

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра микроэлектроники, информационных технологий и управляющих
систем (МИТиУС)

РАБОТА С ПАМЯТЬЮ, ПЕРЕМЕННЫЕ В СИ. ПОБИТОВЫЙ ДОСТУП К
ДАННЫМ В CORTEX-M3
Отчет по лабораторной работе №1
по дисциплине «Аппаратные средства телекоммуникационных систем»
Вариант №16

Студент гр. 735

_____ Д. А. Осипов

Принял:

Преподаватель кафедры

МИТУС

_____ Ю. Б. Шаропин
(оценка)

(дата)

1 Введение

Цель работы: ознакомиться с адресным пространством микроконтроллера MDR1986BE92QI и реализовать побитовый доступ к данным с помощью средств языка программирования Си и технологии Bit-banding.

Задание:

1. ознакомиться с адресным пространством МК;
2. сравнить карту памяти МК из спецификации и настройки линковщика;
3. выбрать область памяти, допускающую работу в режиме bit-band;
4. изменить состояние заданного бита с использованием:
 - Bit-banding;
 - структуры для битовых полей в языке Си;
 - операторов сдвига и побитовых «и», «или» в языке Си;
5. выполнить профилировку кода;
6. дополнительно: выполнить профилировку кода с помощью профилировщика IAR/Keil.

2 Ход работы

2.1 Ознакомиться с адресным пространством МК

При первичной настройке проекта необходимо указать путь к файлу конфигурации линковщика. (Рисунок 2.1)

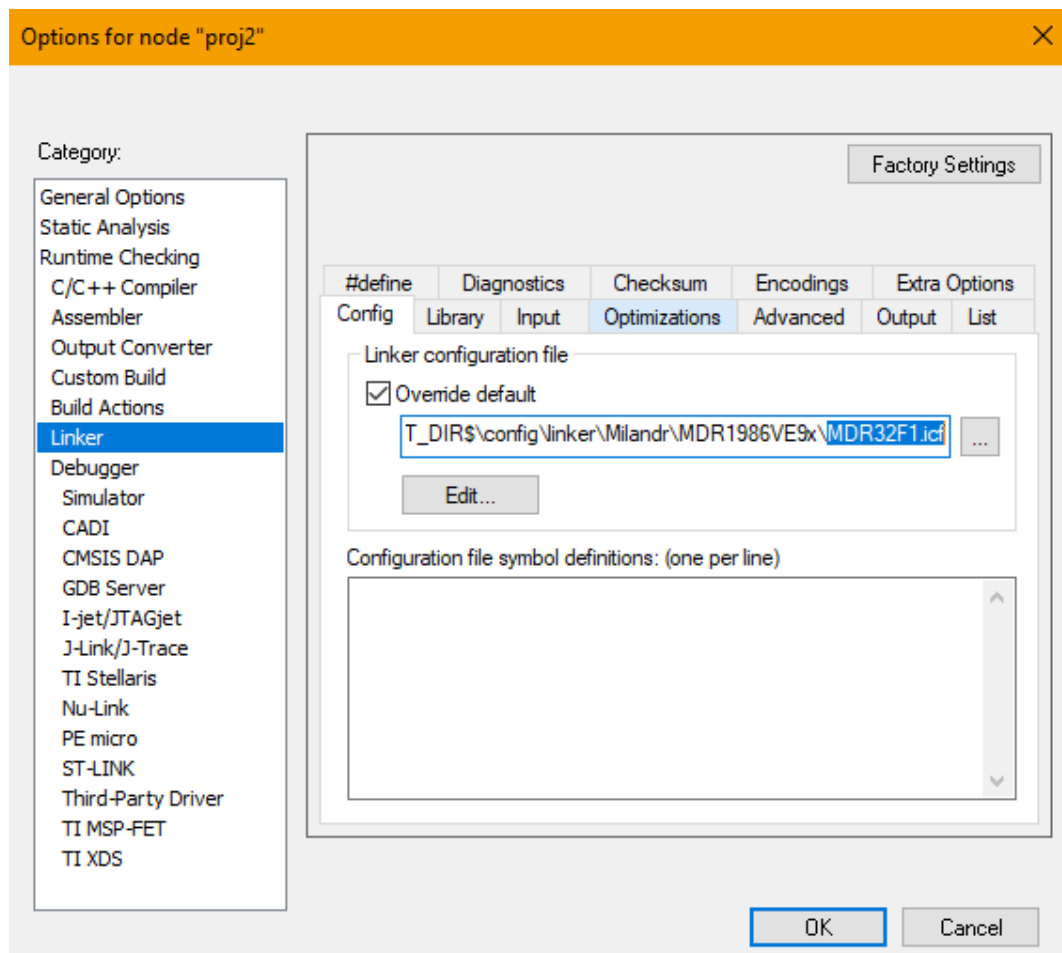


Рисунок 2.1 – Название

В этом файле заданы границы областей памяти, которые линковщиком будут определены как ROM (Read Only Memory), RAM (Random Access Memory), стек, куча и область векторов прерываний. (Рисунок 2.1111)

Файл расположен по адресу:

C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.2\arm\config\linker\Milandr\MDR1986VE9x\MDR32F1.icf

```

/#####ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__ = (0x08000000+0x00020000-1);
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__ = (0x20000000+0x00008000-1);
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__ = 0x2000;
/**** End of ICF editor section. #####ICF###*/

define memory mem with size = 4G;
define region ROM_region = mem:[from __ICFEDIT_region_ROM_start__ to
__ICFEDIT_region_ROM_end__];
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to
__ICFEDIT_region_RAM_end__];

define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ { };
define block HEAP with alignment = 8, size = __ICFEDIT_size_heap__ { };

initialize by copy { readwrite };
do not initialize { section .noinit };

keep { section .intvec };
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region { readonly };
place in RAM_region { readwrite,
    block CSTACK,
    block HEAP };

```

Рисунок 2.2 – Конфигурационный файл MDR32F1.icf – настройки линковщика для IAR Embedded Workbench

