Лабораторная работа №7. Применение API операционной системы реального времени Keil RTX.

Изучение понятия операционной системы реального времени, ее задач и преимуществ. Ознакомление с архитектурой Keil RTX. Изучение принципов многопоточного программирования на примере задачи синхронизации светофоров. Настройка проекта и окружения в Keil uVision.

Учебное время: 4 часа.

Цель работы: Изучение операционной системы реального времени Keil RTX, применение полученных знаний для реализации многопоточного приложения.

1.1 Содержание работы

1) Ознакомиться с информацией об операционной системе реального времени Keil RTX [1] и настроить проект Keil uVision.
2) Написать программу многопоточного приложения, имитирующего работу светофоров согласно таблицам вариантов 1.1 - 1.3.

Необходимо в симуляторе Keil реализовать вывод в окно просмотра отладки (функция printf) состояний светофоров. Светофор состоит из нескольких фонарей, в зависимости от варианта это может быть один, два или три фонаря. Каждый фонарь может принимать следующие цвета: красный (R), оранжевый (O), желтый (Y), зеленый (G). Возможные состояния каждого из цвета: потушен(D), горит (G), мигает (B). Пример обозначения состояния цвета: красный мигает– RB; жёлтый потушен – YD; зеленый горит – GG.

Также обрабатываются события нажатия кнопок К1 - незамедлительно переключает состояние каждого светофора на следующее, К2 - принудительно устанавливает состояния светофоров в назначенные положения.

Варианты приведены в приложении.

1.2 Справочная информация

1.2.1 Понятие операционной системы

Дать точное определение операционной системы довольно трудно. Можно сказать, что это программное обеспечение осуществляющее две значительно отличающиеся друг от друга функции: предоставление прикладным программистам (и прикладным программам, естественно) вполне понятный абстрактный набор ресурсов взамен неупорядоченного набора аппаратного обеспечения и управляют этими ресурсами. Подробнее о всем в операционных системах можно прочитать в источнике [2].

1.2.2 Операционные системы реального времени

Эти системы характеризуются тем, что время для них является ключевым параметром. Например, в системах управления производственными процессами компьютеры, работающие в режиме реального времени, должны собирать сведения о процессе и использовать их для управления станками на предприятии.

Если операция должна быть проведена точно в срок (или в определенный период времени), то мы имеем дело с системой жесткого реального времени. Множество подобных систем встречается при управлении производственными процессами, в авиационно-космическом электронном оборудовании, в военной и других подобных областях применения. Эти системы должны давать абсолютные гарантии того, что определенные действия будут осуществляться в конкретный момент времени.

Другой разновидностью подобных систем является система мягкого реального времени, в которой хотя и нежелательно, но вполне допустимо несоблюдение срока какого-нибудь действия, что не наносит непоправимого вреда. К этой категории относятся цифровые аудио- или мультимедийные системы. Смартфоны также являются системами мягкого реального времени.

1.2.3 Операционная система Keil RTX

RTOS (Real-Time Operating System) Keil RTX - это многозадачная операционная система реального времени, встроенная в среду разработки Keil, которая обеспечивает вытесняющую многозадачность. Она упрощает разработку сложных систем, улучшая управление проектами и способствуя повторному использованию кода. Основным компромиссом использования ОСРВ является увеличение объёма используемой памяти и возможное удлинение времени реакции на прерывания. Однако современные микроконтроллеры с объёмом ОЗУ от 32 КБ и выше предоставляют достаточные ресурсы для работы ОСРВ, типичный размер которой составляет всего 5 КБ.

Keil RTX реализует стандарт CMSIS-RTOS, что обеспечивает совместимость с другими ОСРВ, соответствующими этому стандарту. Это позволяет разработчикам легко переносить код между различными платформами и упрощает интеграцию ОСРВ в существующие проекты.

На сегодняшний день доступны две версии RTX: RTOS v1 (Keil RTX 4) и RTOS v2 (Keil RTX 5). В данной работе будет использоваться первая версия ОС, к особенностям которой можно отнести:

1. Многозадачность (Multitasking);
2. Наличие приоритетов задач;
3. Синхронизация задач;
4. Таймеры и задержки;
5. Управление памятью (динамическое выделение памяти, разделение памяти между задачами);
6. Работа с прерываниями;
7. Поддержка различных типов синхронизации (IPC), в том числе очередей сообщений, почтовых ящиков и сигналов событий;
8. Поддержка энергоэффективности.

RTX состоит из следующих основных компонентов:

- Потоки (tasks): это базовые элементы выполнения. Каждый поток выполняется в своём контексте с отдельным стеком.
- Планировщик задач (scheduler): управляет распределением процессорного времени между потоками, используя механизм вытесняющей многозадачности. В Keil RTX 4 используется планировщик Round Robin – (циклический опрос) - каждому потоку выделяется фиксированное время на выполнение.
- Ядро ОСРВ: обрабатывает события, управляет таймерами и взаимодействием между потоками.
- Интерфейс взаимодействия с оборудованием: использует системный таймер (SysTick) для управления временем выполнения задач.

1.2.4 Элементы операционной системы Keil RTX

Кратко рассмотрим основные элементы операционной системы Keil RTX.

Потоки. Каждому потоку выделяется определенный интервал времени, который можно настроить. Планировщик использует вытесняющую многозадачность, что позволяет более высоким приоритетам прерывать выполнение задач с низким приоритетом.
Потоки могут находиться в одном из трех состояний:

- **Running** - поток выполняется.
- **Ready** - поток готов к выполнению, ожидает выделения процессорного времени.
- **Wait** - поток заблокирован, ожидает наступления события (например, сигнала от другого потока, завершения операции ввода/вывода или истечения таймера).

Для создания потока в Keil RTX используется функция osThreadCreate(). Каждому потоку необходимо назначить уникальную функцию, которая будет выполняться в рамках этого потока. Для завершения потока можно использовать функцию osThreadTerminate(). Пример создания потока:

```
#include "cmsis_os.h"
void Thread (void const *arg);  // function prototype for a Thread
osThreadDef (Thread, osPriorityNormal, 3, 0); // define Thread and
      specify to allow three instances
void ThreadCreate_example (void) {
 osThreadId id1;
  id1 = osThreadCreate (osThread (Thread), NULL); // create the thread
      with id1
 if (id1 == NULL) {            // handle thread creation for id1
```

```
    // Failed to create the thread with id1
  }
  osThreadTerminate (id1);       // stop the thread with id1
}
```

Здесь:

- osThreadDef(Thread, osPriorityNormal, 3, 0) — создание определения потока с функцией Thread, приоритетом osPriorityNormal, с аргументами (3) и размером стека 0.
- osThreadCreate(osThread(Thread), NULL) — создание потока на основе определения.

Мьютексы. Мьютексы в Keil RTX используются для синхронизации доступа к общим ресурсам, чтобы избежать конфликтов при параллельном доступе. Мьютекс блокирует ресурс, пока он используется одним потоком, и освобождает его для других потоков после завершения работы с ним.

Мьютекс создается с помощью функции osMutexCreate(). Для захвата мьютекса используется osMutexWait(), а для освобождения — osMutexRelease(). Пример создания мьютекса:

```
#include "cmsis_os.h"
 osMutexDef (MutexIsr);         // Mutex name definition
 void CreateMutex (void)  {
osMutexId mutex_id;
 mutex_id = osMutexCreate  (osMutex (MutexIsr));
 if (mutex_id != NULL)  {
   // Mutex object created
 }
}
```

Пример использования мьютекса:

```
#include "cmsis_os.h"
osMutexDef (MutexIsr);
void WaitAndReleaseMutex (void)  {
 osMutexId mutex_id;
 osStatus status;
 mutex_id = osMutexCreate  (osMutex (MutexIsr));
 if (mutex_id != NULL)  {
   status  = osMutexWait    (mutex_id, osWaitForever);
   if (status != osOK)  {
     // handle failure code
   }
   status = osMutexRelease(mutex_id);
   if (status != osOK)  {
     // handle failure code
   }
 }
}
```

В примерах выше:

- osMutexDef(MutexIsr) — создание определения мьютекса.
- osMutexCreate(osMutex(MutexIsr)) — создание мьютекса.
- osMutexWait(mutex_id, osWaitForever) — захват мьютекса. Если он уже занят, задача будет ожидать освобождения (вечное ожидание).

