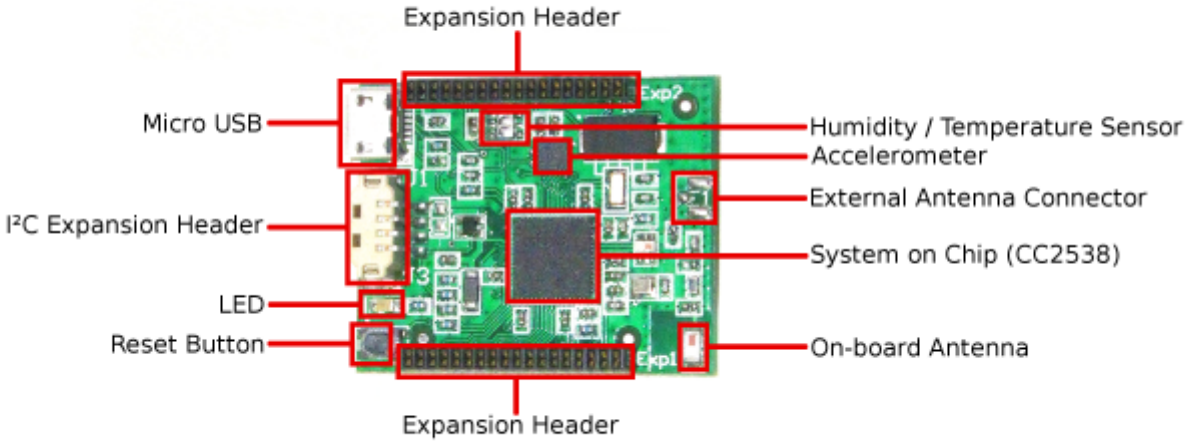
Implementação de driver para linha serial utilizando os protocolos RS422/RS485 no EPOSMote III

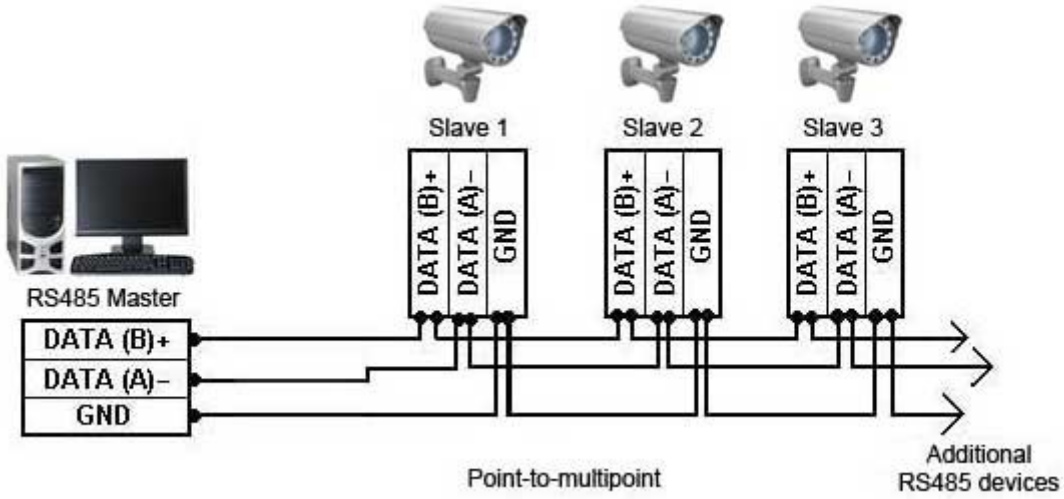
UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE



