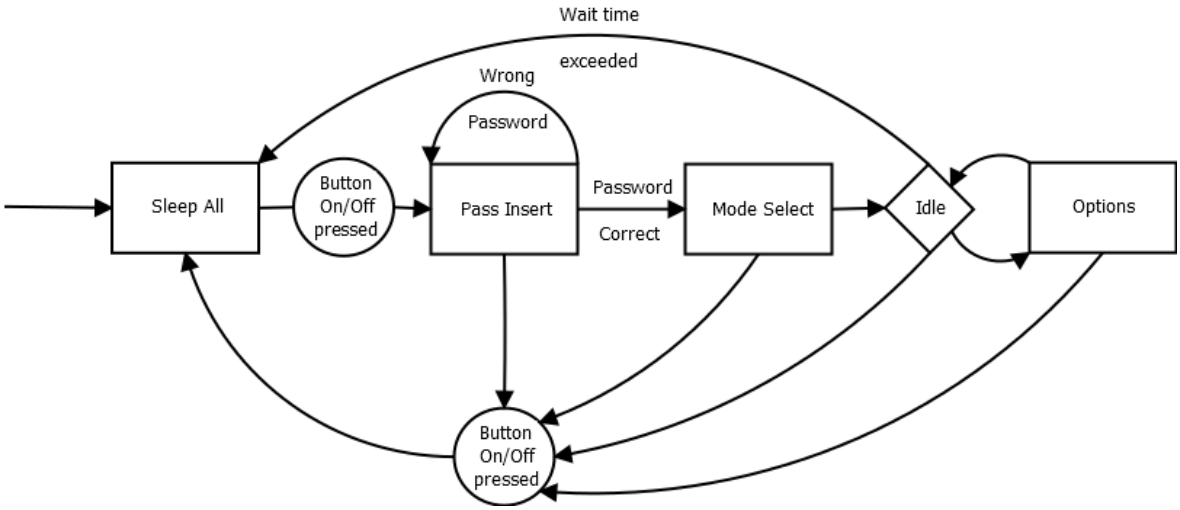


Figure 1: Master's main cycle

In the beginning all nodes are sleeping and, when the On/Off button is pressed the master enters the 'Password Insert' cycle. Once the password is correct the user is asked to select the mode he wants from the existing ones. Once inside one of the modes (except the fourth one) the text "Press fire for more actions" appears. Pressing the joystick will give the user access to four more options. The system can be shut down by pressing the On/Off button, or it will automatically shut down after a predefined time of inactivity (20 seconds).

Looking inside the 'Password insertion' cycle the first thing the system does is to check if there is a password inside a file named 'PASS': if the file is empty or non-existent, the predefined password "uldr" is stored inside the PASS file. The cycle starts with all four mbed leds turned off and will turn them on as the joystick is pushed, turning them off again when the password is inserted (either the correct one or a wrong one).

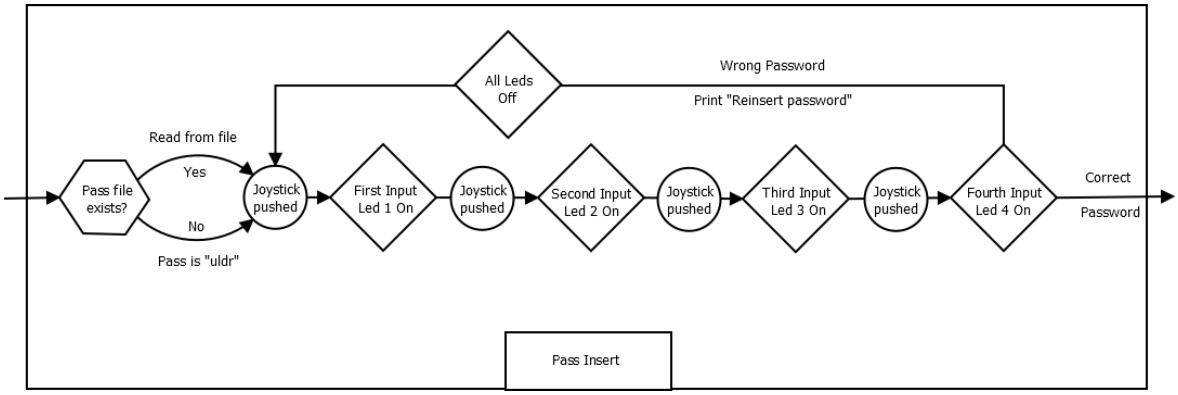


Figure 2: Pass insertion cycle

In the 'Mode select' cycle the user is able to choose the mode he wants from the four existing ones: 'Humidity' mode, 'Temperature' mode, 'Luminosity' mode and 'Change Password' mode, respectively. Selecting one of the three first modes will lead to the 'idle' state in figure 1, while the last mode will open a password cycle similar to the one in figure 2 where one is able to overwrite the existing password, after which the user is redirected to the 'Mode Select' cycle.

