Sistemas Computacionais Embebidos

ANO LECTIVO: 2020/2021

Implementação de um Data Logger com Interface CAN (V2020)

REGIME: Diurno

Autores:

David Drumond, Nª2181042

Edgar Paulo, Nº2181665

Docente:

Professor Luís Conde

---

15 de fevereiro de 2021

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O presente relatório tem como objetivo demonstrar a implementação de um *data logger CAN* que permita disponibilizar a informação proveniente de sensores através de um *Display TFT* e de um a aplicação móvel via *Wifi*, demonstrando desta forma, os conhecimentos adquiridos na cadeira de Sistemas Computacionais Embebidos.

O desenvolvimento do projeto assenta no potencial do módulo *Esp32*, tirando proveito das suas características, nomeadamente do facto de possuir dois *Core's*, diversos protocolos de comunicação *wireless* possibilidade de transmissão e recepção *CAN*. Desta forma, a simulação utiliza dois *Esp32* que controlam separadamente o módulo de aquisição, e o módulo de controlo e visualização, uma estrutura *soft real-time*, com definição de tarefas, prioridades e controlo de acesso aos recursos.

Por outro lado, o aproveitamento dos sensores incorporados no módulo *Pysense v2* assim como a utilização do interface *LCD TFT* exige a implementação de barramentos *I2C* e *SPI*.

O diagrama de blocos da figura 1.1 ilustra as funcionalidades do projecto.

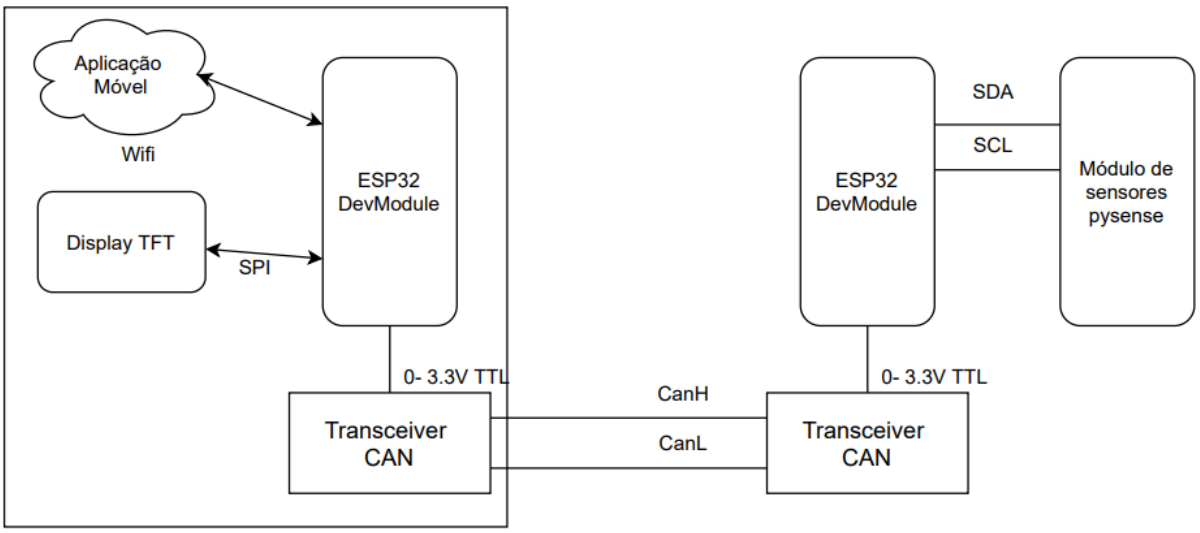


Figura 1.1: Diagrama de blocos.

2 Módulo de Aquisição

O módulo de aquisição tem como função principal a leitura de informação proveniente dos sensores e o reencaminhamento da mesma através do interface *CAN*. A sua composição inclui um *Esp32*, um módulo *Pysense* e um *transceiver CAN MCP2521*.

Os sensores utilizados encontram-se incorporados na placa de extensão *Pysense 2.0*, e são acedidos através de barramento *I2C*. A tabela 2.1 faz a correspondência entre os sensores utilizados e o endereço no barramento *I2C*.

Tabela 2.1: Lista de sensores e endereço *I2*.

Tipo	Circuito integrado	*I2C Address*
Temperatura/Humindade(1)	SI7006A20	0x40
Acelerometro(2)	LIS2HH12	0x1E
Luminosidade(3)	LTR329ALS01	0x29

A figura 2.1 apresenta o esquema eléctrico simplificado de hardware, sendo visível as ligações *I2C* entre o *Pysense* e o *Esp32*, bem como as ligações entre o *Esp32* e o *transceiver CAN*. [1]

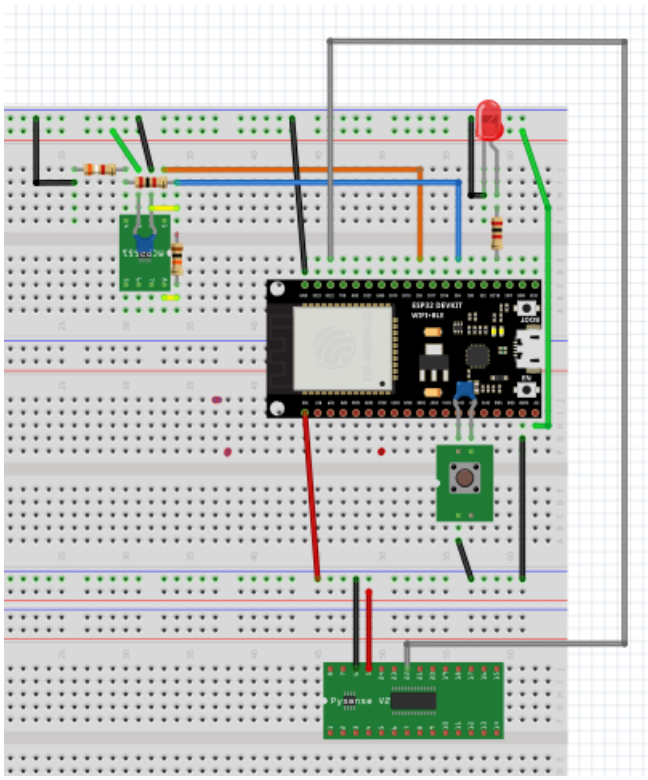


Figura 2.1: Esquema Elétrico Simplificado do Módulo de Aquisição

[1] Para mais detalhes acerca da placa Pysense 2.0 consultar o esquema em anexo.

2.1 Tarefas do módulo de aquisição

Tendo como objetivo um sistema em *real-time* procedeu-se a identificação e escalonamento de tarefas, definindo prioridades e a sua frequência de ativação. Esta informação encontra-se resumida na tabela 2.2.

Tabela 2.2: Escalonamento do módulo aquisição.

Core	**Nome**	**Breve descrição**	**Prioridade**	**VTaskDelayUntil**
1	vGetAccell	Leitura do acelerómetro via I2C	2	1s
1	vGetTemp	Leitura da temperatura via I2C	2	500ms
1	vGetHum	Leitura da humidade via I2C	2	1s
1	vGetLum	Leitura da luminusidade Via I2C	2	1s
0	vCanTx	Envio da Mensagem CAN	3	100ms
0	vMyCanRx	Recepção da mensagem CAN	3	——
0	vManageValues	Gestão da mesnsagem recebida via CAN	2	——

2.2 Queue's, Semáforos e Mutex's do módulo de aquisição

Para gestão de dados entre tarefas e recursos partilhados, recorrer-se-à a *Queue's*, semáforos e *Mutex's*. As *Queue'* podem ser acedidas por múltiplas tarefas, permitindo a transferência de informação entre estas, podendo funcionar como um *Buffer*, o que garante o armazenamento temporário de dados. A *Queue_sensores* será estrutural e recebendo a identificação e o valor das tarefas de aquisição sensorial para a tarefa vCanTx.

Por outro lado, os *Mutex's* possibilitam a gestão de recursos partilhados, como é o caso dos barramentos. A opção do uso de um *Mutex* para controlo do acesso ao *I2C* permite proteger o recurso, evitando que outras tarefas acedam ao mesmo a meio de uma comunicação, o que destruiria a trama de dados.

A tabela 2.3, mostra a utilidade destas ferramentas no projecto em causa.

Tabela 2.3: Lista de Semáforos, *Mutex's* e *Queue's*.

