ГУАП

КАФЕДРА № 23

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

доц. каф. 23, к.т.н., доц.		А. Л. Ляшенко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

МОДЕЛИРОВАНИЕ ТАЙМЕРОВ МИКРОКОНТРОЛЛЕРА STM32 В MATLAB С ВОЗМОЖНОСТЬЮ ОТЛАДКИ

по курсу: ИНТЕЛЛЕКТУАЛЬНЫЕ ЭЛЕКТРОННЫЕ ДАТЧИКИ И УСТРОЙСТВА ИНДИКАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 2337М		А. В. Разваляев
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Цель работы: Разработка симулятора микроконтроллера STM32F407 и его периферийных таймеров, интегрированного в среду MATLAB, для моделирования и отладки программного обеспечения микроконтроллеров

1 Обзор симуляторов микроконтроллеров STM32

Современные микроконтроллеры (МК) широко используются в различных областях, от автоматизации процессов до встраиваемых систем. Однако, тестирование и отладка программного обеспечения для МК требуют наличия физического оборудования, что может быть дорогостоящим и неудобным. Симуляция МК в программной среде, такой как MATLAB, может значительно упростить процесс разработки и отладки, а также позволить тестировать системы в виртуальной среде без физического устройства.

Уже существуют несколько готовых решений.

QEMU (Quick Emulator) – это универсальный симулятор, который поддерживает множество архитектур, включая ARM, на которой основаны микроконтроллеры STM32. Он используется для симуляции не только микроконтроллеров, но и целых компьютерных систем, что делает его очень гибким инструментом. Тем не менее QEMU моделирует только часть периферии микроконтроллера. А также он не имеет интегрируется в MATLAB и довольно сложен в настройке. Что делает процесс отладки и моделирования еще сложнее.

Proteus – система автоматизированного проектирования. Позволяет моделировать микроконтроллеры с визуализацией схем. Proteus также не моделирует всю периферию, но позволяет произвести моделирование МК вместе с схемой, в которую он встроен. Таким образом можно моделировать различные сценарии работы конечного устройства.

Но приведенные варианты симуляции микроконтроллеров не имеют средств отладки исходного кода. Для симуляции необходим уже скомпилированный файл для прошивки МК.

Как видно существующие инструменты для симуляции и моделирования МК не всегда обладают достаточной функциональностью для полноценного тестирования и отладки. Например, некоторые симуляторы могут не обеспечивать моделирование необходимых периферийных устройств. Или же они не интегрируются с популярными средами разработки, такими как MATLAB/Proteus, что не позволяет смоделировать работу всей системы. Это создает трудности для разработчиков, которым необходимо тестировать приложения в виртуальной среде.

1 Разработка симулятора микроконтроллеров в MATLAB

Разработанный симулятор выполняет программу МК в MATLAB, а также симулирует периферию МК. Основное отличие от существующих решений заключается в возможности отладки кода на языке С, а также пользовательская реализация периферии. Это позволяет делать упрощенные симуляции периферии, если это не влияет на работу приложения МК.

Для проведения исследований использованы следующие методы и инструменты:

- MATLAB/Simulink для создания и интеграции симулятора.

- для отладки программы МК в Simulink и компилятор Microsoft Visual C++ (MSVC) для компиляции программы МК для MATLAB.

- стандартные библиотеки и драйверы для STM32, адаптированные для MATLAB.

Программы на большинство МК пишутся на языке программирования С. В MATLAB существует блок S-Function, который представляет собой функцию, написанную на языке C/C++ или MATLAB. Поэтому было решено использовать именно S-Function для портирования программы в MATLAB. Компиляция кода на языке C/C++ происходит с помощью компилятора MSVC. Данный компилятор является сторонним, но он интегрируется в MATLAB.

Основная сложность моделирования программы МК STM32 в MATLAB заключается в различиях принципа работы программ на МК и в среде симуляции MATLAB. В общем случае, программа на микроконтроллере представляет собой бесконечный цикл. Каждую итерацию бесконечного цикла, программа МК считывает порты ввода-вывода, обрабатывает данные, и формирует выходной сигнал на портах ввода-вывода.

S-Function в MATLAB представляет собой функцию, которая вызывается на каждом шаг симуляции. Эта функция принимает входные данные, и выполняет некоторые расчеты. В это время MATLAB ожидает завершения функция, и только после завершения функции формируются выходные данные и симуляция переходит на следующий шаг. Поэтому, если S-Function будет включать бесконечный цикл – симуляция просто зависнет.

Также, микроконтроллеры имеют доступ к периферийным устройствам, которые реализованы аппаратно. Их программы могут взаимодействовать с такими устройствами, как таймеры, АЦП, или UART через регистры периферии и прерывания. В MATLAB программы не имеют никакой периферии. Поэтому, например, если программа МК в MATLAB будет ожидать выставление флага переполнения у таймера, то симуляция также зависнет, т.к. этот флаг аппаратно не выставится.

В связи с этим, для симуляции МК в MATLAB необходимо:

- контролировать ход программы МК, чтобы в определенный момент завершить её выполнение и перейти на следующий шаг симуляции;

- реализовать симулятор периферии, который будет симулировать периферию и её регистры, чтобы программа МК могла полноценно работать.

Для контроля ходом программы МК, было решено поместить её в отдельный поток выполнения. Этот поток возобновляется в начале каждого шага симуляции и выполняется в течение некоторого времени, после чего приостанавливается, позволяя S-Function завершиться. Такой подход позволил создать модель, где программа микроконтроллера может выполняться вместе с основным процессом симуляции, не зависая при ожидании данных от периферийных устройств, которые в реальном МК управляются аппаратно.

Модуль, реализующий управление потоком приложения МК, приведен в приложениях А, Б

Симуляция периферийных устройств - второй важный аспект, который требует детальной проработки. Поскольку микроконтроллеры взаимодействуют с аппаратными компонентами через регистры и прерывания, возникла необходимость имитировать их работу в MATLAB.

Во-первых, все регистры в МК привязаны к определенным адресам, которые недоступны программе в MATLAB. Поэтому надо выделить отдельную область оперативной памяти в программе, чтобы там хранить регистры периферии. Также необходимо переопределить макросы так, чтобы обращение к регистрам происходило по новым адресам в оперативной памяти.

Во-вторых, отсутствие аппаратной периферии вынуждает писать модели для используемой периферии вручную. Можно максимально подробно прописать логику работы периферии, чтобы она полностью эмулировала реальную периферию МК. Или же можно написать упрощенную или абстрактную модель, которая не будет имитировать всех процессов в периферии, но её будет достаточно для симуляции программы.

В данной работе будет рассматриваться программа управления двигателями. В ней активно используются два периферийных устройства: таймеры и интерфейс UART.

Таймеры, которые используются для формирования ШИМ, должны быть симулированы максимально точно. Это связано с тем, что они играют ключевую роль в управлении двигателем и точность их работы существенно влияет на результат. Поэтому для их симуляции была разработана модель, в которой прописана вся основная логика таймера: режимы счета, каналы захвата/сравнения, вывод каналов на порты ввода/вывода. Такой подход позволяет быстро менять настройки ШИМ и по симуляции смотреть, как это будет влиять на устройство, без необходимости переписывать симулятор из-за изменения параметров таймера.

Модуль, для симуляции таймеров МК, приведен в приложениях В, Г.

Интерфейс UART используется для приема параметров ШИМ по протоколу MODBUS. В симуляции работа с этим интерфейсом была упрощена до считывания входов S-Function и записи напрямую в регистры Modbus. Это позволяет программе функционировать корректно и без необходимости моделировать всю сложную логику интерфейса и протокола. Это упрощение было сделано, чтобы снизить нагрузку на симулятор, так как управление двигателем не требует точного воспроизведения работы UART.

Модуль, для симуляции входов/выходов МК, приведен в приложениях Д, Е.

1 Моделирование разработанного симулятора в MATLAB

На рис. 1 приведена модель в Simulink для демонстрации работы симулятора.

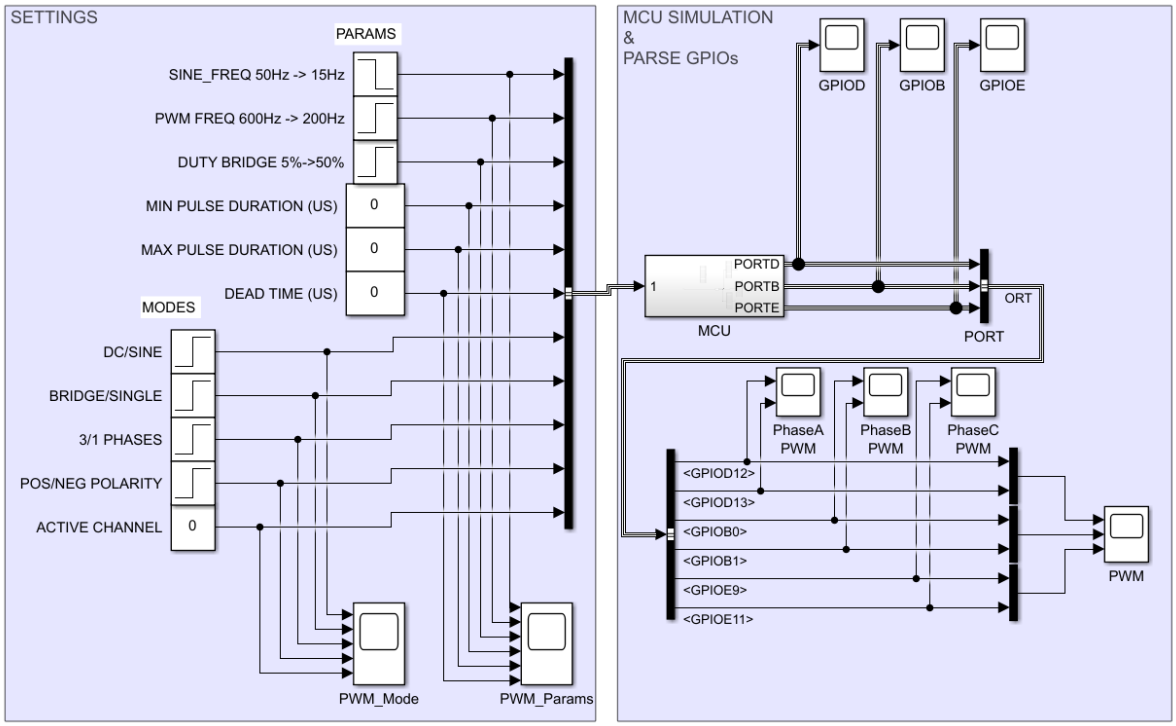

Рисунок 1 – Модель Simulink для демонстрации симулятора МК

На рис. 2 приведен результат моделирования программы МК для управления двигателями. Программа формирует ШИМ в соответствии с разными заданными параметрами, которые передаются в модель МК. Более подробно симуляция ШИМ рассмотрена на рис. 3-6.