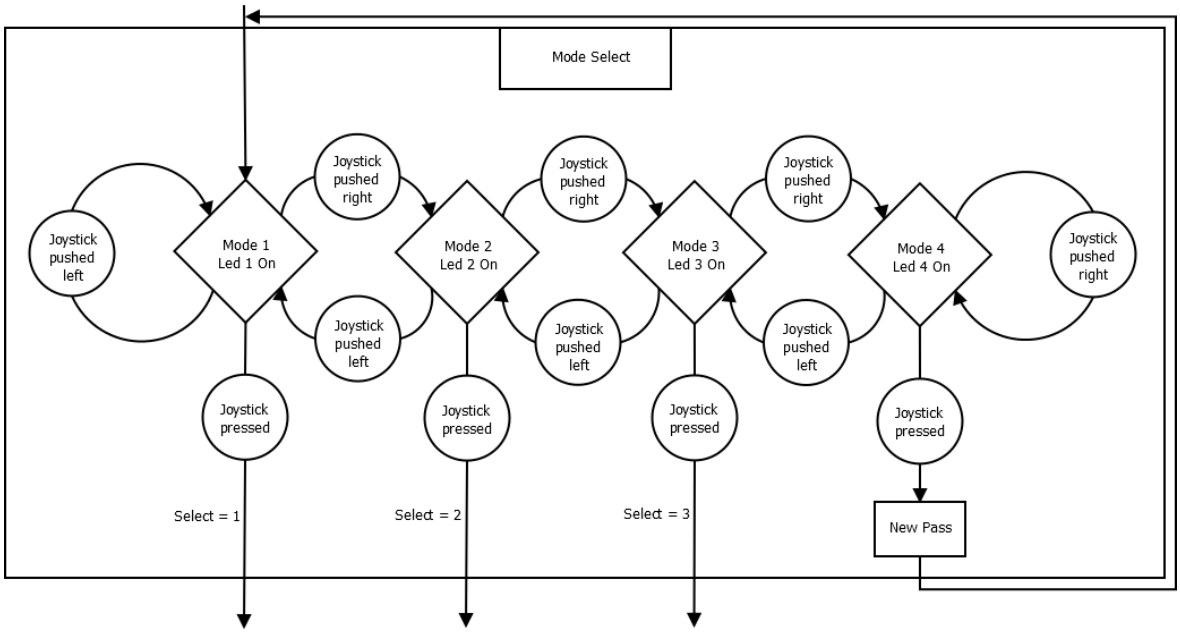


Figure 3: Mode selection insertion cycle

Once a mode is selected the user will be greeted with a ”Press fire for more actions” message on the LCD screen and pressing the joystick will prompt another message to appear explaining the different options the user can choose from. they are as follows:

- UP - pushing the joystick upwards leads to the ’Set Threshold’ cycle where, using the potentiometers on the master’s application board one is able to adjust the threshold limit associated with a certain node and respective sensor. Pressing the joystick will again set the value as the new limit.
- RIGHT - pushing the joystick to the right leads to the ’Set Period’ cycle. There, pushing the joystick right will increment the value of the new period (up to a maximum of 9.9 seconds) and pushing it left will decrease its value (down to a minimum of 2 seconds). Pressing the joystick will set that value as the new period.
- DOWN - pushing the joystick down toggles the ’message receiver’ flag. Once actived, the messages received by the main node will start to be printed on the LCD screen.
- LEFT - pushing the joystick left leads to the ’Mode Select’ cycle explained before.

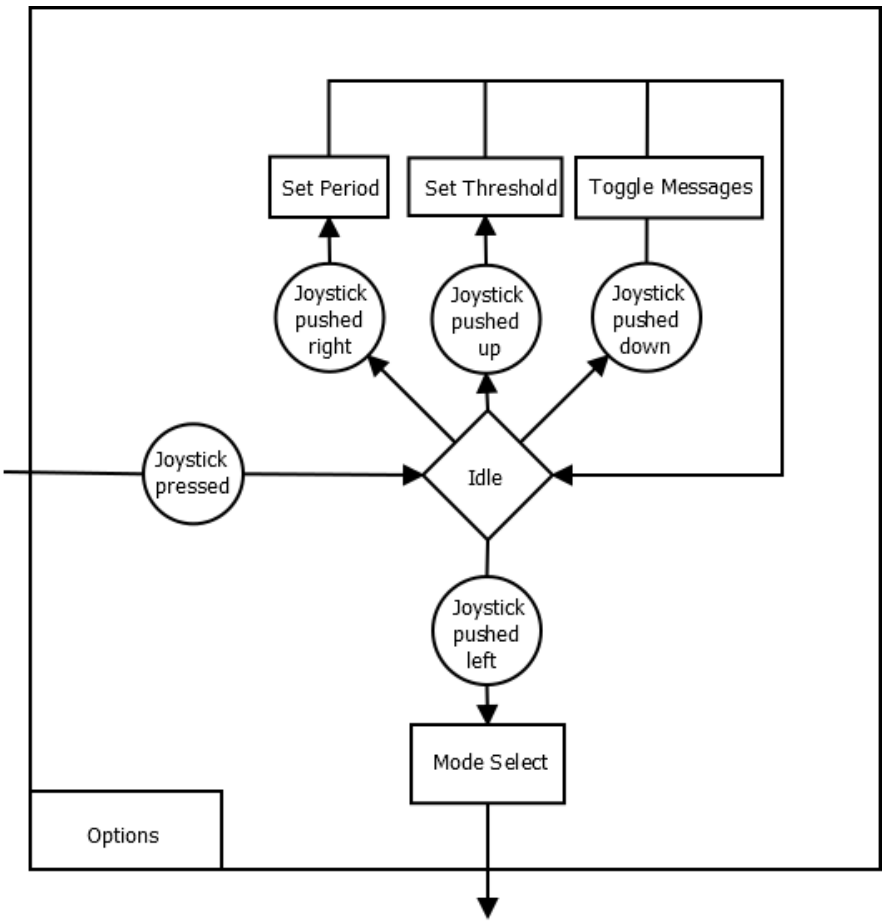


Figure 4: Options selection cycle

Having gone through all the iterations inside the main unit's main loop, we target the auxiliary nodes as our next case study. Let us dive in its main cycle (figure 5).