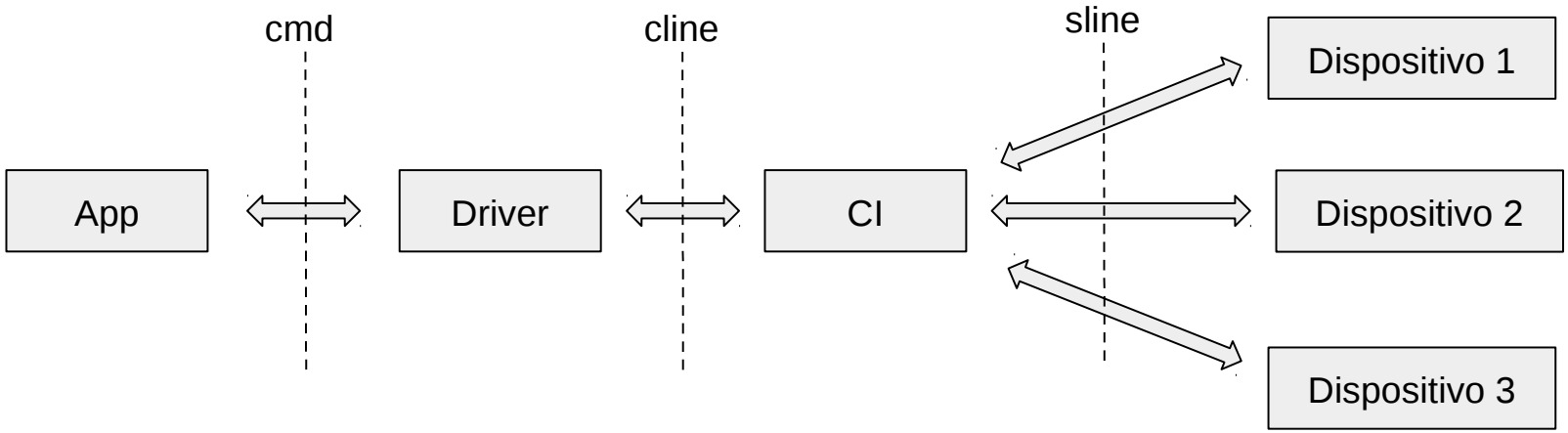
Rudimar Baesso Althof

26/06/2017

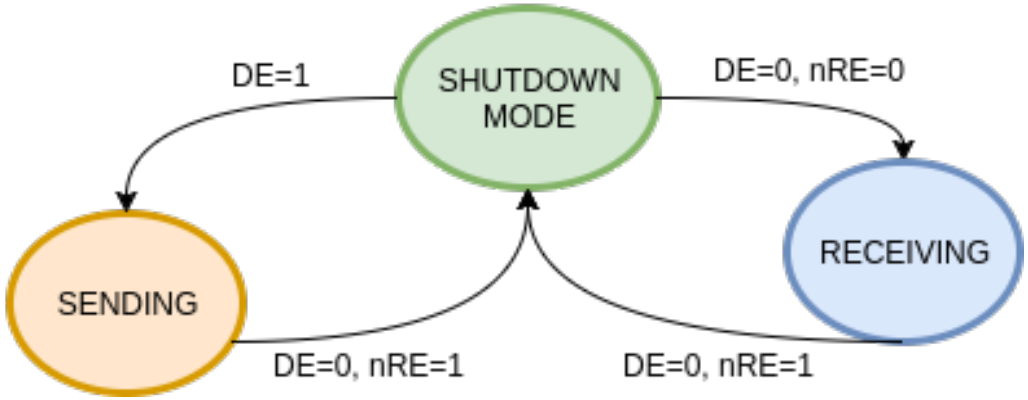
RS485



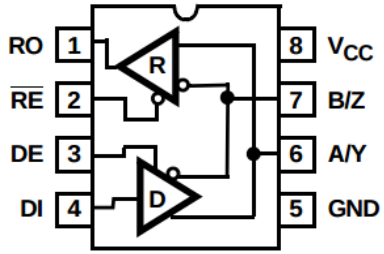
A	B	Valor
1	0	0
0	1	1



Circuito Integrado ISL83483



ISL83483, ISL83485 (PDIP, SOIC)
TOP VIEW

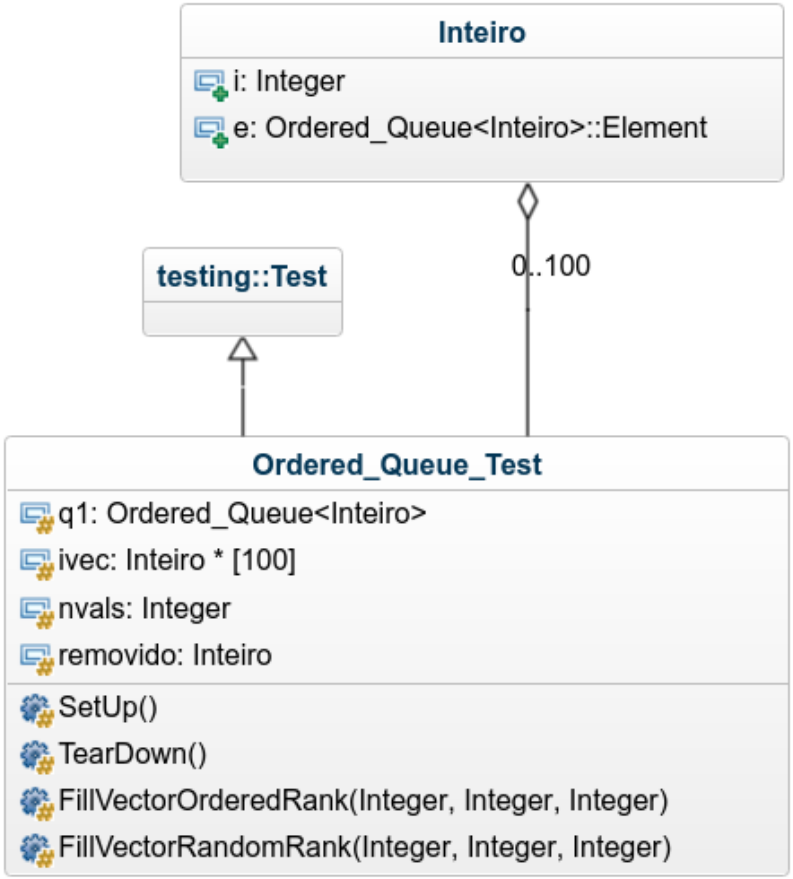


Teste de Fila Ordenada (Ordered_Queue)

Método da classe Ordered_Queue<Inteiro>	Possibilidades	Retorno esperado
insert(Inteiro &)	Elemento existente Elemento nulo	Inteiro inserido Nada ocorre
remove()	Fila com elementos Fila vazia	Inteiro Null
remove(Inteiro & x)	x contido na fila x não presente na fila	x Null
empty()	Fila com elementos Fila vazia	0 1
head()	Fila com elementos Fila vazia	Inteiro Null
tail()	Fila com elementos Fila vazia	Inteiro Null
head() e tail()	Fila com 1 elemento Fila com N>=2 elementos	head() == tail() head() != tail()
size()	Fila com N elementos Fila vazia	N 0

Teste de Fila Ordenada (Ordered_Queue)

- **SetUp:**
  - Atribui o valor 0 a variável \textit{nvalue}, utilizada para controle de memória alocada aos objetos do tipo Inteiro;
- **FillVectorOrderedRank:**
  - Insere *nvalues* valores no vetor de inteiros com rank conhecidos e valores aleatórios entre *min* e *max*;
- **FillVectorRandomRank:**
  - Insere *nvalues* valores no vetor de inteiros com ranks aleatórios entre *min* e *max*. Os valores são iguais ao rank.
- **TearDown:**
  - Libera a memória alocada para os *nvalues* inteiros durante cada teste.



