**РАЗРАБОТКА ЭЛЕКТРОННОГО УСТРОЙСТВА НА БАЗЕ МИКРОКОНТРОЛЛЕРА**

**Текст программы**

**ЮУрГУ.308/415-12.03.01**

**Листов 17**

Челябинск 2022

АННОТАЦИЯ

Ильин С.С. Разработка устройства с возможностью измерять температуру бесконтактным датчиком. – Челябинск: ЮУрГУ, 40 с.

Разработка устройства с возможностью измерять температуру бесконтактным датчиком. Задачи работы: разработать код программного обеспечения;

Работа программы продемонстрирована совместно с платой XNUCLEO-F411RE.

В ходе выполнения данного курсового проекта было разработано устройство, позволяющее измерять температуру бесконтактным способом.

Предусмотрена возможность питания от солнечной батареи за счёт подключения соответствующего модуля.

СОДЕРЖАНИЕ

[1. Main.C 4](#_Toc101871899)

[2. interfacetasks.CPP 7](#_Toc101871900)

[3. interfacetasks.H 8](#_Toc101871901)

[4. interfaceunits.CPP 9](#_Toc101871902)

[5. interfaceunits.H 10](#_Toc101871903)

[6. mailtask.CPP 11](#_Toc101871904)

[7. mailtask.H 12](#_Toc101871905)

[8. messagetransmit.CPP 13](#_Toc101871906)

[9. messagetransmit.H 14](#_Toc101871907)

[10. temperaturetask.CPP 15](#_Toc101871908)

[11. temperaturetask.H 16](#_Toc101871909)

[12. celsius.CPP 17](#_Toc101871910)

[13. celsius.H 18](#_Toc101871911)

[14. kelvin.CPP 19](#_Toc101871912)

[15. kelvin.H 20](#_Toc101871913)

[16. temperature.CPP 21](#_Toc101871914)

[17. temperature.H 22](#_Toc101871915)

[18. smbusdriver.CPP 23](#_Toc101871916)

[19. smbusdriver.H 24](#_Toc101871917)

[20. filter.CPP 26](#_Toc101871918)

[21. filter.H 27](#_Toc101871919)

1. Main.C

#include "rtos.hpp" // for Rtos

#include "mailbox.hpp" // for Mailbox

#include "event.hpp" // for Event

#include "mailtask.h"

#include "messagetransmit.h"

#include "rccregisters.hpp" // for RCC

#include "usart2registers.hpp" //for USART2

#include "buttontask.h" // for MyFirstTask

#include "Application/Diagnostic/GlobalStatus.hpp"

#include <gpioaregisters.hpp> // for GPIOA

#include <gpiobregisters.hpp> // for GPIOB

#include <gpiocregisters.hpp> // for GPIOC

#include "i2c1registers.hpp" //for I2C1

#include "smbusdriver.h"

#include "filter.h"

#include "temperaturetask.h"

#include "temperature.h"

#include <string>

#include <messagetransmit.h>

std::uint32\_t SystemCoreClock = 16'000'000U;

extern "C" {

int \_\_low\_level\_init(void)

{

//Switch on external 16 MHz oscillator

RCC::CR::HSION::On::Set();

while (RCC::CR::HSIRDY::NotReady::IsSet())

{

}

//Switch system clock on external oscillator

RCC::CFGR::SW::Hsi::Set();

while (!RCC::CFGR::SWS::Hsi::IsSet())

{

}

//Switch on clock on PortA and PortC

RCC::AHB1ENRPack<

RCC::AHB1ENR::GPIOCEN::Enable,

RCC::AHB1ENR::GPIOAEN::Enable,

RCC::AHB1ENR::GPIOBEN::Enable

>::Set();

RCC::APB2ENR::SYSCFGEN::Enable::Set();

//LED1 on PortA.5, set PortA.5 as output

GPIOA::MODER::MODER5::Output::Set();

/\* LED2 on PortC.9, LED3 on PortC.8, LED4 on PortC.5 so set PortC.5,8,9 as output \*/

GPIOC::MODERPack<

GPIOC::MODER::MODER5::Output,

GPIOC::MODER::MODER8::Output,

GPIOC::MODER::MODER9::Output

>::Set();

GPIOA::MODER::MODER2::Alternate::Set();

GPIOA::MODER::MODER3::Alternate::Set();

GPIOA::AFRL::AFRL2::Af7::Set(); //Tx usart2

GPIOA::AFRL::AFRL3::Af7::Set(); //Rx usart2

RCC::APB1ENR::USART2EN::Enable::Set();

USART2::CR1::OVER8::OversamplingBy16::Set();

USART2::CR1::M::Data8bits::Set();

USART2::CR1::PCE::ParityControlDisable::Set();

USART2::BRR::Write(16'000'000/(9600));

USART2::CR1::UE::Enable::Set();

//SMBus

RCC::APB1ENR::I2C1EN::Enable::Set();

I2C1::CR1::SMBUS::SmBusMode::Set();