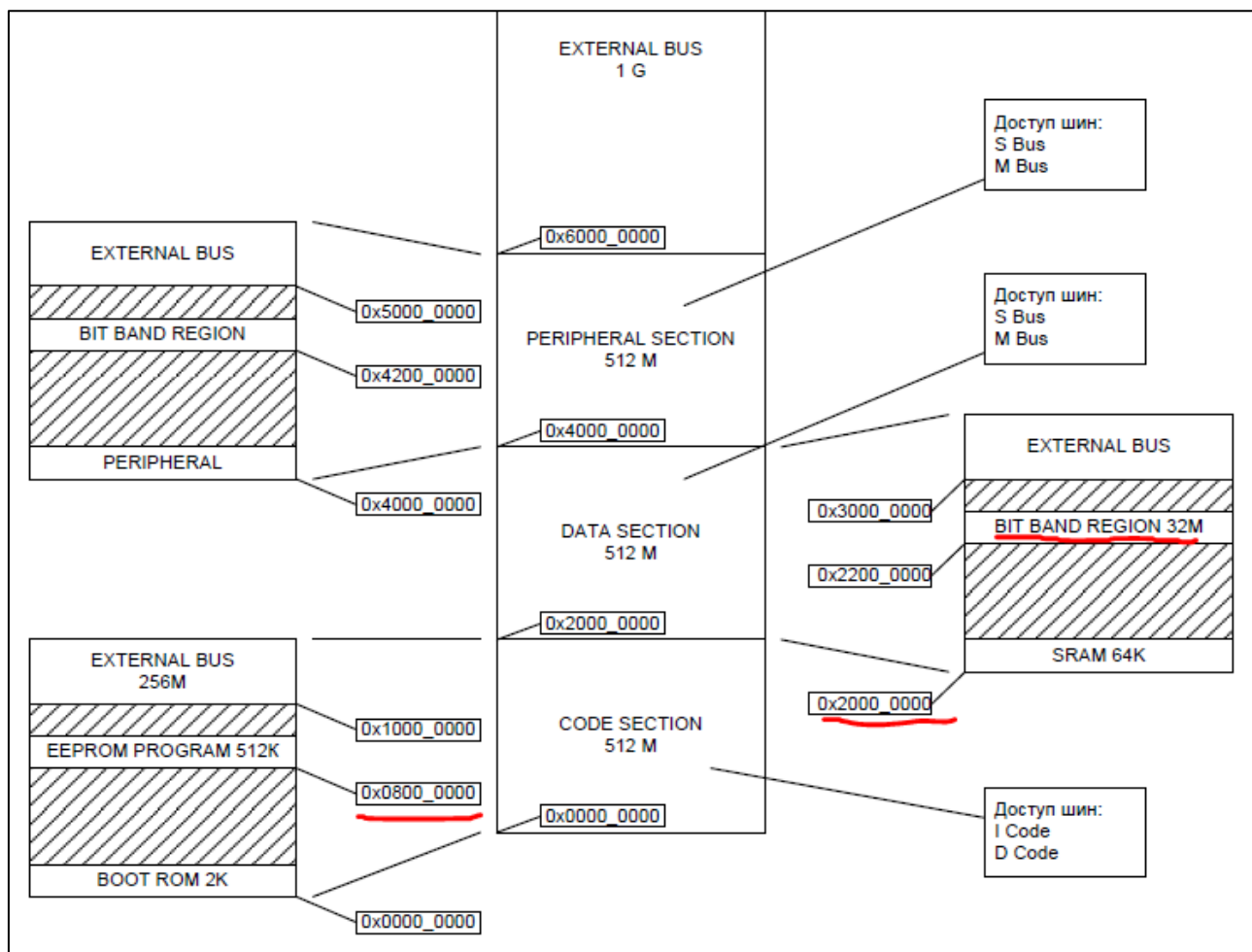


Рисунок 2.3 – Карта распределения основных областей памяти МК (спецификация → структура памяти: стр. 23)

Если обратиться к спецификации микроконтроллера, можно сопоставить адреса из файла конфигурации и карты памяти микроконтроллера.

Область памяти с адреса 0x08000000 по (0x08000000+0x00020000-1) соответствует области EEPROM PROGRAM на рисунке 2.3. Область EEPROM PROGRAM – область энергонезависимой памяти программы, доступной для перепрограммирования пользователем. Память предназначена для хранения основной рабочей программы.

Область памяти с адреса 0x20000000 по (0x20000000+0x00008000-1) соответствует области SRAM (рисунок 2.3). Память области SRAM реализована в виде блока статической памяти. Область памяти для данных. Код программы также может располагаться здесь.

2.2 Bit-banding

Метод bit-band ускоряет выполнение битовых операций, позволяя обойтись меньшим числом команд, он также играет важную роль при организации совместного использования каких-либо ресурсов несколькими процессами. Одним из важнейших достоинств битовых операций с применением метода bit-band является их атомарность. Другими словами, выполнение последовательности «чтение—модификация—запись» не может быть прервано никакими другими операциями на шине

Операции «чтение—модификация—запись» реализуются на аппаратном уровне и являются атомарными — обе пересылки, осуществляемые при выполнении операции, неотделимы друг от друга и возникновение прерываний между этими пересылками исключено

Чтобы воспользоваться в Си-программах возможностями, обеспечиваемыми методом bit-band, можно объявить пару указателей — на адрес в области хранения битов и на соответствующий ему адрес в области доступа к битам. Например:

```
#define DEVICE_REG0 *((volatile unsigned long *) (0x40000000))
#define DEVICE_REG0_BIT0 *((volatile unsigned long *) (0x42000000))
#define DEVICE_REG0_BIT1 *((volatile unsigned long *) (0x42000004))
```

Из учебника «Ядро Cortex-M3»: «Обратите внимание, что при использовании метода bit-band переменные, размещаемые в бит-адресуемой области, должны быть объявлены как volatile. Компиляторы ведь не знают, что к одним и тем же данным можно обращаться по двум различным адресам, а применение модификатора volatile позволяет гарантировать, что при каждом обращении к такой переменной процессор будет обращаться непосредственно к ячейке памяти, а не к локальной копии этой переменной.»

Можно упростить использование бит-адресуемых областей памяти, написав пару макроопределений. Один из макросов будет вычислять адрес в области доступа к битам на основе адреса слова из области хранения битов и номера бита, а второй — преобразовывать значение адреса ячейки памяти в указатель:

```
// Вычисляет адрес в области хранения битов, используя адрес
// в бит-адресуемой области и номер бита
#define BITBAND(addr,bitnum) ((addr & 0xF0000000) + 0x20000000 +
((addr & 0xFFFFF)<<5) + (bitnum <<2))
// Преобразует адрес в указатель
#define MEM_ADDR(addr) *((volatile unsigned long *) (addr))
```

Перепишем код предыдущего примера, используя эти макросы:

```
#define DEVICE_REG0 0x40000000

#define BITBAND(addr,bitnum) ((addr & 0xF0000000) + 0x02000000 +
((addr & 0xFFFFF)<<5) + (bitnum<<2))

#define MEM_ADDR(addr) *((volatile unsigned long *) (addr))

...

MEM_ADDR(DEVICE_REG0) = 0xAB; // Обращение к регистру
периферийного устройства

// с использованием обычной адресации

...

// Установка бита 1 без использования метода bit-band
MEM_ADDR(DEVICE_REG0) = MEM_ADDR(DEVICE_REG0) | 0x2;

...

// Установка бита 1 с использованием метода bit-band
MEM_ADDR(BITBAND(DEVICE_REG0,1)) = 0x1;
```

Обычно ядро процессора не может писать/читать отдельные биты регистров или ячеек памяти. Операция установки/снятия бита занимает одну машинную инструкцию – обеспечивается атомарный доступ к памяти.

Работа с BIT BAND REGION позволяет осуществлять операции «Чтение-Модификация-Запись», «Установка бита» и «Сброс Бита» одной инструкцией. Bit-band область располагается с адреса 0x20000000 по 0x200FFFFFF. Для того чтобы изменить значение отдельного бита в этой области, в программе можно реализовать аналогичное изменение значения нулевого бита соответствующего слова в области bit-band alias, имеющей диапазон 0x22000000-0x23FFFFFF. (Рисунок 2.4)

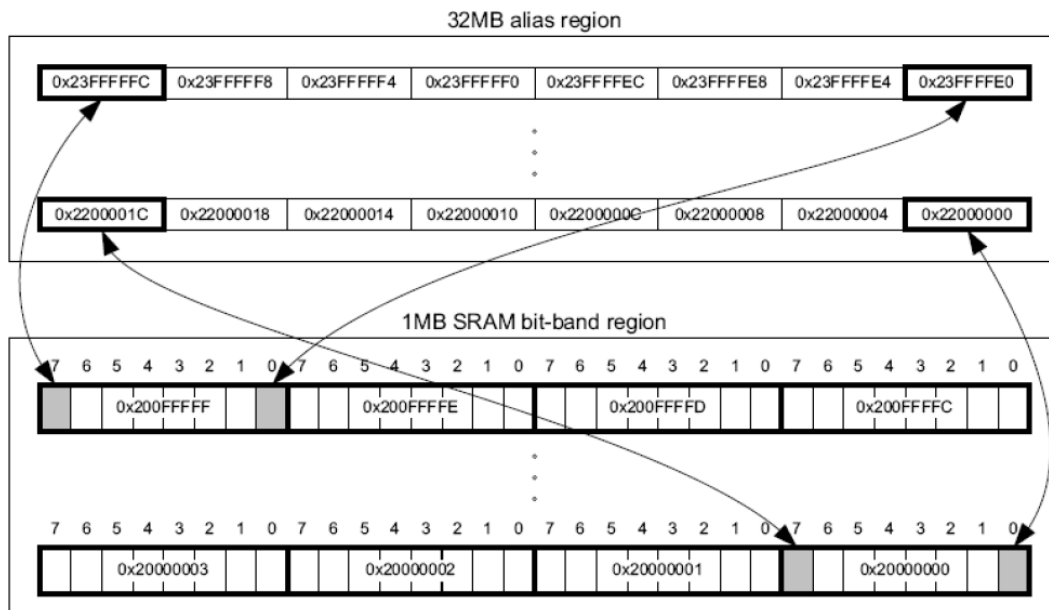


Рисунок 2.4 – Схема отображения региона bit-band alias в регионе bit-band

Битовое поле – это элемент структуры, определенный как некоторое число битов, обычно меньшее, чем число битов в целом числе (оно по величине не превосходит машинного слова и зависит от реализации компилятора). Они предназначены для экономного размещения в памяти данных небольшого диапазона, обеспечивают удобный доступ к отдельным битам данных. Битовое поле, на самом деле, - это просто особый тип структуры, определяющей, какую длину имеет каждый элемент. С помощью битовых полей можно обратиться к биту как к переменной, но операция получения адреса для конкретного элемента недоступна.