Teste de Fila Ordenada (Ordered_Queue)

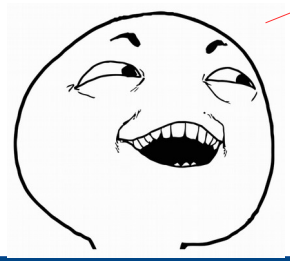
Exemplo de teste utilizando a classe Ordered_Queue_Test:

```
/*
 * Neste teste faz-se o teste do metodo size. Insere-se um numero conhecido de valores
 * e verifica-se o tamanho apos a remocao de cada elemento.
 */
TEST F(Ordered_Queue_Test, SizeTest) {

    if(DEBUG) cout<<"\n\n------ TESTE DO METODO SIZE --------"<<endl;

    //Inserindo valroes de rank conhecido
    FillVectorOrderedRank();

    //Testando o metodo search
    for(int i=nvals;i>0;i--){
        EXPECT EQ(i, q1.size());
        q1.remove();
    }

    EXPECT FALSE(q1.size());
}
```

Teste de Fila Ordenada (Ordered_Queue)

- Criação do tipo inteiro;
- Inserção ordenada;
- Inserção de valor nulo;
- Remoção em fila vazia;
- Remoção de elemento;
- Busca de elemento;
- Retorno da cauda da fila;
- Retorno da cabeça da fila;
- Tamanho da fila;

```
[==========] Running 9 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from Inteiro
[ RUN      ] Inteiro.CriacaoDeInteiro
[       OK ] Inteiro.CriacaoDeInteiro (0 ms)
[----------] 1 test from Inteiro (0 ms total)

[----------] 8 tests from Ordered_Queue_Test
[ RUN      ] Ordered_Queue_Test.RetornoOrdenado
[       OK ] Ordered_Queue_Test.RetornoOrdenado (0 ms)
[ RUN      ] Ordered_Queue_Test.RemocaoElemento
[       OK ] Ordered_Queue_Test.RemocaoElemento (0 ms)
[ RUN      ] Ordered_Queue_Test.RetornoVazia
[       OK ] Ordered_Queue_Test.RetornoVazia (0 ms)
[ RUN      ] Ordered_Queue_Test.EncontraElemento
[       OK ] Ordered_Queue_Test.EncontraElemento (1 ms)
[ RUN      ] Ordered_Queue_Test.TailTest
[       OK ] Ordered_Queue_Test.TailTest (0 ms)
[ RUN      ] Ordered_Queue_Test.HeadTest
[       OK ] Ordered_Queue_Test.HeadTest (0 ms)
[ RUN      ] Ordered_Queue_Test.SizeTest
[       OK ] Ordered_Queue_Test.SizeTest (0 ms)
[ RUN      ] Ordered_Queue_Test.InsersaoElementoNulo
Segmentation fault (core dumped)
```



