ДОДАТОК А
ЛІСТИНГИ ПРОГРАМНИХ КОМПОНЕНТІВ

А.1Програмний код прошивки

А.1.1 Файл CoroUtils.hpp

```
#pragma once

	#include "MetaUtils.hpp"


	#include <coroutine>
	#include <atomic>
	#include <array>
	#include <logger/logger_service.hpp>


	#ifdef _MSC_VER
	#include <coroutine>
	namespace stdcoro = std;
	#elif __GNUC__
	#include <coroutine>
	namespace stdcoro = std;
	#else
	#include <experimental/coroutine>
	namespace stdcoro = std::experimental
	#endif


	#include <etl/vector.h>
	#include <etl/queue_spsc_atomic.h>


	struct Promise
	{
		auto initial_suspend()noexcept
		{
			return stdcoro::suspend_never{};
		}
		auto final_suspend()noexcept
		{
			return stdcoro::suspend_never{};
		}


		void get_return_object()
```

```
	{
	}
	void return_void()
	{
	}


	void unhandled_exception()
	{
		while (1)
		{


		}
	}


};


template <typename... Args>
struct stdcoro::coroutine_traits<void, Args ...>
{
	using promise_type = Promise;
};




namespace CoroUtils
{




struct VoidTask
{
	struct task_promise;
	using promise_type = task_promise;


	struct task_promise
	{
		task_promise()noexcept = default;
		void return_void()noexcept {}
		void unhandled_exception() noexcept
		{
			while (true)
			{


			}
		}
		VoidTask get_return_object()noexcept
		{
			return
VoidTask{ stdcoro::coroutine_handle<task_promise>::from_promise(*thi
s) };
```

```
            }


            auto initial_suspend()noexcept { return
stdcoro::suspend_always{}; }


            struct final_awaitable
            {
                  bool await_ready() noexcept
                  {
                        return false;
                  }
                  template<typename TPromise>
                  void
await_suspend(stdcoro::coroutine_handle<TPromise> coroutine)
noexcept
                  {
                        task_promise& promise = coroutine.promise();
                        if (promise.m_continuation)
                        {
                              promise.m_continuation.resume();
                        }
                  }


                  void await_resume()noexcept {}
            };


            void set_continuation(stdcoro::coroutine_handle<>
continuation)
            {
                  m_continuation = continuation;
            }


            auto final_suspend()noexcept
            {
                  return final_awaitable{};
            }


            stdcoro::coroutine_handle<> m_continuation;
      };


      struct task_awaitable
      {
            task_awaitable(stdcoro::coroutine_handle<promise_type>
coroutine)
                  : m_coroutine{ coroutine }
            {
            }
            bool await_ready()const noexcept
            {
                  return !m_coroutine || m_coroutine.done();
            }
```

```
            void await_suspend(stdcoro::coroutine_handle<>
awaitingRoutine)
            {
                  m_coroutine.resume();

      m_coroutine.promise().set_continuation(awaitingRoutine);
            }


            void await_resume()
            {
            }


            stdcoro::coroutine_handle<promise_type> m_coroutine;
      };


      VoidTask(stdcoro::coroutine_handle<task_promise>
suspendedCoroutine)
            : m_coroutine{ suspendedCoroutine }
      {
      }


      ~VoidTask()
      {
            if (m_coroutine)
                  m_coroutine.destroy();
      }


      bool await_ready() const noexcept
      {
            return !m_coroutine || m_coroutine.done();
      }


      auto operator co_await() noexcept
      {
            return task_awaitable{ m_coroutine };
      }


      void await_resume()
      {
      }


      stdcoro::coroutine_handle<promise_type> m_coroutine;
};


template<typename ... Tasks>
struct WhenAllSequence
{
      std::tuple<Tasks...> m_taskList;
```

```
	explicit WhenAllSequence(Tasks&& ... tasks) noexcept
		: m_taskList(std::move(tasks)...)
	{
	}


	explicit WhenAllSequence(std::tuple<Tasks...>&& tasks)noexcept
		: m_taskList(std::move(tasks))
	{
	}


	bool await_ready() const noexcept { return false; }


	void await_suspend(stdcoro::coroutine_handle<> handle)
noexcept
	{
		co_await
launchAll(std::make_integer_sequence<std::size_t, sizeof...
(Tasks)>{});
		handle.resume();
	}


	template<std::size_t... Indexes>
	VoidTask launchAll(std::integer_sequence<std::size_t,
Indexes...>)
	{
		(co_await std::get<Indexes>(m_taskList),...);
	}


	void await_resume() noexcept
	{
	}
};




template<typename ... Args>
auto when_all_sequence(Args&& ... args) noexcept
{
	return WhenAllSequence{ std::make_tuple(std::move(args)...) };
}




struct CoroQueueMainLoop
{
	static CoroQueueMainLoop& GetInstance()
	{
		static CoroQueueMainLoop instance{};
		return instance;
	}
```

```
	void pushToLater(std::coroutine_handle<> coroHandle)
	{
		executionQueue.push(coroHandle);
	}


	void processQueue()
	{
		std::coroutine_handle<> handle;
		while( executionQueue.pop(handle))
		{
			if(!handle.done())
				handle.resume();
		}
	}


	template<typename Type, const size_t StorageSize = 3>
	using TQueueStorageType = etl::queue_spsc_atomic<Type,
StorageSize,etl::memory_model::MEMORY_MODEL_SMALL>;

	TQueueStorageType<std::coroutine_handle<>> executionQueue;
};


struct Event
{
	Event(bool isSet = false)
		: m_isSet{ isSet }
	{
	}


	bool await_ready()noexcept
	{
		return m_isSet;
	}


	void await_suspend(stdcoro::coroutine_handle<> handle)
	{
		m_continuation = handle;
	}


	void await_resume()
	{
	}


	void set(bool toMainThread = false)
	{
		m_isSet.store(true);
		if (m_continuation && !m_continuation.done())
		{
```

```
                        if (toMainThread){

            CoroQueueMainLoop::GetInstance().pushToLater(m_continuation);
                        }
                        else{
                              m_continuation.resume();
                        }
                  }

            }


            bool isSet() const noexcept
            {
                  return m_isSet;
            }


            std::atomic_bool m_isSet;
            stdcoro::coroutine_handle<> m_continuation;
      };
```