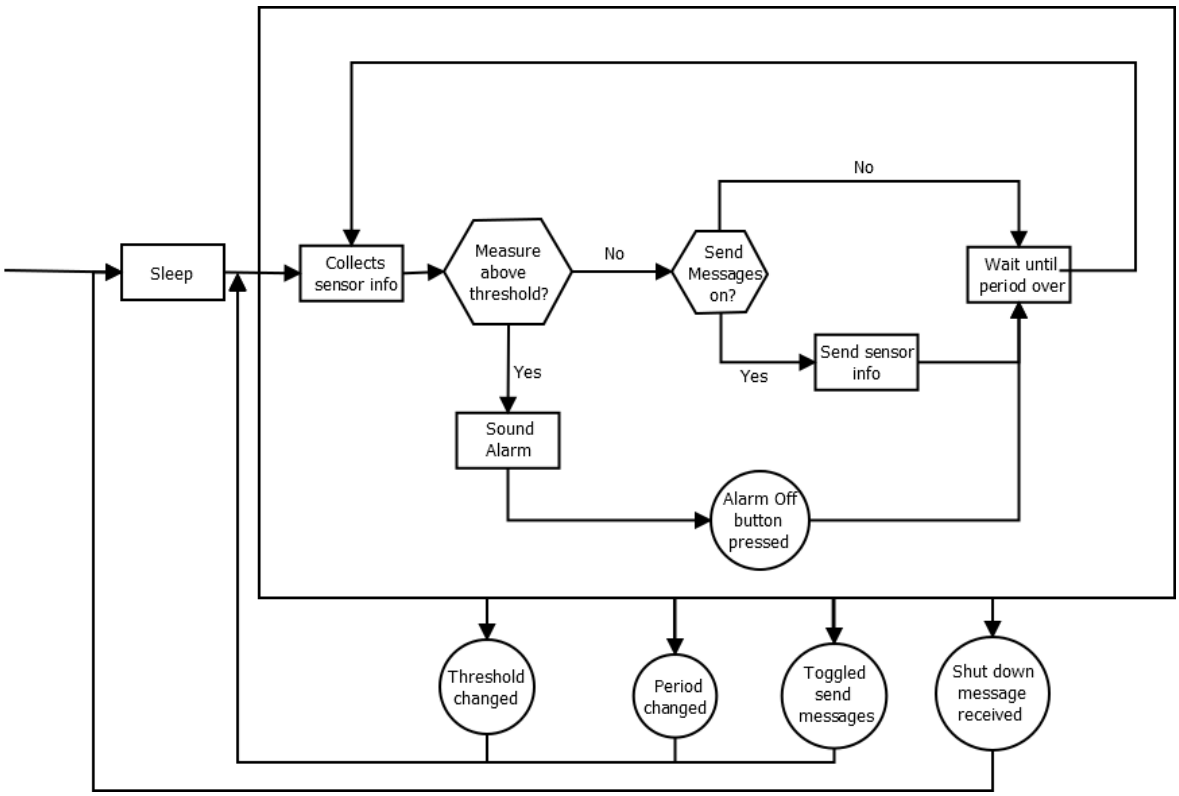


Figure 5: Slaves' main cycle

When the application begins the secondary nodes are sleeping, and are automatically woken up by the master when the user inserts the correct password. From there, the sensors will periodically start collecting information, printing it on the LCD screen, and the node will compare it with the threshold value assign to it, alerting the Master node and sounding the alarm until the alarm button is pressed if the limit value is exceeded. After the comparison the node will send the information to the Master node if the 'receive messages' flag is toggled on.

The node changes the period and threshold values, toggles the 'receive messages' flag or shuts down when he gets a message from the master node according to the protocol from table 1.

2.2 Energy Management

Energy issues are usually regarded as important topics when building a system. Things like energy sources, and energy management and consumption have to be kept under strict supervision especially whenever a project has a high energy dependency. Reducing power is not only green, it saves money by saving power. Over the life of a computer system, the money spent to power and cool the device can cost as much as the device itself! So well engineered designs need to minimize power whenever practical.

To reduce power consumption we can, for example: reduce voltage levels - in recent years, voltage levels have been reduced on logic from 5V to 3.3V, and reduce clock speeds - turning off the clock, or slowing down the clock whenever excess CPU time is available.

Many processors have hardware support to vary the clock frequency or even turn off the clock (Sleep mode) and we take advantage of that using the mbed 'PowerControl' library.

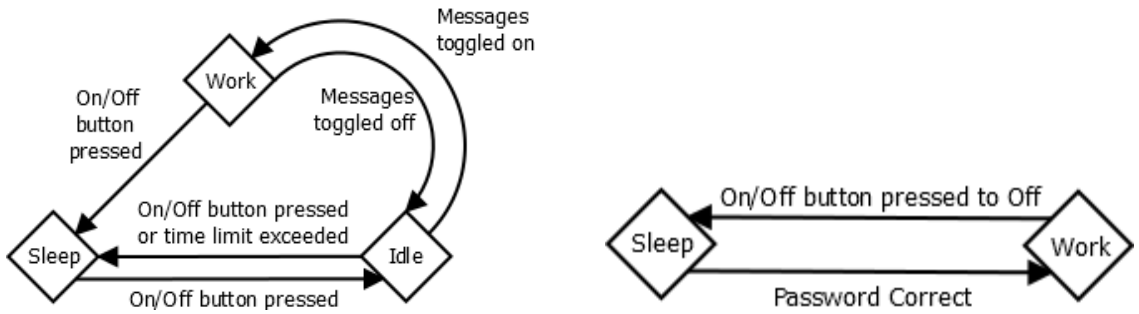
Some reduced static power levels will still be required even with the clock off to save the values in registers and volatile RAM memory (password value and other system settings), so that the device can wake up without a full reboot. Interrupt hardware is used to wake a device from 'sleep' mode, so the hardware used for wakeup can't be turned off. A study concerning the implementation of the 'sleep' state was conducted and the 'PinDetect' library was used to ensure that the interrupts that caused the program to go to 'sleep' mode (or out of it) acted without any problems.