- osMutexRelease(mutex_id) — освобождение мьютекса.

Ресурс памяти (Memory Pool). Memory Pool — это механизм в CMSIS-RTOS RTX для динамического управления памятью. Он позволяет выделять и освобождать блоки памяти фиксированного размера во время выполнения программы (runtime) без использования стандартных функций malloc() и free(). Это позволяет эффективно управлять памятью, например выделять ее динамически для временных данных: буферов, структур, сообщений.

В коде, как правило, создается пул через osPoolCreate, а память выделяется с помощью osPoolAlloc и освобождается с помощью osPoolFree. Рассмотрим пример создания ресурса памяти:

```
#include "cmsis_os.h"
typedef struct {
 uint8_t Buf[32];
 uint8_t Idx;
} MEM_BLOCK;
osPoolDef (MemPool, 8, MEM_BLOCK);
 void CreateMemoryPool (void)  {
osPoolId MemPool_Id;
 MemPool_Id = osPoolCreate (osPool (MemPool));
 if (MemPool_Id != NULL)  {
   // memory pool created
 }
}
```

Выше используются параметры при объявлении и создании ресурса памяти:

- osPoolDef (name, no, type) - определение ресурса памяти, с параметрами: name - имя ресурса памяти, no - максимальное число выделяемых блоков памяти в ресурсе, type - тип данных отдельного выделяемого блока памяти.
- osPoolCreate (const osPoolDef_t * pool_def) - создание ресурса памяти с параметром - ранее созданным объявлением.

Далее рассмотрим пример выделения и возврата отдельного блока памяти:

```
#include "cmsis_os.h"
typedef struct {
 uint8_t Buf[32];
 uint8_t Idx;
} MEM_BLOCK;
osPoolDef (MemPool, 8, MEM_BLOCK);
 void CAlocMemoryPoolBlock (void)  {
 osPoolId   MemPool_Id;
 MEM_BLOCK *addr;
 osStatus   status;
  MemPool_Id = osPoolCreate (osPool (MemPool));
 if (MemPool_Id != NULL)  {
   addr = (MEM_BLOCK *)osPoolCAlloc (MemPool_Id);
   if (addr != NULL) {
     :
     // return a memory block back to pool
     status = osPoolFree (MemPool_Id, addr);
     if (status==osOK)  {
       // handle status code
     }
```

```
    }
  }
}
```

Очереди сообщений (Message Queues). Очереди сообщений позволяют обмениваться данными между потоками. Задачи могут отправлять и получать сообщения через очередь, что полезно для передачи данных или событий между потоками. Важно учесть, что все операции внутри очереди окружены другими примитивами синхронизации, ввиду чего очередь сообщений является безопасным способом организации межпроцессорного взаимодействия.

Очередь сообщений создается с помощью функции osMessageCreate(). Для отправки сообщения используется osMessagePut(), а для получения — osMessageGet(). Пример создания и использования очереди сообщений:

```
#include "cmsis_os.h"
osThreadId tid_thread1;   // ID for thread 1
osThreadId tid_thread2;   // for thread 2
typedef struct {          // Message object structure
 float    voltage;        // AD result of measured voltage
 float    current;        // AD result of measured current
 int      counter;        // A counter value
} T_MEAS;
osPoolDef(mpool, 16, T_MEAS);   // Define memory pool
osPoolId  mpool;
osMessageQDef(MsgBox, 16, &T_MEAS); // Define message queue
osMessageQId  MsgBox;
void send_thread (void const *argument);         // forward reference
void recv_thread (void const *argument);         // forward reference
// Thread definitions
osThreadDef(send_thread, osPriorityNormal, 1, 0);
osThreadDef(recv_thread, osPriorityNormal, 1, 2000);
//
//  Thread 1: Send thread
//
void send_thread (void const *argument) {
 T_MEAS    *mptr;
 mptr = osPoolAlloc(mpool);   // Allocate memory for the message
 mptr->voltage = 223.72;      // Set the message content
 mptr->current = 17.54;
 mptr->counter = 120786;
 osMessagePut(MsgBox, (uint32_t)mptr, osWaitForever);  // Send Message
 osDelay(100);
 mptr = osPoolAlloc(mpool);     // Allocate memory for the message
 mptr->voltage = 227.23;        // Prepare a 2nd message
 mptr->current = 12.41;
 mptr->counter = 170823;
 osMessagePut(MsgBox, (uint32_t)mptr, osWaitForever);  // Send Message
 osThreadYield();         // Cooperative multitasking
                          // We are done here, exit this thread
}
//
//  Thread 2: Receive thread
//
void recv_thread (void const *argument) {
 T_MEAS  *rptr;
 osEvent  evt;

 for (;;) {
```

```
    evt = osMessageGet(MsgBox, osWaitForever);  // wait for message
    if (evt.status == osEventMessage) {
      rptr = evt.value.p;
      printf ("\nVoltage: %.2f V\n", rptr->voltage);
      printf ("Current: %.2f A\n", rptr->current);
      printf ("Number of cycles: %d\n", rptr->counter);
      osPoolFree(mpool, rptr);                  // free memory
        allocated for message
    }
  }
}
void StartApplication (void) {
 mpool = osPoolCreate(osPool(mpool));    // create memory pool
 MsgBox = osMessageCreate(osMessageQ(MsgBox), NULL);  // create msg
       queue

 tid_thread1 = osThreadCreate(osThread(send_thread), NULL);
 tid_thread2 = osThreadCreate(osThread(recv_thread), NULL);
 :
}
```

1.3 Выполнение лабораторной работы

1.3.1 Настройка проекта Keil

Перед использованием объектов операционной системы RTX необходимо создать и настроить новый проект. Создаем проект из среды Keil путем нажатия на кнопку Project >> New µVision Project.

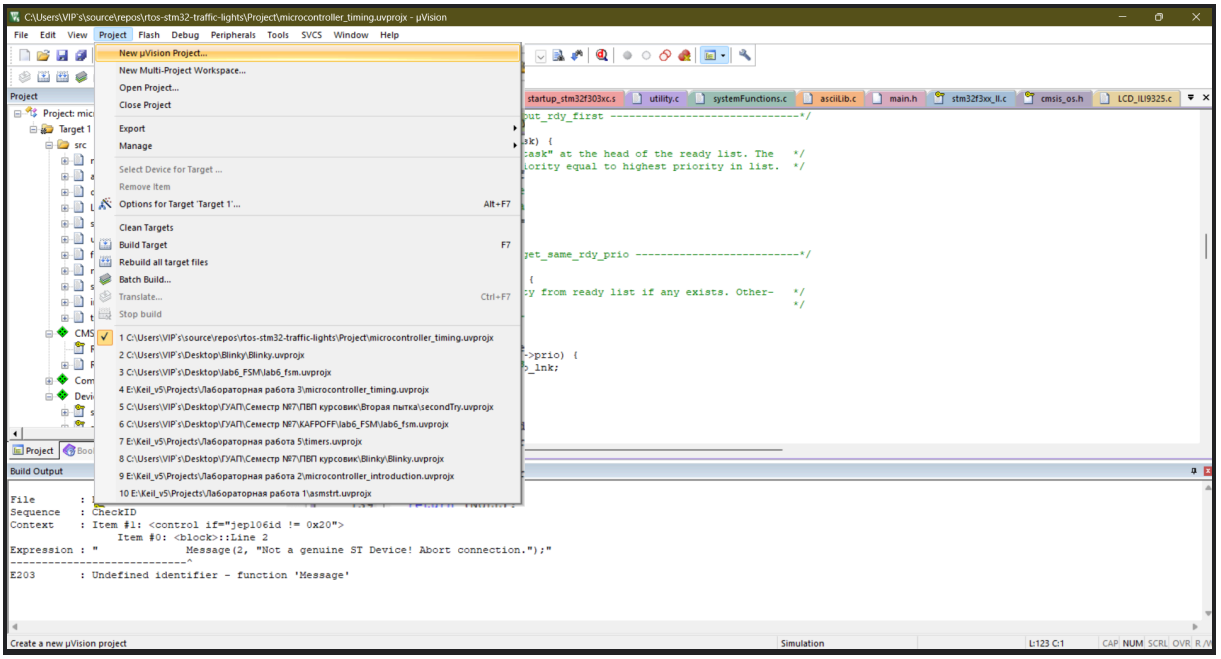


Рисунок. Создание нового проекта Keil.