A.1.3 Файл spi_wrapper_async_templated.hpp

```
#pragma once



      #include <memory>
      #include <atomic>
      #include <coroutine>
      #include <optional>
      #include <cstdint>


      #include <etl/vector.h>
      #include <utils/CoroUtils.hpp>
      #include <utils/Noncopyable.hpp>


      namespace Interface::SpiTemplated
      {




      template<typename SpiBackendImpl>
```

```
class SpiBus
    : private Utils::noncopyable
{


public:


    SpiBus()noexcept
    {
        m_backendImpl.setTransactionCompletedHandler(
            [this] { transmitCompleted(); }
        );
    }
    ~SpiBus() = default;


public:


    static constexpr std::uint16_t DmaBufferSize = 255;
    using DmaBufferType = etl::vector<std::uint8_t, DmaBufferSize>;


    constexpr std::uint16_t getDmaBufferSize() noexcept
    {
        return DmaBufferSize;
    }


    constexpr DmaBufferType& getDmaBufferTransmit() noexcept
    {
        return DmaArrayTransmit;
    }


    void transmitBuffer(
        const std::uint8_t* _pBuffer
        , std::uint16_t _pBufferSize
        , void* _pUserData
        , bool restoreInSpiCtx
    )noexcept
    {
        m_coroHandle =
```

```
std::coroutine_handle<>::from_address(_pUserData);


        const size_t TransferBufferSize = _pBufferSize;
        const size_t FullDmaTransactionsCount =
            TransferBufferSize / DmaBufferSize;
        const size_t ChunkedTransactionsBufSize =
            TransferBufferSize % DmaBufferSize;
        const bool ComputeChunkOffsetWithDma =
FullDmaTransactionsCount >= 1;


        TransactionContext newContext
        {
                .restoreInSpiCtx = restoreInSpiCtx
            ,   .computeChunkOffsetWithDma = ComputeChunkOffsetWithDma
            ,   .pDataToTransmit = _pBuffer
            ,   .fullDmaTransactionsCount = FullDmaTransactionsCount
            ,   .chunkedTransactionBufSize =
ChunkedTransactionsBufSize
            ,   .completedTransactionsCount = 0
        };


        m_transmitContext = std::move(newContext);


        if (FullDmaTransactionsCount)
        {
            --m_transmitContext.fullDmaTransactionsCount;
            m_backendImpl.sendChunk(
                _pBuffer
                , DmaBufferSize
            );
        }
        else {
            m_transmitContext.chunkedTransactionBufSize = 0;
            m_backendImpl.sendChunk(
                _pBuffer
                , _pBufferSize
            );
        }
    }
public:
```

```
    SpiBackendImpl& getBackendImpl()noexcept
    {
        return m_backendImpl;
    }


    const SpiBackendImpl& getBackendImpl()const noexcept
    {
        return m_backendImpl;
    }


private:


    void transmitCompleted()noexcept
    {
        const bool isAllDmaTransactionsProceeded =
            m_transmitContext.fullDmaTransactionsCount == 0;
        const bool isAllChunckedTransactionsCompleted =
            m_transmitContext.chunkedTransactionBufSize == 0;


        if (!isAllDmaTransactionsProceeded)
        {
            --m_transmitContext.fullDmaTransactionsCount;


            m_backendImpl.sendChunk(
                m_transmitContext.pDataToTransmit +
DmaBufferSize*getTransitionOffset()
                , DmaBufferSize
            );
        }
        else if (!isAllChunckedTransactionsCompleted)
        {
            const size_t transmissionOffset =
m_transmitContext.computeChunkOffsetWithDma
                ? DmaBufferSize
                * getTransitionOffset() : 0;
```

```
                const size_t TransmitBufferSize =
                    m_transmitContext.chunkedTransactionBufSize;
                m_transmitContext.chunkedTransactionBufSize = 0;


                m_backendImpl.sendChunk(
                    m_transmitContext.pDataToTransmit + transmissionOffset
                    , TransmitBufferSize
                );
            }
            else {
                if (m_transmitContext.restoreInSpiCtx) {
                    m_coroHandle.resume();
                }
                else {

CoroUtils::CoroQueueMainLoop::GetInstance().pushToLater(m_coroHandle);
                }
            }
        }


    private:
        std::size_t getTransitionOffset() noexcept
        {
            return ++m_transmitContext.completedTransactionsCount;
        }


    private:
        SpiBackendImpl m_backendImpl;
        std::coroutine_handle<> m_coroHandle;


        struct TransactionContext
        {
            bool restoreInSpiCtx = false;
            bool computeChunkOffsetWithDma = false;
            const std::uint8_t* pDataToTransmit = nullptr;
            size_t fullDmaTransactionsCount = 0;
            size_t chunkedTransactionBufSize = 0;
            size_t completedTransactionsCount = 0;
        };
```

```
        TransactionContext m_transmitContext;
        DmaBufferType DmaArrayTransmit;
        DmaBufferType DmaArrayReceive;
    };


    }
```

A.1.5 Класс GC9A01Compact

```
template<typename TSpiBusInstance, std::uint16_t Width, std::uint16_t Height
>
                class GC9A01Compact
                    : public
            BaseSpiDisplayCoroutine<GC9A01Compact<TSpiBusInstance,Width,He
            ight>, TSpiBusInstance,Width,Height >
                {
                    using TBaseSpiDisplay =
                        BaseSpiDisplayCoroutine<
                            GC9A01Compact<
                                TSpiBusInstance,
                                Width,
                                Height
                            >,
                            TSpiBusInstance,
                            Width,
                            Height
                       >;
                public:


                    void turnOn()noexcept{}


                    void turnOff()noexcept{}


                    void fillRectangle(
                        std::uint16_t _x
                        , std::uint16_t _y
                        , std::uint16_t _width
                        , std::uint16_t _height
                        , TBaseSpiDisplay::TColor* _colorToFill
```

```
        ) noexcept
        {
            for( size_t i{}; i< 2400; ++i ){
                _colorToFill[i] = 0xF800;
            }


            const std::uint16_t DisplayHeight =
TBaseSpiDisplay::getHeight();
            const std::uint16_t DisplayWidth =
TBaseSpiDisplay::getWidth();


            const bool isCoordsValid{ !((_x >= DisplayWidth)
|| (_y >= DisplayHeight)) };
            if (isCoordsValid)
            {
                if (_width >= DisplayWidth) _width =
DisplayWidth - _x;
                if (_height >= DisplayHeight) _height =
DisplayHeight - _y;


                const size_t BytesSizeX = (_width - _x + 1);
                const size_t BytesSizeY = (_height - _y + 1);
                const size_t BytesSquare = BytesSizeX *
BytesSizeY;
                const size_t TransferBufferSize = (BytesSquare
* sizeof(typename TBaseSpiDisplay::TColor));


                co_await
TBaseSpiDisplay::m_displayInitialized;


                //co_await setAddrWindow(_x, _y, _width,
_height);


                std::uint16_t width = _width - _x;
                std::uint16_t height = _height - _y;


                std::uint16_t correctedX = _x;
                std::uint16_t correctedY = _y;
```

```
                uint32_t xa = ((uint32_t)correctedX << 16) |
(correctedX + width); // (_x+_width-1);
                int32_t ya = ((uint32_t)correctedY << 16) |
(correctedY + height); //(_y+_height-1);


                using TCommandsArray =
std::array<std::uint8_t,5>;
                static TCommandsArray xAxisCommand{};
                xAxisCommand[0] =
static_cast<std::uint8_t>( 0x2a );
                xAxisCommand[1] =
static_cast<std::uint8_t>( xa >> 24 );
                xAxisCommand[2] =
static_cast<std::uint8_t>( xa >> 16 );
                xAxisCommand[3] =
static_cast<std::uint8_t>( xa >> 8 );
                xAxisCommand[4] =
static_cast<std::uint8_t>( xa );


                static TCommandsArray yAxisCommand{};
                yAxisCommand[0] =
static_cast<std::uint8_t>( 0x2b );
                yAxisCommand[1] =
static_cast<std::uint8_t>( ya >> 24 );
                yAxisCommand[2] =
static_cast<std::uint8_t>( ya >> 16 );
                yAxisCommand[3] =
static_cast<std::uint8_t>( ya >> 8 );
                yAxisCommand[4] =
static_cast<std::uint8_t>( ya );


                constexpr std::uint8_t CmdSize = 1;
                constexpr std::uint8_t ArgSize = 4;


                co_await
TBaseSpiDisplay::sendCommandImplFast(xAxisCommand.data());
                co_await
TBaseSpiDisplay::sendChunkFast(xAxisCommand.data()+CmdSize,
ArgSize);
```

```
                co_await
TBaseSpiDisplay::sendCommandImplFast(yAxisCommand.data());
                co_await
TBaseSpiDisplay::sendChunkFast(yAxisCommand.data()+CmdSize,
ArgSize);


                static std::uint8_t RamWriteCmd{0x2C};


                co_await
TBaseSpiDisplay::sendCommandImplFast(&RamWriteCmd);
//LCD_WriteCMD(GRAMWR);


                TBaseSpiDisplay::setDcPin();
                co_await
TBaseSpiDisplay::sendChunkFast(reinterpret_cast<const
std::uint8_t*>(_colorToFill),     TransferBufferSize);
                TBaseSpiDisplay::resetDcPin();



TBaseSpiDisplay::onRectArreaFilled.emitLater();
            }
        }


        void initialize()noexcept
        {
            initDisplay();
        }


    public:


        void initDisplay() noexcept
        {
            TBaseSpiDisplay::resetResetPin();
            Delay::waitFor(100);
            TBaseSpiDisplay::setResetPin();
            //static std::array Test =
std::array<std::uint8_t,
```

```
CommandsSize>{0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0
A};
            //co_await TBaseSpiDisplay::sendChunk(Test.data(),
Test.size());
            size_t CommandCount =
InitializationCommands::CommandsTransactionsCount;
            const std::uint8_t* pBuffer =
InitializationCommands::Commands.data();
            while (CommandCount--)
            {
                const std::uint8_t* Command = pBuffer++;
                const std::uint8_t NumArgs = *pBuffer++;
                const std::uint8_t Delay = *pBuffer++;


                co_await
TBaseSpiDisplay::sendCommandImpl(Command);
                if (NumArgs) {
                    co_await
TBaseSpiDisplay::sendChunk(pBuffer++, NumArgs);
                    pBuffer += (NumArgs - 1);
                }
                if (Delay)
                    Delay::waitFor(Delay);
            }
            TBaseSpiDisplay::m_displayInitialized.set();
        }


        CoroUtils::VoidTask setAddrWindow(
            std::uint16_t _x
            , std::uint16_t _y
            , std::uint16_t _width
            , std::uint16_t _height
        ) noexcept
        {
            // TODO be careful here;
            std::uint16_t width = _width - _x;
            std::uint16_t height = _height - _y;


            std::uint16_t correctedX = _x;
            std::uint16_t correctedY = _y;
```

```
                uint32_t xa = ((uint32_t)correctedX << 16) |
    (correctedX + width);// (_x+_width-1);
                int32_t ya = ((uint32_t)correctedY << 16) |
    (correctedY + height); //(_y+_height-1);


                static std::array xAxisCommand =
                    std::array{
                            static_cast<std::uint8_t>(0x2a)
                        ,   static_cast<std::uint8_t>(xa >> 24)
                        ,   static_cast<std::uint8_t>(xa >> 16)
                        ,   static_cast<std::uint8_t>(xa >> 8)
                        ,   static_cast<std::uint8_t>(xa)
                };
                static std::array yAxisCommand =
                    std::array{
                            static_cast<std::uint8_t>(0x2b)
                        ,   static_cast<std::uint8_t>(ya >> 24)
                        ,   static_cast<std::uint8_t>(ya >> 16)
                        ,   static_cast<std::uint8_t>(ya >> 8)
                        ,   static_cast<std::uint8_t>(ya)
                };


                co_await CoroUtils::when_all_sequence(

    TBaseSpiDisplay::sendCommandFast(xAxisCommand.data(),
    xAxisCommand.size())
                    ,
    TBaseSpiDisplay::sendCommandFast(yAxisCommand.data(),
    yAxisCommand.size())
                );
            }
        };
    }; // namespace DisplayDriver
```