The mbed LPC1768 board was not designed for very low power levels, but it is still possible to save significant power with a little additional effort. It is possible to turn off some of the external devices and the processor includes support to power down unused devices inside the processor chip, vary the clock rate, and enter low power Sleep modes. Instead of using wait, 'sleep' can be used. Sleep does not clock in new instructions, it turns off the clock on the processor core and waits for a timer interrupt. Keep in mind that this is particularly helpful when running mbed from a limited power source such as batteries. That isn't exactly our case but these are principles that can be applied to our application.

We split the system in three energy consumption states:

- Work: this is the state that consumes the most energy out of all the three because in this state the transceivers are receiving or transmitting, and the processing power is at its highest;
- Idle: in this state the transceivers are ready to receive, but not doing so. Some functionalities are switched off, therefore consuming less energy;
- Sleep: processing power is severely reduced and we are not able to immediately receive something. Recovery time and startup energy to leave sleep state can be significant.

One solution to lower the energy consumption is one we present below (figure 6. The two diagrams illustrate the energy management flow of the different nodes.



(a) Master (b) Slave

Figure 6: Energy management

As our program stands, all nodes are initialized in sleep mode. When the 'on' button is pressed the main node enters the 'idle' state, as it is expecting the user's commands regarding the input of a password and selection of a mode. After those steps are made, he remains in 'idle' until the messages are toggled (enters 'work' state) or until the system is turned 'off'

('sleep' state) - either by the press of a button or by the passage of a certain period of time (20 seconds).

Our secondary nodes (slaves) only enter 'work' state after the password has been correctly inserted. At that time they start reading and printing sensor measurements periodically in the LCD screen of their application board. They remain working, reading sensor values and performing computations until the system is shut down - time at which all the nodes are sleeping (including the main node).

Since this was a small scale project and the whole system wasn't very energy demanding all the common components could, and were powered only by the Master node. The mbed nodes needed to be connected to the computer to function properly, and so, USB cables were used to supply all three mbed application boards with 5 V.

Further improvements can be done to reduce power consumption, such as adding a resistor to the push buttons in order to minimize energy losses. The values of the resistors change as the voltage input power is either 3,3 or 5V.

2.3 Sensors and Actuators

In this section we will refer some sensors and actuators used throughout our project, specifically mentioning their requirements and characteristics.

2.3.1 Humidity sensor DHT22

Firstly we mention the humidity sensor (figure 7) incorporated in our node 1.

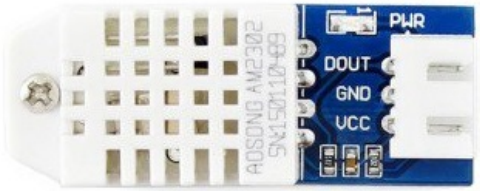


Figure 7: Humidity sensor

The DHT22 is a low-cost digital temperature and humidity sensor. It is a complex sensor as it uses more than one sensor element, and it is a contactless one. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

To use it we simply connect the first pin on the left to 3-5V power, the second pin to our data input pin (pin 23) and the right most pin to ground. Note that although it uses a single-wire to send data if you want multiple sensors, each one must have its own data pin!

The humidity sensor required an external pull-up resistor from the data pin to VCC. In our case we used a 12k Ω resistor for that purpose. DHT22 specifications table can be found at appendix E and here [4].

2.3.2 Light sensor GL5528

Secondly, we mention the Light Dependent Resistor (LDR) used in node 3 (figure 8). The analog sensor can be found here [5].

Figure 8: Light sensor

A photoconductive light sensor does not produce electricity but simply changes its physical properties when subjected to light energy. The most common type of photoconductive device is the photoresistor which changes its electrical resistance in response to changes in the light intensity. Photoconductivity results from light hitting a semiconductor material which controls the current flow through it. Thus, more light increases the current for a given applied voltage.

Like the previous sensor, the LDR required an external pull-up resistor. In our case we used a 100k Ω resistor. We were able to receive its data using one of the analog input pins at the application board (appendix D) - we used pin 17.

2.3.3 Other sensors and actuators

The remaining sensors and actuators were incorporated in our application board (appendix C). We used the joystick (figure 9) as a sensor to iterate through the password insertion cycle of our program, and also the menu and options cycle. It is reachable through pins 12 to 16 in our board.

Figure 9: Joystick

We use the potentiometer (figure 10) as an analog sensor that allows us to change the threshold values of our application. This sensor is available at pins 19 and 20 of our board.

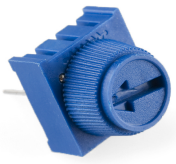


Figure 10: Potentiometer

The temperature sensor (figure 11) is a contactless sensor that converts thermal energy into electrical one, and it is incorporated in our node 2. To interface with the sensor we used the open source LM758D library.

Figure 11: Temperature sensor

We use two distinct push buttons in our project. One of them is a 2-pin momentary contact tactile switch (specifications at appendix G) that required a pull-down resistor; and the other a 5-pin push button (specifications at appendix H) that serves the same purpose but required a pull-up setup.