Далее выбираем необходимую модель контроллера, в нашем случае это STM32F303VCTx. Далее необходимо настроить окружение проекта, выбрав необходимые компоненты. Для использования интерфейса операционной системы реального времени CMSIS RTX необходимо выбрать CMISIS >> RTOS (API) >> Keil RTX, кроме того будем использовать библиотеку LL для настройки RCC и GPIO. Также будем выводить в консоль состояния потоков (светофоров) с помощью printf, поэтому выбираем компонент Compiler >> I/O >> STDOUT. Разрешаем все конфликты зависимостей кнопкой Resolve (в левом нижнем углу окна настройки окружения) и получаем следующую настройку проекта Keil.

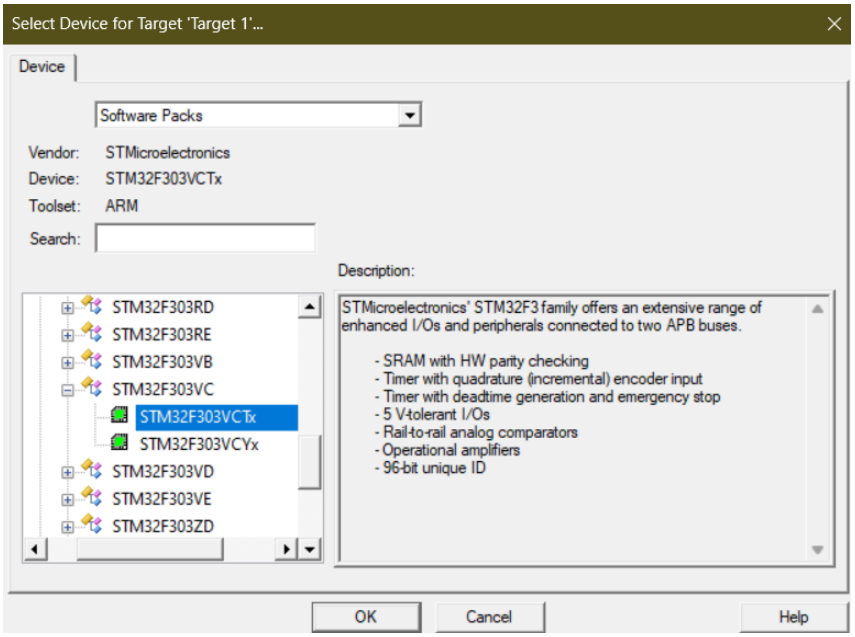
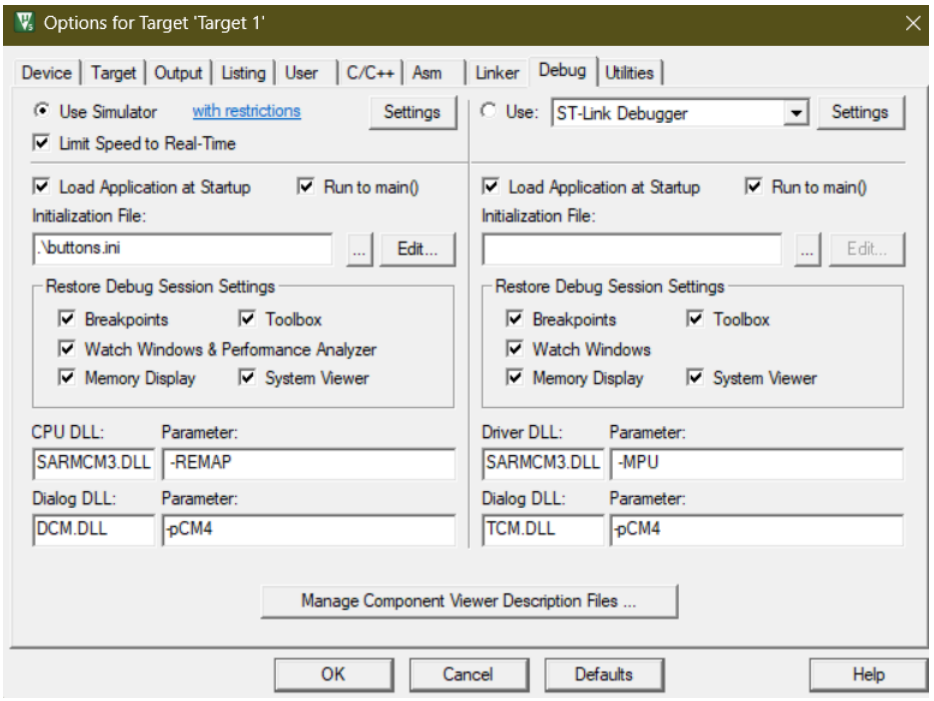


Рисунок. Выбор модели контроллера.

Software Component	Sel.	Variant	Version	Description
Board Support		STM32303E-EVAL	2.1.3	STMicroelectronics STM32303E-EVAL Board
CMSIS				Cortex Microcontroller Software Interface Components
CORE	☑		5.0.0	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M
DSP	☐		1.4.6	CMSIS-DSP Library for Cortex-M, SC000, and SC300
RTOS (API)			1.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
Keil RTX5	☐		5.0.0	CMSIS-RTOS RTX5 implementation for Cortex-M, SC000, and SC300
Keil RTX	☑		4.81.0	CMSIS-RTOS RTX implementation for Cortex-M, SC000, and SC300
RTOS2 (API)			2.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.1.0	Compiler Extensions for ARM Compiler ARMCC and ARMClang
Event Recorder	☐	DAP	1.1.0	Event Recording using Debug Access Port (DAP)
I/O				Retarget Input/Output
File	☐	File System	1.1.0	Use retargeting together with the File System component
STDERR	☐	User	1.1.0	Redirect STDERR to a user defined output target (USART, Graphics Display or other)
STDIN	☐	User	1.1.0	Retrieve STDIN from a user specified input source (USART, Keyboard or other)
STDOUT	☑	User	1.1.0	Redirect STDOUT to a user defined output target (USART, Graphics Display or other)
TTY	☐	User	1.1.0	Redirect TTY to a user defined output target
Device		Standalone	1.11.3	All HAL and LL peripheral APIs are selectable as individual components.
Startup	☑		1.11.3	System Startup for STMicroelectronics
STM32Cube HAL				
STM32Cube LL				
ADC	☐		1.11.3	Analog-to-digital converter (ADC) LL driver
COMMON	☑		1.11.3	Common LL driver
COMP	☐		1.11.3	Analog Comparator (CRC) LL driver
CRC	☐		1.11.3	CRC calculation unit (CRC) LL driver
DAC	☐		1.11.3	Digital-to-analog converter (DAC) LL driver
DMA	☐		1.11.3	DMA controller (DMA) LL driver
EXTI	☐		1.11.3	Extended interrupts and events controler (EXTI) LL driver
FMC	☐		1.11.3	Flexible memory controller (FMC) LL driver
GPIO	☑		1.11.3	General-purpose I/O (GPIO) LL driver
HRTIM	☐		1.11.3	High resolution timer (HRTIM) LL driver
I2C	☐		1.11.3	Inter-integrated circuit (I2C) interface LL driver
OPAMP	☐		1.11.3	Operational amplifier (OPAMP) peripheral LL driver
PWR	☑		1.11.3	Power controller (PWR) LL driver
RCC	☑		1.11.3	Reset and clock control (RCC) LL driver
RTC	☐		1.11.3	Real-time clock (RTC) LL driver
SPI	☐		1.11.3	Serial peripheral interface (SPI) LL driver
TIM	☐		1.11.3	Timers (TIM) LL driver
USART	☐		1.11.3	Universal synchronous asynchronous receiver transmitter (USART) LL driver
USB	☐		1.11.3	Universal serial bus full-speed device interface (USB) LL driver
UTILS	☑		1.11.3	UTILS LL driver
File System		MDK-Pro	6.9.0	File Access on various storage devices

Рисунок. Настройка окружения проекта.