A.1.6 Реалізація семплу FreeRTOS

```
#include <FreeRTOS.h>
            #include <task.h>
            #include <timers.h>


            #include "SEGGER_RTT.h"
            #include "pca10040.h"
```

```
#include "boards.h"
#include "bsp.h"


#include <cstdint>


TaskHandle_t seggerLoggerTaskHandle;
TaskHandle_t ledBlinkTaskHandle;




const char FirstMessage [] = "Before suspending\n";
const char SecondMessage [] = "After restoring from suspended
state\n";







static void loggerTaskFunction (void * pvParameter)
{
    UNUSED_PARAMETER(pvParameter);
    while (true)
    {
        SEGGER_RTT_WriteString(0, FirstMessage);
        taskYIELD();
        SEGGER_RTT_WriteString(0, SecondMessage);
    }
}







static void ledBlinkTaskFunction (void * pvParameter)
{
    UNUSED_PARAMETER(pvParameter);
    while (true)
    {
        bsp_board_led_invert(0);
        taskYIELD();
```

```
        }
    }



int main()
{


    constexpr std::uint8_t Priority = 2;
    UNUSED_PARAMETER(
        xTaskCreate(
            loggerTaskFunction,
            "LogggerTask",
            configMINIMAL_STACK_SIZE + 200,nullptr, Priority,
&seggerLoggerTaskHandle)
    );
    UNUSED_PARAMETER(
        xTaskCreate(
            ledBlinkTaskFunction,
            "LogggerTask",
            configMINIMAL_STACK_SIZE + 200,nullptr, Priority,
&ledBlinkTaskHandle)
    );


    vTaskStartScheduler();
    return 0;
}
```

A.1.7 Класс SimpleTaskScheduler

```
#pragma once


    #include <utils/Noncopyable.hpp>


    #include "SimpleResumable.hpp"


    #include <coroutine>
    #include <etl/queue.h>
    #include <etl/vector.h>
```

```
namespace TaskScheduler
{

class SimpleCooperativeNonPriorityScheduler
{

public:

    void pushTask(Resumable::coro_handle taskHandle)
    {
        m_tasksStorage.push_back(taskHandle);
    }

    void runTasks()
    {
        while(true){
            size_t newTaskIndex = getTaskIndex();
            m_tasksStorage[newTaskIndex].resume();
        }
    }

    size_t getTaskIndex()
    {
        if (m_taskRunningIndex == m_tasksStorage.size())
            m_taskRunningIndex = 0;
        return m_taskRunningIndex++;
    }
private:

    template<typename Type, const size_t StorageSize = 6>
    using TStorageType = etl::vector<Type, StorageSize>;

    using TTaskStorage = TStorageType<Resumable::coro_handle>;
```

```
        TTaskStorage m_tasksStorage;
        size_t m_taskRunningIndex = 0;
    };



};  // namespace TaskScheduler
```