2.3 Структуры для битовых полей в языке Си

Структура

Структура — это совокупность нескольких переменных, чаще всего различных типов, сгруппированных под одним именем.

Структура — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.

Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое. Структура представляет собой сложный тип данных, составленный из простых типов.

```
struct point3d {
    int x;
    int y;
    int z;
};
```


Доступ к элементам структуры:

```
point3d.x = 0;
(*point3d)->x = 2;
```

Под каждый элемент структуры выделяется своя область памяти, а в объединении выбирается наибольший элемент памяти и все типы размещаются в общей ячейке памяти, в таком случае в union можно обращаться к одной ячейке памяти как к разным типам данных.

Запись битового поля:

```
struct {
    uint8_t flag_a : 1; // 1 bit
    uint8_t flag_b : 1; // 1 bit
    uint8_t flag_c : 1; // 1 bit
    uint8_t flag_d : 1; // 1 bit
    uint8_t value : 4; // 4 bits
} bit_field; // 8 bits
```

2.4 Операторы сдвига и побитовые «и», «или» в языке Си

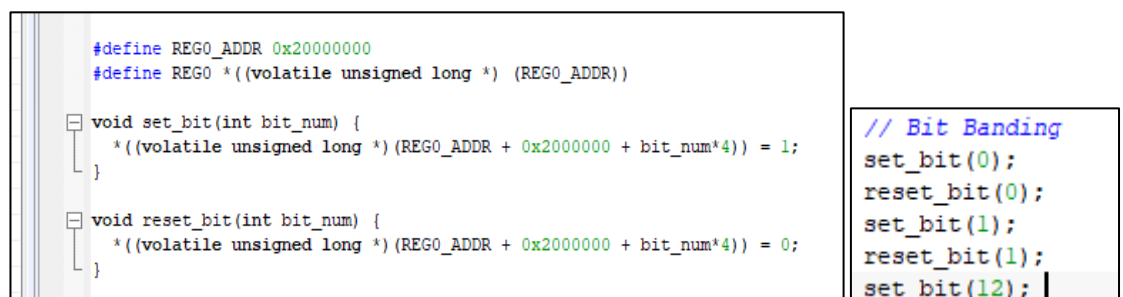
Для манипулирования битами используются следующие операторы:

- «<<» — сдвиг влево;
- «>>» — сдвиг вправо;
- «~» — поразрядная инверсия;
- «|» — поразрядное «ИЛИ»;
- «&» — поразрядное «И»;
- «^» — поразрядное исключающее «ИЛИ».

2.5 Профилировка кода IAR

В IAR 8.32 на языке программирования Си была реализована работа с битовыми полями, «bit-banding» и «ручной» работой с битами.

Для выполнения поставленной задачи была написана программа в среде разработки «IAR Embedded Workbench» в которой были реализованы три способа работы с битами, таким образом «bit-banding» представлен на рисунке 2.5, «битовые поля» представлены на рисунке 2.6 и работа с битами «вручную» представлена на рисунке 2.7. Далее будет разобран каждый из способов, представлен результат профилировки кода.



```
#define REG0_ADDR 0x20000000
#define REG0 *((volatile unsigned long *) (REG0_ADDR))

void set_bit(int bit_num) {
    *((volatile unsigned long *) (REG0_ADDR + 0x20000000 + bit_num*4)) = 1;
}

void reset_bit(int bit_num) {
    *((volatile unsigned long *) (REG0_ADDR + 0x20000000 + bit_num*4)) = 0;
}

// Bit Banding
set_bit(0);
reset_bit(0);
set_bit(1);
reset_bit(1);
set_bit(12);
```

Рисунок 2.5 – Реализация «bit-banding»

```

struct {
    char flag_a : 1;
    char flag_b : 1;
    char flag_c : 1;
    char flag_d : 1;
    char flag_e : 1;
    char flag_f : 1;
    char flag_g : 1;
    char flag_h : 1;
} bit_field; // 8 bits

// Bit Field
bit_field.flag_a = 1;
bit_field.flag_a = 0;
bit_field.flag_c = 1;
bit_field.flag_c = 0;
bit_field.flag_e = 1;
bit_field.flag_a = 1;
bit_field.flag_a = 0;
bit_field.flag_e = 0;
// Bit Banding

```

Рисунок 2.6 – Реализация битового поля

```

static char bait = 0;

int main()

// Handwork with bits
bait |= (1<<2);
bait |= (1<<3);
bait &= ~(1<<2);
bait &= ~(1<<3);
bait ^= (1<<0);

```

Рисунок 2.7 – Реализация работы с битами «вручную»

Перед началом работы производится предварительная настройка окон режима отладки. Включим счетчик инструкций, поле регистров, поле памяти и функции профилировщика. (Рисунок 2.8 – 2.11)