Установим использование симулятора в настройках проекта, дополнительно укажем файл инициализации для использования виртуальных кнопок:



Сам файл инициализации имеет следующее содержимое:

```
// Add read permissions for IDR (GPIOC input data register)
MAP 0x48000810, 0x4800081F READ WRITE

// Change PC4 state (IDR4)
FUNC void toggle_C4() {
    _WDWORD(0x48000810, _RDWORD(0x48000810) ^ 0x10);
}

// Change PC6 state (IDR6)
FUNC void toggle_C6() {
    _WDWORD(0x48000810, _RDWORD(0x48000810) ^ 0x40);
}

// Define buttons for Keil Debug Toolbox
DEFINE BUTTON "K1 (PC4)", "toggle_C4()"
DEFINE BUTTON "K2 (PC6)", "toggle_C6()"
```

Отображать состояния светофоров следует в окне отладки (Debug (printf) Viewer) с помощью перенаправления вывода printf.

Перед компиляцией проекта необходимо зайти в настройки проекта и на вкладке Debug нажать Settings. В открывшемся окне, на вкладке Trace необходимо установить галочку напротив Trace Enable и частоту тактирования ядра Core Clock - 8.0 MHz:

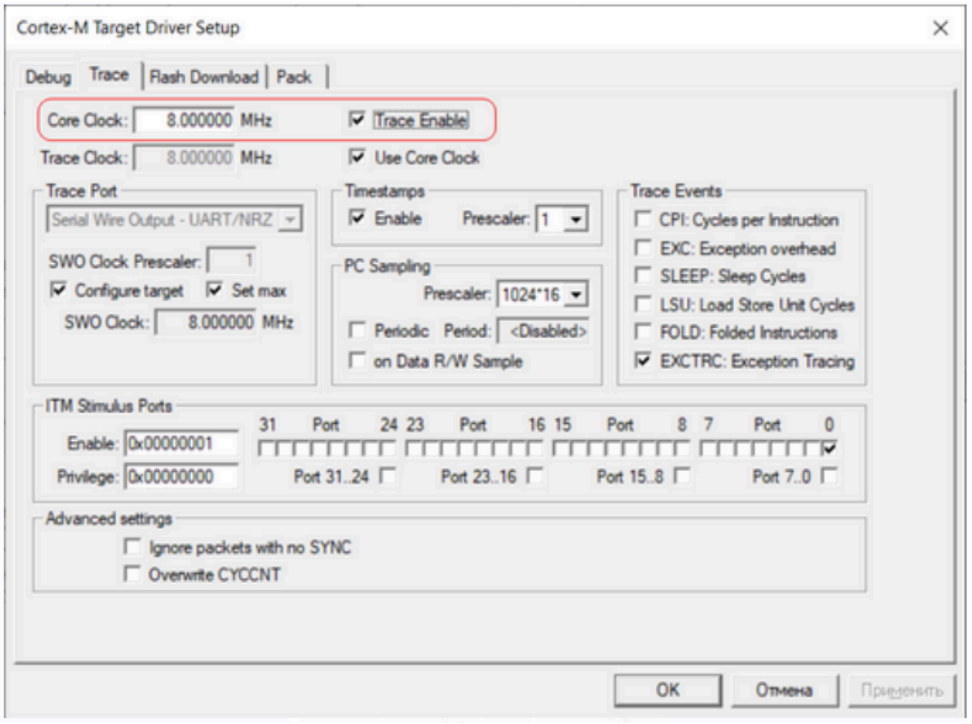


Рисунок. Настройка отладчика для вывода в окно отладки.

Отобразить окно отладки можно во вкладке *View -> Serial Windows -> Debug (printf) Viewer.*

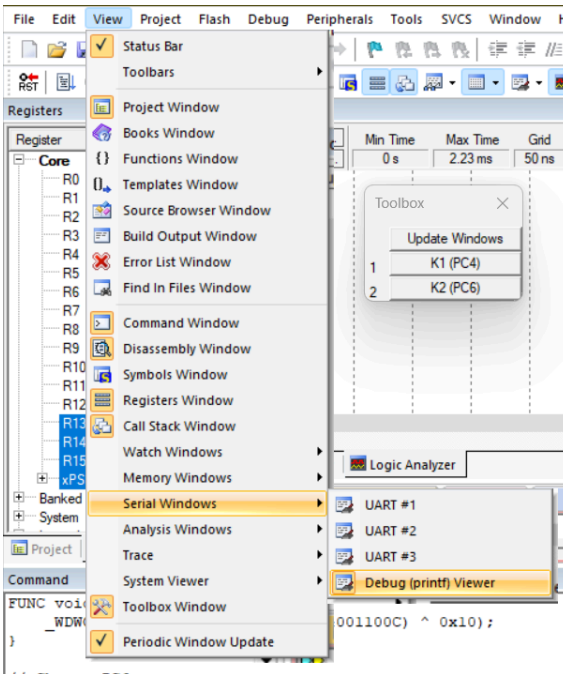


Рисунок. Порядок включения окна отладки.

Теперь можно отображать текущее состояние светофора с помощью функции printf(), например следующим образом:

```
printf("Traffic light id %02d:\t", threadId);
    for (int i = 0; i < colorsCount; i++) {
    printf("%s\t", lights[i].color);
        }
printf("\n");
```

1.3.2 Пример взаимодействия светофоров

Рассмотрим пример реализации варианта 37 с двумя светофорами (каждый имеет один фонарь). В качестве светофоров будут выступать отдельные потоки, кроме того, одним из потоков будет очередь сообщений (Message Queue), обрабатывающая сообщения светофоров о смене состояний. Также не забудем о потоках, отвечающих за нажатие кнопок управления К1 и К2. Реализация также может основываться на прерываниях или событиях, но это не будет рассматриваться.

Определим структуры для состояний потоков и сообщений, принимаемых MessageQueue:

```
typedef struct {
    unsigned char threadId;
    const char* lights[2];
    unsigned int count;
} Message;

typedef struct {
    const char* color;
    unsigned int delay;
} TrafficLightState;
```

Далее необходимо определить сами наборы состояний для потоков, исходя из варианта 37 получим следующее для первого светофора:

```
TrafficLightState firstThreadStates[3] = {
    { "RG", 3000 },
    { "RB", 2000 },
    { "RD", 4000 }
};
```

И для второго светофора:

```
TrafficLightState secondThreadStates[3] = {
    { "YB", 3000 },
    { "YG", 2000 },
    { "YD", 4000 }
};
```

Далее определим функции создания потоков и семафоров для потокобезопасного изменения переменных (номер состояния потока). Приведем пример для одного светофора:

```
	void firstTrafficLightThread (void const *argument);
// thread function
	osThreadId firstThreadId;                 // thread id
	osThreadDef(firstTrafficLightThread, osPriorityNormal, 1, 0);
// thread object

	osSemaphoreDef(firstSemaphore);
	osSemaphoreId firstSemaphoreId;

	extern osSemaphoreId secondSemaphoreId;

	int initFirstTrafficLight() {
	    firstSemaphoreId =
osSemaphoreCreate(osSemaphore(firstSemaphore), 1);  // create first
traffic lights semaphore

	    if (!firstSemaphoreId)
	        return -1;

	    firstThreadId =
osThreadCreate(osThread(firstTrafficLightThread), NULL);  // create
thread with traffic lights function

	    if (!firstThreadId)
	        return -1;

	    firstLightsCount = 2; // number of lights count
	    firstStatesCount = 3; // number of states count
	    firstThreadState = 0; // init state
	    firstTrafficLightsId = 1; // traffic light id

	    return 0;
	}
```