(a) 5-pin (b) 2-pin

Figure 12: Two push buttons used

We use the led (figure 13) as an actuator to visually indicate the distinct parts of our program cycle. In an earlier stage of development it served as a debugging tool.

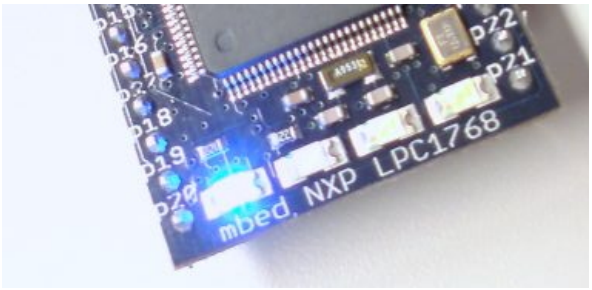


Figure 13: LPC1768 LED

And finally, we used a speaker to signal an alert to everyone around our alerting node, indicating that the sensor values measured were over our defined threshold limit. We used pin 26 to interact with this actuator that transformed an electrical energy into a mechanical one (sound waves).

More specifications and characteristics on the mbed application board components are found here [6]. All the components are layed out at appendix C.

2.4 System Support and Special Requirements

Our program was developed in a cloud compiler environment - an IoT Device Management Platform [3] - and consists of a single iterative cycle based on interruptions, and thus we do not utilize an Operating System (OS) - nor did we need to. . One thing to pay attention to is the real-time requirements of the project and the concurrency support we provide. As we do not take advantage of the FreeRTOS operating system, we need to make sure priorities in communication are linked to priorities in processing operations. Independently of having or not an OS, we must guarantee that the task linked to a higher priority message is executed first. To do so, we take advantage of the thousands of mbed libraries available online and develop a solid communications' protocol (see 2.5) capable of dealing with any concurrency problems.

Concurrency will appear when a measurement (processed in a slave node) exceeds a certain threshold and this information is sent to the master node while the user is interacting with it. To solve this situation, we will have to program an interruption that detects a message to be read in the CAN bus and, when identifying an alarm makes the main node acknowledge it, allowing the alarm to be turned off with the push of a button.

When the user is not interacting with a certain slave node, this node can go to a more energy saving state. He must wake up within a certain period of time to read the sensor and see if there is an alarm to fire (and then, send the information to the master node in case there is). This is done via a periodic interruption. The node also has to wake up when the master node wants to talk to him. In other words, if the user wants to read the value of that specific sensor, the slave node must be awake to send the measurements.

In general, there are some concurrency situations in this project related to the alarms and to the change of states of the slave nodes. Another example of concurrency happens when a node wants to talk to another node and the network is busy, making him wait until the bus is free. The resolution of this situation is done by the CAN interface itself.

As it stands, there are no real-time requirements in our application.

2.5 Network Communications

A Controller Area Network (CAN) is an architecture partitioned in 3 layers: the physical layer - where the bit transfer between the different nodes in the network, including bit-timing, synchronization and encoding is set, a medium access control - where a deterministic collision resolution of a multi-access carrier sense is defined, and the application layer - defined by us. This network uses a two-wire differential bus terminated in each end by a resistor (120 Ω) and it operates in a quasi-stationary mode where it waits for signal propagation and stabilization, before sampling again.

To guarantee a collision free environment nodes delay transmissions until an idle bus is detected, and when the bus is idle any node may start transmitting. The bus access conflicts are resolved through the bitwise comparison of a unique identifier. The bit represented on the bus is either dominant ('0' indicates that at least one node is transmitting a dominant bit level) or recessive ('1' means that no node is transmitting a dominant bit level). Each node compares the transmitted bit level with the level monitored on the bus. A node stops transmitting when a recessive bit was sent but a dominant bit was monitored, becoming then a receiver. A lower identifier translates into a higher priority.

A communications' protocol is introduced below in table 1.

Table 1: Communications protocol

Message identifier (hexadecimal)	Message content
0	Master informs Slave nodes that they should turn 'on' or 'off'
1	Master informs Slave nodes that they should disable their alarms
2 to 4	Master receives an alarm signal from a specific node
5	Master asks for information on the readings of a specific node
6 to 8	A certain node sends sensor readings to the master
9 to B	Master sends the new threshold value to the node
C	Master establishes the new period at which he receives sensor information

Additional information on the protocol established above is given here:

- With a message ID of 0 the message encodes a '1' to shut the node down (sleep mode), or '0' to wake him up;
- With an ID of 5 the message contains '1', '2', '3' to select the humidity, temperature and luminosity slave respectively;
- Additionally, if the message identified by 5 has any other value in it it indicates that the master is requesting the end of the sending of information to him;
- The messages identified from 6 to 8 and from 9 to B refer to the humidity, temperature and luminosity nodes respectively;
- Message identified by C is received by all the nodes - the new period is universal to the network.

In figure 14 we can see an example of two nodes communicating via CAN bus.