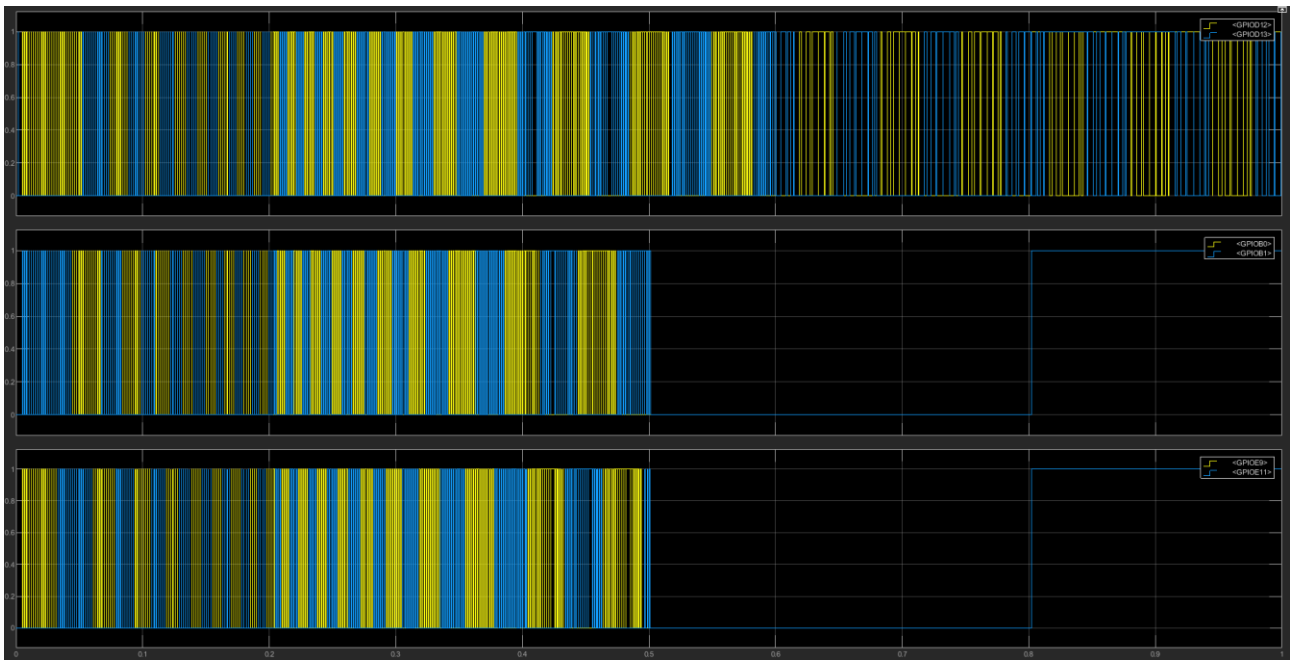

Рисунок 2 – Демонстрация симуляции МК в MATLAB

На рис. 3 приведен фрагмент симуляции трехфазного ШИМ с постоянным заполнением 5%, частотой ШИМ – 600 Гц, частота огибающей – постепенно растет от 0 Гц до 50 Гц.

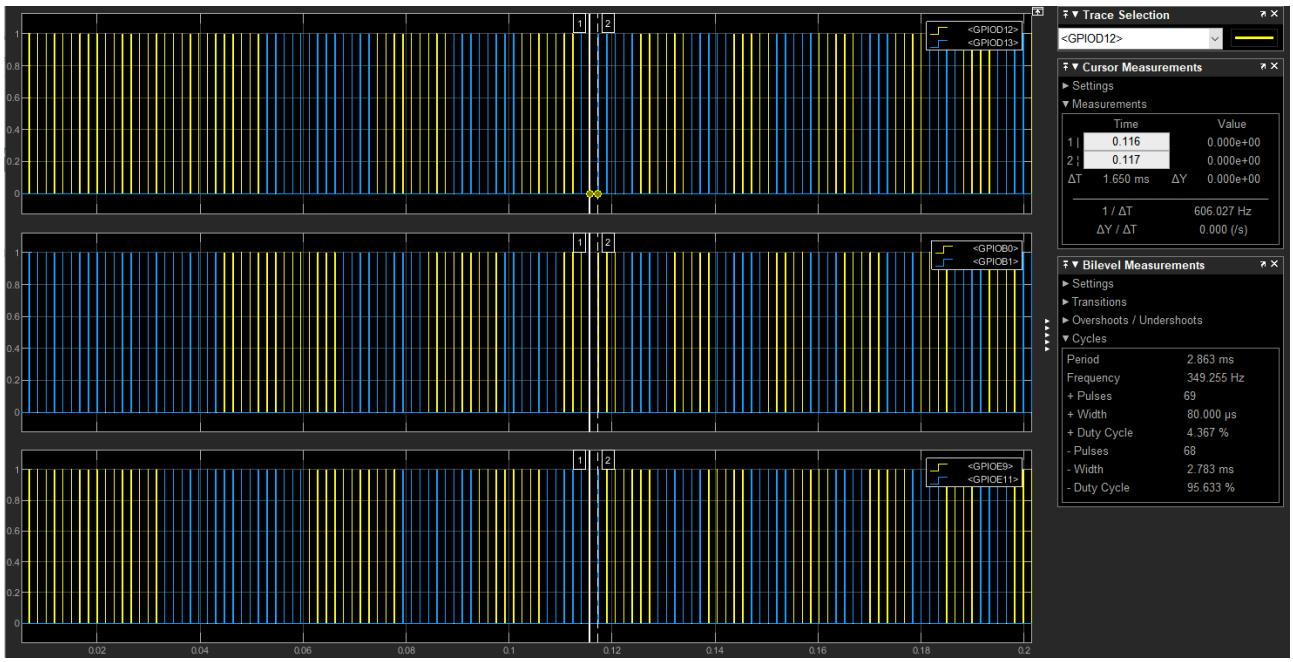

Рисунок 3 – Демонстрация симуляции трехфазного постоянного ШИМ с нарастанием частоты

На рис. 4 приведен фрагмент симуляции трехфазного ШИМ с постоянным заполнением 5%, частотой ШИМ – 600 Гц, частота огибающей – постепенно убывает от 50 Гц до 15 Гц.

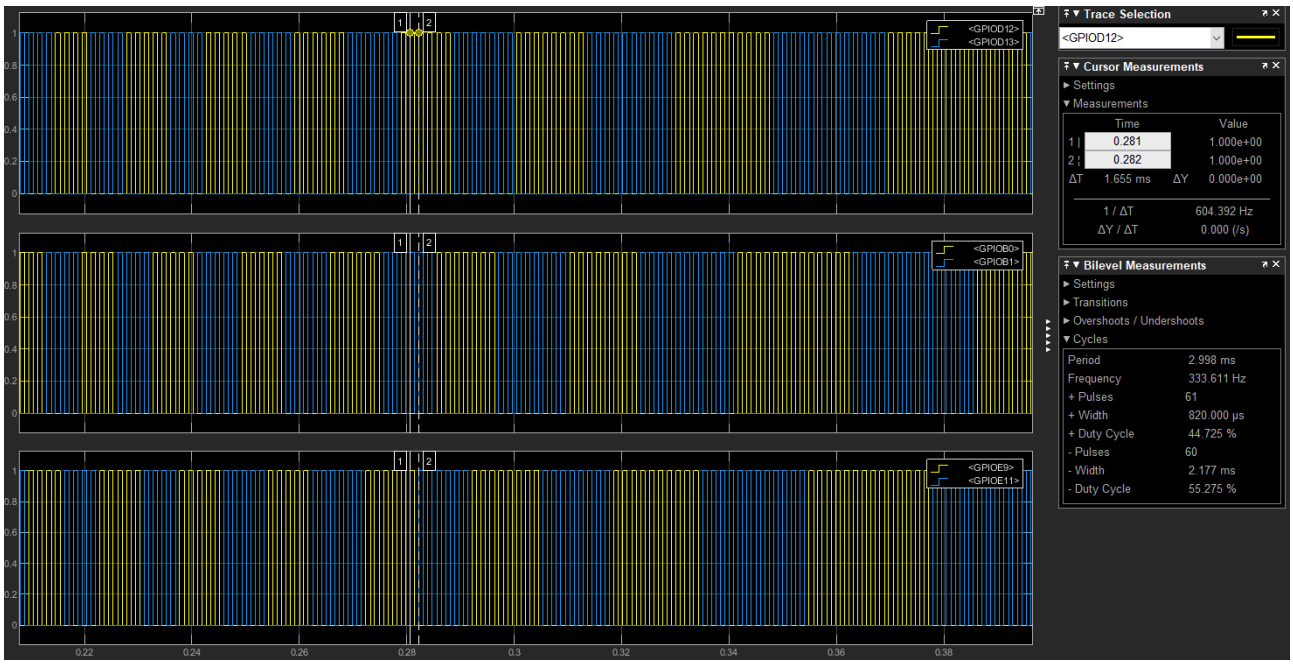

Рисунок 4 – Демонстрация симуляции трехфазного постоянного ШИМ с убыванием частоты

На рис. 5 продемонстрированы режимы генерации ШИМ по синусоидальному закону, где положительная/отрицательная полуволны выведены на разные каналы. Сначала формируется трехфазный ШИМ по синусоидальному закону, потом однофазный. Частота ШИМ – 600 Гц, частота огибающей – 15 Гц.

