| **Gerb-BMSTU_01** | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |
| --- | --- |

ФАКУЛЬТЕТСПЕЦИАЛЬНОЕ МАШИНОСТРОЕНИЕ

КАФЕДРААВТОНОМНЫЕ ИНФОРМАЦИОННЫЕ И УПРАВЛЯЮЩИЕ СИСТЕМЫ

Домашнее задание №2 по курсу

**«Программно-алгоритмическое обеспечение вычислительных**

**средств АИУС»**

Вариант 12

**Группа**: СМ5-32Б

**Студент**: Курган Б.Е.

**Преподаватель***:* Чудников В.В.

Москва, 2019

**Задание:**  
Датчик на валу электродвигателя выдает значения угловой скорости и момента. Диапазон изменения угловой скорости 0...20000 об/мин. Критическими значениями считаются значения меньше 20 об/мин и более 20000 об/мин. Диапазон возможных значений момента 0...1000 Н\*м. Оптимальный момент - 500 Н\*м, допустимое отклонение от номинала 10%. Остальные значения соответствуют аварийному режиму. Произвести проверку значений и вывести пользователю сообщение при аварийном режиме.

Используя наработки первого домашнего задания, необходимо в рамках ОСРВ FreeRTOS реализовать эмуляцию работы модулей передатчика, приемника и канала передачи данных.

**Исходный код: (кроме main.c)**

**transmitter.h**

#ifndef TRANSMITTER\_H\_

#define TRANSMITTER\_H\_

#include "FreeRTOS.h"

#include "queue.h"

#include "semphr.h"

#pragma pack(push,1)

typedef struct

{

short address;

short packetNumber;

unsigned short ang\_vel;

unsigned short torque;

unsigned char instruction;

unsigned short CRC16\_value;

}dataPacket;

typedef struct

{

dataPacket\* pointer;

uint32\_t size;

} qMessage;

#pragma pack(pop)

dataPacket\* createData(int packetNumber);//создает и заполняет пакет с номером packetNumber

int SendData(dataPacket\* data, unsigned int size, QueueHandle\_t destQueue); //SendData

void show\_packet(dataPacket\* packet); //отображение содержимого пакета данных

unsigned short CRC16(unsigned char \*pcBlock, unsigned short len); // расчет КС

int rand25(); //функция вероятности 75%

#endif /\* TRANSMITTER\_H\_ \*/

**transmitter.c**

#include <stdlib.h>

#include <stdio.h>

#include <assert.h>

#include "FreeRTOS.h"

#include "queue.h"

#include "receiver.h"

#include "transmitter.h"

#define SEND\_OK 0

#define SEND\_ERR 1

unsigned short CRC16(unsigned char \*pcBlock, unsigned short len)

{

unsigned short crc = 0xFFFF;

unsigned char n;

while (len--)

{

crc ^= \*pcBlock++ << 8;

for (n = 0; n < 8; n++)

{

crc = crc & 0x8000 ? (crc << 1) ^ 0x1021 : crc << 1;

}

}

return crc;

}

dataPacket\* createData(int packetNumber)

{

dataPacket\* newPacket = (dataPacket\*)pvPortMalloc(sizeof(dataPacket));

assert(newPacket != NULL);

newPacket->address = 12;

newPacket->packetNumber = packetNumber;

newPacket->instruction = 0x01;

newPacket->ang\_vel = ((rand() & 1) == 0) ? (10000+(rand()%1000)+(rand25()\*10000)) : (10000-(rand()%1000)+(rand25()\*20000));

newPacket->torque = ((rand() & 1) == 0) ? (500+(rand()%100)+(rand25()\*1000)) : (500-(rand()%100)+(rand25()\*1000));

newPacket->CRC16\_value = CRC16((unsigned char\*)newPacket,(unsigned short)(sizeof(dataPacket)-sizeof(unsigned short)));

if(newPacket->packetNumber % 5 == 0)

{

newPacket->ang\_vel++;

}

return newPacket;

}

int SendData(dataPacket\* data, unsigned int size, QueueHandle\_t destQueue)

{

qMessage\* message = (qMessage\*)malloc(sizeof(qMessage\*));

message->pointer = data;

message->size = size;

if(xQueueSend(destQueue, &message, 0) != pdPASS)

{

return SEND\_ERR;

}

return SEND\_OK;

}

void show\_packet(dataPacket\* packet)

{

printf("Address: %d\r\n",packet->address);

printf("Packet Number: %d\r\n",packet->packetNumber);

printf("Instruction: %d\r\n",packet->instruction);

printf("Data: {Angular velocity = %5d | Torque = %5d}\r\n",packet->ang\_vel,packet->torque);

printf("CRC: %d\r\n\n",packet->CRC16\_value);