I2C1::CR1::SMBTYPE::Device::Set();

I2C1::CCR::F\_S::StandartMode::Set();

I2C1::CCR::CCR::Set(0xFA);

I2C1::CR2::FREQ::Set(0x10);

I2C1::TRISE::Write(0x11);

I2C1::CR1::PE::Enable::Set();

I2C1::OAR1::ADDMODE::Bits7::Set();

GPIOB::MODER::MODER8::Alternate::Set(); //Alternate moder 8

GPIOB::MODER::MODER9::Alternate::Set(); //Alternate moder 9

GPIOB::AFRH::AFRH8::Af4::Set(); //scl

GPIOB::AFRH::AFRH9::Af4::Set(); //sda

GPIOB::OTYPER::OT8::OutputOpenDrain::Set();

GPIOB::OTYPER::OT9::OutputOpenDrain::Set();

GPIOB::PUPDR::PUPDR8::PullUp::Set();

GPIOB::PUPDR::PUPDR9::PullUp::Set();

return 1;

}

}

Filter filter(9.0F,0.1F);

Temperature temperature;

TemperatureTask temperatureTask(100ms,temperature,filter);

MailTask mailTask1(3000ms, &temperatureTask);

// MailTask mailTask2(testmes2, 110ms);

// OsWrapper::Mutex USARTMutex;

int main()

{

USART2::CR1::TE::Enable::Set();

using namespace OsWrapper;

Rtos::CreateThread(temperatureTask, "TemperatureTask", ThreadPriority::highest);

Rtos::CreateThread(mailTask1, "MailTask", ThreadPriority::normal);

Rtos::Start();

return 0;

}

1. interfacetasks.CPP

#include "interfacetasks.h"

InterfaceTasks::InterfaceTasks()

{

}

1. interfacetasks.H

# pragma once

typedef struct

{

float Value;

const char\* Unit;

} pr;

class InterfaceTasks

{

//pr P={0};

public:

virtual pr GetPair() const =0;

InterfaceTasks();

};

1. interfaceunits.CPP

#include "interfaceunits.h"

1. interfaceunits.H

# pragma once

class InterfaceUnits

{

public:

virtual const char\* GetUnits() const;

virtual float Convert(float temperature) const;

};

1. mailtask.CPP

#include "mailtask.h"

#include "messagetransmit.h"

#include "interfacetasks.h"

#include "translate.h"

#include <array>

#include <string>

MailTask::MailTask(std::chrono::milliseconds delay, InterfaceTasks\* value) : delayTask(delay), ValueUnit(value)

{

}

void MailTask::Execute()

{

Translate translate;

for(;;)

{

Pair = ValueUnit -> GetPair();

messageToSend = translate.ToString(Pair.Value, Pair.Unit);

messageTransmit.Send(messageToSend);

Sleep(delayTask);

}

1. mailtask.H

# pragma once

#include "messagetransmit.h"

#include "thread.hpp"

#include "interfacetasks.h"

#include <string> // for std::string

#include <chrono> // for std::chrono::miliseconds

class MailTask : public OsWrapper::Thread<200>

{

public:

MailTask(std::chrono::milliseconds delay, InterfaceTasks\* value );

void Execute() override;

private:

std::string messageToSend;

std::chrono::milliseconds delayTask;

InterfaceTasks\* ValueUnit = nullptr;

MessageTransmit messageTransmit;

//TemperatureTask& temperatureTask;

pr Pair;

};

1. messagetransmit.CPP

#include "messagetransmit.h"

#include "usart2registers.hpp" //for USART2

#include <array>

#include <string>

#include "rtos.hpp" // for Rtos

#include "mutex.hpp" // for Mutex

extern OsWrapper::Mutex USARTMutex;

void MessageTransmit::Send(std::string& message) // Ïåðåäà÷à ññûëêè íà ñîîáùåíèå

{

while(\*ptr != 0)

{

USART2::DR::Write(\*ptr);

while(USART2::SR::TXE::DataRegisterNotEmpty::IsSet())

{

}

ptr++;

}

if(\*ptr == 0)

{

ptr= message.c\_str();

}

}

1. messagetransmit.H

pragma once

#include <array>

#include <string>

//template <TUSART>

class MessageTransmit

{

public:

void Send(std::string& message);

MessageTransmit() = default;

private:

std::string& MessageToSend;

const char \*ptr;

};

1. temperaturetask.CPP

#include "temperaturetask.h"

#include "temperature.h"

#include "filter.h"

#include "interfacetasks.h"

#include "kelvin.h" // for Kelvin

#include "celsius.h" // for Kelvin

TemperatureTask::TemperatureTask(std::chrono::milliseconds delay,Temperature& temperature, Filter& filter): delayTask(delay), temperature(temperature), filter(filter), InterfaceTasks()

{

temperature.SetCurrentUnits(&celsius);

}

void TemperatureTask::Execute()

{

for(;;)

{

tempfromsens = temperature.TakeTempValue();

filteredvalue = filter.FiltredValue(tempfromsens);