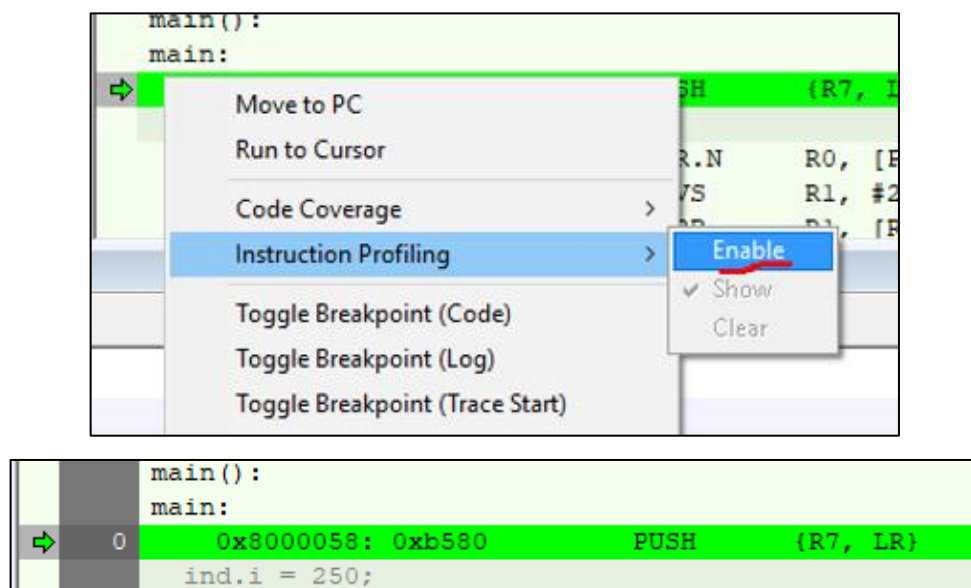


Рисунок 2.8 – Включение счетчика инструкций

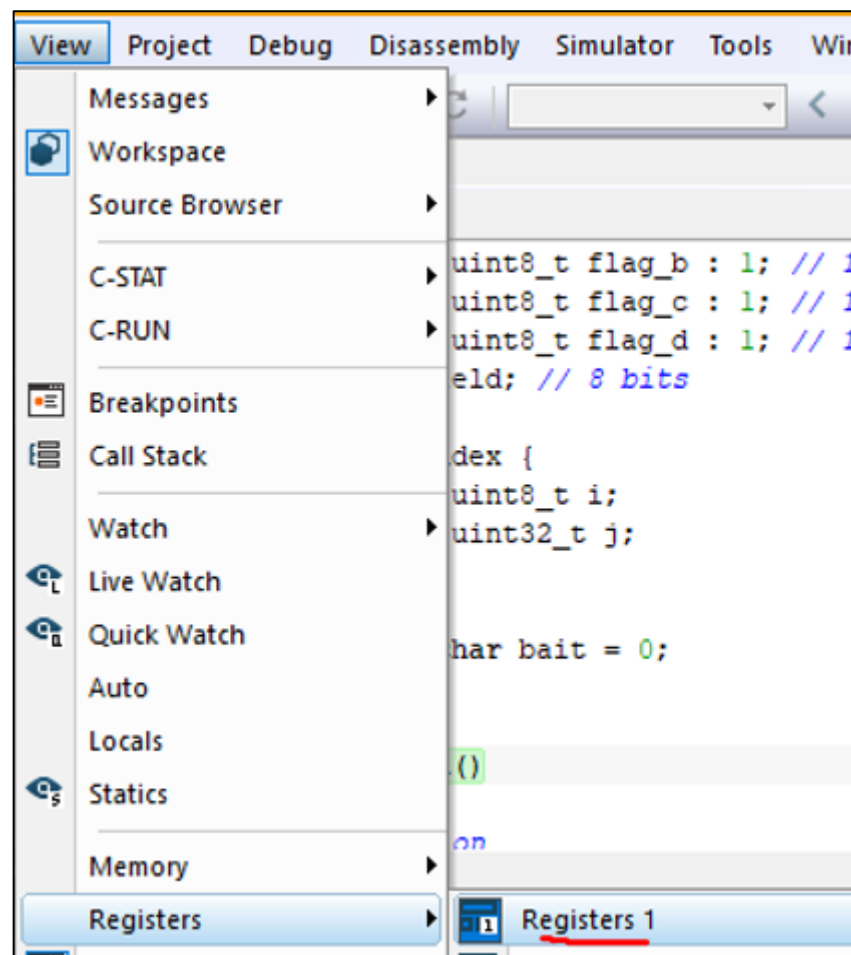


Рисунок 2.9 – Включение поля регистров

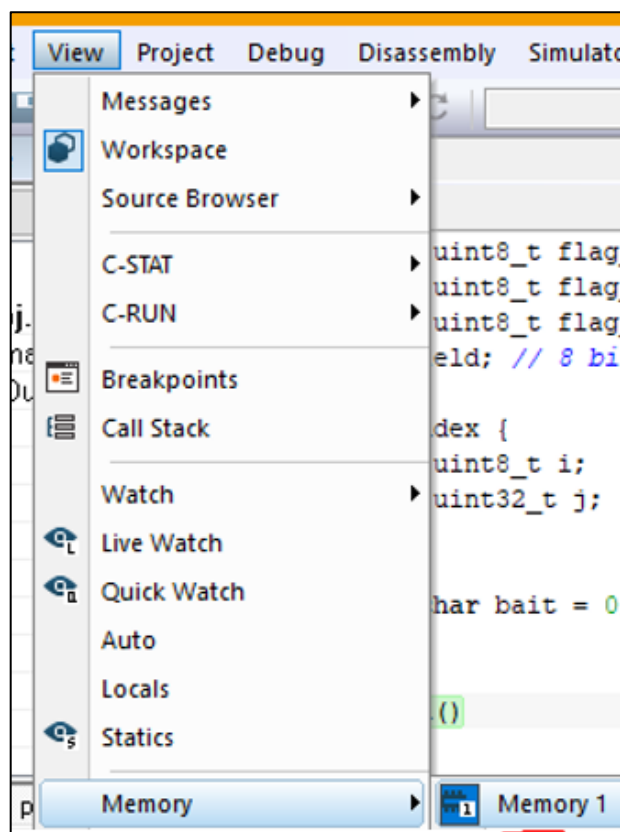


Рисунок 2.10 – Включение поля памяти

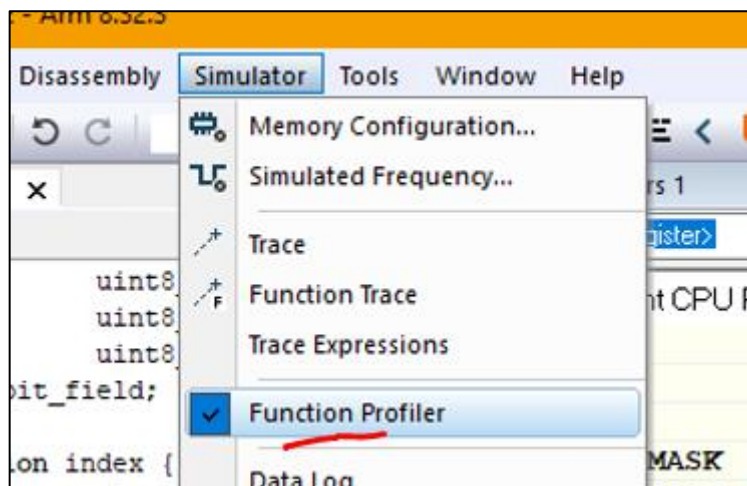


Рисунок 2.11 – Включение профилировщика

Для подсчёта числа тактов в окне «Register» присутствует регистр «CycleCounter», который начинает подсчёт тактов с момента начала работы программы.

Производится работа с битовыми полями, просмотр состояния памяти и состояния регистров. Выполнение записи производилось по адресу 0x20000000 для битового поля и 0x20000001 для «ручной» работы с битами. (Рисунок 2.12 – 2.15)