A.1.8 Файл конфігурації FreeRTOS

```
/*
 * FreeRTOS Kernel V10.0.0
 * Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights
Reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining
a copy of
 * this software and associated documentation files (the "Software"), to
deal in
 * the Software without restriction, including without limitation the
rights to
 * use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of
 * the Software, and to permit persons to whom the Software is furnished
to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
included in all
 * copies or substantial portions of the Software. If you wish to use our
Amazon
 * FreeRTOS name, please do so in a fair use way that does not cause
confusion.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS
 * FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR
 * COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
```

```
WHETHER
 * IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
 *
 * http://www.FreeRTOS.org
 * http://aws.amazon.com/freertos
 *
 * 1 tab == 4 spaces!
 */




#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H


#ifdef SOFTDEVICE_PRESENT
#include "nrf_soc.h"
#endif
#include "app_util_platform.h"


/*-----------------------------------------------------------
 * Possible configurations for system timer
 */
#define FREERTOS_USE_RTC      0 /**< Use real time clock for the system
*/
#define FREERTOS_USE_SYSTICK  1 /**< Use SysTick timer for system */


/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF
THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
```

```
 *-----------------------------------------------------------*/


#define configTICK_SOURCE FREERTOS_USE_RTC


#define configUSE_PREEMPTION 0
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1
#define configUSE_TICKLESS_IDLE 1
#define configUSE_TICKLESS_IDLE_SIMPLE_DEBUG
1 /* See into vPortSuppressTicksAndSleep source code for explanation */
#define configCPU_CLOCK_HZ
( SystemCoreClock )
#define configTICK_RATE_HZ
1024
#define configMAX_PRIORITIES
( 3 )
#define configMINIMAL_STACK_SIZE
( 60 )
#define configTOTAL_HEAP_SIZE
( 4096 )
#define configMAX_TASK_NAME_LEN
( 4 )
#define configUSE_16_BIT_TICKS
0
#define configIDLE_SHOULD_YIELD
1
#define configUSE_MUTEXES
1
#define configUSE_RECURSIVE_MUTEXES
1
#define configUSE_COUNTING_SEMAPHORES
1
#define configUSE_ALTERNATIVE_API
0    /* Deprecated! */
#define configQUEUE_REGISTRY_SIZE
2
#define configUSE_QUEUE_SETS
0
#define configUSE_TIME_SLICING
0
#define configUSE_NEWLIB_REENTRANT
0
#define configENABLE_BACKWARD_COMPATIBILITY
1
```

```
/* Hook function related definitions. */
#define configUSE_IDLE_HOOK
0
#define configUSE_TICK_HOOK
0
#define configCHECK_FOR_STACK_OVERFLOW
0
#define configUSE_MALLOC_FAILED_HOOK
0


/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS
0
#define configUSE_TRACE_FACILITY
0
#define configUSE_STATS_FORMATTING_FUNCTIONS
0


/* Co-routine definitions. */
#define configUSE_CO_ROUTINES
0
#define configMAX_CO_ROUTINE_PRIORITIES
( 2 )


/* Software timer definitions. */
#define configUSE_TIMERS 1
#define configTIMER_TASK_PRIORITY
( 2 )
#define configTIMER_QUEUE_LENGTH
32
#define configTIMER_TASK_STACK_DEPTH
( 80 )


/* Tickless Idle configuration. */
#define configEXPECTED_IDLE_TIME_BEFORE_SLEEP
2


/* Tickless idle/low power functionality. */
```

```
/* Define to trap errors during development. */
#if defined(DEBUG_NRF) || defined(DEBUG_NRF_USER)
#define configASSERT( x )
ASSERT(x)
#endif


/* FreeRTOS MPU specific definitions. */
#define configINCLUDE_APPLICATION_DEFINED_PRIVILEGED_FUNCTIONS
1


/* Optional functions - most linkers will remove unused functions anyway.
*/
#define INCLUDE_vTaskPrioritySet
1
#define INCLUDE_uxTaskPriorityGet
1
#define INCLUDE_vTaskDelete
1
#define INCLUDE_vTaskSuspend
1
#define INCLUDE_xResumeFromISR
1
#define INCLUDE_vTaskDelayUntil
1
#define INCLUDE_vTaskDelay
1
#define INCLUDE_xTaskGetSchedulerState
1
#define INCLUDE_xTaskGetCurrentTaskHandle
1
#define INCLUDE_uxTaskGetStackHighWaterMark
1
#define INCLUDE_xTaskGetIdleTaskHandle
1
#define INCLUDE_xTimerGetTimerDaemonTaskHandle
1
#define INCLUDE_pcTaskGetTaskName
1
#define INCLUDE_eTaskGetState
1
```

```
#define INCLUDE_xEventGroupSetBitFromISR
1
#define INCLUDE_xTimerPendFunctionCall
1


/* The lowest interrupt priority that can be used in a call to a "set
priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY         0xf


/* The highest interrupt priority that can be used by any interrupt
service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO
NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A
HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY    _PRIO_APP_HIGH




/* Interrupt priorities used by the kernel port layer itself.  These are
generic
to all Cortex-M ports, and do not rely on any particular library
functions. */
#define configKERNEL_INTERRUPT_PRIORITY
configLIBRARY_LOWEST_INTERRUPT_PRIORITY
/* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
#define configMAX_SYSCALL_INTERRUPT_PRIORITY
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY


/* Definitions that map the FreeRTOS port interrupt handlers to their
CMSIS
standard names - or at least those used in the unmodified vector table.
*/


#define vPortSVCHandler
SVC_Handler
#define xPortPendSVHandler
```

PendSV_Handler

```
/*-----------------------------------------------------------
 * Settings that are generated automatically
 * basing on the settings above
 */
#if (configTICK_SOURCE == FREERTOS_USE_SYSTICK)
    // do not define configSYSTICK_CLOCK_HZ for SysTick to be configured
automatically
    // to CPU clock source
    #define xPortSysTickHandler     SysTick_Handler
#elif (configTICK_SOURCE == FREERTOS_USE_RTC)
    #define configSYSTICK_CLOCK_HZ  ( 32768UL )
    #define xPortSysTickHandler     RTC1_IRQHandler
#else
    #error  Unsupported configTICK_SOURCE value
#endif


/* Code below should be only used by the compiler, and not the assembler.
*/
#if !(defined(__ASSEMBLY__) || defined(__ASSEMBLER__))
    #include "nrf.h"
    #include "nrf_assert.h"


    /* This part of definitions may be problematic in assembly - it uses
definitions from files that are not assembly compatible. */
    /* Cortex-M specific definitions. */
    #ifdef __NVIC_PRIO_BITS
        /* __BVIC_PRIO_BITS will be specified when CMSIS is being used.
*/
        #define configPRIO_BITS             __NVIC_PRIO_BITS
    #else
        #error "This port requires __NVIC_PRIO_BITS to be defined"
    #endif


    /* Access to current system core clock is required only if we are
ticking the system by systimer */
    #if (configTICK_SOURCE == FREERTOS_USE_SYSTICK)
        #include <stdint.h>
```

```
        extern uint32_t SystemCoreClock;
    #endif
#endif /* !assembler */


/** Implementation note:  Use this with caution and set this to 1 ONLY
for debugging
 * ---------------------------------------------------------
     * Set the value of configUSE_DISABLE_TICK_AUTO_CORRECTION_DEBUG to
below for enabling or disabling RTOS tick auto correction:
     * 0. This is default. If the RTC tick interrupt is masked for more
than 1 tick by higher priority interrupts, then most likely
     *    one or more RTC ticks are lost. The tick interrupt inside RTOS
will detect this and make a correction needed. This is needed
     *    for the RTOS internal timers to be more accurate.
     * 1. The auto correction for RTOS tick is disabled even though few
RTC tick interrupts were lost. This feature is desirable when debugging
     *    the RTOS application and stepping though the code. After
stepping when the application is continued in debug mode, the auto-
corrections of
     *    RTOS tick might cause asserts. Setting
configUSE_DISABLE_TICK_AUTO_CORRECTION_DEBUG to 1 will make RTC and RTOS
go out of sync but could be
     *    convenient for debugging.
     */
#define configUSE_DISABLE_TICK_AUTO_CORRECTION_DEBUG     0


#endif /* FREERTOS_CONFIG_H */
```

A.1.9 Реалізація SpiDriverTest

```
#pragma once
        #include <gtest/gtest.h>


        #include <spi/spi_wrapper_async_templated.hpp>
        #include <backends/spi_backend_desktop.hpp>


        #include "spi_fake_backend.hpp"


        #include <vector>
        #include <string>
        #include <tuple>
```

```
class SpiDriverTest
      : public ::testing::Test
    , public
::testing::WithParamInterface<std::tuple<std::uint16_t,std::string_view
>>
{




public:


      using TTestDriver =
Interface::SpiTemplated::SpiBus<Testing::Spi::SpiBusBackendStub>;


protected:


      void SetUp()override
      {
      }




    struct StubAwaiter
    {
        bool restoreInSpiCtx = false;
        const std::uint8_t* pTransmitBuffer;
        SpiDriverTest* pThis;
        std::uint16_t bufferSize;


        bool await_ready() const noexcept
        {
            const bool isAwaitReady = pTransmitBuffer == nullptr ||
bufferSize == 0;
            return isAwaitReady;
        }
        void await_resume() const noexcept
        {
```

```
            }
            void await_suspend(std::coroutine_handle<> thisCoroutine) const
            {
                pThis->testSpiDriver.transmitBuffer(
                    pTransmitBuffer
                    , bufferSize
                    , thisCoroutine.address()
                    , restoreInSpiCtx
                );
            }
        };




        auto sendChunk(
            const std::uint8_t* _pBuffer
            , std::size_t _bufferSize
        )noexcept
        {
            return StubAwaiter
            {
                    .restoreInSpiCtx = true
                ,   .pTransmitBuffer = _pBuffer
                ,   .pThis = this
                ,   .bufferSize = static_cast<std::uint16_t>(_bufferSize)
            };
        }


        using TDataStream = std::vector<std::byte>;
        TDataStream TransactionsToDataStream()
        {
            return testSpiDriver.getBackendImpl().getTransmittedData();
        }


    protected:
          TTestDriver testSpiDriver;
    };
```