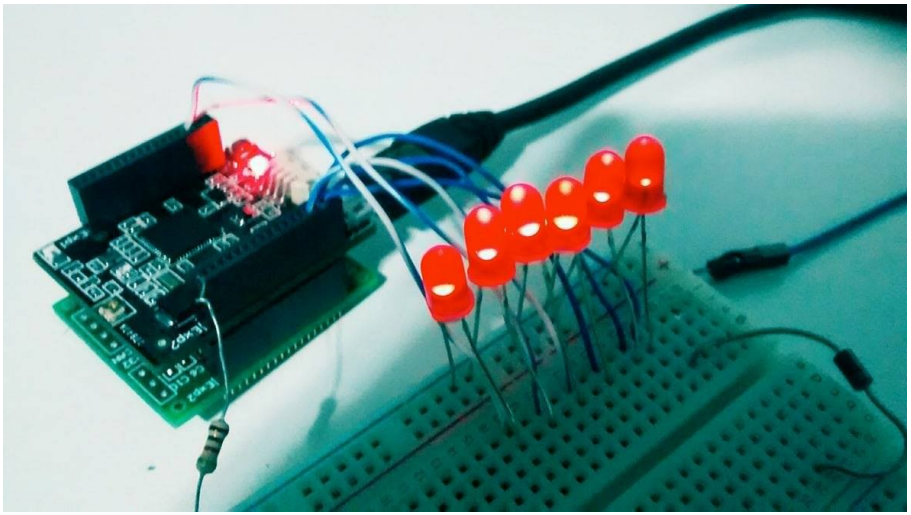
Correção da classe Ordered_Queue

Correção do método insert.

```
//void insert(Element * e) { T::insert(e); }
void insert(Element * e) { if(e==NULL) return; T::insert(e); }
```

```
[==========] Running 9 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from Inteiro
[ RUN      ] Inteiro.CriacaoDeInteiro
[       OK ] Inteiro.CriacaoDeInteiro (0 ms)
[----------] 1 test from Inteiro (0 ms total)

[----------] 8 tests from Ordered_Queue_Test
[ RUN      ] Ordered_Queue_Test.RetornoOrdenado
[       OK ] Ordered_Queue_Test.RetornoOrdenado (0 ms)
[ RUN      ] Ordered_Queue_Test.RemocaoElemento
[       OK ] Ordered_Queue_Test.RemocaoElemento (0 ms)
[ RUN      ] Ordered_Queue_Test.RetornoVazia
[       OK ] Ordered_Queue_Test.RetornoVazia (0 ms)
[ RUN      ] Ordered_Queue_Test.EncontraElemento
[       OK ] Ordered_Queue_Test.EncontraElemento (0 ms)
[ RUN      ] Ordered_Queue_Test.TailTest
[       OK ] Ordered_Queue_Test.TailTest (0 ms)
[ RUN      ] Ordered_Queue_Test.HeadTest
[       OK ] Ordered_Queue_Test.HeadTest (1 ms)
[ RUN      ] Ordered_Queue_Test.SizeTest
[       OK ] Ordered_Queue_Test.SizeTest (0 ms)
[ RUN      ] Ordered_Queue_Test.InsersaoElementoNulo
[       OK ] Ordered_Queue_Test.InsersaoElementoNulo (0 ms)
[----------] 8 tests from Ordered_Queue_Test (1 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 2 test cases ran. (1 ms total)
[  PASSED  ] 9 tests.
```

Teste das GPIOS



Criado a partir do exemplo *led_blink*, testando todos os pinos utilizados na implementação.

Primeiro teste de comunicação RS485

Primeira versão do driver implementada e testada com conversor RS485-serial.


```
class SerialRS485 : public UART {

    public:

        //------ Pinos de acesso ao chip -------
        GPIO * nRE;//PC5 - RS485 RE - Receiving Enable
        GPIO * DE; //PC6 - RS485 DE - Driver Enable

        SerialRS485(unsigned int unit, unsigned int baud rate, unsigned int data bits, unsigned int parity, unsigned int stop bits)
        : UART(unit, baud rate, data bits, parity, stop bits)
        {
            uartNumber = unit;
            nRE = new GPIO('C',5, GPIO::OUT);
            DE = new  GPIO('C',6, GPIO::OUT);
        }
```

Testes da Classe SerialRS485

‘Templatização’ da classe SerialRS485 para facilitar os testes de classes internas.

```
template <class GPIOClass, class UARTClass>
class SerialRS485 {

    public:

        //PC5 - RS485 RE - Receiving Enable
        GPIO * nRE;
        //PC6 - RS485 DE - Driver Enable
        GPIO * DE;
        UART * uart;

        GPIOClass * gpioCreator;
        UARTClass * uartCreator;


        SerialRS485(unsigned int baud_rate, unsigned int data_bits, unsigned int parity, unsigned int stop_bits)
        {
            gpioCreator = new GPIOClass();
            uartCreator = new UARTClass();

            nRE     = gpioCreator->newGPIO('C',5, GPIO_Common::OUT);
            DE      = gpioCreator->newGPIO('C',6, GPIO_Common::OUT);
            uart    = uartCreator->newUART(1, baud_rate, data_bits, parity, stop_bits);
```

Classe de mock UARTCreator

A classe recebe como parâmetro o número de leituras e escritas.

```
template <int nWrites=0, int nReads=0>
class MockUARTCreator {

 public:

    UART * uart1;
```