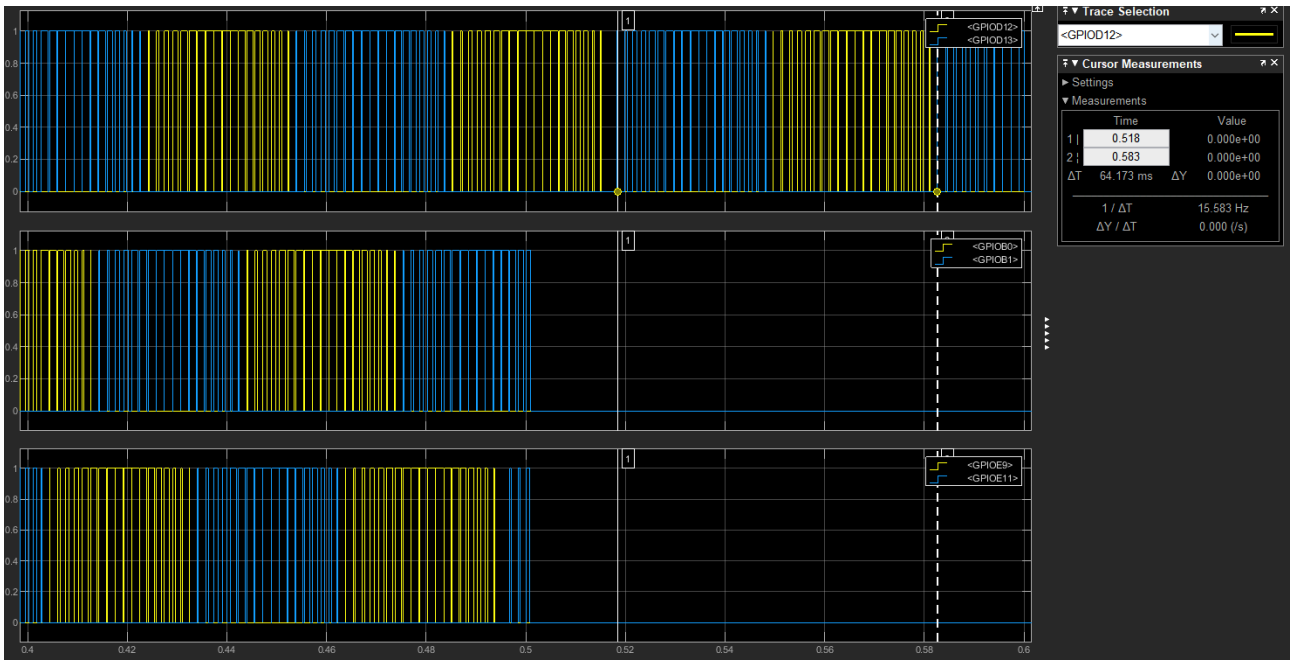

Рисунок 5 – Демонстрация симуляции трехфазного и однофазного ШИМ по синусоидальному закону

На рис. 6 продемонстрированы режимы генерации ШИМ по синусоидальному закону только на одной фазе. Сначала генерируется ШИМ с положительной полярностью, а потом с полярность меняется на противоположную. Полярность остальных фаз также меняется, Частота ШИМ – 200 Гц, частота огибающей – 15 Гц

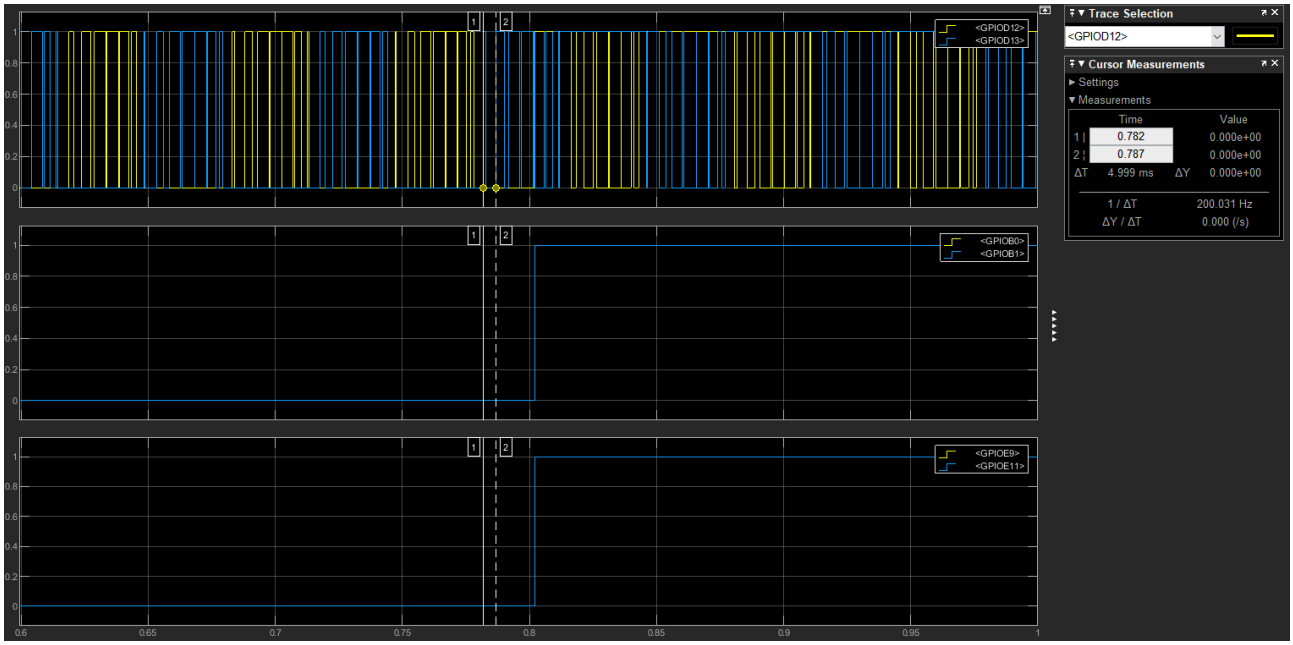

Рисунок 6 – Демонстрация смены полярности в симуляции однофазного ШИМ по синусоидальному закону

Компилятор MSVC позволяет скомпилировать код для отладки, следовательно появляется возможность отладки программы МК непосредственно в MATLAB. Для этого необходимо открыть папку с исходными файлами в среде разработки, например, Visual Studio. Далее перед запуском симуляции подключиться к процессу MATLAB. После этого при запуске симуляции появиться возможность отладки программы, т.е. возможность смотреть текущие значения всех доступных переменных и выполнять программу пошагово (рис. 7).

ЗАКЛЮЧЕНИЕ

Разработка симулятора микроконтроллеров для MATLAB продемонстрировала его эффективность в моделировании и тестировании программного обеспечения. Симулятор позволяет проводить отладку и тестирование без необходимости физического устройства, что упрощает процесс разработки. Результаты моделирования совпадают с экспериментальными:

- ШИМ корректной симулируется с заданной частотой 600 или 200 Гц (рис. 3, 4, 6);
- постоянная скважность ШИМ корректно симулируется в соответствии с заданной скважностью 5% или 50% (рис. 3, 4);
- переменная скважность ШИМ симулируется корректно (рис. 5, 6);
- частота вращения трехфазного ШИМ корректно симулируется в соответствии с заданной частотой 15 Гц (рис. 5).

В результате был написан функциональный симулятор периферийных таймеров МК STM32, который позволяет моделировать и тестировать программное обеспечение для МК в среде MATLAB. В дальнейшем планируется расширение функциональности симулятора для поддержки большего количества моделей микроконтроллеров, периферийных устройств и более комплексных программ МК.