A.1.10 Реалізація SpiBusBackendStub

```
#pragma once
```

```
#include <vector>
#include <functional>
#include <queue>
#include <span>


namespace Testing::Spi
{


class SpiBusBackendStub
{


public:


    void sendChunk(
        const std::uint8_t* _pBuffer
        , const size_t _bufferSize
    )noexcept
    {
        BusTransactions.emplace_back(_pBuffer, _bufferSize);
        m_completedTransaction();
    }


public:


    using TTransactionCompletedHandler = std::function<void()>;
    void setTransactionCompletedHandler(TTransactionCompletedHandler&&
_handler)
    {
        m_completedTransaction = std::move(_handler);
    }


    size_t getTransactionsSize()const
    {
        return BusTransactions.size();
    }
```

```
        using TDataStream = std::vector<std::byte>;
        TDataStream getTransmittedData()const
        {
            TDataStream stream;
            std::for_each(
                BusTransactions.cbegin(),
                BusTransactions.cend(),
                [&stream](const auto& _transaction)
                {
                    const auto& [pArray, blockSize] = _transaction;
                    auto arraySpan = std::span{ pArray,blockSize };
                    std::transform(
                        arraySpan.begin(),
                        arraySpan.end(),
                        std::back_inserter(stream),
                        [](std::uint8_t _dataByte) {
                            return std::byte{ _dataByte };
                        }
                    );
                }
            );
            return stream;
        }


    protected:
        using TTransacation = std::pair<const std::uint8_t*, size_t>;
        std::vector<TTransacation> BusTransactions;


    private:


        TTransactionCompletedHandler m_completedTransaction;
    };


    } // namespace Testing::Spi
```

A.1.11 Реалізація тестів SPI-інтерфейсу

```
#include <gmock/gmock.h>
            #include <gtest/gtest.h>


            #include <random>
```

```
#include "spi_driver_fixture.hpp"



constexpr inline std::uint16_t Null = 0;
constexpr inline std::uint16_t SmallerThanSingleDmaTransaction =
SpiDriverTest::TTestDriver::DmaBufferSize - 2;
constexpr inline std::uint16_t EqualsToSingleDmaTransaction =
SpiDriverTest::TTestDriver::DmaBufferSize;
constexpr inline std::uint16_t ThreeDmaTransactions =
EqualsToSingleDmaTransaction * 3;
constexpr inline std::uint16_t SingleDmaAndChunk =
EqualsToSingleDmaTransaction + SmallerThanSingleDmaTransaction;
constexpr inline std::uint16_t ThreeDmaAndChunk =
ThreeDmaTransactions + SmallerThanSingleDmaTransaction;
constexpr inline std::uint16_t StressChunked =
ThreeDmaTransactions*10 + SmallerThanSingleDmaTransaction;


TEST_F(SpiDriverTest, ExpectAnyTransmissionOccured)
{
	static constexpr std::byte DataBlock{ 223 };
	static const TDataStream ExpectedStream{ DataBlock };


	co_await sendChunk(reinterpret_cast<const
std::uint8_t*>(ExpectedStream.data()), ExpectedStream.size());


	constexpr size_t TransactionsCount = 1;



EXPECT_EQ(testSpiDriver.getBackendImpl().getTransactionsSize(),
TransactionsCount);
	EXPECT_EQ(TransactionsToDataStream(), ExpectedStream);
}


TEST_P(SpiDriverTest,
CheckRandomSequenceWithLengthTransmissionCorrect)
{
```

```
	const std::uint16_t
TransactionLength{ std::get<0>(GetParam()) };
	TDataStream ExpectedStream{ TransactionLength };


	std::random_device randomDevice;
	std::mt19937 generator(randomDevice());
	std::uniform_int_distribution<> distribution(0x00, 0xFF);
	std::generate(
		ExpectedStream.begin(),
		ExpectedStream.end(),
		[&distribution, generator]()mutable { return
std::byte(distribution(generator)); }
	);


	co_await sendChunk(reinterpret_cast<const
std::uint8_t*>(ExpectedStream.data()), ExpectedStream.size());
	EXPECT_EQ(TransactionsToDataStream(), ExpectedStream);
}


INSTANTIATE_TEST_SUITE_P(
	SpiDriverTesting,
	SpiDriverTest,
	::testing::Values(
		std::make_tuple( Null,"Null" ),


std::make_tuple( SmallerThanSingleDmaTransaction ,"SmallerThanSi
ngleDmaTransaction"),


std::make_tuple( EqualsToSingleDmaTransaction ,"EqualsToSingleDm
aTransaction"),


std::make_tuple( ThreeDmaTransactions,"ThreeDmaTransactions"),

	std::make_tuple( SingleDmaAndChunk ,"SingleDmaAndChunk"),

	std::make_tuple( ThreeDmaAndChunk ,"ThreeDmaAndChunk"),
		std::make_tuple( StressChunked,"StressChunked")
	),
	[](const auto& testParam)
```

```
                {
                        const auto& suiteName =
            std::get<1>(testParam.param);
                        return std::string(suiteName.data());
                }
            );
```

A.1.12 Реалізація SpiBusDesktopBackend

```
#pragma once


        #include <iostream>
        #include <array>
        #include <chrono>
        #include <thread>
        #include <cstdint>
        #include <vector>
        #include <queue>
        #include <atomic>
        #include <functional>
        #include <mutex>
        #include <condition_variable>


        #include <utils/MetaUtils.hpp>


        #define USE_THREADING_ASYNC_BACKEND




        namespace Interface::SpiTemplated
        {


        class SpiBusDesktopBackend
        {


        public:
            SpiBusDesktopBackend()
                :   m_newDataArrived{ false }
                ,   m_pDataBuffer{ nullptr }
                ,   m_bufferTransmitSize{}
```

```
        ,   m_processSpiTransactions{true}
    {
#ifdef  USE_THREADING_ASYNC_BACKEND


        m_dmaThread = std::make_unique<std::thread>(
            [this]
            {
                while (m_processSpiTransactions)
                {
                    if(m_newDataArrived)
                    {
                        using namespace std::chrono_literals;
                        //std::this_thread::sleep_for(1500ms);


                        std::cout << "TRANSMIT SOME DATA:" << ' ';


                        for (size_t i{}; i < m_bufferTransmitSize; +
+i)
                        {
                            std::cout << std::hex <<
static_cast<std::int16_t>(m_pDataBuffer[i]) << ' ';
                        }
                        std::cout << std::endl;


                        m_newDataArrived.store(false);
                        m_transactionCompleted();
                    }
                }
            }
       );
#endif  //  USE_THREADING_ASYNC_BACKEND
    }


    ~SpiBusDesktopBackend()
    {
        m_processSpiTransactions.store(false);
        m_dmaThread->detach();
        m_dmaThread.reset();
    }
```

```
public:


    void sendChunk(
        const std::uint8_t* _pBuffer
        , const size_t _bufferSize
    )noexcept
    {
        m_bufferTransmitSize.store( _bufferSize );
        m_pDataBuffer.store( _pBuffer );
        m_newDataArrived.store( true, std::memory_order_release );


#ifndef USE_THREADING_ASYNC_BACKEND
        std::cout << "TRANSMIT SOME DATA:" << ' ';


        for ( size_t i{}; i < m_bufferTransmitSize; ++i )
        {
            std::cout << std::hex
                      << static_cast<std::int16_t>( m_pDataBuffer[i] )
<< ' ';
        }
        std::cout << std::endl;


        m_newDataArrived.store( false );
        m_transactionCompleted();
#endif
    }


    using TTransactionCompletedHandler = std::function<void()>;
    void setTransactionCompletedHandler(TTransactionCompletedHandler&&
_handler)
    {
        m_transactionCompleted = std::move(_handler);
    }


private:
```

```
        std::atomic_bool m_newDataArrived;
        std::atomic_bool m_processSpiTransactions;
        std::atomic<const std::uint8_t*> m_pDataBuffer;
        std::atomic<size_t> m_bufferTransmitSize;


        std::unique_ptr<std::thread> m_dmaThread;
        TTransactionCompletedHandler m_transactionCompleted;
    };
    }
```