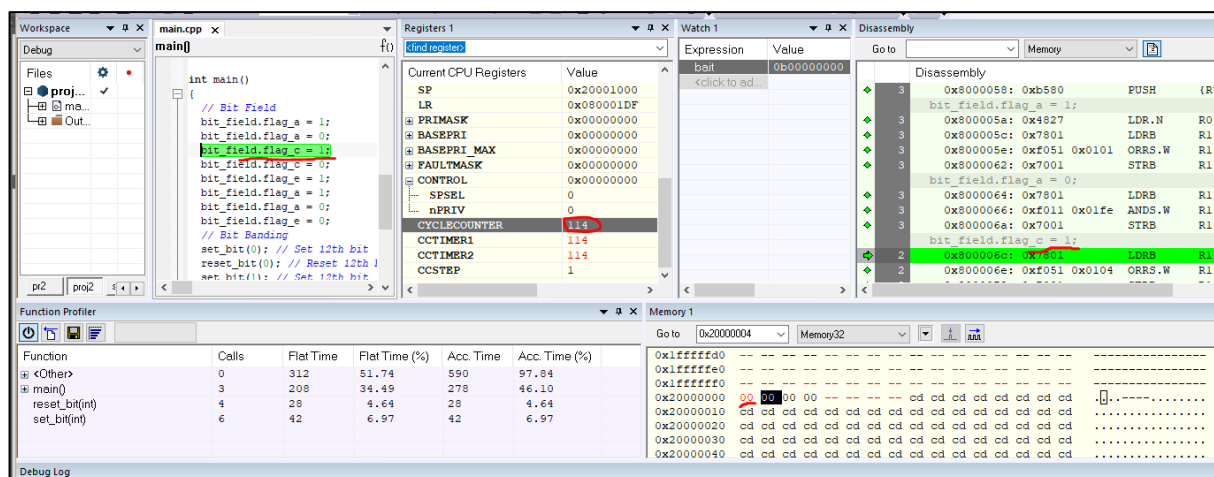


Рисунок 2.12 – До выполнения записи «1» в переменную «flag_c» битового поля

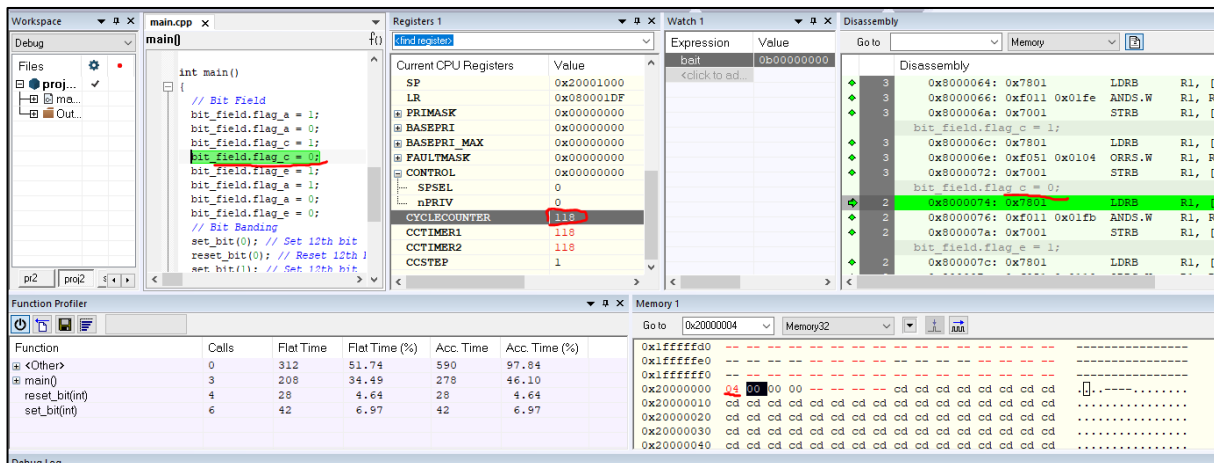


Рисунок 2.13 – После выполнения записи «1» в переменную «flag_c» битового поля

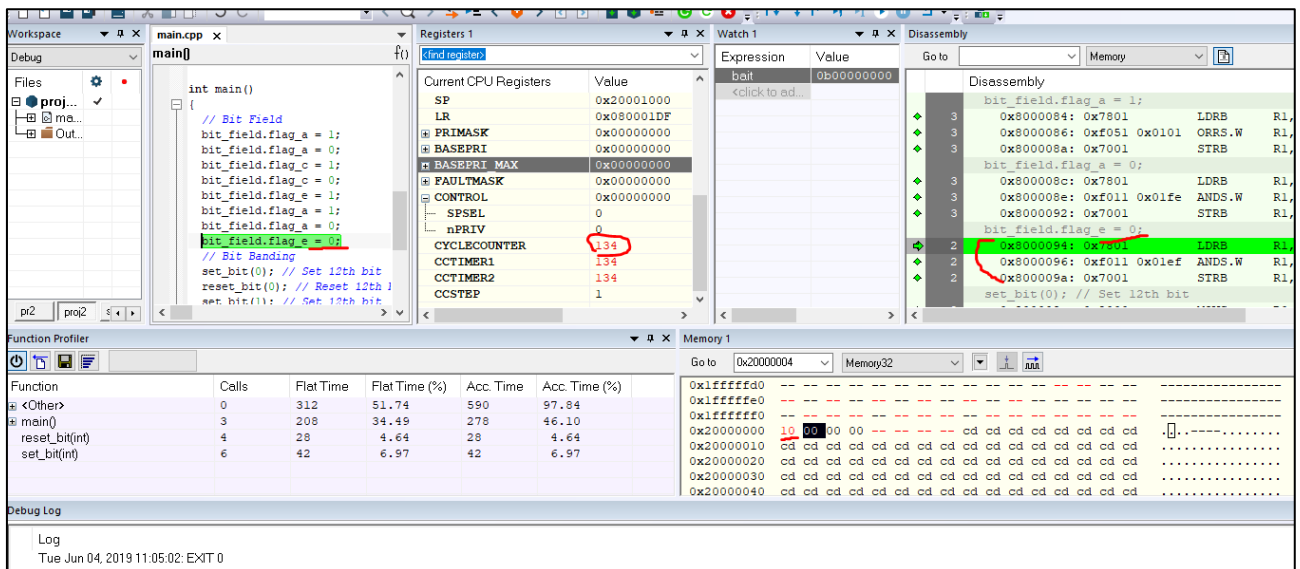


Рисунок 2.14 – После выполнения записи «1» в переменную «flag_e» битового поля

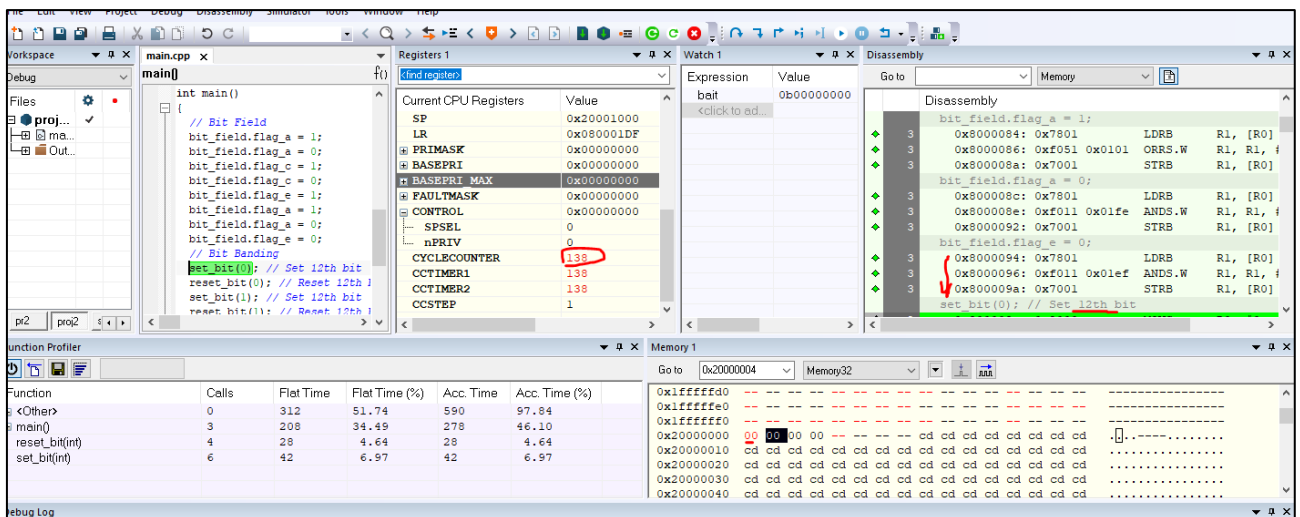


Рисунок 2.15 – После выполнения записи «0» в переменную «flag_e» битового поля

Для выполнения записи «1» в переменную «flag_c» битового поля необходимо 4 такта ($118-114=4$).

Для выполнения записи «0» в переменную «flag_e» битового поля необходимо 4 такта ($138-134=4$).