Далее перейдем непосредственно к определению функции потока (светофора). Предлагается следующая логика: цикл последовательно перебирает массив состояний светофора. Следует учесть, что потоки (светофоры) имеют возможность принудительной установки состояния другого (внешнего) светофора, поэтому ожидание переключения между состояниями (обычно исполняется функцией osDelay(delay) - системная функция задержки (переводит поток в режим ожидания))

предлагается реализовать как множество кратковременных задержек с проверкой значения состояния светофора (на случай изменения ее другим светофором). Пример логики ожидания:

```
int customDelay(unsigned int microDelay, unsigned int totalDelay,
int stateCopy, volatile int* state) {
    unsigned int currentDelayTime;
    currentDelayTime = 0;

    while (currentDelayTime < totalDelay) {
        if (stateCopy != *state) {
            return 1;
        }

        osDelay(microDelay);

        currentDelayTime = currentDelayTime + microDelay;
    }

    return 0;
}
```

Соответственно, в функцию передаются общая задержка, размер краткосрочной задержки (обычно 50 - 100 мс), копия состояния светофора и указатель на значение состояния светофора. При неравенстве копии и разыменованного указателя прерывается ожидание, светофор немедленно возобновляет работу. Заметим, что указатель помечен ключевым словом volatile, указывающим компилятору не использовать любые оптимизации для этой переменной, т.к. она может изменяться извне. Это помогает избежать неопределенного поведения работы функции.

Перейдем непосредственно к функции, выполняющейся в потоке (имитация работы светофора), ее реализацией может стать следующий код:

```
void firstTrafficLightThread (void const *argument) {
    int stateCopy;

    while (1) {
        osSemaphoreWait(firstSemaphoreId, osWaitForever);

        stateCopy = firstThreadState;

        osSemaphoreRelease(firstSemaphoreId);

        switch (stateCopy) {
            case 0: {
```

```
processLights(firstThreadStates[stateCopy].lights, firstLightsCount,
firstTrafficLightsId);
                    break;
                }
                case 1: {

processLights(firstThreadStates[stateCopy].lights, firstLightsCount,
firstTrafficLightsId);

                    break;
                }
                case 2: {

processLights(firstThreadStates[stateCopy].lights, firstLightsCount,
firstTrafficLightsId);

                    osSemaphoreWait(secondSemaphoreId,
osWaitForever);

                    secondThreadState = 0;

                    osSemaphoreRelease(secondSemaphoreId);

                    break;
                }
            }

            int result = customDelay(microDelay,
firstThreadStates[stateCopy].delay, stateCopy, &firstThreadState);

            if (result != 0) continue;

            osSemaphoreWait(firstSemaphoreId, osWaitForever);

            firstThreadState = (firstThreadState + 1) %
firstStatesCount;

            osSemaphoreRelease(firstSemaphoreId);
        }
```

Стоит уделить внимание функции, обрабатывающей текущее состояние потока (функция processLights). Внутренняя логика предусматривает выделение динамической

памяти (посредством Memory Pool) для сообщения с информацией о состоянии светофора. Данное сообщение отправляется в очередь сообщений (Message Queue). Например, функция обработки состояния потока может выглядеть следующим образом:

```
	void processLights(Ligth const* lights, unsigned int const count, unsigned char threadId) {
		Message* message = (Message*)allocMemory(); //allocate memory for message

		if (message != NULL) {
			message->lights = lights;
			message->count = count;
			message->threadId = threadId;

			sendMessage(message);
		}
	}
```

Реализацией же функции по выделению динамической памяти и отправке сообщений в очередь могут служить:

```
	void sendMessage(Message* message) {
		osMessagePut(messageQueue, (uint32_t) message, osWaitForever);  // Send Message
	}

	void* allocMemory() {
		return osPoolCAlloc(memoryPool); // Allocate memory
	}
```

После отправления сообщения в очередь необходимо обработать его. Как уже было сказано ранее, одной из функций CMSIS RTOS RTX является osMessageGet(), позволяющая получать сообщения из очереди. Все что нам нужно - получить сообщение, вывести информацию в окно вывода и очистить выделенную память:

```
	void consumeMessageThread(void const *argument) {
		Message* message;
		osEvent  evt;

		// Start thread of consuming messages.
		for (;;) {
			evt = osMessageGet(messageQueue, osWaitForever);  // wait for message
			if (evt.status == osEventMessage) {
				message = (Message*)evt.value.v;  // we can get valid message.
```

```
                printf("Thread id %02d:\t", message->threadId);

                unsigned int count;
                count = message->count;

                for (int i = 0; i < count; i++) {
                    printf("%s\t", message->lights[i].color);
                }

                printf("\n");

                osStatus status;
                status = osPoolFree(memoryPool, message);  // clear
the dynamic memory

                if (status != osOK) {
                    printf("Error clearing allocated memory");
                }
            }
        }
    }
```

1.3.3 Работа кнопок

Перейдем к логике работы кнопок. Как уже было сказано ранее кнопка К1 - незамедлительно переключает состояние каждого светофора на следующее, К2 - принудительно устанавливает состояния светофоров в назначенные положения. Предусмотрена работа на симуляторе, следовательно придется эмулировать работу кнопок. Для добавления виртуальных кнопок необходимо сделать одно из:

- Настроить проект, а именно добавить файл .ini в настройках проекта (окно Debug) - рассмотрено в пункте настройки проекта.
- Непосредственно в отладке проекта добавить файл инициализации (рассмотрено ниже).

При начале отладки проекта (Ctrl + F5) открываем окно настройки функций: Debug >> Function Editor. Далее либо открываем готовый файл .ini, либо вручную вводим настройки, пример которых был указан выше, далее сохраняем (кнопка Save) и применяем (кнопка Compile). Данная конфигурация позволяет имитировать работу кнопок, изменяя значения регистров IDR4 и IDR6 порта ввода/вывода GPIOC.

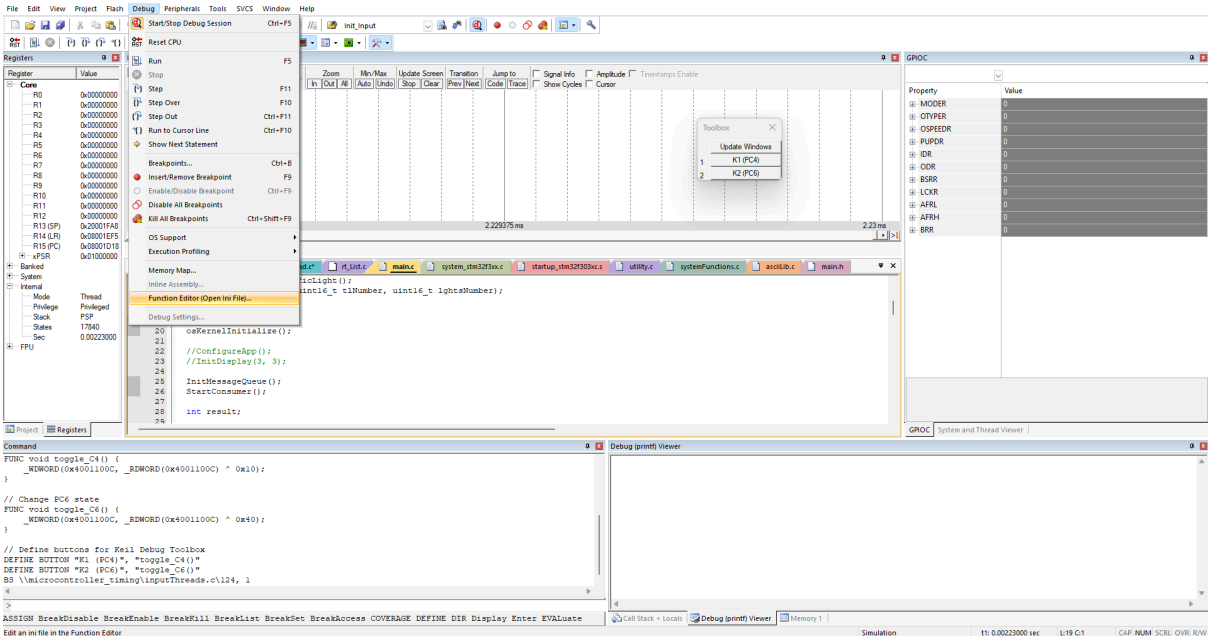


Рисунок. Открытие окна настроек отладки.