A.1.13 Скрпит збирання та прошивки додатку:

Скрипт доступний за посиланням на github проекту: https://github.com/ValentiWorkLearning/GradWork/blob/dev/coroutine/Firmware/CMakeLists.txt

A.1.14 Реалізація сервісу логування даних:

```
#include "inc/logger/logger_service.hpp"

                                #include "utils/CallbackConnector.hpp"


                                #include <atomic>
                                #include <type_traits>


                                #if defined(LoggerUart)
                                #include "nrf.h"
                                #include "boards.h"
                                #include "app_uart.h"
                                #include "app_error.h"
                                #include "bsp.h"
                                #elif defined(LoggerSwo)
                                #include "nrf.h"
                                #include "boards.h"
                                #include "app_uart.h"
                                #include "app_error.h"
                                #include "bsp.h"
                                #elif defined(LoggerSegger)
                                #include "SEGGER_RTT.h"
                                #elif defined(LoggerDesktop)
```

```
#include <cstdio>
    #if defined USE_MSVC_DEBUG_OUT
        #include <Windows.h>
    #endif


#endif


namespace
{
    std::string_view CaretReset = "\r\n";
}


#if defined (LoggerUart)


class Logger::LoggerImpl
{


public:


    LoggerImpl() noexcept
    {
        initLogInterface();
    }


    void logString( std::string_view _toLog )
const noexcept
    {
        for( const auto ch : _toLog )

app_uart_put( static_cast<std::uint8_t>( ch )
);
    }




private:
```

```
    void initLogInterface() noexcept
    {
        std::uint32_t errorCode{};


        const app_uart_comm_params_t
appUartParams =
        {
                RX_PIN_NUMBER
            ,   TX_PIN_NUMBER
            ,   RTS_PIN_NUMBER
            ,   CTS_PIN_NUMBER
            ,   APP_UART_FLOW_CONTROL_DISABLED
            ,   false
            ,
UART_BAUDRATE_BAUDRATE_Baud115200
        };


        auto uartEventCallback =
cbc::obtain_connector(
            [ this ]( app_uart_evt_t * _pEvent
)
            {
                return
uartEventHandler( _pEvent );
            }
        );


        APP_UART_FIFO_INIT(
                &appUartParams
            ,   RxSize
            ,   TxSize
            ,   uartEventCallback
            ,   APP_IRQ_PRIORITY_LOWEST
            ,   errorCode
        );


        APP_ERROR_CHECK( errorCode );
    }
```

```
private:


    static constexpr std::uint16_t TxSize =
256;        /**< UART TX buffer size. */
    static constexpr std::uint16_t RxSize =
TxSize;     /**< UART RX buffer size. */
    static constexpr std::uint8_t
TestDataBytes = 15;   /**< max number of test
bytes to be used for tx and rx. */


    void uartEventHandler(app_uart_evt_t *
_pEvent)noexcept
    {
        if (_pEvent->evt_type ==
APP_UART_COMMUNICATION_ERROR)
        {
            APP_ERROR_HANDLER( _pEvent-
>data.error_communication );
        }
        else if ( _pEvent->evt_type ==
APP_UART_FIFO_ERROR )
        {
            APP_ERROR_HANDLER( _pEvent-
>data.error_code );
        }
    }
};
#endif


#if defined (LoggerSwo)
class Logger::LoggerImpl
{


public:


    LoggerImpl()noexcept
    {
        initLogInterface();
    }
```

```
public:


    void logString( std::string_view _toLog )
const noexcept
    {
        for( const auto ch : _toLog )
            ITM_SendChar( ch );
    }


private:
    void initLogInterface()noexcept
    {
        NRF_CLOCK->TRACECONFIG =
            ( NRF_CLOCK->TRACECONFIG &
~CLOCK_TRACECONFIG_TRACEPORTSPEED_Msk )
        |
( CLOCK_TRACECONFIG_TRACEPORTSPEED_4MHz <<
CLOCK_TRACECONFIG_TRACEPORTSPEED_Pos );


        ITM->TCR |= 1;
        ITM->TER |= 1;
    }
};
#endif



#if defined (LoggerSegger)
class Logger::LoggerImpl
{


public:


    void logString( std::string_view _toLog )
const noexcept
    {
        SEGGER_RTT_WriteString( 0,
_toLog.data() );
```

```
    }
};
#endif


#if defined (LoggerDesktop)
class Logger::LoggerImpl
{


public:
    void logString(std::string_view _toLog)
const noexcept
    {
        printf(_toLog.data());
#if defined USE_MSVC_DEBUG_OUT
        OutputDebugString(_toLog.data());
#endif
    }
};
#endif




Logger::Logger()noexcept = default;


Logger::~Logger() = default;


Logger& Logger::Instance()noexcept
{
    static Logger intance;
    return intance;
}


void Logger::logDebugEndl( std::string_view
_toLog )noexcept
{
        m_pLoggerImpl->logString( _toLog );
        m_pLoggerImpl-
>logString( CaretReset );
}
```

```
void Logger::logDebug( std::string_view _toLog
)noexcept
{
        m_pLoggerImpl->logString( _toLog );
}
```

ДОДАТОК Б

https://doi.org/10.1007/s10479-021-04417-1

Development of coroutines usage model for cooperative multitasking implementation on the systems with limited resources

Amin Salih Mohammed[1,2] · Inna Filippenko[3] · B. Saravana Balaji[4] · Olesia Barkovska[3] · Ivan Semenenko[3] · Valentyn Korniienko[3]

Accepted: 8 November 2021

Abstract

According to the growing complexity of tasks, which embedded systems should solve, there arises a need to use Real-Time Operation Systems (RTOS). However, RTOS usage and imple- mentation require a set of restrictions for a system, especially on the memory usage and time delays occurrence. The available implementations of coroutines neither cross-platform solutions nor accessible without a third-party library. The article proposes a model of cross- platform coroutine usage from C 20 for embedded systems programming and lightweight cooperative multitasking implementation. The proposed model contains the improved mini- mal scheduler without priorities mechanism and tasks descriptors. The model of coroutines usage for resource-constrained systems is developed. The task inside the model is presented as a coroutine, which can communicate with available coroutines, call them, or be suspended. Also, the minimal primitives of coroutines usage for cooperative multitasking, architec- ture decisions, and non-blocking data transmission were implemented. For the experimental research, the following list of program components was implemented in the proposed model: cooperative coroutines scheduler, tasks for data transmission, and awaitable type prototypes. The testing and analysis were completed on the chosen microcontroller (MCU). The perspec- tive of coroutines usage on resource-constrained systems was shown in the research based on obtained results of FLASH and RAM consumption.