ПРИЛОЖЕНИЕ А

Листинг mcu_wrapper.c

```
/**
******************************************************************************
* @file mcu_wrapper.c
* @brief Исходный код оболочки МК.
******************************************************************************
@details
Данный файл содержит функции для симуляции МК в Simulink (S-Function).
******************************************************************************/
#include "mcu_wrapper_conf.h"

/**
    * @addtogroup   WRAPPER_CONF
    * @{
    */

SIM__MCUHandleTypeDef hmcu;          ///< Хендл для управления потоком
программы МК

double SystemClockDouble = 0;        ///< Счетчик в формате double для точной
симуляции системных тиков С промежуточными значений
uint64_t SystemClock;                ///< Счетчик тактов для симуляции
системных тиков (в целочисленном формате)
double SystemClock_step = 0;         ///< Шаг тиков для их симуляции, в
формате double

/** MCU_WRAPPER
    * @}
    */
//-----------------------------------------------------------------//
//-----------------CONTROLLER SIMULATE FUNCTIONS---------------//
/**
  * @brief      Главная функция приложения МК.
  * @details    Функция с которой начинается выполнение кода МК. Выход из
данной функции происходит только в конце симуляции @ref mdlTerminate
  */
extern int main(void);                  // extern while from main.c
/**
  * @brief      Поток приложения МК.
  * @details    Поток, который запускает и выполняет код МК (@ref main).
  */
unsigned __stdcall MCU_App_Thread(void) {
    main();     // run MCU code
    return 0;   // end thread
    // note: this return will reached only at the end of simulation, when
all whiles will be skipped due to @ref sim_while
}

/**
  * @brief      Симуляция МК на один такт симуляции.
 *  @param      S    - указатель на структуру S-Function из "simstruc.h"
  * @param      time - текущее время симуляции.
  * @details    Запускает поток, который выполняет код МК и управляет ходом
потока:
  *             Если прошел таймаут, поток прерывается, симулируется
периферия
  *             и на следующем шаге поток возобнавляется.
  *
  *             Вызывается из mdlUpdate()
  */
void MCU_Step_Simulation(SimStruct* S, time_T time)
```

```
{
    MCU_readInputs(S); // считывание портов

    MCU_Periph_Simulation(); // simulate peripheral

    ResumeThread(hmcu.hMCUThread);
    for (int i = DEKSTOP_CYCLES_FOR_MCU_APP; i > 0; i--)
    {
    }
    SuspendThread(hmcu.hMCUThread);

    MCU_writeOutputs(S); // запись портов (по факту запись в буфер. запись в
порты в mdlOutputs)
}

/**
  * @brief      Симуляция периферии МК
  * @details    Пользовательский код, который симулирует работу периферии
МК.
  */
void MCU_Periph_Simulation(void)
{
    /* USER CODE FOR SIMULATING MCU PERIPHERAL START */

    /* USER CODE FOR SIMULATING MCU PERIPHERAL END */
}

/**
  * @brief      Считывание входов S-Function в порты ввода-вывода.
 *  @param      S - указатель на структуру S-Function из "simstruc.h"
  * @details    Пользовательский код, который записывает порты ввода-вывода
из входов S-Function.
  */
void MCU_readInputs(SimStruct* S)
{
    /* Get S-Function inputs */
    real_T* IN = ssGetInputPortRealSignal(S, 0);

    /* USER CODE FOR READING INPUTS START */

    /* USER CODE FOR READING INPUTS END */
}

/**
  * @brief      Запись портов ввода-вывода в буфер выхода S-Function
 *  @param      S - указатель на структуру S-Function из "simstruc.h"
  * @details    Пользовательский код, который записывает буфер выходов S-
Function из портов ввода-вывода.
  */
void MCU_writeOutputs(SimStruct* S)
{
    /* Get S-Function descrete array */
    real_T* Out_Buff = ssGetDiscStates(S);

    /* USER CODE FOR WRITING OUTPUT BUFFERS START */

    /* USER CODE FOR WRITING OUTPUT BUFFERS END */
}
//-----------------CONTROLLER SIMULATE FUNCTIONS---------------//
//------------------------------------------------------------//
```

```
//-------------------------------------------------------------//
//----------------------SIMULINK FUNCTIONS---------------------//
/**
  * @brief      Формирование выходов S-Function.
 *  @param      S - указатель на структуру S-Function из "simstruc.h"
  * @details    Пользовательский код, который записывает выходы S-Function
из буфера.
  */
void SIM_writeOutputs(SimStruct* S)
{
    real_T* GPIO;
    real_T* Out_Buff = ssGetDiscStates(S);

    //-------------WRITTING GPIOS---------------
    for (int j = 0; j < PORT_NUMB; j++)
    {
        GPIO = ssGetOutputPortRealSignal(S, j);
        for (int i = 0; i < PORT_WIDTH; i++)
        {
            GPIO[i] = Out_Buff[j * PORT_WIDTH + i];
            Out_Buff[j * PORT_WIDTH + i] = 0;
        }
    }
    //-----------------------------------------
}

/**
  * @brief      Инициализация симуляции МК.
  * @details    Пользовательский код, который создает поток для приложения
МК
               и настраивает симулятор МК для симуляции.
  */
void SIM_Initialize_Simulation(void)
{
    // инициализация потока, который будет выполнять код МК
    hmcu.hMCUThread = (HANDLE)CreateThread(NULL, 0, MCU_App_Thread, 0,
CREATE_SUSPENDED, &hmcu.idMCUThread);

    /* USER CODE FOR INITIALIZATION SIMULATOR START */

    /* USER CODE FOR INITIALIZATION SIMULATOR END */
}

/**
  * @brief      Деинициализация симуляции МК.
  * @details    Пользовательский код, который будет очищать все структуры
после окончания симуляции.
  */
void SIM_deInitialize_Simulation(void)
{
    /* USER CODE FOR DEINITIALIZATION SIMULATOR START */

    /* USER CODE FOR DEINITIALIZATION SIMULATOR END */
}
//-------------------------------------------------------------//
```

ПРИЛОЖЕНИЕ Б

Листинг mcu_wrapper_conf.h

```
/**
******************************************************************************
* @dir ../MCU_Wrapper
* @brief <b> Папка с исходным кодом оболочки МК. </b>
* @details
В этой папке содержаться оболочка(англ. wrapper) для запуска и контроля
эмуляции микроконтроллеров в MATLAB (любого МК, не только STM).
Оболочка представляет собой S-Function - блок в Simulink, который работает
по скомпилированому коду. Компиляция происходит с помощью MSVC-компилятора.
******************************************************************************/

/**
******************************************************************************
* @file mcu_wrapper_conf.h
* @brief Заголовочный файл для оболочки МК.
******************************************************************************
@details
Главный заголовочный файл для матлаба. Включает дейфайны для S-Function,
объявляет базовые функции для симуляции МК и подключает базовые библиотеки:
- для симуляции         "stm32fxxx_matlab_conf.h"
- для S-Function     "simstruc.h"
- для потоков         <process.h>
******************************************************************************/
#ifndef _CONTROLLER_H_
#define _CONTROLLER_H_

// Includes
#include "stm32f4xx_matlab_conf.h"  // For stm simulate functions
#include "simstruc.h"               // For S-Function variables
#include <process.h>                // For threads


/**
    * @defgroup MCU_WRAPPER     MCU Wrapper
    * @brief        Всякое для оболочки МК
    */

/**
    * @addtogroup   WRAPPER_CONF    Wrapper Configuration
    * @ingroup      MCU_WRAPPER
    * @brief        Параметры конфигурации для оболочки МК
    * @details      Здесь дефайнами задается параметры оболочки, которые
определяют как она будет работать
    * @{
    */

// Parametrs of MCU simulator
#define CREATE_SUSPENDED                0x00000004  ///< define from
WinBase.h. We dont wanna include "Windows.h" or smth like this, because of
HAL there are a lot of redefine errors.

#define DEKSTOP_CYCLES_FOR_MCU_APP      0xFFFF      ///< number of for()
cycles after which MCU thread would be suspended
#define PORT_WIDTH                      16          ///< width of one port
#define PORT_NUMB                       3           ///< amount of ports

// Parameters of S_Function
#define NPARAMS                 1                           ///< number of
input parametrs (only Ts)
```

```
#define IN_PORT_WIDTH           (8)                         ///< width of
input ports
#define IN_PORT_NUMB            1                           ///< number of
input ports
#define OUT_PORT_WIDTH          PORT_WIDTH                  ///< width of
output ports
#define OUT_PORT_NUMB           PORT_NUMB                   ///< number of
output ports
#define DISC_STATES_WIDTH       PORT_WIDTH*PORT_NUMB        ///< width of
discrete states array

/** WRAPPER_CONF
  * @}
  */


/**
  * @addtogroup MCU_WRAPPER
  * @{
  */

typedef void* HANDLE; ///< MCU handle typedef

/**
  * @brief  MCU handle Structure definition.
  * @note   Prefixes: h - handle, s - settings, f - flag
  */
typedef struct {
    // MCU Thread
    HANDLE          hMCUThread;                 ///< Хендл для потока МК
    uint32_t        idMCUThread;                ///< id потока МК (unused)
    // Flags
    unsigned        fMCU_Stop : 1;              ///< флаг для выхода из
потока программы МК
    double          SIM_Sample_Time;            ///< sample time of
simulation

}SIM__MCUHandleTypeDef;
extern SIM__MCUHandleTypeDef hmcu;              // extern для видимости
переменной во всех файлах

//-------------------------------------------------------------//
//------------------  SIMULINK WHILE DEFINES  -----------------//
/* DEFINE TO WHILE WITH SIMULINK WHILE */
/**
  * @brief      Redefine C while statement with sim_while() macro.
  * @param      _expression_ - expression for while.
  * @details    Это while который будет использоваться в симулинке @ref
sim_while для подробностей.
  */
#define while(_expression_)                         sim_while(_expression_)

/* SIMULINK WHILE */
/**
  * @brief      While statement for emulate MCU code in Simulink.
  * @param      _expression_ - expression for while.
  * @details    Данный while необходим, чтобы в конце симуляции, завершить
поток МК:
  *             При выставлении флага окончания симуляции, все while будут
пропускаться
  *             и поток сможет дойти до конца функции main и завершить себя.
  */
```

```
#define sim_while(_expression_)
while((_expression_)&&(hmcu.fMCU_Stop == 0))

/* DEFAULT WHILE */
/**
  * @brief      Default/Native C while statement.
  * @param      _expression_ - expression for while.
  * @details    Данный while - аналог обычного while, без дополнительного
функционала.
  */
#define native_while(_expression_)                    for(; (_expression_); )
  /*************************************************************/

//------------------  SIMULINK WHILE DEFINES  -----------------//
//-------------------------------------------------------------//



//-------------------------------------------------------------//
//---------------- SIMULATE FUNCTIONS PROTOTYPES  -------------//
/* Step simulation */
void MCU_Step_Simulation(SimStruct *S, time_T time);

/* MCU peripheral simulation */
void MCU_Periph_Simulation(void);

/* Initialize MCU simulation */
void SIM_Initialize_Simulation(void);

/* Deinitialize MCU simulation */
void SIM_deInitialize_Simulation(void);

/* Read inputs S-function  */
void MCU_readInputs(SimStruct* S);

/* Write pre-outputs S-function (out_buff states) */
void MCU_writeOutputs(SimStruct* S);

/* Write outputs of block of S-Function*/
void SIM_writeOutput(SimStruct* S);
//---------------- SIMULATE FUNCTIONS PROTOTYPES  -------------//
//-------------------------------------------------------------//

/** MCU_WRAPPER
    * @}
    */
#endif // _CONTROLLER_H_


//-------------------------------------------------------------//
//---------------------BAT FILE DESCRIBTION--------------------//
/**
  * @file run_mex.bat
  * @brief Батник для компиляции оболочки МК.
  * @details
  * Вызывается в матлабе из mexing.m.
  *
  * Исходный код батника:
  * @include
F:\Work\Projects\MATLAB\matlab_stm_emulate\MCU_Wrapper\run_mex.bat
  */
```