Tipo	Designação	Função
Mutex	xMutex_i2c	Controlo de acesso ao recurso *I2C*
Queue	xI2CQueue	Comunicação entre as tasks de leiruta sensorial e VCanTx
Queue	xDataQueue	Comunicação de informação recebida via *CAN* para *vManageValues*
Queue	callbackQueue[a]	Guarda temporariamente a mensagem recebida

[a] Queue já presente na biblioteca esp32_can_builtin.cpp

2.3 Descrição Funcional do Subsistema de aquisição

Para para demonstrar o funcionamento do subsistema fazer-se-á a descrição genérica do ciclo de execução das tarefas em função das prioridades, interrupções, tempo de bloqueio, utilização das *Queue's* e *mutex's*. A figura 2.2 facilita a interpretação do funcionamento. .

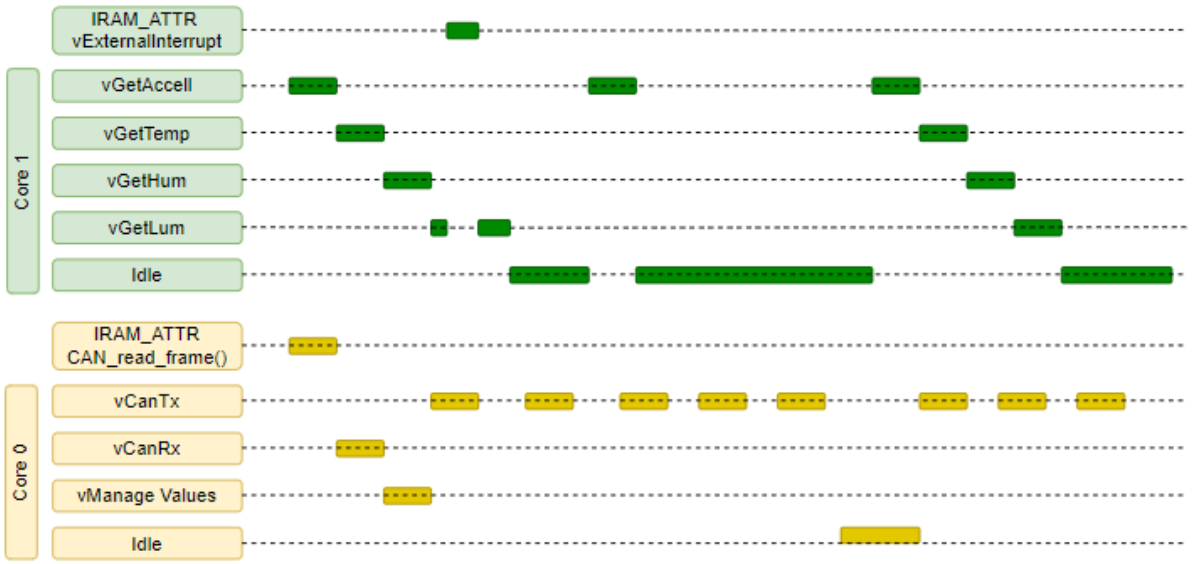


Figura 2.2: Diagrama de funcionamento do sistema de Aquisição

2.3.1 Tarefas VGetAccell, vGetTemp, vGetHum, e vGetLum

Estas tarefas estão alocadas no core 1 e têm como função a leitura periódica dos sensores via I2C, sendo que o período primeira de 0.5s e o das restantes de 1s. Estas tem todas a mesma prioridade, pelo que a ordem de desbloqueio do primeiro ciclo é indefinida, contudo nos seguintes é mantida a ordem inicial. O A protecção do recurso I2C é garantida através da utilização do mutex xMutex_I2C, como referido na subsecção anterior.

Estas tarefas escrevem o valor obtido dos sensores na Queue "xI2CQueue"através do comando xQueueSend(xDataQueue, &xDataStruct, 0);

2.3.2 Interrupção vExternalInterrupt

Está também previsto a ocorrência de uma interrupção externa, que também escreve Queue xDataQueue, contudo a mesma coloca o seu valor "na frente"da Queue através do comando xQueueSendToFrontFromISR(xI2CQueue, &xSenderStructure, 0); , garantindo assim um intervalo de detecção por parte do Módulo de Controlo e Visualização[2] menor.

[2]Módulo abordado na secção3

As tarefas vCanTx, vCanRx e vManageValues estão alocadas no core 0, podendo ser executadas em simultâneo face às restantes.

2.3.3 Tarefa vCanTx

A tarefa vCanTx apenas pode ocorrer a cada 100ms, definindo assim a taxa máxima de transferência de dados para cerca de 10 tramas por segundo,[3] para além desta restrição, a mesma só ocorre se existirem dados na Queue xDataQueue.

2.3.4 Tarefa vMyCanRx

Quando é recebida uma mensagem Can é despoletada a Interrupção CAN_read_frame(), que escreve na Queue callbackQueue [4] a trama de dados recebida. Por sua vez a tarefa vCanRx é desbloqueada sempre que existam dados nesta Queue. A sua função consiste apenas em mover estes dados para Queue xDataQueue.

2.3.5 Tarefa vManageValues

Esta tarefa é desbloqueada sempre que existam dados na Queue xDataQueue. Esta apenas descodifica a mensagem pelo seu id e pelos bytes de dados.

Para este módulo previu-se apenas a recepção de uma mensagem CAN, que permite ligar ou desligar Um led.

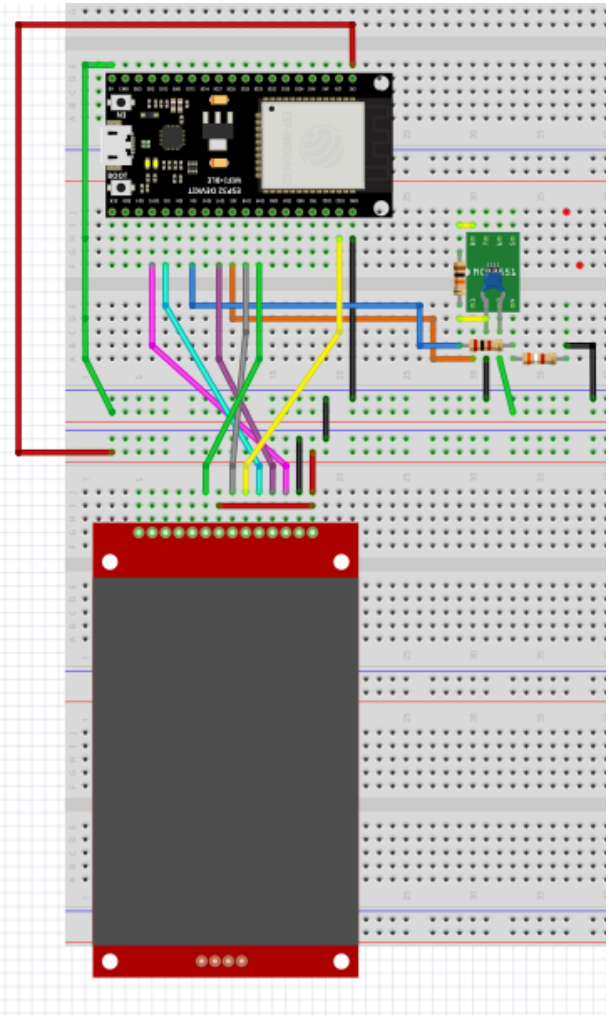
[3] Apesar do Barramento Can permitir um débito binário muito mais elevado, surgiram limitações no módulo de controlo e visualização sendo assim necessário baixar o débito

[4] Esta Queue está definida na biblioteca xxxxx, bem como a propri a interrupção

3 Módulo de Controlo e Visualização

O módulo de controlo permite reunir a informação recebida pelo interface *CAN*, proveniente do módulo de aquisição, facultando esses dados ao utilizador. A informação é disponibilizada através de um *LCD* e enviada via *Wifi* para um *Broker MQTT*, permitindo assim o acesso aos dados de uma aplicação móvel. A composição deste módulo inclui um *Esp32*, um *transceiver CAN MCP2551* e um display *TFT*.

A figura 3.1 apresenta o esquema eléctrico simplificado do hardware deste subsistema, bem como o *dashboard MQTT* desenvolvido para visualização e controlo.



(a) Esquema Elétrico simplificado

(b) Dasboard MQTT

Figura 3.1: Módulo de visualização e controlo

3.1 Tarefas do módulo de controlo

O escalonamento e identificação das tarefas do módulo de controlo e visualização encontram-se resumidos na tabela 3.1

Tabela 3.1: Escalonamento do módulo de controlo.

Core	**Tarefas**	**Descrição da funcionalidade**	**Prioridade**
0	vTFTDisplayValues	Controlo da informação para o *LCD*	1
0	vWifiKeepAlive	Garante a comunicação com o broker *LCD*	1
0	CanRx	Receber a mensagem *CAN*	1
1	CANTx	Enviar a mensagem *CAN*	1
0	vTFTPresentation	Apresentação dos autores e do projecto	1

3.2 Queue's, Semáforos e Mutex's do módulo de controlo

À semelhança do módulo de aquisição, são implementadas *Queue's* para a interacção entre tarefas. A tabela 3.2 apresenta a utilidade destas ferramentas para este módulo.