}

**receiver.h**

#ifndef RECEIVER\_H\_

#define RECEIVER\_H\_

#include "FreeRTOS.h"

#include "queue.h"

#include "semphr.h"

#include "transmitter.h"

int ReceiveData(qMessage\* from\_buff, SemaphoreHandle\_t hSemaphore); //ReceiveData

#endif /\* RECEIVER\_H\_ \*/

**receiver.c**

#include <stdlib.h>

#include <stdio.h>

#include "FreeRTOS.h"

#include "semphr.h"

#include "transmitter.h"

#include "receiver.h"

int ReceiveData(qMessage\* from\_buff,SemaphoreHandle\_t hSemaphore)

{

qMessage\* message;

xSemaphoreTake(hSemaphore, portMAX\_DELAY);

message = from\_buff;

dataPacket\* packet = message->pointer;

uint32\_t size = (message->size)-sizeof(unsigned short);

free(message);

unsigned short CRC\_check = CRC16((unsigned char\*)packet,(unsigned short)(size));

if((CRC\_check) != (packet->CRC16\_value))

{

printf("%d: CRC doesn't match\r\n",packet->packetNumber);

vPortFree(packet);

return 1;

}

else

{

printf("%d: ",packet->packetNumber);

if((packet->ang\_vel) < 20 || (packet->ang\_vel>20000))

{

printf("Warning: Critical angular velocity\r\n");

}

else if((packet->torque < 400) ||(packet->torque > 600))

{

printf("Warning: Critical torque\r\n");

}

else

{

printf("All systems nominal\r\n");

}

vPortFree(packet);

return 0;

}

}

int rand25()

{

return (rand()&1)&(rand()&1);

}

**data\_channel.h**

#include "receiver.h"

#include "transmitter.h"

#include "FreeRTOS.h"

#include "queue.h"

#include "semphr.h"

#ifndef DATA\_CHANNEL\_H\_

#define DATA\_CHANNEL\_H\_

void relayData(QueueHandle\_t fromQueue, SemaphoreHandle\_t hSemaphore);

#endif /\* DATA\_CHANNEL\_H\_ \*/

**data\_channel.c**

#include "data\_channel.h"

#include "FreeRTOS.h"

#include "queue.h"

#include "receiver.h"

#include "transmitter.h"

#include "hwTask.h"

void relayData(QueueHandle\_t fromQueue, SemaphoreHandle\_t hSemaphore)

{

qMessage\* transport;

xQueueReceive(fromQueue, &transport, portMAX\_DELAY);

inputBuffer = \*transport;

xSemaphoreGive(hSemaphore);

}

**hwTask.h**

#ifndef HWTASK\_H\_

#define HWTASK\_H\_

#include "receiver.h"

#include "transmitter.h"

#include "data\_channel.h"

qMessage inputBuffer;

void hwTaskInit(void);

#endif /\* HWTASK\_H\_ \*/

**hwTask.c**

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "semphr.h"

#include "receiver.h"

#include "transmitter.h"

#include "data\_channel.h"

TaskHandle\_t hTransmitter;

TaskHandle\_t hReceiver;

TaskHandle\_t hDataChannel;

QueueHandle\_t hQueue = NULL;

SemaphoreHandle\_t hSemaphore = NULL;

extern qMessage inputBuffer;

void ReceiverTask(void \*pvParameters);

void TransmitterTask(void \*pvParameters);

void DataChannelTask(void \*pvParameters);

void hwTaskInit(void)

{

hQueue = xQueueCreate(5,sizeof(qMessage\*));

hSemaphore = xSemaphoreCreateBinary();

xTaskCreate(ReceiverTask, "Rx", configMINIMAL\_STACK\_SIZE, NULL, tskIDLE\_PRIORITY + 2, &hReceiver);

xTaskCreate(TransmitterTask, "Tx", configMINIMAL\_STACK\_SIZE, NULL, tskIDLE\_PRIORITY + 1, &hTransmitter);

xTaskCreate(DataChannelTask, "Data Channel", configMINIMAL\_STACK\_SIZE, NULL, tskIDLE\_PRIORITY + 2, &hDataChannel);

}

void ReceiverTask(void \*pvParameters)

{

(void) pvParameters;

for(;;)

{

ReceiveData(&inputBuffer,hSemaphore);

}

}

void TransmitterTask(void \*pvParameters)

{

(void) pvParameters;

uint8\_t counter = 1;

for(;;)

{

SendData(createData(counter),sizeof(dataPacket), hQueue);

counter++;

if(counter == 51)

{

vTaskDelete(NULL);

}

}

}

void DataChannelTask(void \*pvParameters)

{

(void) pvParameters;

for(;;)

{

relayData(hQueue, hSemaphore);

}

}