ПРИЛОЖЕНИЕ В

Листинг stm32f4xx_matlab_tim.c

```
/**
******************************************************************************
* @file stm32f4xx_matlab_tim.c
* @brief Исходный код симулятора таймеров.
******************************************************************************
@details
Данный файл содержит функции для симуляции таймеров STM32F407xx.
******************************************************************************/
#include "stm32f4xx_matlab_tim.h"



struct SlaveChannels Slave_Channels; ///< структура для связи и
синхронизации таймеров


//----------------------TIMER BASE FUNCTIONS----------------------//
/**
  * @brief    Симуляция таймера на один такт симуляции.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура таймера для симуляции.
  * @details  Это базовая функция для симуляции таймера: она вызывается
каждый шаг симуляции
  *           и вызывает все другие функции, необходимые для симуляции:
  *           - Overflow_Check()
  *           - Slave_Mode_Check_Source()
  *           - TIMx_Count()
  *           - Channels_Simulation()
  */
void TIM_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
{
    Overflow_Check(TIMx, TIMS);



    // Выбор режима работы таймера
    switch (TIMx->SMCR & TIM_SMCR_SMS) // TIMER MODE
    {
        // обычный счет
        case(TIM_SLAVEMODE_DISABLE):// NORMAL MODE counting
            TIMx_Count(TIMx, TIMS);
            Channels_Simulation(TIMx, TIMS); // CaptureCompare and PWM
channels simulation
            break;


        // включение слейв таймера по ивенту
        case(TIM_SLAVEMODE_TRIGGER): // SLAVE MODE: TRIGGER MODE
            Slave_Mode_Check_Source(TIMx, TIMS);
            TIMx_Count(TIMx, TIMS);
            Channels_Simulation(TIMx, TIMS); // CaptureCompare and PWM
channels simulation
            break;
    }

}
/**
  * @brief    Симуляция счетчика таймера на один такт симуляции.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура таймера для симуляции.
```

```
  * @details  Данная функция проверяет направление таймера и увеличивает или
уменьшает
  *           значение счетчика на то число, на которое оно бы увеличилось
за шаг симуляции.
  * @note     Для счетчика используется double формат, т.к. кол-во счетов за
шаг симуляции
              может быть дробным. После в конце функции double счетчик
записывает с округлением
              в регистр таймера CNT.
  */
void TIMx_Count(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS)
{
    if ((TIMx->CR1 & TIM_CR1_DIR) && TIMx->CR1) // up COUNTER and COUNTER
ENABLE
        TIMS->tx_cnt -= TIMS->tx_step / TIMx->PSC;
    else if (((TIMx->CR1 & TIM_CR1_DIR) == 0) && TIMx->CR1) // down COUNTER
and COUNTER ENABLE
        TIMS->tx_cnt += TIMS->tx_step / TIMx->PSC;
    TIMx->CNT = (uint32_t)TIMS->tx_cnt;
}

/**
  * @brief    Проверка на переполнение и дальнейшая его обработка.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура таймера для симуляции.
  * @details  Данная функция проверяет когда таймер переполниться и если
надо,
              вызывает соответствующее прерывание:
              - call_IRQHandller()
  */
void Overflow_Check(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS)
{
    // Переполнение таймера: сброс таймера и вызов прерывания
    if ((TIMx->CR1 & TIM_CR1_UDIS) == 0) // UPDATE enable
    {
        if ((TIMx->CR1 & TIM_CR1_ARPE) == 0)  TIMS->RELOAD = TIMx->ARR; //
PRELOAD disable - update ARR every itteration
        if (TIMS->tx_cnt > TIMS->RELOAD || TIMS->tx_cnt < 0) // OVERFLOW
        {
            TIMS->RELOAD = TIMx->ARR; // RELOAD ARR

            if (TIMS->tx_cnt > TIMx->ARR) // reset COUNTER
                TIMS->tx_cnt = 0;
            else if (TIMS->tx_cnt < 0)
                TIMS->tx_cnt = TIMx->ARR;

            if(TIMx->DIER & TIM_DIER_UIE) // if update interrupt enable
                call_IRQHandller(TIMx); // call HANDLER
        }
    }
}
//-----------------------------------------------------------------//


//----------------------------CHANNELS-----------------------------//
/**
  * @brief    Симуляция каналов таймера.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура таймера для симуляции.
  * @details  Данная функция симулирует работу всех каналов таймера.
  *           Она вызывает функции:
  *           - CC_PWM_Ch1_Simulation()
  *           - CC_PWM_Ch2_Simulation()
```

```
  *            - CC_PWM_Ch3_Simulation()
  *            - CC_PWM_Ch4_Simulation()
  *            - Write_OC_to_GPIO()
  *            - Write_OC_to_TRGO()
  */
void Channels_Simulation(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS)
{
    CC_PWM_Ch1_Simulation(TIMx, TIMS);
    CC_PWM_Ch2_Simulation(TIMx, TIMS);
    CC_PWM_Ch3_Simulation(TIMx, TIMS);
    CC_PWM_Ch4_Simulation(TIMx, TIMS);

    Write_OC_to_GPIO(TIMx, TIMS);

    Write_OC_to_TRGO(TIMx, TIMS);
}
//-----------------CAPTURE COPMARE & PWM FUNCTIONS-----------------//
/**
  * @brief    Выбор режима первого канала и его симуляция.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура таймера для симуляции.
  * @details  Данная функция по регистрам таймера проверяет как настроен
              первый канал и соответствующе симулирует его работу.
  */
void CC_PWM_Ch1_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
{ // определяет режим канала
switch (TIMx->CCMR1 & TIM_CCMR1_OC1M)
{
    case (TIM_OCMODE_ACTIVE): // ACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR1) < 2*TIMS->tx_step)
            TIMS->Channels.OC1REF = 1;
        break;

    case (TIM_OCMODE_INACTIVE): // INACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR1) < 2*TIMS->tx_step)
            TIMS->Channels.OC1REF = 0;
        break;

    case (TIM_OCMODE_TOGGLE): // TOOGLE mode
        if (abs(TIMx->CNT - TIMx->CCR1) < 2*TIMS->tx_step)
            TIMS->Channels.OC1REF = ~TIMS->Channels.OC1REF;
        break;

    case (TIM_OCMODE_PWM1): // PWM MODE 1 mode
        if (TIMx->CNT < TIMx->CCR1)
            TIMS->Channels.OC1REF = 1;
        else
            TIMS->Channels.OC1REF = 0;
        break;

    case (TIM_OCMODE_PWM2): // PWM MODE 2 mode
        if (TIMx->CNT < TIMx->CCR1)
            TIMS->Channels.OC1REF = 0;
        else
            TIMS->Channels.OC1REF = 1;
        break;

    case (TIM_OCMODE_FORCED_ACTIVE): // FORCED ACTIVE mode
        TIMS->Channels.OC1REF = 1; break;

    case (TIM_OCMODE_FORCED_INACTIVE): // FORCED INACTIVE mode
        TIMS->Channels.OC1REF = 0; break;
```

```
}
}
/**
  * @brief    Выбор режима второго канала и его симуляция.
  * @param    TIMx – таймер, каналы которого надо записать.
  * @param    TIMS – структура таймера для симуляции.
  * @details  Данная функция по регистрам таймера проверяет как настроен
              второй канал и соответствующе симулирует его работу.
  */
void CC_PWM_Ch2_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
{ // определяет режим канала
switch (TIMx->CCMR1 & TIM_CCMR1_OC2M)
{
    case ((TIM_OCMODE_ACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): // ACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR2) < 2*TIMS->tx_step)
            TIMS->Channels.OC2REF = 1;
        break;

    case ((TIM_OCMODE_INACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): // INACTIVE
mode
        if (abs(TIMx->CNT - TIMx->CCR2) < 2*TIMS->tx_step)
            TIMS->Channels.OC2REF = 0;
        break;

    case ((TIM_OCMODE_TOGGLE) << (TIM_OCMODE_SECOND_SHIFT)): // Toogle mode
        if (abs(TIMx->CNT - TIMx->CCR2) < 2*TIMS->tx_step)
            TIMS->Channels.OC2REF = ~TIMS->Channels.OC2REF;
        break;

    case ((TIM_OCMODE_PWM1) << (TIM_OCMODE_SECOND_SHIFT)): // PWM mode 1
mode
        if (TIMx->CNT < TIMx->CCR2)
            TIMS->Channels.OC2REF = 1;
        else
            TIMS->Channels.OC2REF = 0;
        break;

    case ((TIM_OCMODE_PWM2) << (TIM_OCMODE_SECOND_SHIFT)): // PWM mode 2
mode
        if (TIMx->CNT < TIMx->CCR2)
            TIMS->Channels.OC2REF = 0;
        else
            TIMS->Channels.OC2REF = 1;
        break;

    case ((TIM_OCMODE_FORCED_ACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): //
FORCED ACTIVE mode
        TIMS->Channels.OC2REF = 1; break;