Далее производится работа с битами с помощью bit-banding. Выполнение записи производилось по адресу 0x20000000. (Рисунок 2.16 – 2.20)

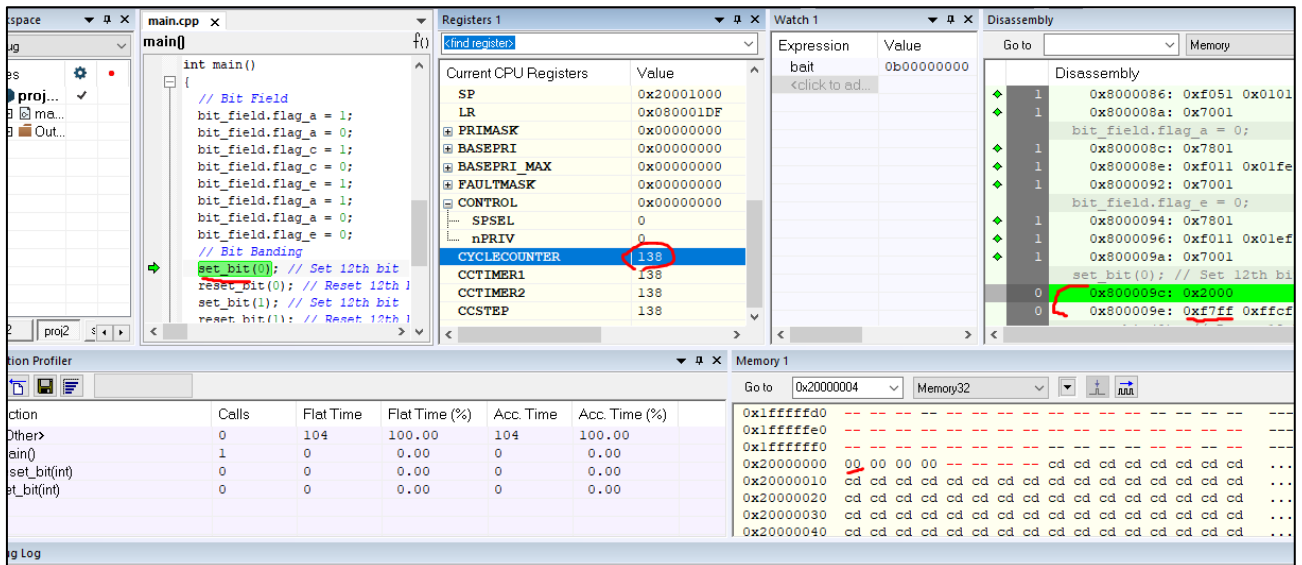


Рисунок 2.16 – Состояние перед выполнением записи

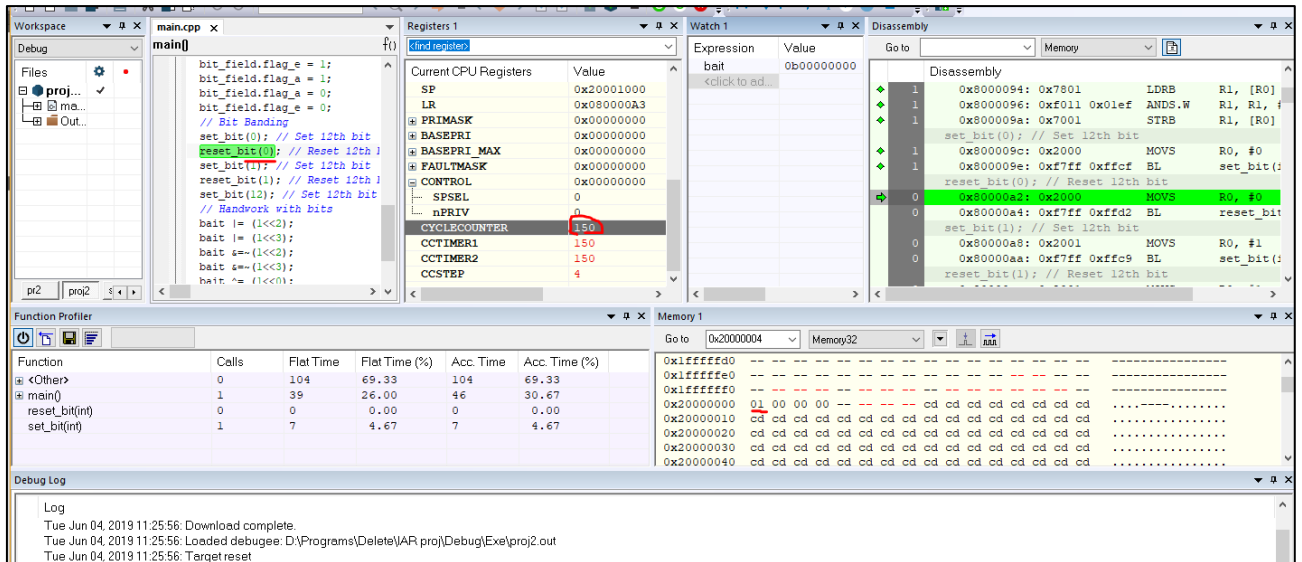


Рисунок 2.17 – Запись 1 в первый бит 0x20000000 с помощью bit-banding

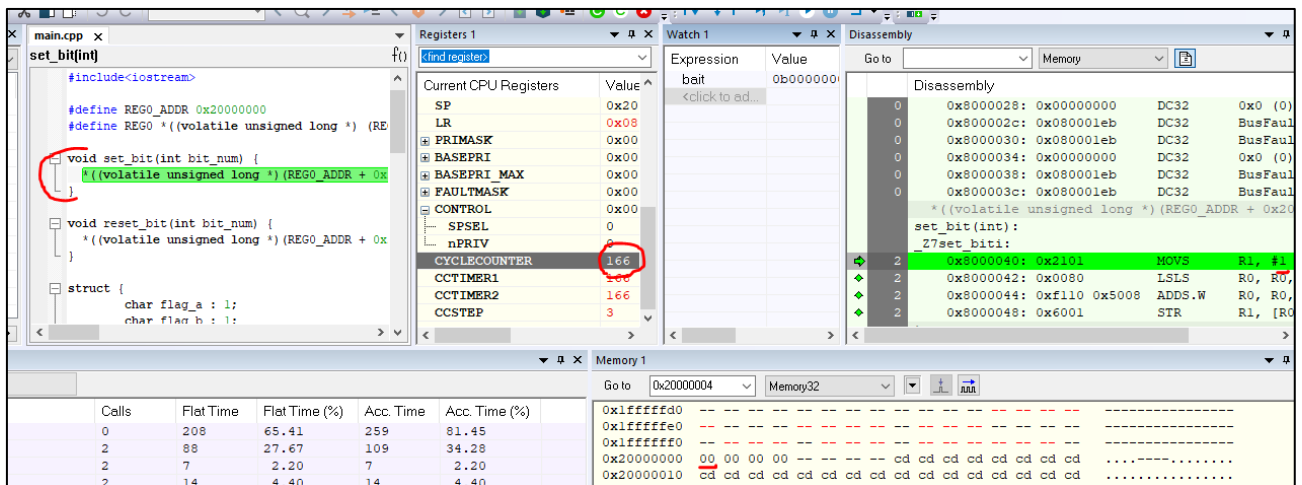


Рисунок 2.18 – Запись 2 в первый бит 0x20000000 с помощью bit-banding