Tabela 3.2: Lista de Semáforos e *Queue's* do módulo de controlo

Tipo	Designação	Função
Queue	xQueueBroker	Guarda temporariamente a mensagem a enviar
Queue	callbackQueue	Guarda temporariamente a mensagem recebida
Queue	xDataQueue	Comunicação de informação entre CanRx e vTFTDisplayValues

3.3 Descrição Funcional do Subsistema de Visualização e Controlo

A principal função do subsistema de visualização e controlo é de receber os dados do subsistema de aquisição e disponibilizar os mesmo ao utilizador, contudo, é também possível enviar comandos ao subsistema de aquisição.

A figura 7.1 ilustra o funcionamento deste módulo.

Figura 3.2: Diagrama de funcionamento do sistema de Aquisição

3.3.1 Tarefa vTFTPresentation

Esta tarefa é a primeira a correr e escreve no Diplay TFT o nome do projecto e os autores. Após 10 segundos cria as restantes tarefas e apaga-se a si mesma.

3.3.2 Tarefa vWifiKeepAlive

Esta tarefa é a única periódica, sendo que corre a cada 2 segundos. A sua função é a de manter a ligação ao broker MQTT, sem a mesma não seria possível receber mensagens via MQTT, mas apenas enviar.

3.3.3 Tarefa CanRx

Sempre que é recebida uma messagem CAN, é despoletada a interrupção CANreadframe();, descrita na biblioteca esp_32_builtin.cpp, esta coloca os valores na Queue callbackQueue, que por sua vez desbloqueia a tarefa CanRx. A tarefa CanRX, á semelhança do que já foi descrito no subsistema anterior, coloca os valores recebidos na Queue xDataQueue.

3.3.4 Tarefa TFTDisplayValues

Esta tarefa corre sempre que existam valores na Queue xDataQueue e descodifica a mensagem pelo seu id e pelos bytes de dados, apresentando estes valores no *Display TFT* e disponibilizando-os via *MQTT broker*.

3.3.5 CanTX

Quando é recebida uma mensagem via *MQTT broker* é despoletada a função "Broker MQTT Callback", que coloca dados na Queue xQueueBroker.

4 Principais alterações e configurações de bibliotecas

As Bibiloteca esp_32_builtin.cpp e o código forncecido no exemplo do arduino ”TFT Meters”foram alterados, pelo que o projecto apenas funcionará se as devidas alterações forem feitas.

4.1 Alteração da função Plot Pointer

O código presente em 8.4 é fornecido nos exemplos do arduino e serve para ajustar cada ponteiro do TFT ao seu novo valor. Na configuração original, sempre que o programa entrava nesta função, todos os ponteiros seriam ajustados. Como o que se prentende é apenas ajustar o ponteiro correspondente ao sensor lido, foi passado o índice do sensor a ler e o valor a ajustar, reduzindo significativamente o impacto computacional.

```
void plotPointer(int i, int sValue)
{
  int dy = 187;
  byte pw = 16;
  tft.setTextColor(TFT_GREEN, TFT_BLACK);
  // Move the 6 pointers one pixel towards new value
  //for (int i = 0; i < 6; i++){
    char buf[8]; dtostrf(sValue, 4, 0, buf);
    tft.drawRightString(buf, i * 40 + 36 - 5, 187 - 27 + 155 - 18, 2);
    int dx = 3 + 40 * i;
    if (sValue < 0)   sValue = 0; // Limit value to emulate needle end stops
    if (sValue > 100) sValue = 100;

    while (!(sValue == old_value[i])) {
      dy = 187 + 100 - old_value[i];
      if (old_value[i] > sValue)
      {
        tft.drawLine(dx, dy - 5, dx + pw, dy, TFT_WHITE);
        old_value[i]--;
        tft.drawLine(dx, dy + 6, dx + pw, dy + 1, TFT_RED);
      }
      else
      {
        tft.drawLine(dx, dy + 5, dx + pw, dy, TFT_WHITE);
        old_value[i]++;
        tft.drawLine(dx, dy - 6, dx + pw, dy - 1, TFT_RED);
      }
    }
 // }
 // }
}
```

Algoritmo 4.1: Plot Pointer

4.2 Biblioteca esp32_can_builtin.cpp

Dentro desta biblioteca na função init são criadas tasks que permitem processar a mesnagem CAN recebida, contudo uma das tasks foi rescrita no programa principal.

```
void ESP32CAN::_init()
{
    if (debuggingMode) Serial.println("Built in CAN Init");
    for (int i = 0; i < BI_NUM_FILTERS; i++)
    {
        filters[i].id = 0;
        filters[i].mask = 0;
        filters[i].extended = false;
        filters[i].configured = false;
    }

    if (!initializedResources) {
                                //Queue size, item size
        CAN_cfg.rx_queue = xQueueCreate(BI_RX_BUFFER_SIZE,sizeof(CAN_frame_t));
        CAN_cfg.tx_queue = xQueueCreate(BI_TX_BUFFER_SIZE,sizeof(CAN_frame_t));
        callbackQueue = xQueueCreate(16, sizeof(CAN_FRAME));
        CAN_initRXQueue();
        //A linha abaixo foi comentada uma vez que a task task_CAN foi reescrita no programa principal
                    //func        desc    stack, params, priority, handle to task
        //xTaskCreate(&task_CAN, "CAN_RX", 8192, this, 15, NULL);
        xTaskCreatePinnedToCore(&task_LowLevelRX, "CAN_LORX", 4096, this, 19, NULL, 1);
        xTaskCreatePinnedToCore(&CAN_WatchDog_Builtin, "CAN_WD_BI", 2048, this, 10, NULL, 1);
        initializedResources = true;
    }
}
}
```

Algoritmo 4.2: Plot Pointer

4.3 Ficheiro de configurações na biblioteca TFT_eSPI

O ficheiro user_setup.h desta biblioteca também teve que ser alterado para a seguinte configuração.

```
#define TFT_MISO 19
#define TFT_MOSI 23
#define TFT_SCLK 18
#define TFT_CS   15  // Chip select control pin
#define TFT_DC    2  // Data Command control pin
#define TFT_RST   17  // Reset pin (could connect to RST pin)
//#define TFT_RST  -1  // Set TFT_RST to -1 if display RESET is connected to ESP32 board RST
}
```

Algoritmo 4.3: Plot Pointer

5 Conclusão

O desenvolvimento deste projecto permitiu consolidar todos os conteúdos abordados ao longo do semestre.

De um modo geral todos os objetivos foram compridos e o sistema está 100% funcional.

As Bibiloteca xx e o código exemplo yy foram alterados, pelo que o projecto apenas funcionará se as devidas alterações forem feitas.

Todo o projecto e as suas bibliotecas estão disponíveis através do seguinte Link :

`https://github.com/engineeringforlife/SCE_Data_Logger_CAN`

Link do vídeo no Youtube:

`https://www.youtube.com/watch?v=EKvW6zoHNWc&feature=youtu.be`

Referências

[1] S. Labs, “I2c humidity and temperature sensor,” pp. 18–25, 2016. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/Si7006-A20.pdf

[2] ST, “Ultra-low-power high-performance 3-axis ”pico”accelerometer,” pp. 12–29, 2015. [Online]. Available: https://www.st.com/resource/en/datasheet/lis2hh12.pdf

[3] L. optoeletronics, “Optical sensor,” pp. 10–24, 9 April 2014. [Online]. Available: https://optoelectronics.liteon.com/upload/download/DS86-2014-0006/LTR-329ALS-01_DS_V1.pdf

Anexos

1. Esquema elétrico
2. PinOut Pysense
3. Código Fonte ”Projecto SCE”(Emissor CAN)
4. Código Fonte ”Projecto SCE”(Receptor CAN)