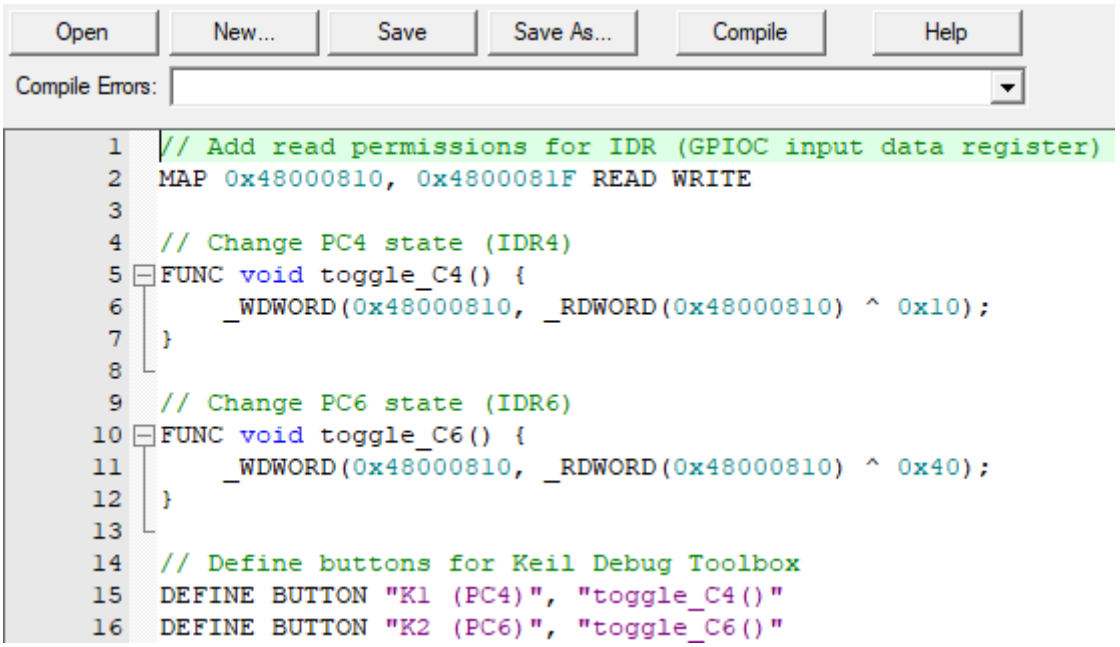


Рисунок. Настройка работы виртуальных кнопок.

После успешной настройки работы виртуальных кнопок во время отладки можем вызвать окно инструментов: View >> Toolbox Window. Во время отладки можем нажимать кнопки в панели и наблюдать как изменяются значения регистра IDR.

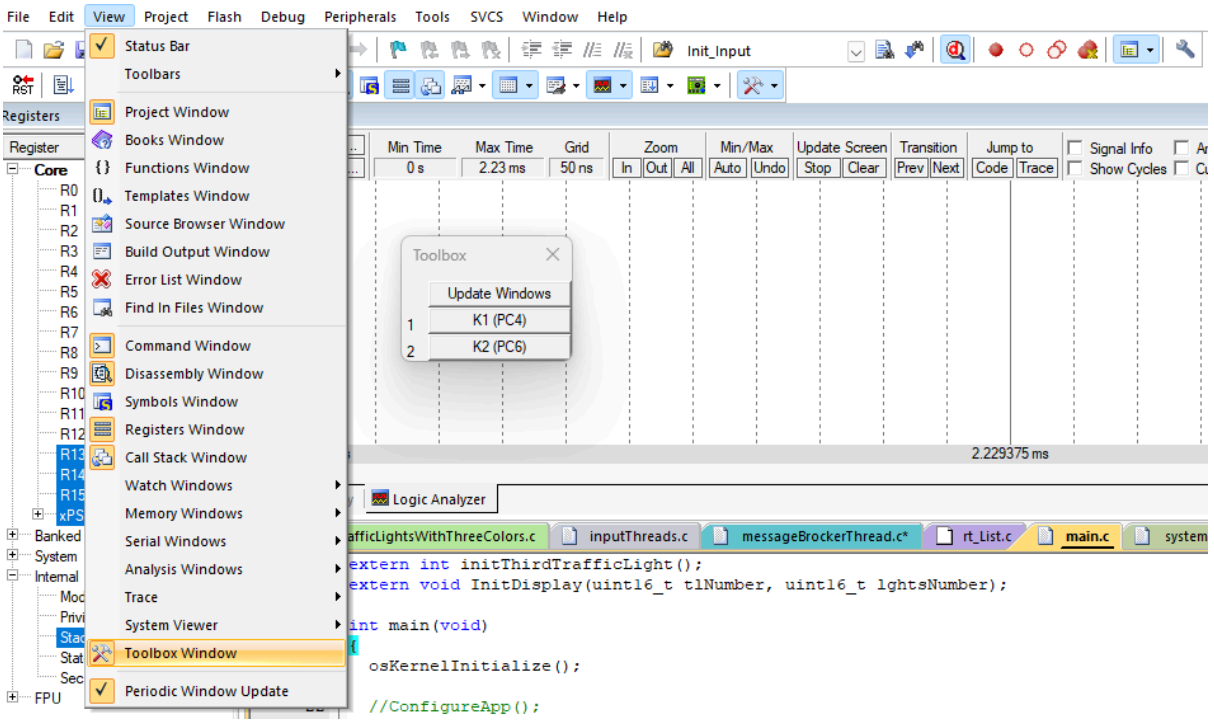


Рисунок. Вызов панели инструментов.

Теперь, имея возможность изменять значения регистра с помощью виртуальных кнопок, переходим к непосредственной реализации логики использования кнопок. Кнопка K1 должна незамедлительно переключать состояние каждого светофора на следующее. Предлагается следующая логика: в цикле проверяем значение регистра, и если оно равно единице, увеличиваем значение всех состояний потоков на единицу:

```
void Input_K1_Thread (void const *argument) {
    while (1) {
        // Check is PC4 (k1) pressed
        if ((GPIOC->IDR & GPIO_IDR_4) == 0) {
            // k1 pressed -> k2 cannot be checked
            osSemaphoreWait(semaphore_input_id, osWaitForever);
// wait for ability to check k1 (when k2 is not pressed)

            // Debonce delay
            osDelay(50);

            if ((GPIOC->IDR & GPIO_IDR_4) != 0) {
                // k2 now available
                osSemaphoreRelease(semaphore_input_id);
                continue;
            }

            do {
                // Increase states
```

```
                    osSemaphoreWait(firstSemaphoreId, osWaitForever);
                    firstThreadState++;
                    osSemaphoreRelease(firstSemaphoreId);

                    osSemaphoreWait(secondSemaphoreId,
osWaitForever);
                    secondThreadState++;
                    osSemaphoreRelease(secondSemaphoreId);


                    osSemaphoreWait(thirdSemaphoreId, osWaitForever);
                    secondThreadState++;
                    osSemaphoreRelease(thirdSemaphoreId);

                    // Sleep for 100 ms
                    osDelay(100);
                } while ((GPIOC->IDR & GPIO_IDR_4) == 0);

                // k1 unpressed
                // k2 now available
                osSemaphoreRelease(semaphore_input_id);
            }

            osDelay(50);
        }
    }
```