Keywords Coroutines · Embedded systems · Memory consumption reducing · Embedded interfaces · Cooperative multitasking

B. Saravana Balaji saravanabalaji.b@gmail.com

[1] Department of Computer Engineering, Lebanese French University, Erbil, Iraq

[2] Department of Software and Informatics, Salahaddin University, Erbil, Iraq

[3] Department of Automation and Computer Engineering Design, Kharkiv National University of Radio Electronics, Kharkiv Oblast, Ukraine

[4] Department of Information Technology, Lebanese French University, Erbil, Iraq

Published online: 25 November 2021

УДК 681.326

ОГЛЯД ГРАФІЧНИХ БІБЛІОТЕК ДЛЯ ВБУДОВАНИХ ПЛАТФОРМ

ФІЛІППЕНКО І.В., КОРНІЄНКО В.Р., КУЛАК Г.К.

Наводиться огляд існуючих на поточний момент графічних бібліотек для вбудованих систем для створення панелей HMI та приладів носимої електроніки. Розглядаються основні етапи, які необхідні для адаптації однієї з існуючих бібліотек на процесор Nordic. Наведено вимоги до ресурсів системи (ОЗП, ПЗП, різновидам інтерфейсів, що наявні на конкретній платформі) для проектування вбудованих систем з використанням графічних пристроїв. Вивчаються особливості SDK для одного з процесорних вендорів та рішень, що необхідно реалізувати для запуску бібліотеки.

Ключові слова: графічна бібліотека, вбудована си- стема, інтерфейси обміну даними, способи організації обміну, процесорний вендор, дисплейні модулі.

Key words: graphical library, embedded system, data transfer interface, data transferring organization, processors vendor, display modules.

1. Вступ

Одним з етапів проектування вбудованої системи є створення графічного інтерфейсу. Особливо важливим це є для області проектування HMI-пристроїв (Human-Machine Interface). З розповсюдженням на ринку 32-бітних проце- сорів на базі ядра ARM зростає тенденція до до- давання графічного інтерфейсу та створення йо- го близьким за виглядом до вже існуючих стан- дартів дизайну Material.

Змінюються тенденції в області підбору цільово- го мікроконтролера, який буде використано у пристрої. Поступово до периферії у процесорних вендорів, таких як STMicroelectronics, Nordic Semiconductor, Texas Instruments, Infineon, дода- ються просунуті модулі роботи з пам’яттю, графічні прискорювачі.

З метою забезпечення тривалої автономної робо- ти системи (зазвичай це стосується пристроїв носимої електроніки) використовують просунуті режими енергоспоживання та його профілюван- ня. У випадку наявності графічної складової у системі потрібно забезпечити коректність інформації, що доступна на дисплеї під час роботи у режимі зниженого енергоспоживання.

Беручи до уваги постійне вдосконалення та існуючий різновид дисплейних інтерфейсів, необхідно враховувати можливості бібліотеки з боку виведення зображення, тому що іноді по- трібна якість зображення не може бути досягнута через особливості конкретного інтерфейсу.

14. 2 Існуючі інтерфейси обміну з дисплейними модулями

За типом обміну між контролером та периферій- ним дисплейним модулем інтерфейси поділяють на послідовні та паралельні. У випадку інтер- фейсів з високою швидкістю слід відмітити па- ралельні FSMC та LTDC [1]. Розглянемо деякі з них.

SPI (Serial Peripheral Interface) – один з найвідоміших і широко використовуваних на даний час інтерфейсів зв'язку між різними мікро- схемами в електронних пристроях. Його основ- ними перевагами є простота, невелика кількість сигнальних ліній (у багатьох випадках достатньо лише трьох ліній), достатня простота реалізації друкованої плати при розробці пристрою.

Ключовою відмінністю SPI від паралельних ін- терфейсів є те, що він має всього одну лінію пе- редачі даних, та швидкість передачі часто не пе- ревищує 20 Мбіт/с, що значно обмежує пропуск- ну здатність, а отже і максимальний розмір керо- ваного дисплея. Це робить практично неможли- вим створення складних динамічних зображень. Крім того, управління дисплеєм відбувається за допомогою постійного пересилання команд, а це змушує ядро контролера виконувати безліч зай- вих задач, що підвищує його завантаження і ро- бота з зображенням виявляється неефективною. У процесорів від вендора Espressif (сімейство ESP32) наразі доступні процесори з модулями SPI, що працюють на частоті до 60 МГц. У випадку старших сімейств STM32H7 доступна робота з шинами SPI на швидкості до 120МГц. Слід зауважити, що дисплейні контролери зазви- чай підтримують роботу з шиною SPI на частоті не вище 50МГц.

FSMC (Flexible Static Memory Controller) – кон- тролер зовнішньої пам'яті, який дозволяє взаємодіяти з TFT-дисплеями, забезпеченими інтерфейсами Motorola 6800 і Intel 8080. При певних умовах FSMC може використовуватися для роботи з TFT-дисплеями з RGB-інтерфейсом.

LTDC (LCD-TFT Display Controller) – спеціалізований TFT-контролер, який дозволяє працювати з TFT-дисплеями з RGB-інтерфейсом.

Послідовний інтерфейс, що набуває популяр- ності у процесорів старших сімейств з ядром ARM, – **MIPI DSI** (Display Serial Interface специфікація Mobile Industry Processor Interface), який є спеціалізованим інтерфейсом для роботи з TFT-дисплеями, з MIPI-DSI [2].

Ці інтерфейси доступні лише на останніх сімей- ствах процесорів, таких як STM32L4, STM32H7, STM32F7. На більш розповсюджених моделях зазвичай є у наявності інтерфейси SPI та I2C, в окремих випадках – інтерфейс LTDC.

У більшості випадків внутрішньої пам'яті мікро- контролера не вистачає для зберігання графічних зображень і розміщення екранної пам'яті. З цієї причини дуже часто потрібна зовнішня Flash- пам’ять, або ОЗП. Для підключення зовнішньої

ДОДАТОК В

МОДЕЛЬ ВИКОРИСТАННЯ СОПРОГРАМ ПРИ ПРОГРАМУВАННІ ВБУДОВАНИХ СИСТЕМ

Виконав:
студент групи СКСм-20-1
Корнієнко Валентин
Русланович

Керівник:
доц. каф. АПОТ
Філіппенко І.В.

Харківський національний університет радіоелектроніки, Харків, Україна,
2021р

Зміст

- Тенденції розвитку вбудованих систем
- Актуальність проблеми асинхронної взаємодії компонентів
- Огляд вбудованих ОСРВ
- Існуючі методи асинхронної взаємодії
- Сопрограмна модель взаємодії компонентів
- Підтримка у сучасних компіляторах
- Реалізація
- Висновки

NORDIC® SEMICONDUCTOR

Актуальність проблеми

- Додавання периферійних компонентів, що працюють за моделью fire-and-forget
- Необхідність додавання ОСРВ та/або POSIX-реалізацій setcontext/makecontext/swapcontext
- Необхідність інтеграції з існуючими компонентами програмного забезпечення
- інвазивні підходи до збереження контексту сьогодні(std::future/std::promise, teleports, reactive Cpp, ProtoThreads)

Тенденції розвитку вбудованих систем

- домінуюча архітектура процесорів для вбудованих систем-ARM
- підтримка Realtime-відладки цільового пристрою (JLink-RTT/OpenOCD/BlackMagicProbe)
- необхідність крос-платформеної роботи проекту
- зростаюча обчислювальна потужність->зростаючи можливості розгортання бізнес-логіки:
  - графічні бібліотеки(Qt/LVGL/TouchGFX)
  - протоколи зв'язку(MQTT/BLE/LoraWAN/ZigBEE)
  - MotorControl рішення з HWPWM(Hardware PWM)
  - потужні інтерфейси комунікації(MIPI/LTDC/CAN/USB)