6 Esquema elétrico

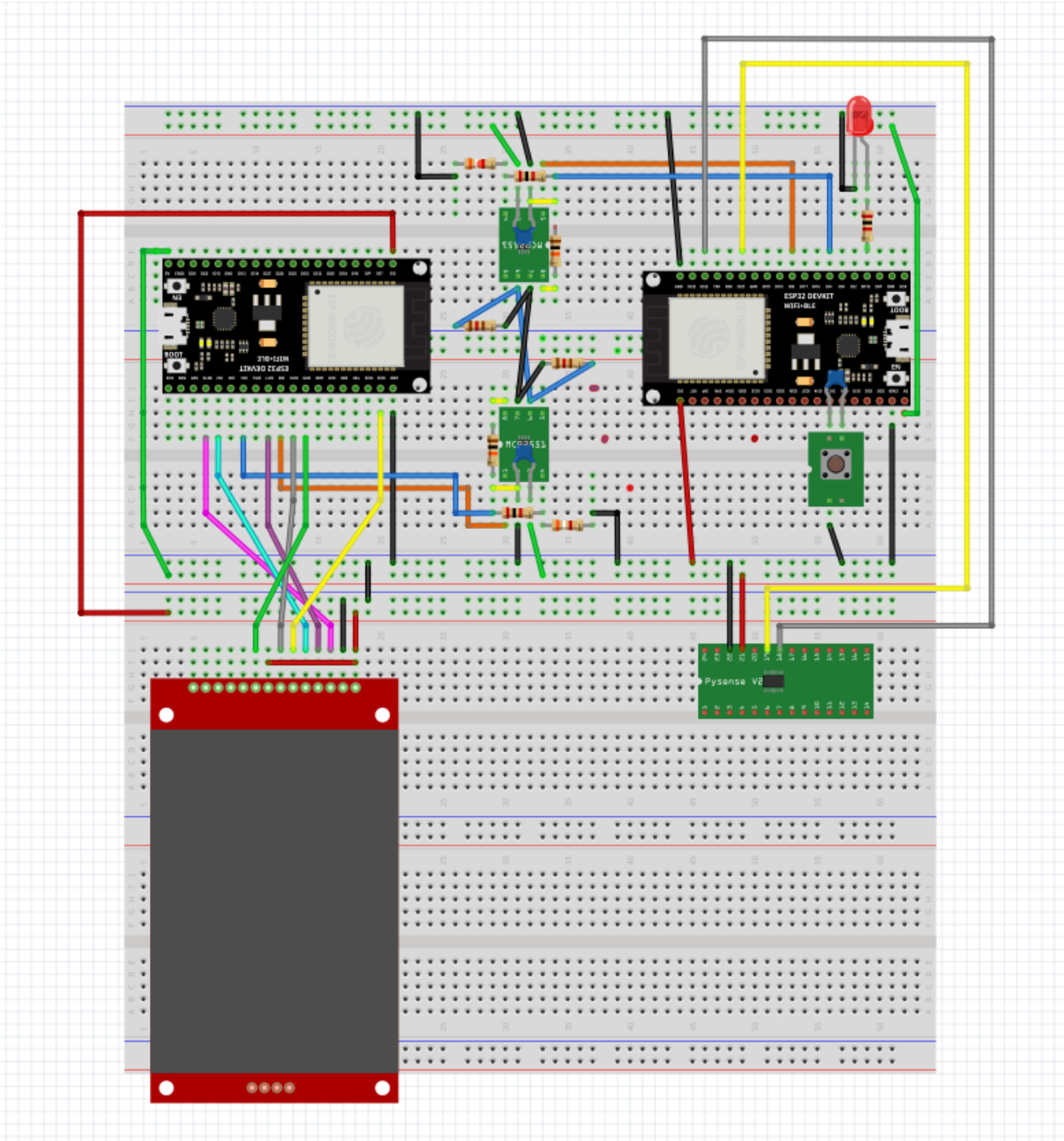


Figura 6.1: Diagrama de funcionamento do sistema de Aquisição

7 PinOut Pysense

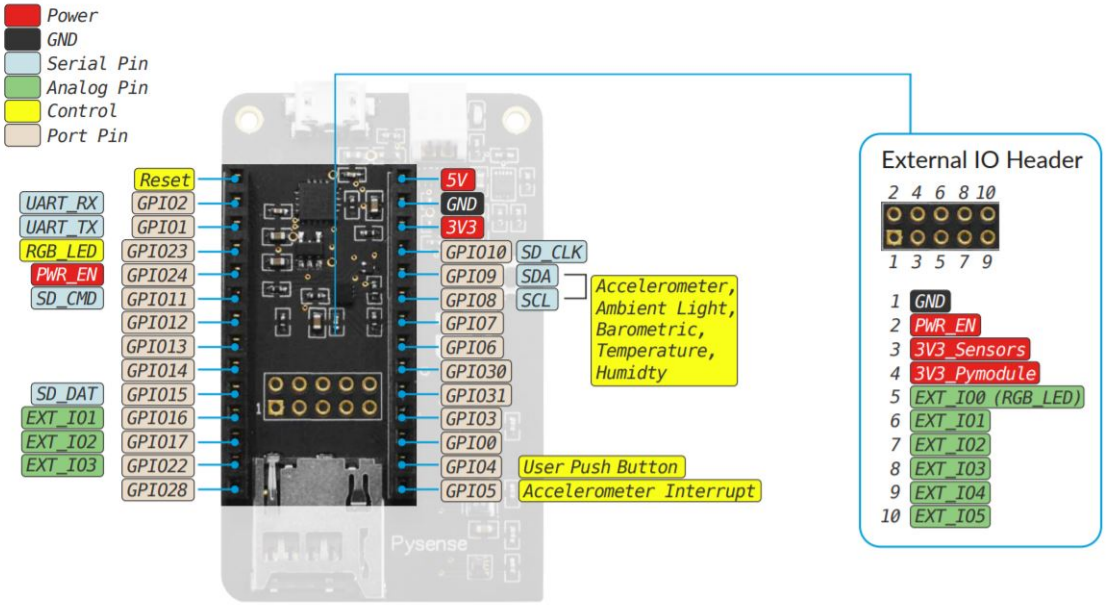


Figura 7.1: Diagrama de funcionamento do sistema de Aquisição

8 Codigo Fonte ”Projecto SCE”(Emissor CAN)

```
void plotPointer(int i, int sValue)
{
#include "Arduino.h"
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include <Wire.h>
#include <rhio-LIS2HH12.h>
#include <esp32_can.h>
#include <SPI.h>

#define mainDELAY_LOOP_COUNT    400000 //( 0xffffff )

// SI7006-A20 I2C address is 0x40(64)
#define AddrSI7006 0x40
#define AddrLTR329AL 0x29

#define mainSENDER_1    1
#define mainSENDER_2    2
#define mainSENDER_3    3

char tempComand = 0xF3;
char humComand = 0xF5;


static void vGetHum( void *pvParameters );
static void vGetTemp( void *pvParameters );
static void vGetLum( void *pvParameters );
static void vGetAccell( void *pvParameters );
static void vCanTx( void *pvParameters );
static void vMyCanRx( void *pvParameters );
static void vManageValues( void *pvParameters );


/* The service routine for the interrupt.  This is the interrupt that the task
will be synchronized with. */
static void IRAM_ATTR vExternalInterruptHandler( void );

// pin to generate interrupts
const uint8_t interruptPin = 12;
const uint8_t LED_BUILTIN = 2;
/*-----------------------------------------------------------*/

QueueHandle_t xI2CQueue;
extern QueueHandle_t callbackQueue;
QueueHandle_t xDataQueue;
SemaphoreHandle_t xMutex;

typedef struct
{
  float ucValue1;
  float ucValue2;
  float ucValue3;
  int ucSource;
} xData;

LIS2HH12 lis = LIS2HH12();


/*Imprime a mensagem CAN
 *
 */
void printFrame(CAN_FRAME *message)
{
  Serial.print(message->id, HEX);
  if (message->extended) Serial.print(" X ");
  else Serial.print(" S ");
  Serial.print(message->length, DEC);
  Serial.print(" ");
  for (int i = 0; i < message->length; i++) {
    Serial.print(message->data.byte[i], HEX);
    Serial.print(" ");
  }
  Serial.println();
}