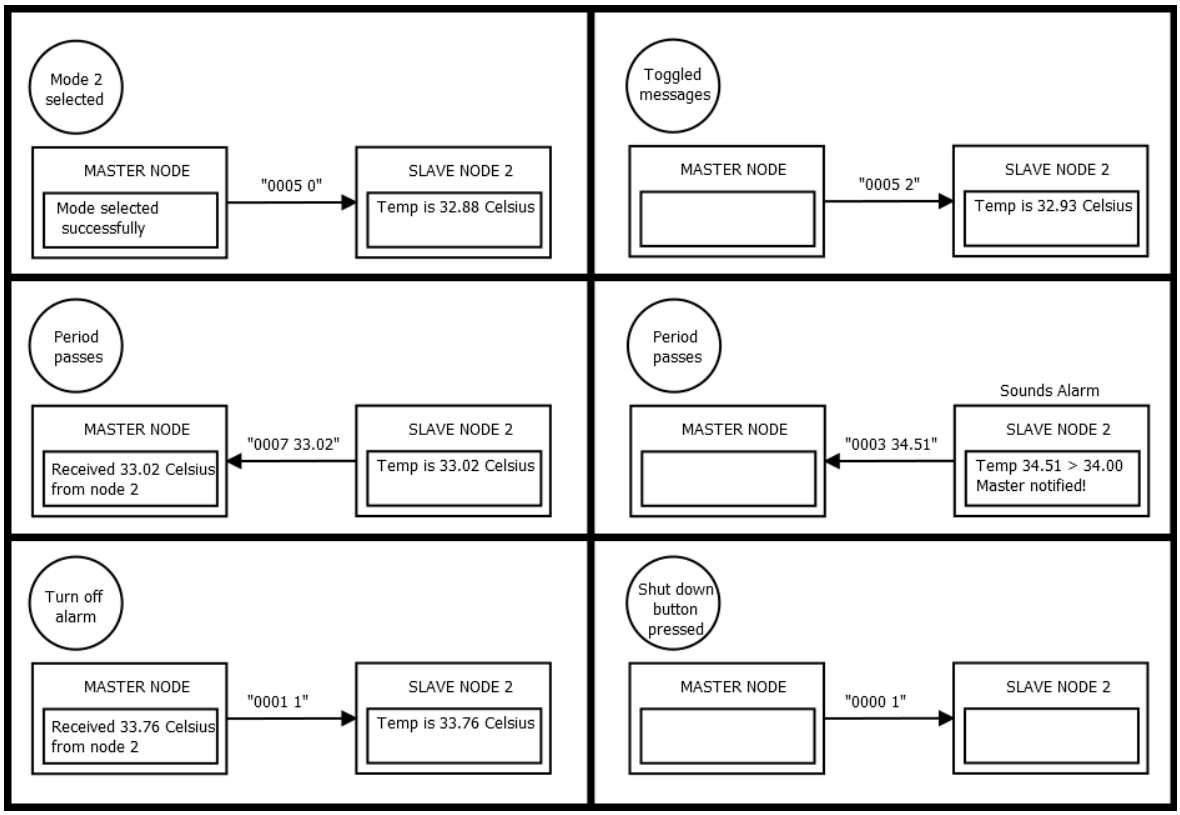


Figure 14: Example routine of our program

It goes as follows:

1. On the mode selection menu the second mode is selected and the Master informs the respective node;
2. The option ”Toggle messages” is select, indicating that the Master wants to receive information from the node;
3. When the period has passed the Slave node will send a message with the information it gathered in this iteration;
4. After a while, the measurement exceeds the current threshold. The node proceeds to notify the Master and enters the cycle that sounds the alarm;
5. The user presses the push button that turns off the alarm and the Master sends a message to the node that prompts the node to get out of the alarm cycle;
6. If the shut down button is pressed, all nodes receive the message to shut down and after receiving confirmation, the Master node will also shut down.

The network provides a good error detection and handling. It detects mismatches between transmitted and received streams, and lack of acknowledgment from receiving nodes. It has one particularity: a complementary bit is inserted upon detection of five identical consecutive bits, and an error appears every time a stream has more than five consecutive bits with the same polarity. The comprehensive set of error detection mechanisms aims to enforce reliability of communication objects transfer and allows the distinction between temporary errors and permanent node failure. As a general rule, errors are signaled as soon as they are detected, and are globally handled by transmitting and receiving nodes.

3 Setup

Our current setup consists of three LPC1768 mbeds (appendix A), three mbed application boards (appendix C and D) that can house several sensors, three high-speed CAN transceivers (2 $\times$ MCP2551 and 1 $\times$ MCP2561) that serve as an interface between a CAN protocol controller and the physical two-wire CAN bus, five resistors (two 120 Ω, one 12 $k\Omega$, one 100 $k\Omega$ and one Light Dependent GL5528 (appendix F)), two push buttons (one 5-pin button (appendix H) and one 2-pin button (appendix G)) and one DHT22 humidity sensor (appendix E).

In terms of interface thew two transceivers are very similar, being able to be replaced with one another as long as the eighth pin (appendix B) is connected to ground with a resistance between 0 and 100 $k\Omega$.

4 Conclusions

Having raised all of the project's requirements and after figuring out all the specifications we needed to oblige with, we successfully implemented all of the work referred to in the last sections of the project.

The communications protocol in conjunction with the characteristics of the CAN bus allowed us to achieve the intended results and deal with all of the concurrency issues that could have caused us trouble. Furthermore, the extensive list of available libraries gave us the flexibility to search and find suitable options for our problems. Our manipulation of existing libraries was minimal, as we found that the work already developed suited our needs perfectly.

A video showcasing a demonstration of our working system is provided here [7].