    case ((TIM_OCMODE_FORCED_INACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): //
FORCED INACTIVE mode
        TIMS->Channels.OC2REF = 0; break;

}
}
/**
  * @brief    Выбор режима третьего канала и его симуляция.
  * @param    TIMx – таймер, каналы которого надо записать.
  * @param    TIMS – структура таймера для симуляции.
  * @details  Данная функция по регистрам таймера проверяет как настроен
              третий канал и соответствующе симулирует его работу.
  */
void CC_PWM_Ch3_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
```

```
{ // определяет режим канала
switch (TIMx->CCMR2 & TIM_CCMR1_OC1M)
{
    case (TIM_OCMODE_ACTIVE): // ACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR3) < 2*TIMS->tx_step)
            TIMS->Channels.OC3REF = 1;
        break;

    case (TIM_OCMODE_INACTIVE): // INACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR3) < 2*TIMS->tx_step)
            TIMS->Channels.OC3REF = 0;
        break;

    case (TIM_OCMODE_TOGGLE): // Toogle mode
        if (abs(TIMx->CNT - TIMx->CCR3) < 2*TIMS->tx_step)
            TIMS->Channels.OC3REF = ~TIMS->Channels.OC3REF;
        break;

    case (TIM_OCMODE_PWM1): // PWM mode 1 mode
        if (TIMx->CNT < TIMx->CCR3)
            TIMS->Channels.OC3REF = 1;
        else
            TIMS->Channels.OC3REF = 0;
        break;

    case (TIM_OCMODE_PWM2): // PWM mode 2 mode
        if (TIMx->CNT < TIMx->CCR3)
            TIMS->Channels.OC3REF = 0;
        else
            TIMS->Channels.OC3REF = 1;
        break;

    case (TIM_OCMODE_FORCED_ACTIVE): // FORCED ACTIVE mode
        TIMS->Channels.OC3REF = 1; break;

    case (TIM_OCMODE_FORCED_INACTIVE): // FORCED INACTIVE mode
        TIMS->Channels.OC3REF = 0; break;

}
}
/**
  * @brief    Выбор режима четвертого канала и его симуляция.
  * @param    TIMx – таймер, каналы которого надо записать.
  * @param    TIMS – структура таймера для симуляции.
  * @details  Данная функция по регистрам таймера проверяет как настроен
              четвертый канал и соответствующе симулирует его работу.
  */
void CC_PWM_Ch4_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
{ // определяет режим канала
switch (TIMx->CCMR2 & TIM_CCMR1_OC2M)
{
    case ((TIM_OCMODE_ACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): // ACTIVE mode
        if (abs(TIMx->CNT - TIMx->CCR4) < 2*TIMS->tx_step)
            TIMS->Channels.OC4REF = 1;
        break;

    case ((TIM_OCMODE_INACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): // INACTIVE
mode
        if (abs(TIMx->CNT - TIMx->CCR4) < 2*TIMS->tx_step)
            TIMS->Channels.OC4REF = 0;
        break;

    case ((TIM_OCMODE_TOGGLE) << (TIM_OCMODE_SECOND_SHIFT)): // Toogle mode
```

```
        if (abs(TIMx->CNT - TIMx->CCR4) < 2*TIMS->tx_step)
            TIMS->Channels.OC4REF = ~TIMS->Channels.OC4REF;
        break;

    case ((TIM_OCMODE_PWM1) << (TIM_OCMODE_SECOND_SHIFT)): // PWM mode 1
mode
        if (TIMx->CNT < TIMx->CCR4)
            TIMS->Channels.OC4REF = 1;
        else
            TIMS->Channels.OC4REF = 0;
        break;

    case ((TIM_OCMODE_PWM2) << (TIM_OCMODE_SECOND_SHIFT)): // PWM mode 2
mode
        if (TIMx->CNT < TIMx->CCR4)
            TIMS->Channels.OC4REF = 0;
        else
            TIMS->Channels.OC4REF = 1;
        break;

    case ((TIM_OCMODE_FORCED_ACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): //
FORCED ACTIVE mode
        TIMS->Channels.OC4REF = 1; break;

    case ((TIM_OCMODE_FORCED_INACTIVE) << (TIM_OCMODE_SECOND_SHIFT)): //
FORCED INACTIVE mode
        TIMS->Channels.OC4REF = 0; break;

}
}

/**
  * @brief    Запись каналов таймера в порты GPIO.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура того же таймера для симуляции.
  * @details  Данная функция записывает каналы OC в порты GPIO, определенные
в TIMS.
  *           Запись происходит только если пин настроен на альтернативную
функцию.
  */
void Write_OC_to_GPIO(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS)
{
    // write gpio pin if need
    if (Check_OC1_GPIO_Output(TIMS)) // check OC OUTPUT 4 enable (GPIO AF
MODE)
    {
        uint32_t temp2 = ~(uint32_t)(1 << (TIMS->Channels.OC1_PIN_SHIFT));
        if (TIMx->CCER & TIM_CCER_CC1P) // POLARITY check
        {   // low POLARITY
            if (TIMS->Channels.OC1REF)
                TIMS->Channels.OC1_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC1_PIN_SHIFT));
            else
                TIMS->Channels.OC1_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC1_PIN_SHIFT);
        }
        else
        {   // high POLARITY
            if (TIMS->Channels.OC1REF)
                TIMS->Channels.OC1_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC1_PIN_SHIFT);
            else
```

```
                TIMS->Channels.OC1_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC1_PIN_SHIFT));
        }
    }
    if (Check_OC2_GPIO_Output(TIMS)) // check OC OUTPUT 4 enable (GPIO AF
MODE)
    {
        if (TIMx->CCER & TIM_CCER_CC2P) // POLARITY check
        {   // low POLARITY
            if (TIMS->Channels.OC2REF)
                TIMS->Channels.OC2_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC2_PIN_SHIFT));
            else
                TIMS->Channels.OC2_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC2_PIN_SHIFT);
        }
        else
        {   // high POLARITY
            if (TIMS->Channels.OC2REF)
                TIMS->Channels.OC2_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC2_PIN_SHIFT);
            else
                TIMS->Channels.OC2_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC2_PIN_SHIFT));
        }
    }
    if (Check_OC3_GPIO_Output(TIMS)) // check OC OUTPUT 4 enable (GPIO AF
MODE)
    {
        if (TIMx->CCER & TIM_CCER_CC3P) // POLARITY check
        {   // low POLARITY
            if (TIMS->Channels.OC3REF)
                TIMS->Channels.OC3_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC3_PIN_SHIFT));
            else
                TIMS->Channels.OC3_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC3_PIN_SHIFT);
        }
        else
        {   // high POLARITY
            if (TIMS->Channels.OC3REF)
                TIMS->Channels.OC3_GPIOx->ODR |= 1 << (TIMS-
>Channels.OC3_PIN_SHIFT);
            else
                TIMS->Channels.OC3_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC3_PIN_SHIFT));
        }
    }
    if (Check_OC4_GPIO_Output(TIMS)) // check OC CHANNEL 4 enable (GPIO AF
MODE)
    {
        if (TIMx->CCER & TIM_CCER_CC4P) // POLARITY check
        {   // low POLARITY
            if (TIMS->Channels.OC4REF)
                TIMS->Channels.OC4_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC4_PIN_SHIFT));
            else
                TIMS->Channels.OC4_GPIOx->ODR |= (1) << (TIMS-
>Channels.OC4_PIN_SHIFT);
        }
        else
        {   // high POLARITY
            if (TIMS->Channels.OC4REF)
```

```
                TIMS->Channels.OC4_GPIOx->ODR |= (1) << (TIMS-
>Channels.OC4_PIN_SHIFT);
            else
                TIMS->Channels.OC4_GPIOx->ODR &= ~(uint32_t)(1 << (TIMS-
>Channels.OC4_PIN_SHIFT));
        }
    }
}
/** Запись результата compare в глабальную структуру с TRIGGER OUTPUT */
/**
  * @brief    Запись каналов таймера в глабальную структуру с TRIGGER
OUTPUT.
  * @param    TIMx - таймер, каналы которого надо записать.
  * @param    TIMS - структура того же таймера для симуляции.
  * @details  Данная функция считывает каналы OC и записывает их в внешний
канал триггера TRGO.
  */
void Write_OC_to_TRGO(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS)
{
    // write trigger output from OCxREF pin if need
    unsigned temp_trgo;
    if ((TIMx->CR2 & TIM_CR2_MMS) == (0b100 << TIM_CR2_MMS_Pos))
    {
        temp_trgo = TIMS->Channels.OC1REF;
    }
    else if ((TIMx->CR2 & TIM_CR2_MMS) == (0b101 << TIM_CR2_MMS_Pos))
    {
        temp_trgo = TIMS->Channels.OC2REF;
    }
    else if ((TIMx->CR2 & TIM_CR2_MMS) == (0b110 << TIM_CR2_MMS_Pos))
    {
        temp_trgo = TIMS->Channels.OC3REF;
    }
    else if ((TIMx->CR2 & TIM_CR2_MMS) == (0b111 << TIM_CR2_MMS_Pos))
    {
        temp_trgo = TIMS->Channels.OC4REF;
    }
    // select TIMx TRGO