```

```
79 void gotHundred(CAN_FRAME *frame)
80 {
81
82   Serial.print("gotHundred ");
83 }
84
85 /*
86  * Configura es iniciais
87  */
88 void setup( void )
89 {
90   Serial.begin(115200);
91   while(!Serial);
92
93   pinMode(interruptPin, INPUT_PULLUP);
94   pinMode(15, OUTPUT);
95   attachInterrupt(digitalPinToInterrupt(interruptPin), &vExternalInterruptHandler,
       CHANGE);
96
97   Serial.println("Initializing CAN");
98   pinMode(GPIO_NUM_16, OUTPUT);
99   digitalWrite(GPIO_NUM_16, LOW); //enable CAN transceiver
100  CAN0.setCANPins(GPIO_NUM_4, GPIO_NUM_5);
101  CAN0.init(500000);
102
103  Serial.println("SENDER Ready ...!");
104
105  CAN0.watchFor(); //then let everything else through anyway
106  CAN0.setCallback(0, gotHundred); //callback on that first special filter
107
108  Wire.begin();
109  Wire.beginTransmission(AddrSI7006);
110  Wire.endTransmission();
111  delay(300);
112
113  //Wire.begin();
114  Wire.beginTransmission(AddrLTR329AL);
115  Wire.write(0x80); //control register
116  Wire.write(0x01); //gain=1, active mode
117  Wire.endTransmission();
118  delay(500);
119
120  lis.begin();
121  lis.setBasicConfig();
122  xI2CQueue = xQueueCreate( 5, sizeof( xData ) );
123  xMutex = xSemaphoreCreateMutex();
124  xDataQueue = xQueueCreate( 10, sizeof( xData ) );
125  if ( xI2CQueue != NULL )
126  {
127    if( xMutex != NULL ){
128        xTaskCreatePinnedToCore( vGetHum, "vGetHum", 2048, NULL, 2, NULL, 1);
129        xTaskCreatePinnedToCore( vGetTemp, "vGetTemp", 2048, NULL, 2, NULL, 1);
130        xTaskCreatePinnedToCore( vGetLum, "vGetLum", 2048, NULL, 2, NULL, 1);
131        xTaskCreatePinnedToCore( vGetAccell, "vGetAccell", 2048, NULL, 2, NULL, 1);
132        xTaskCreatePinnedToCore( vCanTx, "vCanTx", 1024, NULL, 1, NULL, 0); //core 0
       e 1???
133        xTaskCreatePinnedToCore( vMyCanRx, "vMyCanRx", 4096, NULL, 1, NULL, 0);
134        xTaskCreatePinnedToCore( vManageValues, "vManageValues", 2048, NULL, 1, NULL
     , 0);
135
136    }else{
137      Serial.print( "The mutex not be created.\r\n" );
138    }
139  }
140  else
141  {
142    Serial.print( "The queue not be created.\r\n" );
143  }
144 }
145
146 void loop()
147 {
148   vTaskDelete( NULL );
149 }
150 /*
151  * Handler da Interrup o externa
152  */
153 static void  IRAM_ATTR  vExternalInterruptHandler( void )
154 {
155 static signed portBASE_TYPE xHigherPriorityTaskWoken;
156 xData xSenderStructure;
```

```
157
158   xHigherPriorityTaskWoken = pdFALSE;
159
160   xSenderStructure.ucSource = 0x05;
161
162   xSenderStructure.ucValue1 =0x01;
163
164   xQueueSendToFrontFromISR( xI2CQueue, &xSenderStructure, 0);
165
166   Serial.print( "An Interrupt was generated!!!!!!!!11.\r\n" );
167   if( xHigherPriorityTaskWoken == pdTRUE )
168   {
169     Serial.print( "Context Switch!!!!!!!11.\r\n" );
170     portYIELD_FROM_ISR();
171   }
172 }
173
174 /*
175  * L   o valor do sensor de humidade
176  * Periodo de 1s
177  */
178 static void vGetHum( void *pvParameters )
179 {
180   xData xSenderStructure;
181   portBASE_TYPE xStatus;
182   const TickType_t xTicksToWait = 50 / portTICK_PERIOD_MS;
183   float humidity =0.0;
184   uint8_t dataT[2] = {0};
185   uint8_t dataH[2] = {0};
186   TickType_t xLastWakeTime;
187   xLastWakeTime = xTaskGetTickCount();
188
189   for ( ;; )
190   {
191
192     if( xSemaphoreTake( xMutex, portMAX_DELAY)){
193       Serial.print( "vSenderSI7006_HUM got access.\r\n" );
194         // Start I2C transmission
195         Wire.beginTransmission(AddrSI7006);
196         // Send HUMIDITY measurement command, NO HOLD MASTER
197         Wire.write(0xF5);
198         // Stop I2C transmission
199         Wire.endTransmission();
200         delay(50);
201         // Request 2 bytes of data
202         Wire.requestFrom(AddrSI7006, 2);
203         // Read 2 bytes of data
204         // humidity msb, humidity lsb
205         if(Wire.available() == 2)
206         {
207           dataH[0] = Wire.read();
208           dataH[1] = Wire.read();
209         }
210
211         xSemaphoreGive(xMutex);
212         // Convert the data
213         humidity  = ((dataH[0] * 256.0) + dataH[1]);
214         humidity = ((125 * humidity) / 65536.0) - 6;
215
216         xSenderStructure.ucSource = 0x03;
217         //xSenderStructure.ucValue1 = (float)millis();
218         xSenderStructure.ucValue1 =int(humidity);
219       xStatus = xQueueSendToBack( xI2CQueue, &xSenderStructure, xTicksToWait );
220
221       if ( xStatus != pdPASS )
222       {
223         Serial.print( "SI7006_HUM Could not send to the queue.\r\n" );
224       }
225     }else{
226       Serial.print( "vSenderSI7006_HUM couldn't access.\r\n" );
227       }
228
229     vTaskDelayUntil( &xLastWakeTime, ( 1000/portTICK_PERIOD_MS ) );
230   }
231 }
232
233
234 /*
235  * L   o sensor de temperatura
236  * Periodo de 1s
237  */
```

```
238 static void vGetTemp( void *pvParameters )
239 {
240   xData xSenderStructure;
241   portBASE_TYPE xStatus;
242   const TickType_t xTicksToWait = 500 / portTICK_PERIOD_MS;
243   float humidity =0.0;
244   float temp= 0.0;
245   float ctemp = 0.0;
246   uint8_t dataT[2] = {0};
247   TickType_t xLastWakeTime;
248   xLastWakeTime = xTaskGetTickCount();
249
250
251   for ( ;; )
252   {
253
254     if( xSemaphoreTake( xMutex, portMAX_DELAY)){
255       Serial.print( "vGetTemp got access.\r\n" );
256         // Start I2C transmission
257       Wire.beginTransmission(AddrSI7006);
258       // Send temperature measurement command, NO HOLD MASTER
259       Wire.write(0xF3);
260       // Stop I2C transmission
261       Wire.endTransmission();
262       delay(10);
263       // Request 2 bytes of data
264       Wire.requestFrom(AddrSI7006, 2);
265       // Read 2 bytes of data
266       // temp msb, temp lsb
267       if(Wire.available() == 2)
268       {
269         dataT[0] = Wire.read();
270         dataT[1] = Wire.read();
271       }
272       xSemaphoreGive(xMutex);
273
274       // Convert the data
275       temp  = ((dataT[0] * 256.0) + dataT[1]);
276       ctemp = ((175.72 * temp) / 65536.0) - 46.85;
277         xSenderStructure.ucSource = 0x02;
278         //xSenderStructure.ucValue1 = (float)millis();;
279         xSenderStructure.ucValue1 = int(ctemp);
280       xStatus = xQueueSendToBack( xI2CQueue, &xSenderStructure, xTicksToWait );
281
282       if ( xStatus != pdPASS )
283       {
284         Serial.print( "vGetTemp could not send to the queue.\r\n" );
285       }
286     }else{
287       Serial.print( "vGetTemp failed to got access.\r\n" );
288     }
289     vTaskDelayUntil( &xLastWakeTime, ( 1000/portTICK_PERIOD_MS ) );
290
291   }
292 }
293
294 /*