References

[1] Embedded Systems Design & Networked Embedded Systems, Richard Zurawski (ed.), 2009, CRC Press

[2] Protocols and Architectures for Wireless Sensor Networks, Holger Karl, Andreas Willig, 2005, Wiley

[3] Several manuals and datasheets available at https://www.mbed.com/

[4] Humidity sensor information at https://www.adafruit.com/product/385

[5] Light sensor information at
https://www.pcboard.ca/gl5528-light-dependent-resistor.html

[6] Application board components' specifications at
https://developer.mbed.org/cookbook/mbed-application-board#details

[7] Demonstration video at https://youtu.be/znFD7n8ZaAA

Appendix

A Mbed pin layout

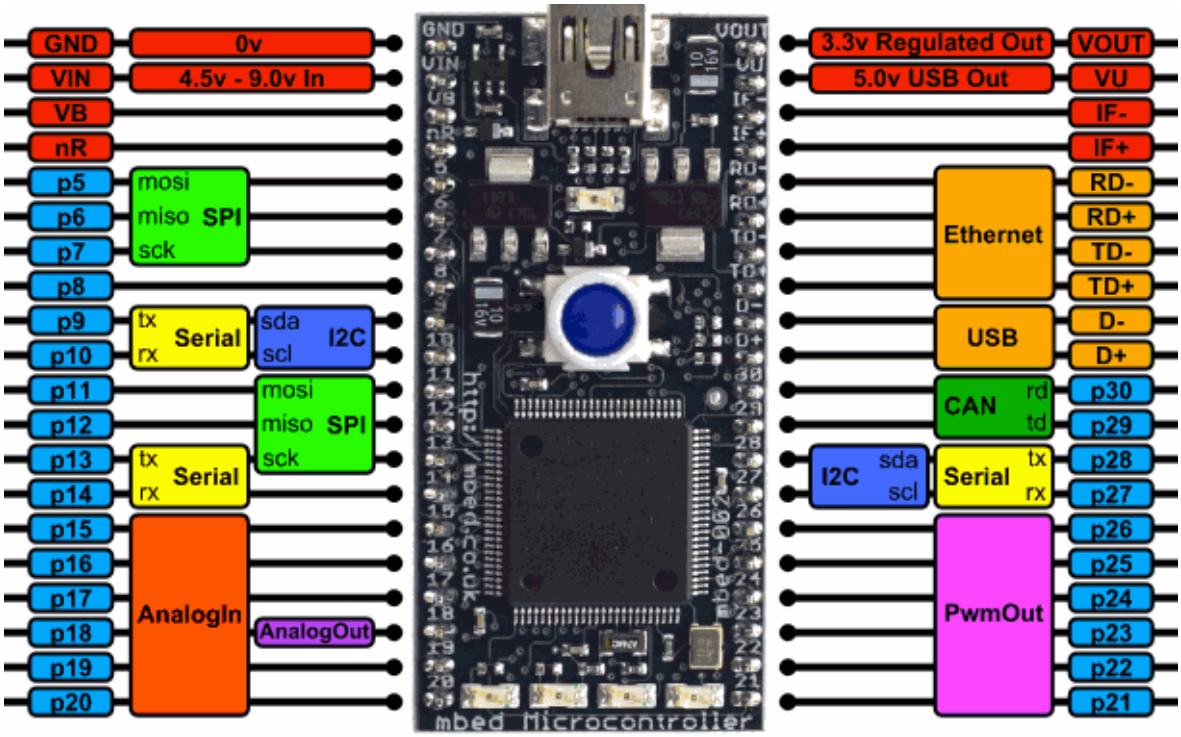


Figure 15: Mbed pin layout

B Transceiver pin layout

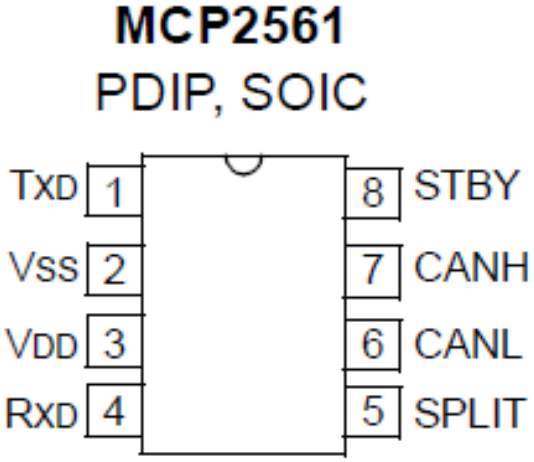


Figure 16: Transceiver used

C Front of the application board

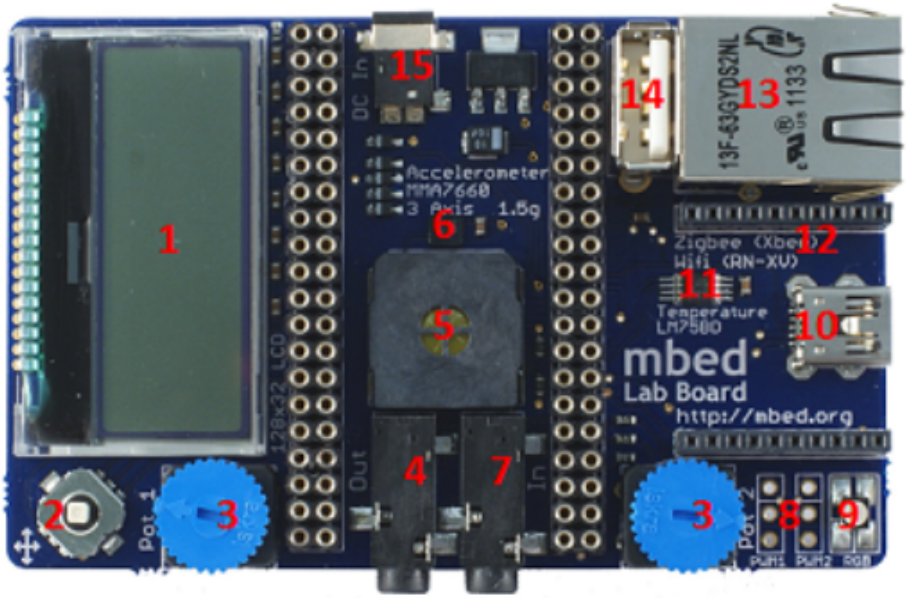


Figure 17: Application board layout

Listed as:

1. 128x32 Graphics LCD
2. 5 way joystick
3. 2 x Potentiometers
4. 3.5mm Audio jack (Analog Out)
5. Speaker, PWM connected
6. 3 Axis +/1 1.5g Accelerometer
7. 3.5mm Audio jack (Analog In)
8. 2 x Servo motor headers
9. RGB LED, PWM connected
10. USB-mini-B Connector
11. Temperature sensor
12. Socket for for Xbee (Zigbee) or RN-XV (Wifi)
13. RJ45 Ethernet Connector
14. USB-A Connector
15. 1.3mm DC Jack input

D Back of the application board

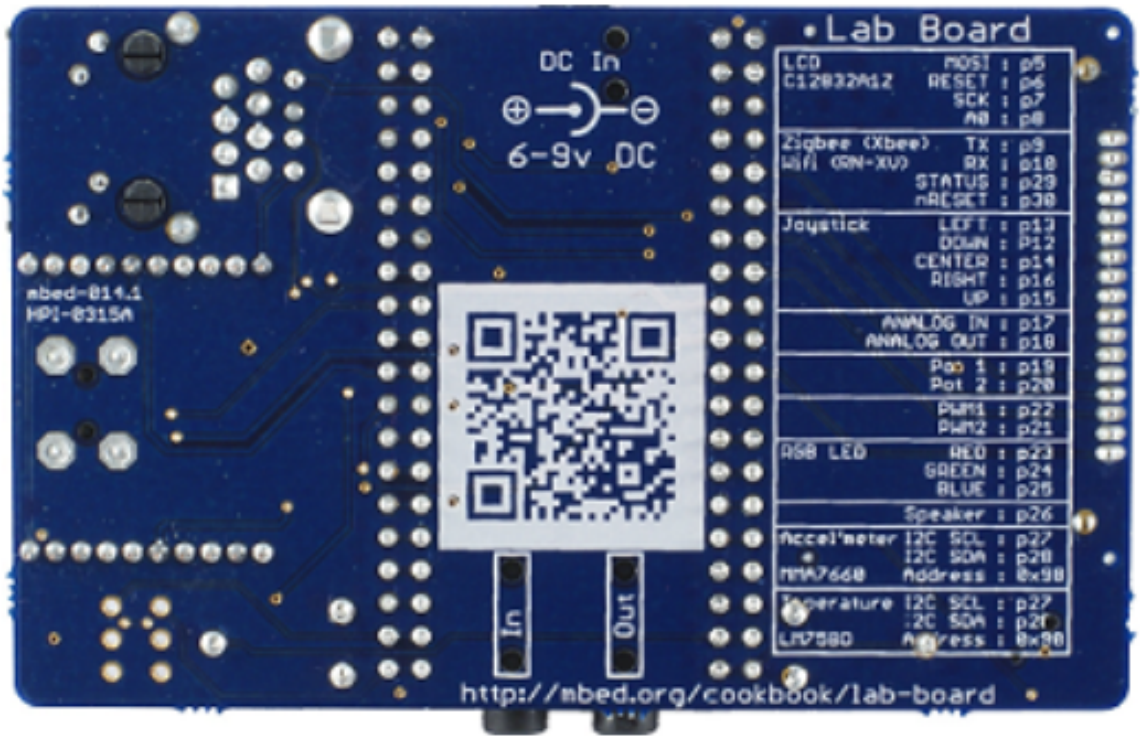


Figure 18: Application board layout

E Humidity sensor specifications

Table 2: DHT22

Model	AM2303
Power supply	3.3-6 V DC
Output signal	Digital signal via single-bus
Sensing element	Polymer humidity capacitor & DS18B20 for detecting temperature
Measuring range	Humidity 0-100%RH; Temperature -40$\tilde{1}$25 Celsius
Accuracy	Humidity $\pm$2%RH(Max +-5%RH); Temperature $\pm$0.2 Celsius
Resolution or sensitivity	Humidity 0.1%RH; Temperature 0.1 Celsius
Repeatability	Humidity $\pm$1%RH; Temperature $\pm$0.2Celsius
Humidity hysteresis	$\pm$0.3%RH
Long-term Stability	$\pm$0.5%RH/year
Sensing period	Average: 2s

F Light sensor specifications

Table 3: GL5528

Model	GL5528
Maximum Voltage	150 V DC
Maximum Wattage	100 mw
Spectral Peak	540 nm
Light Resistance	8 to 20 KΩ
Dark Resistance	1 MΩ
Sensitivity	0.8
Response Time (ms)	Up: 20/ Down: 30
Material	Carbon
Shape	Cylindrical
Size	5 x 3mm/0.2 x 0.12”

G 2-Pin push button

Table 4: Push button characteristics

Switch Type	Tact Switch
Contact Type	Momentary Contact
Dimension	3 x 6 x 4.3 mm / 0.12” x 0.24” x 0.17” (L*W*H)
Button Size	3 x 1.5 mm/ 0.12” x 0.06” (L*W)
Pin Pitch	6mm / 0.24”
Pin Length	5mm / 0.2”
Net Weight	6g

H 5-Pin push button

Table 5: Push button specifications

Model	TS1304
Rating	DC 12V 50 mA
Contact Resistance	$\leq$50 mΩ
Insulation Resistance	$\geq$100 MΩ
Dielectric Strength	AC 250 V 1minute
Operating Force	160$\pm$50 gf
Travel	0.3$\pm$0.1 mm
Life	$\geq$100,000 Cycles