    if (TIMx == TIM1)
        Slave_Channels.TIM1_TRGO = temp_trgo;
    else if (TIMx == TIM2)
        Slave_Channels.TIM2_TRGO = temp_trgo;
    else if (TIMx == TIM3)
        Slave_Channels.TIM3_TRGO = temp_trgo;
    else if (TIMx == TIM4)
        Slave_Channels.TIM4_TRGO = temp_trgo;
    else if (TIMx == TIM5)
        Slave_Channels.TIM5_TRGO = temp_trgo;
    else if (TIMx == TIM6)
        Slave_Channels.TIM6_TRGO = temp_trgo;
    else if (TIMx == TIM7)
        Slave_Channels.TIM7_TRGO = temp_trgo;
    else if (TIMx == TIM8)
        Slave_Channels.TIM8_TRGO = temp_trgo;
    temp_trgo = 0;
}
//-----------------------------------------------------------------------//

//--------------------MISC (temporary) FUNCTIONS--------------------//
/** Определение источника для запуска таймера в SLAVE MODE */
/**
  * @brief    Определение источника для запуска таймера в SLAVE MODE.
  * @param    TIMx - таймер, который надо включить.
```

```
  * @param    TIMx – таймер, прерываний которого надо вызвать.
  * @details  Данная функция проверяет какой триггер выбран для запуска
таймера,
  *           после записывает значение канала триггера в бит включения
таймера.
  *           Таким образом, при лог.1 в канале триггера - таймер
включиться.
  */
void Slave_Mode_Check_Source(TIM_TypeDef* TIMx)
{
    if (TIMx == TIM2)
    {
        if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR0)
            TIMx->CR1 |= (Slave_Channels.TIM1_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR1)
            TIMx->CR1 |= (Slave_Channels.TIM1_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR2)
            TIMx->CR1 |= (Slave_Channels.TIM1_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR3)
            TIMx->CR1 |= (Slave_Channels.TIM8_TRGO << TIM_CR1_CEN_Pos);
    }
    else if (TIMx == TIM3)
    {
        if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR0)
            TIMx->CR1 |= (Slave_Channels.TIM8_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR1)
            TIMx->CR1 |= (Slave_Channels.TIM2_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR2)
            TIMx->CR1 |= (Slave_Channels.TIM2_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR3)
            TIMx->CR1 |= (Slave_Channels.TIM3_TRGO << TIM_CR1_CEN_Pos);
    }
    else if (TIMx == TIM4)
    {
        if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR0)
            TIMx->CR1 |= (Slave_Channels.TIM3_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR1)
            TIMx->CR1 |= (Slave_Channels.TIM5_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR2)
            TIMx->CR1 |= (Slave_Channels.TIM3_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR3)
            TIMx->CR1 |= (Slave_Channels.TIM4_TRGO << TIM_CR1_CEN_Pos);
    }
    else if (TIMx == TIM5)
    {
        if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR0)
            TIMx->CR1 |= (Slave_Channels.TIM4_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR1)
            TIMx->CR1 |= (Slave_Channels.TIM4_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR2)
            TIMx->CR1 |= (Slave_Channels.TIM7_TRGO << TIM_CR1_CEN_Pos);
        else if ((TIMx->SMCR & TIM_SMCR_TS) == TIM_TS_ITR3)
            TIMx->CR1 |= (Slave_Channels.TIM7_TRGO << TIM_CR1_CEN_Pos);
    }
}
//-----------------------------------------------------------------------//

//------------------------SIMULINK FUNCTIONS-----------------------------//
/** Симулирование выбранных через дефайн таймеров */
/**
  * @brief    Симуляция выбранных таймеров.
  * @details  Таймеры для симуляции выбираются через дефайны в
stm32f4xx_matlab_conf.h.
```

```
                Функция вызывается на каждом шаге симуляции.
  *
  * @attention  Добавить это в функцию MCU_Periph_Simulation()
  */
void Simulate_TIMs(void)
{
#ifdef USE_TIM1
    TIM_Simulation(TIM1, &tim1s);
#endif
#ifdef USE_TIM2
    TIM_Simulation(TIM2, &tim2s);
#endif
#ifdef USE_TIM3
    TIM_Simulation(TIM3, &tim3s);
#endif
#ifdef USE_TIM4
    TIM_Simulation(TIM4, &tim4s);
#endif
#ifdef USE_TIM5
    TIM_Simulation(TIM5, &tim5s);
#endif
#ifdef USE_TIM6
    TIM_Simulation(TIM6, &tim6s);
#endif
#ifdef USE_TIM7
    TIM_Simulation(TIM7, &tim7s);
#endif
#ifdef USE_TIM8
    TIM_Simulation(TIM8, &tim8s);
#endif
#ifdef USE_TIM9
    TIM_Simulation(TIM9, &tim9s);
#endif
#ifdef USE_TIM10
    TIM_Simulation(TIM10, &tim10s);
#endif
#ifdef USE_TIM11
    TIM_Simulation(TIM11, &tim11s);
#endif
#ifdef USE_TIM12
    TIM_Simulation(TIM12, &tim12s);
#endif
#ifdef USE_TIM13
    TIM_Simulation(TIM13, &tim13s);
#endif
#ifdef USE_TIM14
    TIM_Simulation(TIM14, &tim14s);
#endif
}
/**
  * @brief    Деинициализирование выбранных таймеров.
  * @details  Таймеры для деинициализизации выбираются через дефайны в
stm32f4xx_matlab_conf.h.
                Функция вызывается в конце симуляции.
  */
void TIM_SIM_DEINIT(void)
{
#ifdef USE_TIM1
    memset(&tim1s, 0, sizeof(tim1s));
#endif
#ifdef USE_TIM2
    memset(&tim2s, 0, sizeof(tim2s));
#endif
```

```
#ifdef USE_TIM3
    memset(&tim3s, 0, sizeof(tim3s));
#endif
#ifdef USE_TIM4
    memset(&tim4s, 0, sizeof(tim4s));
#endif
#ifdef USE_TIM5
    memset(&tim5s, 0, sizeof(tim5s));
#endif
#ifdef USE_TIM6
    memset(&tim6s, 0, sizeof(tim6s));
#endif
#ifdef USE_TIM7
    memset(&tim7s, 0, sizeof(tim7s));
#endif
#ifdef USE_TIM8
    memset(&tim8s, 0, sizeof(tim8s));
#endif
#ifdef USE_TIM9
    memset(&tim9s, 0, sizeof(tim9s));
#endif
#ifdef USE_TIM10
    memset(&tim10s, 0, sizeof(tim10s));
#endif
#ifdef USE_TIM11
    memset(&tim11s, 0, sizeof(tim11s));
#endif
#ifdef USE_TIM12
    memset(&tim12s, 0, sizeof(tim12s));
#endif
#ifdef USE_TIM13
    memset(&tim13s, 0, sizeof(tim13s));
#endif
#ifdef USE_TIM14
    memset(&tim14s, 0, sizeof(tim14s));
#endif
}
//-----------------------------------------------------------------------//

//------------------TIM'S HANDLERS (BETA) FUNCTIONS-----------------//
// Определение обработчиков, которые не используются
// Т.к. в MSVC нет понятия weak function, необходимо объявить все колбеки
// И если какой-то колбек не используется, его надо определить
#ifndef USE_TIM1_UP_TIM10_HANDLER
void TIM1_UP_TIM10_IRQHandler(void) {}
#endif
#ifndef USE_TIM2_HANDLER
void TIM2_IRQHandler(void) {}
#endif
#ifndef USE_TIM3_HANDLER
void TIM3_IRQHandler(void) {}
#endif
#ifndef USE_TIM4_HANDLER
void TIM4_IRQHandler(void) {}
#endif
#ifndef USE_TIM5_HANDLER
void TIM5_IRQHandler(void) {}
#endif
#ifndef USE_TIM6_HANDLER
void TIM6_DAC_IRQHandler(void) {}
#endif
#ifndef USE_TIM7_HANDLER
void TIM7_IRQHandler(void) {}
```

```
#endif
#ifndef USE_TIM8_UP_TIM13_HANDLER
void TIM8_UP_TIM13_IRQHandler(void) {}
#endif
#ifndef USE_TIM1_BRK_TIM9_HANDLER
void TIM1_BRK_TIM9_IRQHandler(void) {}
#endif
#ifndef USE_TIM1_TRG_COM_TIM11_HANDLER
void TIM1_TRG_COM_TIM11_IRQHandler(void) {}
#endif
#ifndef USE_TIM8_BRK_TIM12_HANDLER
void TIM8_BRK_TIM12_IRQHandler(void) {}
#endif
#ifndef USE_TIM8_TRG_COM_TIM14_HANDLER
void TIM8_TRG_COM_TIM14_IRQHandler(void) {}
#endif

/**
  * @brief    Вызов прерывания таймера TIMx.
  * @param    TIMx - таймер, прерываний которого надо вызвать.
  * @details  Данная функция симулирует аппаратный вызов прерывания
              таймера по какому-либо событию.
  */
void call_IRQHandller(TIM_TypeDef* TIMx)
{ // calling HANDLER
    if ((TIMx == TIM1) || (TIMx == TIM10))
        TIM1_UP_TIM10_IRQHandler();
    else if (TIMx == TIM2)
        TIM2_IRQHandler();
    else if (TIMx == TIM3)
        TIM3_IRQHandler();
    else if (TIMx == TIM4)
        TIM4_IRQHandler();
    else if (TIMx == TIM5)
        TIM5_IRQHandler();
    else if (TIMx == TIM6)
        TIM6_DAC_IRQHandler();
    else if (TIMx == TIM7)
        TIM7_IRQHandler();
    else if ((TIMx == TIM8) || (TIMx == TIM13))
        TIM8_UP_TIM13_IRQHandler();
    else if ((TIMx == TIM1) || (TIMx == TIM9))
        TIM1_BRK_TIM9_IRQHandler();
    else if ((TIMx == TIM1) || (TIMx == TIM11))
        TIM1_TRG_COM_TIM11_IRQHandler();
    else if ((TIMx == TIM8) || (TIMx == TIM12))
        TIM8_BRK_TIM12_IRQHandler();
    else if ((TIMx == TIM8) || (TIMx == TIM14))
        TIM8_TRG_COM_TIM14_IRQHandler();
}
//--------------------------------------------------------------------//
```