Apenas um objeto UART deve ser instanciado.

```
uart1 = new UART();

EXPECT_CALL(*this, newUART(_,_,_,_,_))
        .Times(1)
        .WillOnce(ReturnPointee(&uart1));
```

Classe de mock UARTCreator

O método *put* da UART é chamado apenas uma vez a cada escrita.

```
EXPECT_CALL(*uart1, put(_))
    .Times(nWrites);
```

O método *get* da UART é chamado apenas uma vez a cada leitura.

```
for (int i = nReads; i > 0 ; ) {
    EXPECT_CALL(*uart1, get())
        .WillOnce(Return(--i))
        .RetiresOnSaturation();
}
```

Classe de mock GPIOCreator

A classe recebe como parâmetro o número de leituras e escritas.

```
template <int nWrites=0, int nReads=0>
class MockGPIOCreator {
    public:

        GPIO * nRE;
        GPIO * DE;

        enum { OUT = 0, IN = 1 };
```

Objetos GPIO bem definidos são retornados a cada chamada.

```
    nRE = new GPIO();
    DE = new GPIO();

    EXPECT CALL(*this, newGPIO( , , ))
        .Times(2)
        .WillOnce(ReturnPointee(&nRE))
        .WillOnce(ReturnPointee(&DE));
    }
```

Classe de mock GPIOCreator

Testes do pino DE:

```
/*
 * constructor  -> shutdown state
 * receiving    -> nReads vezes
 * destructor   -> shutdown state
 */
EXPECT CALL(*DE, set(0))
    .Times(nReads+1+1);

/*
 * writeWord  -> nWrites vezes
 */
EXPECT CALL(*DE, set(1))
    .Times(nWrites);
```

Testes do pino nRE:

```
/*
 * construtor -> shutdown state
 * destructor -> shutdown state
 */
EXPECT CALL(*nRE, set(1))
    .Times(1+1);

/*
 * readWord     -> nReads vezes
 */
EXPECT CALL(*nRE, set(0))
    .Times(nReads);
```

Testes da Classe SerialRS485

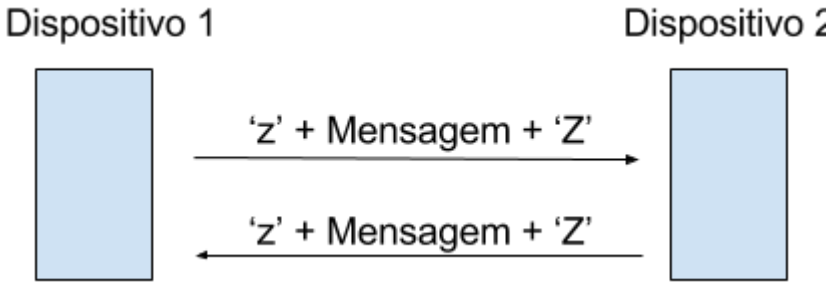
Exemplo de teste falho:

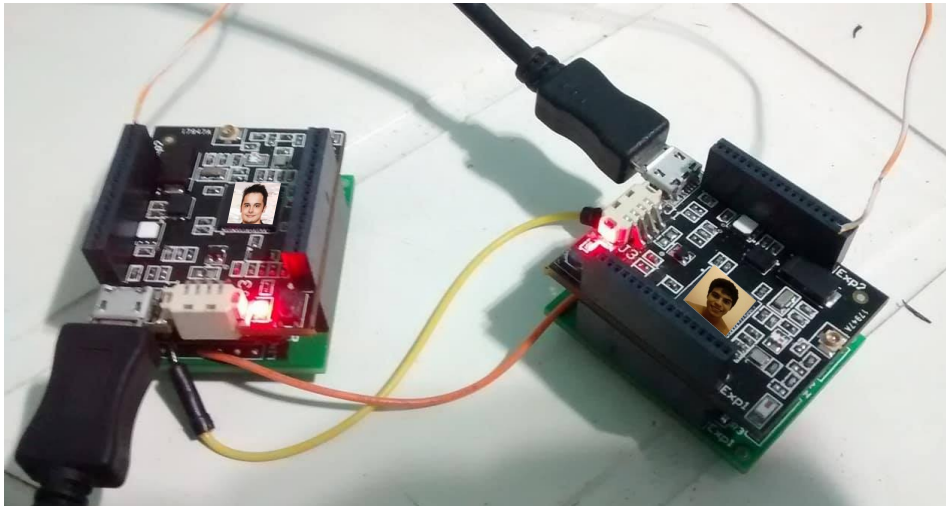
```
TEST(SerialRS485Test, testFailExample){

    const int nWrites=2;
    const int nReads=1;
    SerialRS485 < MockGPIOCreator<nWrites, nReads>, MockUARTCreator<nWrites, nReads> > r(9600, 8, UART Common::NONE, 1);

    r.writeWord('1');
    r.writeWord('2');
}
```

```
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from SerialRS485Test
[ RUN      ] SerialRS485Test.testWriteWord
[       OK ] SerialRS485Test.testWriteWord (0 ms)
[ RUN      ] SerialRS485Test.testReadWord

[       OK ] SerialRS485Test.testReadWord (2 ms)
[ RUN      ] SerialRS485Test.testFailExample
mockGPIO.h:100: Failure
Actual function call count doesn't match EXPECT_CALL(*nRE, set(0))...
         Expected: to be called once
           Actual: never called - unsatisfied and active
mockGPIO.h:108: Failure
Actual function call count doesn't match EXPECT_CALL(*DE, set(0))...
         Expected: to be called 3 times
           Actual: called twice - unsatisfied and active
mockUART.h:73: Failure
Actual function call count doesn't match EXPECT_CALL(*uart1, get())...
         Expected: to be called once
           Actual: never called - unsatisfied and active
[  FAILED  ] SerialRS485Test.testFailExample (0 ms)
[----------] 3 tests from SerialRS485Test (2 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (2 ms total)
[  PASSED  ] 2 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] SerialRS485Test.testFailExample
```

Teste de comunicação 2

EPOSsível a troca de mensagens entre dispositivos?





```
Enviando mensagem no protocolo zZ:
z Teste comunicacao rs485, Rodrigo voce esta ai? Z

Iniciando leitura de mensagem no protocolo zZ:
zTeste de comunicacao RS485, Sim Rudimar, bora testar!Z
```

```
/dev/ttyACM0 - PuTTY
 Criando classe RS485
Esperando mensagem
 Teste comunicacao rs485, Rodrigo voce esta ai?
Enviando mensagem
Mensagem enviada
```

Conclusões

Teste de componente do EPOS:

```
[ RUN      ] Ordered_Queue_Test.TailTest
[       OK ] Ordered_Queue_Test.TailTest (0 ms)
[ RUN      ] Ordered_Queue_Test.HeadTest
[       OK ] Ordered_Queue_Test.HeadTest (0 ms)
[ RUN      ] Ordered_Queue_Test.SizeTest
[       OK ] Ordered_Queue_Test.SizeTest (0 ms)
[ RUN      ] Ordered_Queue_Test.InsersaoElementoNulo
Segmentation fault (core dumped)
```

Uso de mock:

mockUART.h:65: ERROR: this mock object (used in test SerialTest.testWriteWord) should be deleted but never is. Its address is @0x1159a50.

Driver de comunicação:

```
Enviando mensagem no protocolo zZ:
z Teste comunicacao rs485, Rodrigo voce esta ai? Z

Iniciando leitura de mensagem no protocolo zZ:
zTeste de comunicacao RS485, Sim Rudimar, bora testar!Z
```

```
/dev/ttyACM0 - PuTTY
Criando classe RS485
Esperando mensagem
 Teste comunicacao rs485, Rodrigo voce esta ai?
Enviando mensagem
Mensagem enviada
```


I
RU