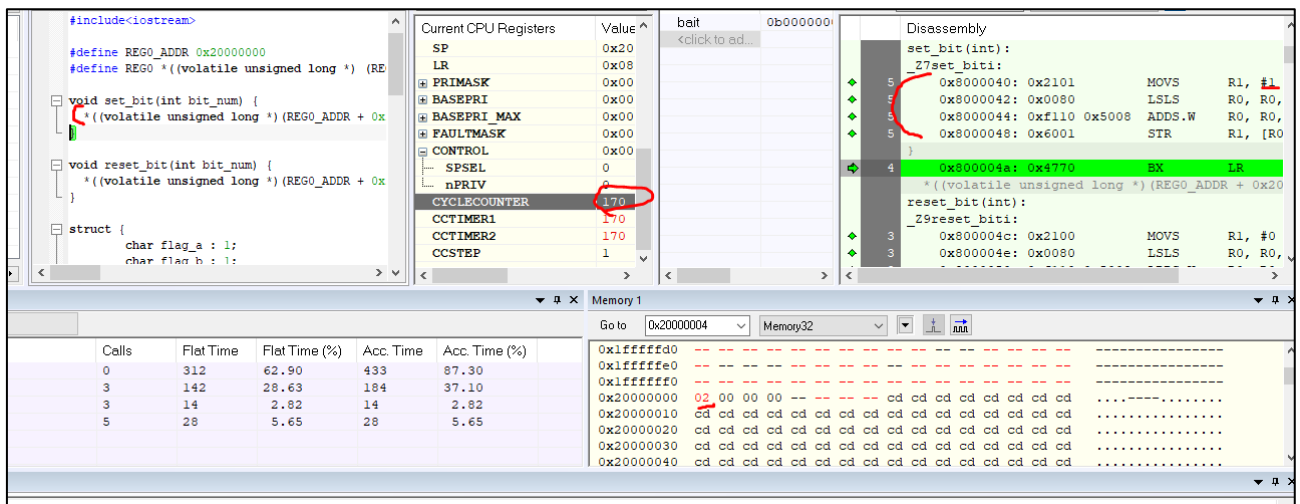


Рисунок 2.19 – Запись 2 в первый бит 0x20000000 с помощью bit-banding

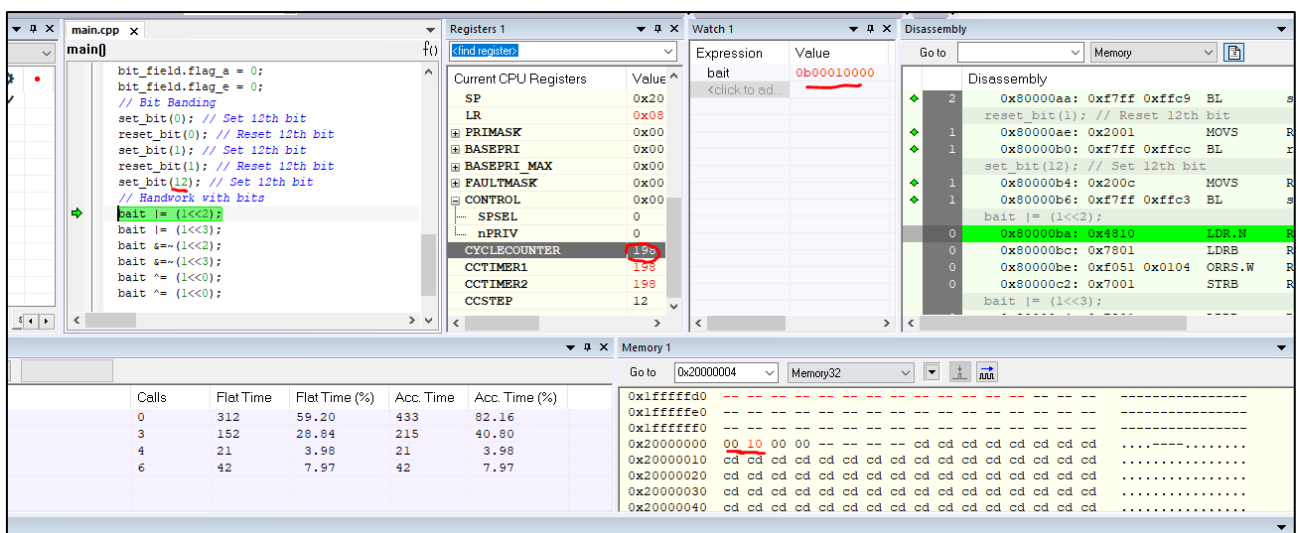


Рисунок 2.21 – Состояние до сложения с «100»

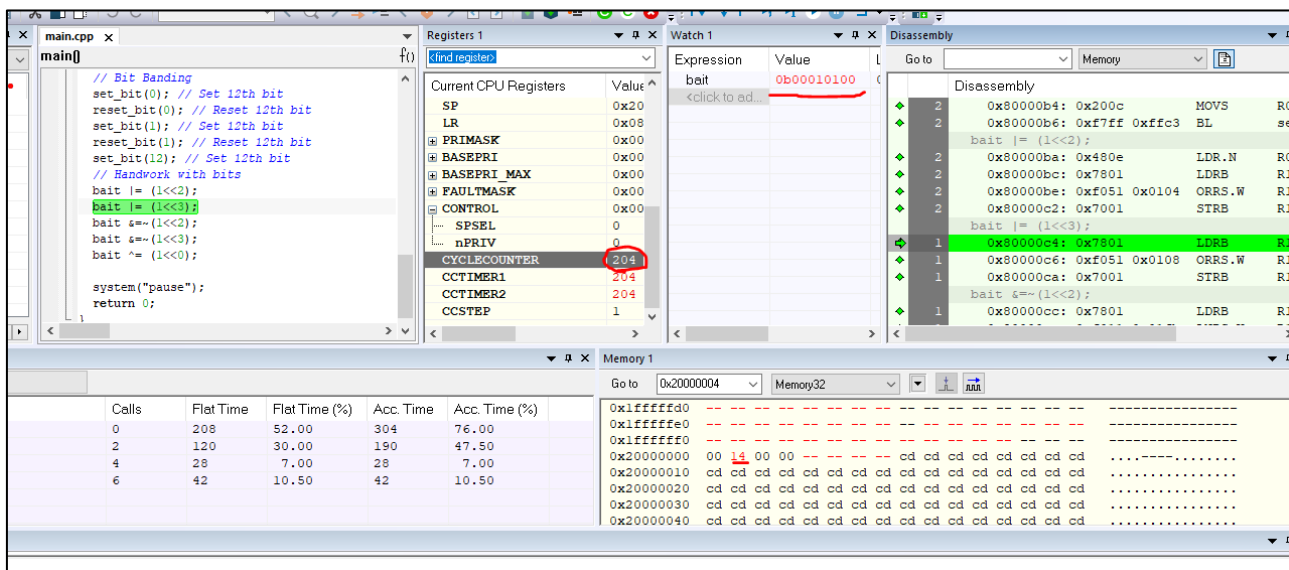


Рисунок 2.22 – Состояние после сложения с «100»

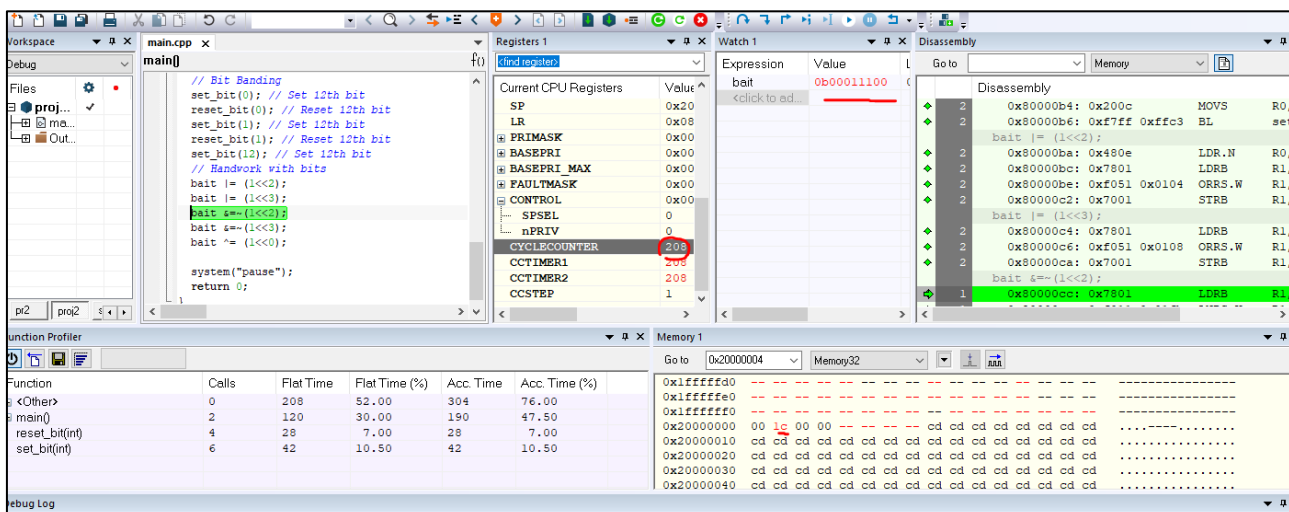


Рисунок 2.23 – Состояние после сложения с «1000»