295  * L   o valor da luminusidade
296  * Periodo de 1s
297  */
298 static void vGetLum( void *pvParameters )
299 {
300   xData xSenderStructure;
301   portBASE_TYPE xStatus;
302   const TickType_t xTicksToWait = 500 / portTICK_PERIOD_MS;
303
304   byte msb=0, lsb=0;
305   uint16_t l;
306   TickType_t xLastWakeTime;
307   xLastWakeTime = xTaskGetTickCount();
308
309       for ( ;; )
310       {
311         if( xSemaphoreTake( xMutex, portMAX_DELAY)){
312             Serial.print( "vGetLum got access.\r\n" );
313
314             //channel 0
315             Wire.beginTransmission(AddrLTR329AL);
316             Wire.write(0x8A); //low
317             Wire.endTransmission();
318             Wire.requestFrom((uint8_t)AddrLTR329AL, (uint8_t)1);
```

```
319             delay(10);
320             if(Wire.available())
321             lsb = Wire.read();
322 
323             Wire.beginTransmission(AddrLTR329AL);
324             Wire.write(0x8B); //high
325             Wire.endTransmission();
326             Wire.requestFrom((uint8_t)AddrLTR329AL, (uint8_t)1);
327             delay(10);
328             if(Wire.available())
329             msb = Wire.read();
330             xSemaphoreGive(xMutex);
331 
332             l = (msb<<8) | lsb;
333             Serial.print( "ABOUT TO PINT CHANNEL OF LTR329ALS01\r\n" );
334             Serial.println(l, DEC); //output in steps (16bit)
335           xSenderStructure.ucSource = 0x04;
336           //xSenderStructure.ucValue1 = (float)millis();
337           xSenderStructure.ucValue1 = l;
338           xStatus = xQueueSendToBack( xI2CQueue, &xSenderStructure, xTicksToWait );
339           if ( xStatus != pdPASS )
340           {
341             Serial.print( "LTR329ALS01 Could not send to the queue.\r\n" );
342           }
343         }else{
344           Serial.print( "vGetLum failed to got access.\r\n" );
345         }
346         vTaskDelayUntil( &xLastWakeTime, ( 1000/portTICK_PERIOD_MS ) );
347       }
348 }
349 
350 /*
351  * l   o valor da acelera  o
352  * Periodo de 0.5s
353  */
354 
355 static void vGetAccell( void *pvParameters )
356 {
357   xData xSenderStructure;
358   portBASE_TYPE xStatus;
359   const TickType_t xTicksToWait = 500 / portTICK_PERIOD_MS;
360   float x=1, y=2, z=3;
361   TickType_t xLastWakeTime;
362   xLastWakeTime = xTaskGetTickCount();
363 
364   for ( ;; )
365   {
366     if( xSemaphoreTake( xMutex, xTicksToWait)){
367       Serial.print( "vGetAccell got access.\r\n" );
368         lis.getAccel(&x, &y, &z);
369         xSemaphoreGive(xMutex);
370         xSenderStructure.ucSource = 0x01;
371        // xSenderStructure.ucValue1 = (float)millis();
372         xSenderStructure.ucValue1 = x;
373         xSenderStructure.ucValue2 = y;
374         xSenderStructure.ucValue3 = z;
375       xStatus = xQueueSendToBack( xI2CQueue, &xSenderStructure, xTicksToWait );
376       if ( xStatus != pdPASS )
377       {
378         Serial.print( "LIS2HH12 Could not send to the queue.\r\n" );
379       }else{
380         Serial.print( "LIS2HH12 has sent to the queue.\r\n" );
381       }
382     }else{
383       Serial.print( "vGetAccell failed to got access.\r\n" );
384     }
385     vTaskDelayUntil( &xLastWakeTime, ( 1000/portTICK_PERIOD_MS ) );
386   }
387 }
388 
389 
390 /*-----------------------------------------------------------*/
391 
392 
393 /*
394  * Envia a mensagem CAN
395  * Desbloqueia sempre que tem dados na QUEUE xI2CQueue
396  */
397 static void vCanTx( void *pvParameters )
398 {
399   xData xReceivedStructure;
```

```
  TickType_t xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();

  for ( ;; )
  {
    //exemplo do exerc cio 16
      /* Wait for a message to arrive. */
      xQueueReceive( xI2CQueue, &xReceivedStructure, 0 );
      /* There is no need to check the return value as the task will block
      indefinitely and only run again when a message has arrived.  When the
      next line is executed there will be a message to be output. */

    CAN_FRAME txFrame;
    txFrame.rtr = 0;
    txFrame.id = int(xReceivedStructure.ucSource);
    txFrame.extended = false;
    txFrame.length = 3;
    txFrame.data.uint8[0] = int(xReceivedStructure.ucValue1);
    txFrame.data.uint8[1] = int(xReceivedStructure.ucValue2);
    txFrame.data.uint8[2] = int(xReceivedStructure.ucValue3);


      Serial.println("SENT ...!");

    Serial.println(xReceivedStructure.ucSource);
    Serial.println(xReceivedStructure.ucValue1);
    CAN0.sendFrame(txFrame);
    vTaskDelayUntil( &xLastWakeTime, ( 100/portTICK_PERIOD_MS ) );

  }
}

/*
 * L   a mesnsagem CAN e reencaminha para a Queue xDataQueue.
 * Desbloqueia sempre que tiver dados na Queue callbackQueue
 */
static void vMyCanRx( void *pvParameters )
{
  xData xDataStruct;
  CAN_FRAME msg;
  //CAN_frame_t rxFrame;
  portBASE_TYPE xStatus;
  const TickType_t xTicksToWait = 50 / portTICK_PERIOD_MS;
  TickType_t xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();


  for ( ;; )
  {

    //Nesta fase  j   recebe os dados filtrados
      if(xQueueReceive(callbackQueue, &msg, portMAX_DELAY)==pdTRUE)
      {
        Serial.println("task_LowLevelRX Drumond
    ************************************");
        Serial.println("task_LowLevelRX Drumond
    ************************************************************");
        printFrame(&msg);
        xDataStruct.ucSource=msg.id;
        xDataStruct.ucValue1 = msg.data.uint8[0];
        xDataStruct.ucValue2 = msg.data.uint8[1];
        xDataStruct.ucValue3 = msg.data.uint8[2];

        xQueueSend(xDataQueue, &xDataStruct, 0 );

      }
  }
}
/*
 * Desbloqueia sempre que tiver dados na Queue xDataQueue
 * Permite Ligar ou desligar o LED quando    recebida uma mensagem.
 */
static void vManageValues( void *pvParameters )
{
  xData xSenderStructure;
  portBASE_TYPE xStatus;
  int i=1;
  const TickType_t xTicksToWait = 50 / portTICK_PERIOD_MS;
  char accell[10];
  char lum[5];
  char hum[5];
```

```
479   char temp[5];
480   uint8_t dataT[2] = {0};
481   uint8_t dataH[2] = {0};
482   TickType_t xLastWakeTime;
483   xLastWakeTime = xTaskGetTickCount();
484
485
486   for ( ;; )
487   {
488       if(xQueueReceive(xDataQueue, &xSenderStructure, portMAX_DELAY)==pdTRUE){
489         if(xSenderStructure.ucSource == 6){
490           Serial.print("\n**************************************\n" );
491           Serial.print("\nLigar LED\n" );
492           Serial.print("\n**************************************\n" );
493
494           digitalWrite(15, uint8_t(xSenderStructure.ucValue1));
495           Serial.print("\n**************************************\n" );
496       }
497
498   }
499 }
500 }
501
502
503 }
```