Характеристика роботи

- Актуальність - створення та запровадження ефективних структур збереження та відтворення контексту виконання без використання ОСРВ для вбудованих систем.
- Мета дослідження- розробка сопрограмних примітивів взаємодії, що надають можливості гнучкого кооперативного планування задач у рамках прошивки пристрою.
- Задачі дослідження
  - Аналіз методів забезпечення багатозадачності у сучасних рішеннях
  - Розробка компонентів синхронізації
  - Верифікація працездатності розроблених компонентів

Огляд вбудованих ОСРВ

З сучасних рішень слід виділити:

- FreeRTOS - домінуюча ОС для вбудованих рішень
- ChibiOS - компактна та швидка операційна система реального часу з ліцензією GPL3
- KeilRTX - ОС з відкритим кодом, розроблена під архітектури ARM та ARM-Cortex-M
- managarm- ОС з відкритим кодом та асинхронним I/O з POSIX-сумісним API

Існуючі методи асинхронної взаємодії

У вбудованих системах сьогодні визначається декілька архітектурних патернів до проектування:

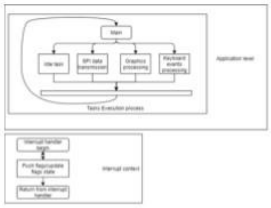
- Модель “суперциклу”
- додавання у проект ОСРВ з використанням розбиття системи на задачі(у випадку витісняючої ОС з’являються додаткові кошти на синронізацію)
- написання task scheduler під конкретну систему та робота у режимі bare-metal

Недоліки існуючих компонентів

- Надмірні кошти у використанні примітивів ОС
- Тяжкість у підтримці проекту, що працює у режимі bare-metal при необхідності розширення функціоналу
- Написання власного task scheduler зазвичай переходить у використання існуючого варіанту ОСРВ для вбудованої системи

Сопрограмна модель взаємодії

- До станів функції, що виконується додається можливість перевести функцію у стан Suspended(призупинена)
- Типові задачі у вбудованих системах містять фази отримання, передачі та обробки інформації, що зазвичай потребують збереження контексту між фазами передачі та виконання задачі.



NRF52832, FreeRTOS

З використанням FreeRTOS, неблокуюче переключення стану світлодіода

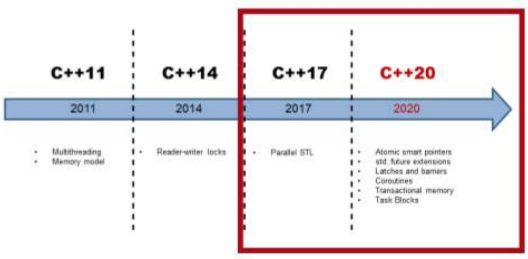
```
xTaskCreate(ledBlinkTask, "ledBlinkTask", 128, NULL, 2, NULL);

void ledBlinkTask(void* pvParameters) {
 while (true) {
  vTaskDelay(300 / portTICK_PERIOD_MS);
  bsp_board_led_invert(0);
 }
}
```

amazon

freeRTOS

NRF52832, C++20 Coroutines:

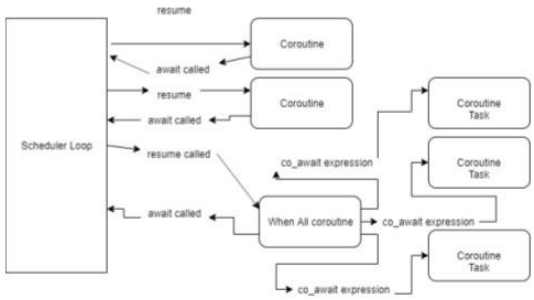
```
void Board::ledToggle() {
 using namespace std::chrono_literals;
 while (true) {
  co_await 300ms;
  LOG_DEBUG_ENDL("LED TIMER EXPIRED");
  bsp_board_led_invert(0);
 }
}
```



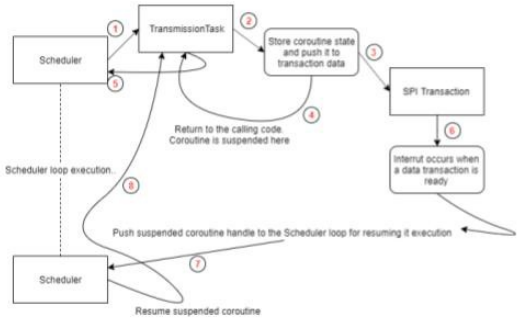
Повний контроль над середовищем виконання

```
void await_suspend(std::coroutine_handle<> _coroLedHandle) {

  ret_code_t errorCode{};

  errorCode =

    app_timer_start(m_ledDriverTimer, APP_TIMER_TICKS(m_duration.count()),
_coroLedHandle.address());

  APP_ERROR_CHECK(errorCode);}

void await_resume() { app_timer_stop(m_ledDriverTimer); }
```

Модель взаємодії сопрограм



Стани сопрограми для задачі передачі даних



Тестування SPI драйверу

```
TEST_P(SpiDriverTest, CheckRandomSequenceWithLengthTransmissionCorrect)
{
    // Отримуємо розмір послідовності
    const std::uint16_t TransactionLength{std::get<0>(GetParam())};
    TDataStream ExpectedStream{TransactionLength};

    std::random_device randomDevice;
    std::mt19937 generator(randomDevice());
    std::uniform_int_distribution<> distribution(0x00, 0xFF);
    // формуємо випадкову посилку
    std::ranges::generate(ExpectedStream, [&distribution, generator]() mutable {
        return std::byte(distribution(generator));
    });

    CoroUtils::syncWait(sendChunk(
        reinterpret_cast<const std::uint8_t*>(ExpectedStream.data()), ExpectedStream.size()));
    // Звіряємо відправлені та прийняті данні у spi-backend
    EXPECT_EQ(TransactionsToDataStream(), ExpectedStream);
}
```


Концепція when_all

- Дозволяє запустити на виконання декілька асинхронних компонентів
- Реалізація дозволяє гнучко здійснювати обробку помилок
- Мінімальна аллокація пам'яті під задачі сопрограм

```
template<typename ...Args>
auto when_all(Args&& ... args)
{
    return WhenAllTask{ std::forward_as_tuple(args...) };
}

void initializeProcedure()
{
    co_await when_all(
            sendCommand(0xFF),
            sendCommand(0xAD),
            sendCommand(0x00)
      );
}
```

Підтримка у сучасних компіляторах

- GCC - починаючи з версії 10.1 - перша реалізація, з 10.2 - стабільна реалізація
- CLang- з 9 версії компілятора
- MSVC- починаючи з 15.3 як working draft у стандартній бібліотеці.

Заключення

Наукова новізна

- Наведено аналіз існуючих механізмів багатозадачності у вбудованих системах. Наведено реалізацію примітивів з використанням механізмів сопрограм C++20.
- Запропоновано модель використання сопрограм при програмуванні вбудованих систем та виконано її реалізацію на базі примітивів C++20. Модель може бути використана у вбудованих системах для провадження low-memory-footprint механізмів кооперативної багатозадачності та/або спрощення написання асинхронного коду.

Практична значимість.

- У порівнянні з існуючими механізмами істотно зменшується кількість необхідної пам'яті програм та пам'яті даних для запуску та підтримки сопрограм. Мінімально необхідні версії компіляторів також включають до себе механізми оптимізацій збереження стеку сопрограми, надаючи максимально шлях використання нових механізмів.

Перспективи дослідження

- Мета- підвищення ефективності існуючих C/C++ бібліотек для вбудованих систем за рахунок мінімізації неблокуючого I/O з мінімальною не-інвазивною інтеграцією до існуючих рішень.
- Сутність дослідження - створення примітивів асинхронної взаємодії з інтеграцією до існуючого проекту. Можливість створення структури та набору бібліотек для роботи з периферією та зовнішніми пристроями з мінімальними накладними витратами на супроводження та початкове внесення до проекту.