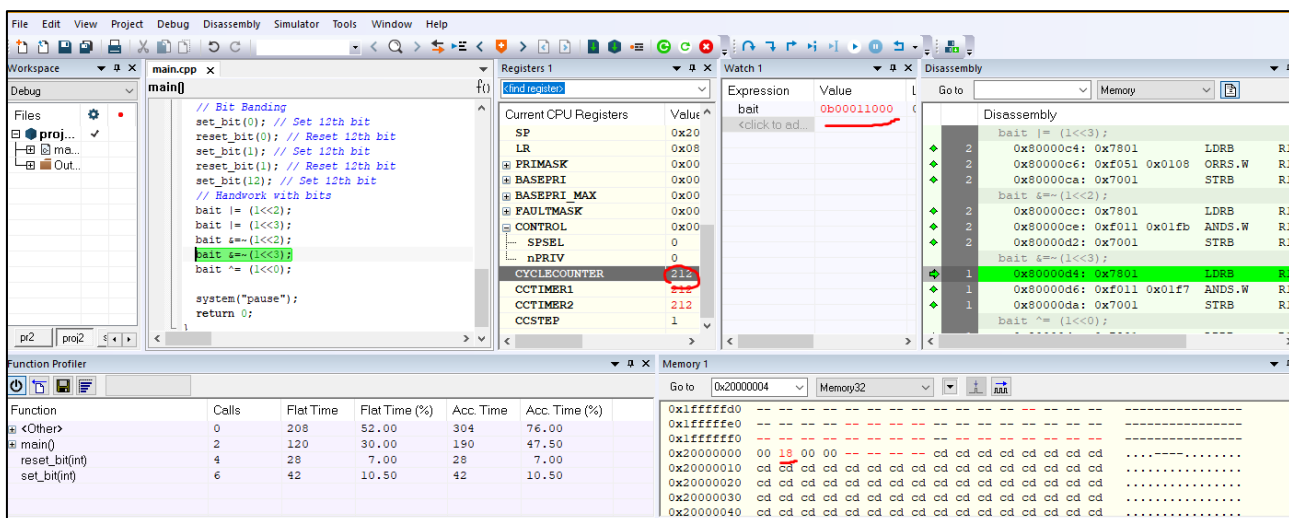


Рисунок 2.24 – Состояние после умножения на «11111011»

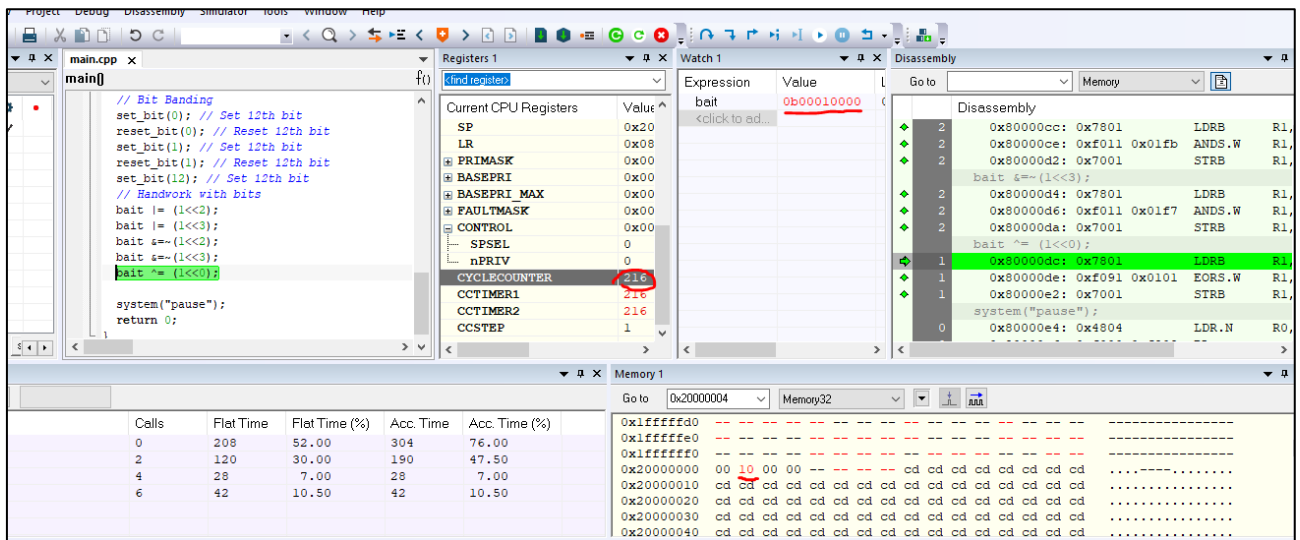


Рисунок 2.25 – Состояние после умножения на «11110111»

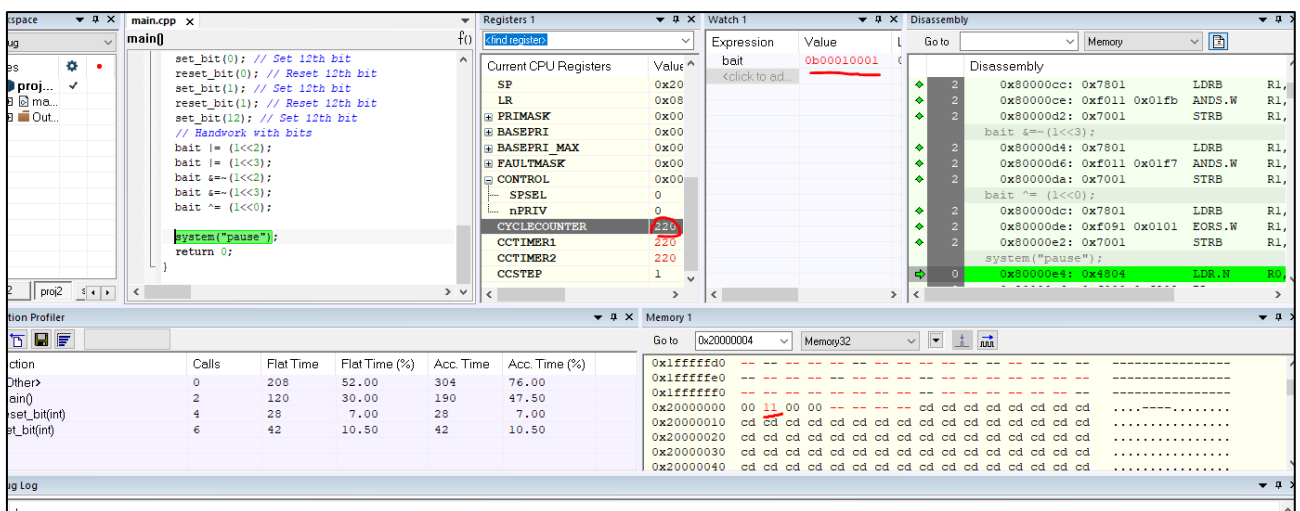


Рисунок 2.26 – Состояние после сложения с «1»

Видно, что «ручная» работа с битами занимает 4 такта (вычисляется как и в предыдущих методах по значению в CycleCounter).

3 Заключение

В ходе выполнения лабораторной работы прошло ознакомление с адресным пространством микроконтроллера MDR1986BE92QI

Была написана программа, реализующая три способа побитового доступа к данным: «bit-banding», битовое поле и «ручная» работа с битами.

Была выполнена профилировка и было выполнено сравнение разных методов доступа к битам. Было использовано встроенное средство в среде программирования IAR для проведения про-филировки кода. Отладка осуществлялась только в режиме симулятора, то встроенное средство профилировки позволяло отслеживать только количество раз выполнения той или иной команды.

Был написан отчёт согласно требованиям ОС ТУСУР 01-2013.