Кнопка К2 принудительно устанавливает состояния светофоров в назначенные положения, например, начальные. Логика схожа с первой кнопкой, но вместо увеличения состояния на единицу, присваиваем конкретное значение:

```
    void Input_K2_Thread (void const *argument) {
        while (1) {
            // Check is PC6 (k2) pressed
            if ((GPIOC->IDR & GPIO_IDR_6) == 0) {
                // k2 pressed -> k1 cannot be checked
                osSemaphoreWait(semaphore_input_id, osWaitForever);
// wait for ability to check k2 (when k1 is not pressed)

                // Debounce delay
                osDelay(50);

                if ((GPIOC->IDR & GPIO_IDR_6) != 0) {
                    // k2 now available
```

```
                    osSemaphoreRelease(semaphore_input_id);
                    continue;
                }

                do {
                    // Set states to initial
                    osSemaphoreWait(firstSemaphoreId, osWaitForever);
                    firstThreadState = 0;
                    osSemaphoreRelease(firstSemaphoreId);

                    osSemaphoreWait(secondSemaphoreId,
osWaitForever);
                    secondThreadState = 0;
                    osSemaphoreRelease(secondSemaphoreId);


                    osSemaphoreWait(thirdSemaphoreId, osWaitForever);
                    thirdThreadState = 0;
                    osSemaphoreRelease(thirdSemaphoreId);

                    // Sleep for 100 ms
                    osDelay(100);
                } while ((GPIOC->IDR & GPIO_IDR_6) == 0);

                // k2 unpressed
                // k1 now available
                osSemaphoreRelease(semaphore_input_id);
            }

            osDelay(50);
        }
    }
```

1.3.4 Запуск приложения

После реализации логики всех элементов необходимо запустить приложение. Это осуществляется в функции main. Изначально вызывается функция osKernelInitialize(), которая инициализирует ядро операционной системы. После успешной инициализации (osKernelInitialize() возвращает один из статус - кодов) возможно обращаться к ресурсам операционной системы (создание потоков, семафоров и т.п.). Далее следует вызов функции osKernelStart(), которая запускает планировщик задач операционной системы. Для нашего случая это может быть:

```
#include "RTE_Components.h"
#include CMSIS_device_header
```

```c
#include "cmsis_os.h"

void ConfigureApp();
extern int InitMessageQueue();
extern int InitMemoryPool();
extern int StartConsumer();
extern int Init_Input (void);
extern int initFirstTrafficLight();
extern int initSecondTrafficLight();

int main(void)
{
    osKernelInitialize();

    int result;

    result = InitMessageQueue();

    if (result != 0 )
        return -1;

    result = InitMemoryPool();

    if (result != 0 )
        return -1;

    result = StartConsumer();

    if (result != 0 )
        return -1;

    result = Init_Input();

    if (result != 0 )
        return -1;

    result = initFirstTrafficLight();

    if (result != 0 )
        return -1;

    result = initSecondTrafficLight();
```

```
    if (result != 0 )
        return -1;

    osKernelStart();

    while(1) { }
}
```

Для понимания в каком состоянии находятся созданные потоки можно воспользоваться встроенным окном просмотра состояний потоков и системы, для его вызова необходимо следовать навигации: Debug >> OS Support >> System and Thread Viewer:

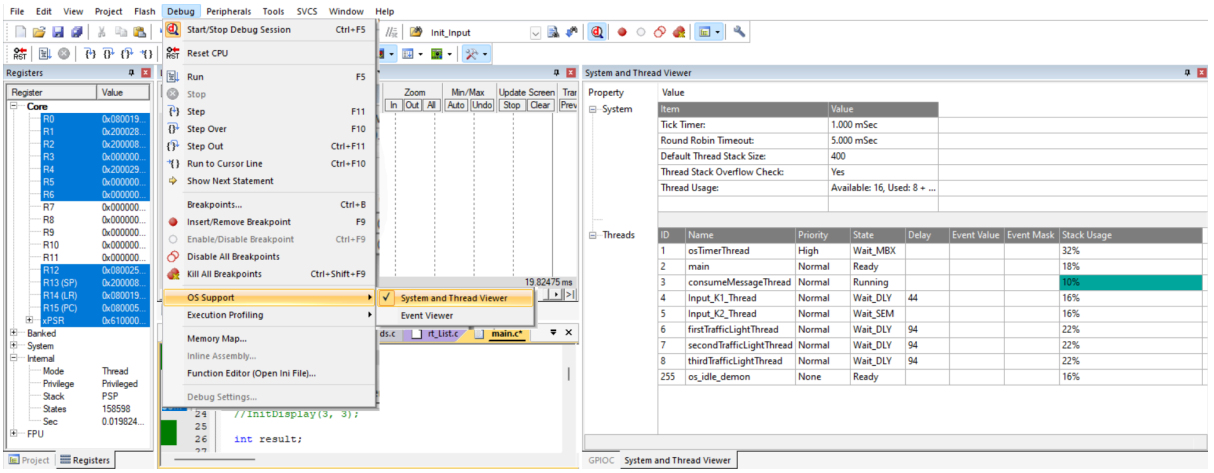


Рисунок. Вызов окна просмотра состояний потоков и системы.

После запуска приложения в окне вывода сможем наблюдать отображаемые состояния потоков, например:

Рисунок. Пример вывода состояний потоков.

1.3.5 Дополнительная настройка операционной системы

При настройке окружения проекта и включении элементов CMSIS RTOS автоматически добавится конфигурационный файл RTX_Conf_CM.c:

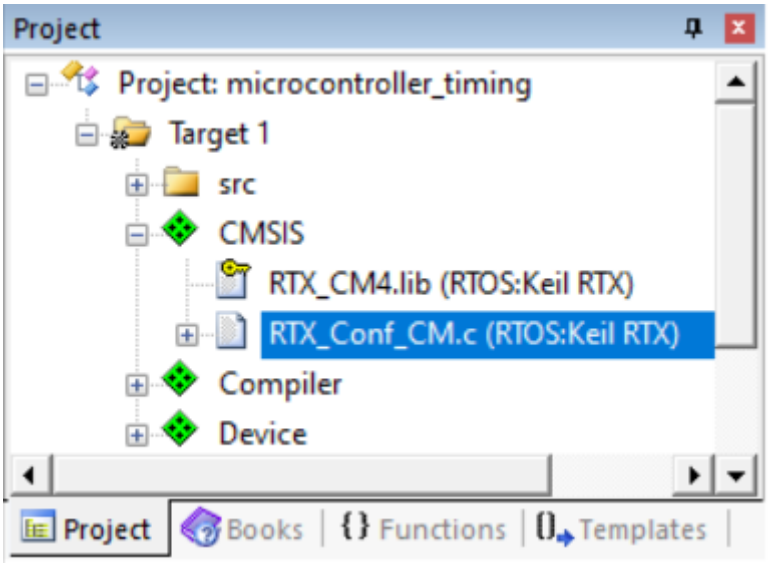


Рисунок. Файл конфигурации CMSIS RTOS.

Файл содержит основные настройки операционной системы, например: количество потоков, мьютексов, размер стека потоков, проверка переполнения стека потоков, настройка планировщика.

В том случае, если потоки не создаются, не запускаются или останавливаются в процессе своего выполнения, наиболее вероятная проблема кроется именно в конфигурации операционной системы. Для ее решения следует изменить некоторые настройки, например в рассмотренной выше реализации лабораторной работы одновременно могут быть запущены 9 потоков, тогда как по умолчанию конфигурация ОС поддерживает только 6. Поэтому максимальное число потоков необходимо увеличить:

```
// <h>Thread Configuration
// ======================
//
//   <o>Number of concurrent running user threads <1-250>
//   <i> Defines max. number of user threads that will run at the same time.
//   <i> Default: 6
#ifndef OS_TASKCNT
 #define OS_TASKCNT     15
#endif
```

Аналогичная ситуация может возникнуть с ограничением на количество мьютексов. Если реализация подразумевает большее число примитивов синхронизации, чем указано в конфигурации по умолчанию, то необходимо изменить настройку:

```
// Standard library system mutexes
// ===============================
//  Define max. number system mutexes that are used to protect
//  the arm standard runtime library. For microlib they are not used.
#ifndef OS_MUTEXCNT
 #define OS_MUTEXCNT    10
#endif
```