Pair.Value = filteredvalue;

Pair.Unit = temperature.TakeTempUnits();

Sleep(delayTask);

}

}

pr TemperatureTask::GetPair() const

{

return Pair;

}

1. temperaturetask.H

# pragma once

#include "temperature.h"

#include "filter.h"

#include "interfacetasks.h"

class TemperatureTask: public OsWrapper::Thread<512>, public InterfaceTasks

{

public:

TemperatureTask(std::chrono::milliseconds delay, Temperature& temperature, Filter& filter);

void Execute() override;

pr GetPair() const override;

private:

pr Pair;

Temperature& temperature;

float filteredvalue;

float tempfromsens;

Filter& filter;

std::chrono::milliseconds delayTask;

};

1. celsius.CPP

#include "celsius.h"

const char\* Celsius::GetUnits() const

{

return (char\*)unitsName;

}

float Celsius::Convert( float temperature) const

{

return temperature;

}

1. celsius.H

# pragma once

#include "interfaceunits.h"

class Celsius: public InterfaceUnits

{

public:

const char\* GetUnits() const override;

float Convert(float temperature) const override;

private:

const char\* unitsName = "C";

};

inline Celsius celsius;

1. kelvin.CPP

#include "kelvin.h"

const char\* Kelvin::GetUnits() const

{

return (char\*)unitsName;

}

float Kelvin::Convert( float temperature) const

{

temperature = temperature+273.15;

return temperature;

}

1. kelvin.H

# pragma once

#include "interfaceunits.h"

class Kelvin: public InterfaceUnits

{

public:

const char\* GetUnits() const override;

float Convert(float temperature) const override;

private:

const char\* unitsName = "K";

};

inline Kelvin kelvin;

1. temperature.CPP

#include "temperature.h"

#include "interfaceunits.h"

#include "smbusdriver.h"

#include "kelvin.h"

const char\* Temperature::TakeTempUnits() const

{

const char\* result = &NoUnits[0];

if(currentunits != nullptr)

{

result = currentunits->GetUnits();

}

return result;

}

float Temperature::TakeTempValue()

{

temperature = smbus.Read(0x07);

return currentunits -> Convert(temperature);

}

1. temperature.H

# pragma once

#include "interfaceunits.h"

#include "smbusdriver.h"

class Temperature

{

public:

const char\* TakeTempUnits() const;

float TakeTempValue();

void SetCurrentUnits(InterfaceUnits\* units)

{

currentunits = units;

}

private:

SMBusDriver smbus;

InterfaceUnits\* currentunits = nullptr;

float temperature;

const char NoUnits[8] = "NoUnits";

};

1. smbusdriver.CPP

#include "smbusdriver.h"

#include "i2c1registers.hpp" //for I2C1

1. smbusdriver.H

# pragma once

#include "thread.hpp"

#include "smbusdriver.h"

#include "i2c1registers.hpp" //for I2C1

class SMBusDriver

{

public:

float Read(uint8\_t address)

{

I2C1::CR1::START::Enable::Set();

while(I2C1::SR1::SB::Value0::IsSet())

{

}

I2C1::SR1::Get();

I2C1::CR1::ACK::Acknowledge::Set() ;

I2C1::DR::Write(SlaveAddress);

while(I2C1::SR1::ADDR::Value0::IsSet())

{

}

I2C1::SR1::Get();

I2C1::SR2::Get();

// while(I2C1::SR1::TxE::Value0::IsSet())

// {

// }

I2C1::DR::Write(address);

while(I2C1::SR1::TxE::Value0::IsSet())

{

}

I2C1::CR1::START::Enable::Set();

while(I2C1::SR1::SB::Value0::IsSet())

{

}

I2C1::DR::Write(AddressToSend);

while(I2C1::SR1::ADDR::Value0::IsSet())

{

}

I2C1::CR1::ACK::NoAcknowledge::Set() ;

I2C1::CR1::POS::NextByte::Set();

I2C1::SR1::Get();

I2C1::SR2::Get();

while (I2C1::SR1::BTF::Value0::IsSet())

{

}

I2C1::CR1::STOP::Enable::Set();

uint16\_t value = I2C1::DR::Get();

value = value + (I2C1::DR::Get()<<8);

float valueC = value\*0.02F - 273.15F;

return valueC;

}

private:

uint8\_t SlaveAddress =0x00;

uint8\_t AddressToSend =0x01;

};

1. filter.CPP

#include "filter.h"

#include <math.h>

Filter::Filter(float RC, float dt)

{

tau = 1 - exp(-dt/RC);

}

float Filter::FiltredValue(float Value)

{

FilteredValue = oldFilterValue + (Value - oldFilterValue) \* tau;

oldFilterValue = FilteredValue;

return FilteredValue;

}

1. filter.H

#pragma once

#include <chrono> // for 'ms' literal

class Filter

{

public:

float FiltredValue(float Value);

Filter(float RC, float dt);

private:

float oldFilterValue;

float tau;

float FilteredValue;

};