Algoritmo 8.4: Código do emissor CAN

9 Codigo Fonte ”Projecto2”(Receptor CAN)

```
#include "Arduino.h"
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include <Wire.h>

//Broker MQTT Libraries
#include <WiFi.h>
#include <PubSubClient.h>
//Can Library
#include "esp32_can.h"

#include <TFT_eSPI.h> // Hardware-specific library
#include <SPI.h>

TFT_eSPI tft = TFT_eSPI();       // Invoke custom library
#define TFT_GREY 0x5AEB
//vari vel necess ria paraos mostradores
int old_value[6] = { -1, -1, -1, -1, -1, -1};

#define mainDELAY_LOOP_COUNT    400000 //( 0xffffff )

//Prototipos das Tasks
static void vTFTPresentation( void *pvParameters );
static void vTFTDisplayValues( void *pvParameters );
static void CANRx( void *pvParameters );
static void CANTx( void *pvParameters );
static void vWifiKeepAlive( void *pvParameters );


extern QueueHandle_t callbackQueue;

QueueHandle_t xQueueBroker;
QueueHandle_t xDataQueue;
SemaphoreHandle_t xCountingSemaphore;
SemaphoreHandle_t xMutex; //=0?
TaskHandle_t xTFTPresentationHandle;

typedef struct
{
  float ucValue1;
  float ucValue2;
  float ucValue3;
  int ucSource;
} xData;


/* Defini  es do LED */
#define PIN_LED     25
/* Defines do MQTT */
#define TOPIC_LED    "topic_led"
#define TOPIC_TEMP   "topic_temp"
#define TOPIC_LUM    "topic_lum"
#define TOPIC_ACCELL "topic_accell"
#define TOPIC_HUM    "topic_hum"
#define TOPIC_ALARM  "topic_alarm"


#define ID_MQTT  "43a21d06b4a0454c903ddeeb425e814f"
const char* SSID = "MEO-351390"; // SSID / nome da rede WI-FI que deseja se conectar
const char* PASSWORD = "df82e87bf3"; // Senha da rede WI-FI que deseja se conectar
const char* BROKER_MQTT = "broker.hivemq.com"; //URL do broker MQTT que se deseja
    utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT

//Vari veis e objetos globais
WiFiClient espClient; // Cria o objeto espClient
PubSubClient MQTT(espClient); // Instancia o Cliente MQTT passando o objeto
    espClient

//PROTOTIPES
void initWiFi(void);
void initMQTT(void);
void mqtt_callback(char* topic, byte* payload, unsigned int length);
void reconnectMQTT(void);
void reconnectWiFi(void);
```

```
77 void VerificaConexoesWiFIEMQTT(void);
78
79
80 void initWiFi(void)
81 {
82     delay(10);
83     Serial.println("------Conexao WI-FI------");
84     Serial.print("Conectando-se na rede: ");
85     Serial.println(SSID);
86     Serial.println("Aguarde");
87     reconnectWiFi();
88 }
89
90 /* Fun  o: inicializa par metros de conex o MQTT(endere o do
91  *         broker, porta e seta fun  o de callback)
92  * Par metros: nenhum
93  * Retorno: nenhum
94  */
95 void initMQTT(void)
96 {
97     MQTT.setServer(BROKER_MQTT, BROKER_PORT);   //informa qual broker e porta deve
    ser conectado
98     MQTT.setCallback(mqtt_callback);            //atribui fun  o de callback (
    fun  o chamada quando qualquer informa  o de um dos t picos subescritos
    chega)
99 }
100
101 /* Fun  o: fun  o de callback
102  *         esta fun  o    chamada toda vez que uma informa  o de
103  *         um dos t picos subescritos chega)
104  * Par metros: nenhum
105  * Retorno: nenhum
106  */
107 void mqtt_callback(char* topic, byte* payload, unsigned int length)
108 {
109
110     xData xSenderStructure;
111     String msg;
112     portBASE_TYPE xStatus;
113
114     /* obtem a string do payload recebido */
115     for(int i = 0; i < length; i++)
116     {
117        char c = (char)payload[i];
118        msg += c;
119     }
120
121     //a MESNSAGEM DEVE SER COLOCADA NUMA QUEUE, E A TAREFA CANTX DEVE SER
    DESBLOQUEADA
122     Serial.print("Chegou a seguinte string via MQTT: ");
123     Serial.println(msg);
124
125     /* toma a   o dependendo da string recebida */
126     if (msg.equals("1"))
127     {
128       xSenderStructure.ucSource= 6;
129       xSenderStructure.ucValue1=1;
130         Serial.print("LED aceso mediante comando MQTT");
131     }else if(msg.equals("0"))
132     {
133       xSenderStructure.ucSource= 6;
134     xSenderStructure.ucValue1=0;
135         Serial.print("LED apagado mediante comando MQTT");
136     }
137     xStatus = xQueueSendToBack( xQueueBroker, &xSenderStructure, 0 );
138     if ( xStatus != pdTRUE )
139     {
140       Serial.print( "mqtt_callback could not send to the queue.\r\n" );
141     }else{
142       Serial.print( "mqtt_callback send to the queue------------------------------.\
    r\n" );
143     }
144
145 }
146
147 /* Fun  o: reconecta-se ao broker MQTT (caso ainda n o esteja conectado ou em
    caso de a conex o cair)
148  *         em caso de sucesso na conex o ou reconex o, o subscribe dos t picos
     refeito.
149  * Par metros: nenhum
150  * Retorno: nenhum
```

```
 */
void reconnectMQTT(void)
{
    while (!MQTT.connected())
    {
        Serial.print("* Tentando se conectar ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe(TOPIC_LED);
        }
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}

/* Fun   o: verifica o estado das conex es WiFI e ao broker MQTT.
 *         Em caso de desconex o (qualquer uma das duas), a conex o
 *            refeita.
 * Par metros: nenhum
 * Retorno: nenhum
 */
void VerificaConexoesWiFIEMQTT(void)
{
    if (!MQTT.connected())
        reconnectMQTT(); //se n o h   conex o com o Broker, a conex o    refeita

     reconnectWiFi(); //se n o h   conex o com o WiFI, a conex o    refeita
}

/* Fun   o: reconecta-se ao WiFi
 * Par metros: nenhum
 * Retorno: nenhum
 */
void reconnectWiFi(void)
{
    //se j   est  conectado a rede WI-FI, nada    feito.
    //Caso contr rio,  s o efetuadas tentativas de conex o
    if (WiFi.status() == WL_CONNECTED)
        return;
    WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }

    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(SSID);
    Serial.println("IP obtido: ");
    Serial.println(WiFi.localIP());
}

/*Fun   o que desenha os mostradores
 * Fornecida nos exemplos do TFT
 */
void plotLinear(char *label, int x, int y)
{
  int w = 36;
  tft.drawRect(x, y, w, 155, TFT_GREY);
  tft.fillRect(x + 2, y + 19, w - 3, 155 - 38, TFT_WHITE);
  tft.setTextColor(TFT_CYAN, TFT_BLACK);
  tft.drawCentreString(label, x + w / 2, y + 2, 2);
  for (int i = 0; i < 110; i += 10)
  {
    tft.drawFastHLine(x + 20, y + 27 + i, 6, TFT_BLACK);
  }
  for (int i = 0; i < 110; i += 50)
  {
    tft.drawFastHLine(x + 20, y + 27 + i, 9, TFT_BLACK);
  }
  tft.fillTriangle(x + 3, y + 127, x + 3 + 16, y + 127, x + 3, y + 127 - 5, TFT_RED)
    ;
  tft.fillTriangle(x + 3, y + 127, x + 3 + 16, y + 127, x + 3, y + 127 + 5, TFT_RED)
    ;
```

```
230   tft.drawCentreString("---", x + w / 2, y + 155 - 18, 2);
231 }
232
233 /*Fun  o que ajusta os ponteiros nos mostradores
234  * A fun  o foi alterada face    vers o exemplo do arduino
235  *
236  * */
237 void plotPointer(int i, int sValue)
238 {
239   int dy = 187;
240   byte pw = 16;
241   tft.setTextColor(TFT_GREEN, TFT_BLACK);
242   // Move the 6 pointers one pixel towards new value
243   //for (int i = 0; i < 6; i++){
244     char buf[8]; dtostrf(sValue, 4, 0, buf);
245     tft.drawRightString(buf, i * 40 + 36 - 5, 187 - 27 + 155 - 18, 2);
246     int dx = 3 + 40 * i;
247     if (sValue < 0)   sValue = 0; // Limit value to emulate needle end stops
248     if (sValue > 100) sValue = 100;
249
250     while (!(sValue == old_value[i])) {
251       dy = 187 + 100 - old_value[i];
252       if (old_value[i] > sValue)
253       {
254         tft.drawLine(dx, dy - 5, dx + pw, dy, TFT_WHITE);
255         old_value[i]--;
256         tft.drawLine(dx, dy + 6, dx + pw, dy + 1, TFT_RED);
257       }
258       else
259       {
260         tft.drawLine(dx, dy + 5, dx + pw, dy, TFT_WHITE);
261         old_value[i]++;
262         tft.drawLine(dx, dy - 6, dx + pw, dy - 1, TFT_RED);
263       }
264     }
265  // }
266  // }
267 }
268
269 /*Fun  o que imprime a mensagem CAN
270  */
271 void printFrame(CAN_FRAME *message)
272 {
273   Serial.print(message->id, HEX);
274   if (message->extended) Serial.print(" X ");
275   else Serial.print(" S ");
276   Serial.print(message->length, DEC);
277   Serial.print(" ");
278   for (int i = 0; i < message->length; i++) {
279     Serial.print(message->data.byte[i], HEX);
280     Serial.print(" ");
281   }
282   Serial.println();
283 }
284
285 void gotHundred(CAN_FRAME *frame)
286 {
287
288 //  static portBASE_TYPE xHigherPriorityTaskWoken;
289 //    xHigherPriorityTaskWoken = pdFALSE;
290   Serial.print("gotHundred ");
291  // xSemaphoreGiveFromISR( xCountingSemaphore, (BaseType_t*)&
       xHigherPriorityTaskWoken );
292  // printFrame(frame);
293 }
294
295
296
297
298 /*Cont m todas as configura es necess rias
299  */
300 void setup( void )
301 {
302
303   //Inicializa  o da porta s rie(debug
304   Serial.begin(115200);