ПРИЛОЖЕНИЕ Г

Листинг stm32f4xx_matlab_tim.h

```
/**
******************************************************************************
* @file stm32f4xx_matlab_tim.h
* @brief Заголовочный файл для симулятора таймеров.
******************************************************************************
@details
Данный файл содержит объявления всякого для симуляции таймеров STM32F407xx.
******************************************************************************/
#ifndef _MATLAB_TIM_H_
#define _MATLAB_TIM_H_

#include "stm32f4xx_hal.h"
#include "stm32f4xx_it.h"
#include "mcu_wrapper_conf.h"

/**
    * @addtogroup   TIM_SIMULATOR   TIM Simulator
    * @ingroup      MAIN_SIMULATOR
    * @brief        Симулятор для таймеров
    * @details      Дефайны и функции для симуляции таймеров.
    * @{
    */

//////////////////////////////---DEFINES---//////////////////////////////
/**
  * @brief Дефайн для сдвига между первой и второй половиной CCMRx регистров
  */
#define TIM_OCMODE_SECOND_SHIFT                 (TIM_CCMR1_OC2M_Pos -
TIM_CCMR1_OC1M_Pos)

/**
  * @brief    Дефайн для проверки выводить ли канал таймера на GPIO
  * @details  Данный дефайн проверяет, настроен ли пин GPIO на
альтернативную функцию. Если да - то таймер выводиться на этот пин
  */
#define Check_OCx_GPIO_Output(_tims_, _OCx_GPIOx_, _OCx_PIN_SHIFT_)
(_tims_->Channels._OCx_GPIOx_->MODER & (0b11<<(2*_tims_-
>Channels._OCx_PIN_SHIFT_))) == (0b10<<(2*_tims_->Channels._OCx_PIN_SHIFT_))
/**
  * @brief Дефайн для проверки выводить ли канал 1 на GPIO (настроен ли GPIO
на альтернативную функцию)
  */
#define Check_OC1_GPIO_Output(_tims_)
Check_OCx_GPIO_Output(_tims_, OC1_GPIOx, OC1_PIN_SHIFT)
/**
  * @brief Дефайн для проверки выводить ли канал 2 на GPIO (настроен ли GPIO
на альтернативную функцию)
  */
#define Check_OC2_GPIO_Output(_tims_)
Check_OCx_GPIO_Output(_tims_, OC2_GPIOx, OC2_PIN_SHIFT)
/**
  * @brief Дефайн для проверки выводить ли канал 3 на GPIO (настроен ли GPIO
на альтернативную функцию)
  */
#define Check_OC3_GPIO_Output(_tims_)
Check_OCx_GPIO_Output(_tims_, OC3_GPIOx, OC3_PIN_SHIFT)
/**
  * @brief Дефайн для проверки выводить ли канал 4 на GPIO (настроен ли GPIO
на альтернативную функцию)
  */
```

```
#define Check_OC4_GPIO_Output(_tims_)
Check_OCx_GPIO_Output(_tims_, OC4_GPIOx, OC4_PIN_SHIFT)

////////////////////////////////////////////////////////////////////////



////////////////////////////---STRUCTURES---////////////////////////////
/**
  * @brief Структура для управления Слейв Таймерами
  */
struct SlaveChannels
{
    unsigned TIM1_TRGO : 1;     ///< Синган синхронизации таймера 1
    unsigned TIM2_TRGO : 1;     ///< Синган синхронизации таймера 2
    unsigned TIM3_TRGO : 1;     ///< Синган синхронизации таймера 3
    unsigned TIM4_TRGO : 1;     ///< Синган синхронизации таймера 4
    unsigned TIM5_TRGO : 1;     ///< Синган синхронизации таймера 5
    unsigned TIM6_TRGO : 1;     ///< Синган синхронизации таймера 6
    unsigned TIM7_TRGO : 1;     ///< Синган синхронизации таймера 7
    unsigned TIM8_TRGO : 1;     ///< Синган синхронизации таймера 8

};

/**
  * @brief Структура для моделирования каналов таймера
  */
struct Channels_Sim
{
    // Каналы таймера
    unsigned OC1REF:1;          ///< Первый канал
    unsigned OC2REF:1;          ///< Второй канал
    unsigned OC3REF:1;          ///< Третий канал
    unsigned OC4REF:1;          ///< Четвертый канал

    // связанные с каналами GPIO порты и пины
    GPIO_TypeDef *OC1_GPIOx;    ///< Порт первого канала
    uint32_t OC1_PIN_SHIFT;     ///< Пин первого канала

    GPIO_TypeDef *OC2_GPIOx;    ///< Порт второго канала
    uint32_t OC2_PIN_SHIFT;     ///< Пин второго канала

    GPIO_TypeDef *OC3_GPIOx;    ///< Порт третьего канала
    uint32_t OC3_PIN_SHIFT;     ///< Пин третьего канала

    GPIO_TypeDef *OC4_GPIOx;    ///< Порт четвертого канала
    uint32_t OC4_PIN_SHIFT;     ///< Пин четвертого канала

};

/**
  * @brief Структура для моделирования таймера
  */
struct TIM_Sim
{
    double tx_cnt;                  ///< Счетчик таймера (double, т.к. кол-
во тактов за шаг симуляции может быть дробным)
    double tx_step;                 ///< Шаг счета за один шаг симуляции
(double, т.к. кол-во тактов за шаг симуляции может быть дробным)
    int RELOAD;                     ///< Буфер для периода таймера (для
реализации функции PRELOAD)
    struct Channels_Sim Channels;   ///< Структура для симуляции каналов
};
```

```
////////////////////////////////////////////////////////////////////////////



////////////////////////////---FUNCTIONS---////////////////////////////

//----------------------TIMER BASE FUNCTIONS-----------------------//
/* Базовая функция для симуляции таймера: она вызывается каждый шаг
симуляции */
void TIM_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS);
/* Счет таймера за один такт */
void TIMx_Count(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
/* Проверка на переполнение и дальнейшая его обработка */
void Overflow_Check(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
/* Вызов прерывания */
void call_IRQHandller(TIM_TypeDef *TIMx);
//----------------------------------------------------------------//


//------------------------CHANNELS FUNCTIONS----------------------//
/* Симуляция каналов таймера */
void Channels_Simulation(TIM_TypeDef *TIMx, struct TIM_Sim *TIMS);
/*---------------- - CAPTURE COPMARE & PWM FUNCTIONS-----------------*/
/* Выбор режима CaptureCompare или PWM и симуляция для каждого канала */
void CC_PWM_Ch1_Simulation(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
void CC_PWM_Ch2_Simulation(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
void CC_PWM_Ch3_Simulation(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
void CC_PWM_Ch4_Simulation(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
/* Запись каналов таймера в порты GPIO */
void Write_OC_to_GPIO(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
/* Запись результата compare в глабальную структуру с TRIGGER OUTPUT */
void Write_OC_to_TRGO(TIM_TypeDef* TIMx, struct TIM_Sim* TIMS);
//----------------------------------------------------------------//


//--------------------MISC (temporary) FUNCTIONS------------------//
/* Определение источника для запуска таймера в SLAVE MODE */
void Slave_Mode_Check_Source(TIM_TypeDef* TIMx);
//----------------------------------------------------------------//


//------------------------SIMULINK FUNCTIONS----------------------//
// Симулирование выбранных таймеров
void Simulate_TIMs(void);
// Деинициализирование выбранных таймеров (вызывается в конце симуляции)
void TIM_SIM_DEINIT(void);
//----------------------------------------------------------------//
/** TIM_SIMULATOR
  * @}
  */
#endif // _MATLAB_TIM_H_
```

ПРИЛОЖЕНИЕ Д

Листинг stm32f4xx_matlab_gpio.c

```
#include "stm32f4xx_matlab_gpio.h"

#include "modbus.h"

void GPIO_to_SFUNC(real_T* disc)
{
    for (int i = 0; i < PORT_WIDTH; i++)
    {
        if (GPIOB->ODR & (1 << i))
        {
            disc[i] = 1;
        }

        if (GPIOD->ODR & (1 << i))
        {
            disc[PORT_WIDTH + i] = 1;
        }

        if (GPIOE->ODR & (1 << i))
        {
            disc[2*PORT_WIDTH + i] = 1;
        }
    }
}

void SFUNC_to_GPIO(real_T* in)
{
    // write pwm ctrl registers
    for (int i = 0; i < 5; i++)
    {
        pwm_ctrl[i] = in[i];
    }
    // write pwm ctrl coils
    if (in[5] > 0.5)
    {
        MB_Set_Coil_Local(coils_regs, COIL_PWM_DC_MODE);
    }
    else
    {
        MB_Reset_Coil_Local(coils_regs, COIL_PWM_DC_MODE);
    }
    if (in[6] > 0.5)
    {
        MB_Set_Coil_Local(coils_regs, COIL_PWM_CH_MODE);
    }
    else
    {
        MB_Reset_Coil_Local(coils_regs, COIL_PWM_CH_MODE);
    }
    if (in[7] > 0.5)
    {
        MB_Set_Coil_Local(coils_regs, COIL_PWM_PHASE_MODE);
    }
    else
    {
        MB_Reset_Coil_Local(coils_regs, COIL_PWM_PHASE_MODE);
    }
}
```

ПРИЛОЖЕНИЕ Е

Листинг stm32f4xx_matlab_gpio.h

```
#ifndef _MATLAB_GPIO_H_
#define _MATLAB_GPIO_H_

#include "stm32f4xx_hal.h"
#include "simstruc.h"
#include "mcu_wrapper_conf.h"

void SFUNC_to_GPIO(real_T* disc);
void GPIO_to_SFUNC(real_T* in);

#endif // _MATLAB_GPIO_H_
```