Размер стека потоков по умолчанию (то есть размер стека для тех потоков, при создании которых он не был указан в качестве аргумента) возможно придется уменьшить или увеличить. В первую очередь, необходимо учитывать размер стека программы, указанный в настройках проекта (файл Startup.h). Если потоки будут требовать больше пространства в памяти, чем им может быть предоставлено, то их создание станет невозможным. С другой стороны, слишком малый размер стека потоков может сделать невозможным их использование, т.к. для размещения переменных просто не хватит памяти:

```
//   <o>Default Thread stack size [bytes] <64-4096:8><#/4>
//   <i> Defines default stack size for threads with osThreadDef stacksz = 0
//   <i> Default: 200
#ifndef OS_STKSIZE
 #define OS_STKSIZE     100      // this stack size value is in words
#endif
```

Литература

1. Операционная система реального времени Keil RTX. URL: https://arm-software.github.io/CMSIS_5/RTOS/html/rtxImplementation.html
2. Таненбаум Э. - Современные операционные системы (Классика Computer Science) - 2015 г. URL - https://github.com/AVGaidai/Technical_Literature/blob/master/Operating%20Systems/Таненбаум%20Э.%20-%20Современные%20операционные%20системы%20(Классика%20Computer%20Science)%20-%202015.pdf
3. Пример выполнения лабораторной работы с использованием LCD дисплея: https://github.com/nikarpoff/rtos-stm32-traffic-lights

Приложение. Варианты.

Вариант 1: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	-
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 3 в RD YD GG
RD YD GG, RD YD GB, RG YD GD	4	1 переключает 2 в RB YD GD

Вариант 2: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	4 переключает 1 в RD YD
RD YG, RD YB, RD YB, RD YG	3	2 переключает 3 в RD YG
RD YD, RD YD, RD YD, RG YD	2	1 переключает 4 в RD YG

Вариант 3: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD

- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	7 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	3 переключает 5 в YD
RD, YD, GD, RD, YD, GD, RD	3	1 переключает 4 в RB

Вариант 4: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	1 переключает 3 в RD YG GD
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 1 в RG YD GD
RD YD GG, RD YD GB, RG YD GD	4	-

Вариант 5: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	-
RD YG, RD YB, RD YB, RD YG	3	1 переключает 2 в RD YD
RD YD, RD YD, RD YD, RG YD	2	4 переключает 3 в RD YB

Вариант 6: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	3 переключает 4 в RB
RB, YB, GB, RB, YB, GB, RB	1	1 переключает 2 в YD
RD, YD, GD, RD, YD, GD, RD	3	5 переключает 6 в GB

Вариант 7: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	1 переключает 3 в RD YG GD

RD YG GD, RD YB GD, RD YG GD	2	-
RD YD GG, RD YD GB, RG YD GD	4	3 переключает 1 в RG YD GD

Вариант 8: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	-
RD YG, RD YB, RD YB, RD YG	3	1 переключает 2 в RD YD
RD YD, RD YD, RD YD, RG YD	2	4 переключает 3 в RD YB

Вариант 9: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	1 переключает 3 в GB
RB, YB, GB, RB, YB, GB, RB	1	5 переключает 7 в RD

RD, YD, GD, RD, YD, GD, RD	3	4 переключает 2 в YB

Вариант 10: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	1 переключает 3 в RD YG GD
RD YG GD, RD YB GD, RD YG GD	2	3 переключает 2 в RB YD GD
RD YD GG, RD YD GB, RG YD GD	4	2 переключает 3 в RD YG GD

Вариант 11: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	1 переключает 2 в RD YD
RD YG, RD YB, RD YB, RD YG	3	2 переключает 3 в RD YD
RD YD, RD YD, RD YD, RG YD	2	3 переключает 4 в RD YG

Вариант 12: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD

- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	5 переключает 1 в RB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 6 в GD
RD, YD, GD, RD, YD, GD, RD	3	4 переключает 7 в RD

Вариант 13: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	1 переключает 3 в RG YD GD
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 1 в RG YD GD
RD YD GG, RD YD GB, RG YD GD	4	3 переключает 1 в RG YD GD

Вариант 14: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**

RG YD, RB YD, RD YG, RD YD	2	4 переключает 2 в RD YB
RD YG, RD YB, RD YB, RD YG	3	2 переключает 1 в RG YD
RD YD, RD YD, RD YD, RG YD	2	1 переключает 3 в RD YB

Вариант 15: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	4 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 1 в RD
RD, YD, GD, RD, YD, GD, RD	3	1 переключает 3 в GB

Вариант 16: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	3 переключает 1 в RD YG GD
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 3 в RD YB GD

RD YD GG, RD YD GB, RG YD GD	4	1 переключает 3 в RD YD GG

Вариант 17: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	4 переключает 2 в RD YB
RD YG, RD YB, RD YB, RD YG	3	-
RD YD, RD YD, RD YD, RG YD	2	1 переключает 3 в RD YG

Вариант 18: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	4 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 1 в RD
RD, YD, GD, RD, YD, GD, RD	3	1 переключает 3 в GB

Вариант 19: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	3 переключает 1 в RD YG GD
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 3 в RD YB GD
RD YD GG, RD YD GB, RG YD GD	4	1 переключает 3 в RD YD GG

Вариант 20: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	4 переключает 2 в RD YB
RD YG, RD YB, RD YB, RD YG	3	2 переключает 1 в RG YD
RD YD, RD YD, RD YD, RG YD	2	1 переключает 3 в RD YD

Вариант 21: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	5 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 4 в RD
RD, YD, GD, RD, YD, GD, RD	3	3 переключает 1 в RD

Вариант 22: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	3 переключает 1 в RD YG GD
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 3 в RG YD GD
RD YD GG, RD YD GB, RG YD GD	4	1 переключает 3 в RD YD GG

Вариант 23: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	4 переключает 2 в RD YB
RD YG, RD YB, RD YB, RD YG	3	2 переключает 1 в RG YD

RD YD, RD YD, RD YD, RG YD	2	1 переключает 3 в RD YD

Вариант 24: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	5 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 4 в RD
RD, YD, GD, RD, YD, GD, RD	3	3 переключает 1 в RD

Вариант 25: Три трехцветных светофора

- **Светофор 1**: RG YD GD -> RD YG GD -> RD YD GG
- **Светофор 2**: RB YD GD -> RD YB GD -> RD YD GB
- **Светофор 3**: RD YD GG -> RD YG GD -> RG YD GD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD GD, RB YD GD, RD YD GG	3	3 переключает 1 в RD YD GG
RD YG GD, RD YB GD, RD YG GD	2	2 переключает 1 в RG YD GD
RD YD GG, RD YD GB, RG YD GD	4	1 переключает 3 в RD YG GD

Вариант 26: Четыре двухцветных светофора

- **Светофор 1**: RG YD -> RD YG -> RD YD
- **Светофор 2**: RB YD -> RD YB -> RD YD
- **Светофор 3**: RD YG -> RD YB -> RD YD
- **Светофор 4**: RD YD -> RD YG -> RG YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG YD, RB YD, RD YG, RD YD	2	-
RD YG, RD YB, RD YB, RD YG	3	2 переключает 1 в RG YD
RD YD, RD YD, RD YD, RG YD	2	1 переключает 3 в RD YD

Вариант 27: Семь одноцветных светофоров

- **Светофор 1**: RD -> RB -> RD
- **Светофор 2**: YD -> YB -> YD
- **Светофор 3**: GD -> GB -> GD
- **Светофор 4**: RD -> RB -> RD
- **Светофор 5**: YD -> YB -> YD
- **Светофор 6**: GD -> GB -> GD
- **Светофор 7**: RD -> RB -> RD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RD, YD, GD, RD, YD, GD, RD	2	5 переключает 2 в YB
RB, YB, GB, RB, YB, GB, RB	1	2 переключает 4 в RD
RD, YD, GD, RD, YD, GD, RD	3	3 переключает 1 в RD

Вариант 37: Два одноцветных светофора

- **Светофор 1**: RG -> RB -> RD

- **Светофор 2**: YB -> YG -> YD

Состояние светофоров	**Время (сек)**	**Принудительное переключение**
RG, YB	3	-
RB, YG	2	1 переключает 2 в YB
RD, YD	4	2 переключает 1 в RB