305   while(!Serial);
306
307   xDataQueue = xQueueCreate( 10, sizeof( xData ) );
308
309   xQueueBroker = xQueueCreate( 5, sizeof( xData ) );
```

```
310   xMutex = xSemaphoreCreateMutex();
311
312   xCountingSemaphore = xSemaphoreCreateCounting( 10, 0 );
313
314   //Inicializa  o do barramento CAN
315   Serial.println("Initializing CAN");
316   pinMode(GPIO_NUM_16, OUTPUT);
317   digitalWrite(GPIO_NUM_16, LOW); //enable CAN transceiver
318   // void ESP32CAN::setCANPins(gpio_num_t rxPin, gpio_num_t txPin)
319   CAN0.setCANPins(GPIO_NUM_4, GPIO_NUM_5);
320   //CAN0.begin(500000);
321   CAN0.init(500000);
322   Serial.println("RECEIVER Ready ...!");
323
324   //CAN0.watchFor(0x00, 0x0A); //setup a special filter
325   CAN0.watchFor(); //then let everything else through anyway
326   CAN0.setCallback(0, gotHundred); //callback on that first special filter
327
328   Serial.print( "Vamos ver se come a aqui\n" );
329   //Inicia o display TFT
330   tft.init();
331   tft.setRotation(0);
332   tft.fillScreen(TFT_BLACK);
333
334   /* Inicializa a conexao wi-fi */
335   initWiFi();
336   /* Inicializa a conexao ao broker MQTT */
337   initMQTT();
338   VerificaConexoesWiFIEMQTT();
339   MQTT.subscribe(TOPIC_LED);
340   //MQTT.loop();
341
342   if ( xDataQueue != NULL ){
343     if( xMutex != NULL ){
344       if(xCountingSemaphore != NULL){
345         if(xQueueBroker != NULL){
346           Serial.print( "Success\n" );
347           xTaskCreatePinnedToCore( vTFTPresentation, "vTFTPresentation", 1024, NULL,
     2, &xTFTPresentationHandle, 1);
348         }else{
349           Serial.print( "The queueBroker not be created.\r\n" );
350         }
351       }else{
352         Serial.print( "The counting semaphore could not be created\r\n" );
353       }
354     }else{
355       Serial.print( "The mutex not be created.\r\n" );
356     }
357   }else
358   {
359     Serial.print( "The queue not be created.\r\n" );
360   }
361 }
362
363 void loop()
364 {
365   vTaskDelete( NULL );
366 }
367
368 /*
369  * Tarefa responsavel pelo envio da mendsagem CAN
370  * Desbloqueia assim que existam dados na Queue xQueueBroker
371  */
372 static void CANTx( void *pvParameters )
373 {
374   xData xReceivedStructure;
375   TickType_t xLastWakeTime;
376   xLastWakeTime = xTaskGetTickCount();
377
378   for ( ;; )
379   {
380       /* Wait for a message to arrive. */
381       xQueueReceive( xQueueBroker, &xReceivedStructure, portMAX_DELAY );
382       /* There is no need to check the return value as the task will block
383       indefinitely and only run again when a message has arrived.  When the
384       next line is executed there will be a message to be output. */
385
386     CAN_FRAME txFrame;
387     txFrame.rtr = 0;
388     txFrame.id = int(xReceivedStructure.ucSource);
389     txFrame.extended = false;
```

```
    txFrame.length = 3;
    txFrame.data.uint8[0] = int(xReceivedStructure.ucValue1);
    txFrame.data.uint8[1] = int(xReceivedStructure.ucValue2);
    txFrame.data.uint8[2] = int(xReceivedStructure.ucValue3);
    Serial.println("SENT ...!");
    Serial.println(xReceivedStructure.ucSource);
    Serial.println(xReceivedStructure.ucValue1);
    CAN0.sendFrame(txFrame);
    vTaskDelayUntil( &xLastWakeTime, ( 100/portTICK_PERIOD_MS ) );

  }
}



/*Esta tarefa    a primeira a correr neste n   e faz a apresenta  o
 * do grupo, espera 10 segundos, de seguida cria as restantes tarefas
 * e apaga-se a si mesma.
 */
static void vTFTPresentation( void *pvParameters )
{

      //Strings de apresenta  o do projecto
    tft.drawString("Este projecto tem o ", 0, 0, 4);
    tft.drawString("nome de \"data- ", 0, 25, 4);
    tft.drawString("-logguer CAN\" e foi", 0, 50, 4);
    tft.drawString("desenvolvido para a", 0, 75, 4);
    tft.drawString("cadeira de SCE por:", 0, 100, 4);
    tft.drawString("David Drumond", 15, 175, 4);
    tft.drawString("Edgar Paulo", 15, 200, 4);

    delay(10000);
    //Limpa o ecr
    tft.fillScreen(TFT_BLACK);

    Serial.print("\n*********************************************************\n" );
    Serial.print("\nvTFTPresentation is running and about to delete itself\r\n" );
    //Create other tasks here
      xTaskCreatePinnedToCore( vTFTDisplayValues, "vTFTDisplayValues", 8196, NULL,
    1, NULL, 0);
      xTaskCreatePinnedToCore( CANRx, "CANRx", 4098, NULL, 1, NULL, 1);
      xTaskCreatePinnedToCore( CANTx, "CANTx", 4098, NULL, 1, NULL, 1);
      xTaskCreatePinnedToCore( vWifiKeepAlive, "vWifiKeepAlive", 4098, NULL, 1, NULL
    , 1);
    vTaskDelete(xTFTPresentationHandle);
}

/*
 * Esta tarefa corre a cada 2 segundos, apra manter a liga  o ao broke est vel
 */
static void vWifiKeepAlive( void *pvParameters )
{
  const TickType_t xTicksToWait = 2000 / portTICK_PERIOD_MS;
  TickType_t xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();


  for ( ;; )
  {
    Serial.print("\nMQTT RECONECT*************************************\r\n" );
    VerificaConexoesWiFIEMQTT();
    MQTT.loop();

    vTaskDelayUntil( &xLastWakeTime, xTicksToWait);
  }
}

/*
 * Tarefa que recebe a mesnsagem CAN
 *  S   corre Quando existem dados na Queue callbackQueue
 * Reencaminha os dados para a Queue xDataQueue
 */
static void CANRx( void *pvParameters )
{
  xData xSenderStructure;
  CAN_FRAME msg;

  for ( ;; )
  {

    //Nesta fase  j   recebe os dados filtrados
```

```
      if(xQueueReceive(callbackQueue, &msg, portMAX_DELAY)==pdTRUE)
      {
        Serial.println("task_LowLevelRX Drumond");
        printFrame(&msg);

        xSenderStructure.ucSource=msg.id;
        xSenderStructure.ucValue1 = msg.data.uint8[0];
        xSenderStructure.ucValue2 = msg.data.uint8[1];
        xSenderStructure.ucValue3 = msg.data.uint8[2];

        xQueueSend(xDataQueue, &xSenderStructure, 0 );

      }
  }
}


/*
 * Ajusta os ponteiros do display TFT e publica os valores
 * dos sensores nos t picos correspondentes do MQTT
 */
static void vTFTDisplayValues( void *pvParameters )
{
  xData xSenderStructure;
  int i=1;
  char accell[10];
  char lum[5];
  char hum[5];
  char temp[5];

  byte d = 40;
  plotLinear("TEMP", 0, 160);
  plotLinear("HUM", 1 * d, 160);
  plotLinear("LUM", 2 * d, 160);
  plotLinear("X", 3 * d, 160);
  plotLinear("Y", 4 * d, 160);
  plotLinear("Z", 5 * d, 160);
  Serial.print("\nvTFTDisplayValues\n" );

  for ( ;; )
  {


      if(xQueueReceive(xDataQueue, &xSenderStructure, portMAX_DELAY)==pdTRUE)
      {
        VerificaConexoesWiFIEMQTT();
        MQTT.loop();
        Serial.print("QueueReceive(xQueuexData, &xSenderStructure, portMAX_DELAY)==
    pdTRUE\n" );
        switch(xSenderStructure.ucSource) {
          case 1:

            Serial.print("\nValor 1 publicado no topico\n" );
            plotPointer(3, int(xSenderStructure.ucValue1));
            plotPointer(4, int(xSenderStructure.ucValue2));
            plotPointer(5, int(xSenderStructure.ucValue3));
            sprintf(accell, "(%d, %d, %d)", int(xSenderStructure.ucValue1), int(
    xSenderStructure.ucValue2), int(xSenderStructure.ucValue3) );
            MQTT.publish(TOPIC_ACCELL, accell );
            break;
          case 2:
            Serial.print("\nValor 2 publicado no topico\n" );
            plotPointer(0, int(xSenderStructure.ucValue1));
            sprintf(temp, "%d", int(xSenderStructure.ucValue1));
            MQTT.publish(TOPIC_TEMP, temp );

            break;
          case 3:
            Serial.print("\nValor 3 publicado no topico\n" );
            plotPointer(1, int(xSenderStructure.ucValue1));
            sprintf(hum, "%d", int(xSenderStructure.ucValue1));
        MQTT.publish(TOPIC_HUM, hum );
            break;
          case 4:
            Serial.print("\nValor 4 publicado no topico\n" );
            plotPointer(2, int(xSenderStructure.ucValue1));
            sprintf(lum, "%d", int(xSenderStructure.ucValue1));
            MQTT.publish(TOPIC_LUM, lum );
            break;
          case 5:
            i++;
```

```
548             Serial.print("\nFoi gerada uma interrup o externa");
549             if(i>1){
550               i=0;
551               tft.drawString("Alarme ON! ", 0, 0, 4);
552               MQTT.publish(TOPIC_ALARM, "Alarme ON!" );
553
554             }else{
555               MQTT.publish(TOPIC_ALARM, "Alarme OFF!" );
556               tft.fillRect(0, 0, 240, 25,TFT_BLACK );
557               Serial.print("\n
    **********************************************************************\n
    ************************************************\n");
558             }
559
560             break;
561           default:
562             // code block
563             Serial.print("\nDefault\n" );
564         }
565       }
566
567
568
569   }
570 }
```

Algoritmo 9.5: Código do receptor CAN