БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 2
Тема: «Прерывания. Таймеры»

Вариант №7

Выполнил:

Проверил:

Минск
2024

1. ПОСТАНОВКА ЗАДАЧИ

Написать программу, используя таймеры и прерывания в соответствии с заданием варианта.

2. АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Различают системные немаскируемые (SMNI), пользовательские немаскируемые (UNMI) и маскируемые прерывания. К системным немаскируемым относятся: сигнал RST/NMI в режиме NMI, сбой генератора, ошибка доступа Flash памяти. К пользовательским немаскируемым - сбой напряжения питания (от подсистемы PMM), доступ к несуществующей (vacant) памяти, события с буфером (mailslot) JTAG интерфейса. Маскируемые прерывания могут быть отключены (замаскированы) индивидуально или все сразу (бит GIE регистра состояния SR).

Из-за конвейерной архитектуры процессора, команда, следующая за EINT (разрешение прерывания), всегда выполняется, даже если запрос на прерывание возник до его разрешения. Если за EINT сразу следует DINT, прерывание, ожидающее обработки может быть не обслужено. Команды, следующие за DINT в этом случае могут сработать некорректно. Аналогичные последствия вызываются альтернативными командами, которые устанавливают и сразу сбрасывают флаг GIE регистра состояний. Рекомендуется вставлять хотя бы одну команду между EINT и DINT.

Возврат из прерывания выполняется командой RETI, которая выполняется за 5 циклов и загружает из стека SR, PC. Таблица векторов прерываний располагается по адресам 0FFFFh – 0FF80h и содержит 64 вектора. Бит SYSRIVECT регистра SYSCTL позволяет определить альтернативную таблицу векторов, в старших адресах RAM. По сигналу сброса этот бит автоматически сбрасывается.

За прерывания отвечают ряд системных регистров. Пользовательские маскируемые прерывания рассматриваются отдельно при обсуждении соответствующего функционального узла архитектуры микроконтроллера, в частности, ранее уже рассматривались регистры для работы с прерываниями от цифровых портов ввода-вывода.

Работа с прерываниями достаточно проста. Вначале необходимо разрешить соответствующее прерывание, например, P1IE |= BIT7; - разрешает прерывание по входу 7 вывода порта 1, в экспериментальной плате к нему подключена кнопка S1.

После того, как режим инициализирован, хорошим тоном считается

перевод контроллера в режим пониженного энергопотребления. Сделать это можно, используя вызов __bis_SR_register, например, следующий фрагмент переводит контроллер в режим LPM0 с разрешением прерываний:

```
__bis_SR_register(LPM0_bits + GIE);
```

Еще одной особенностью запуска в среде отладки Code Composer Studio является необходимость вызова __no_operation() перед завершением функции main, если она не использует некоторого цикла. Без этого вызова с завершением функции main завершится и выполнение кода в оболочке.

Собственно обработчик прерывания описывается с использованием директивы #pragma vector. Например, фрагмент кода ниже описывает обработчик прерывания от порта ввода-вывода 1:

```
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{...}
```

Таймер MSP430F5529 содержит 32-разрядный сторожевой таймер WDT (базовый адрес 015Ch), 3 таймера TAx (базовые адреса соответственно 0340h, 0380h, 0400h), таймер TBx (базовый адрес 03C0h) и таймер часов реального времени RTC_A (базовый адрес 04A0h).

Основная функция сторожевого таймера WDT – генерация сигнала сброса при программном сбое, например, зацикливании: если заданный интервал времени истек, генерируется сигнал сброса. WDT может быть сконфигурирован как интервальный и генерировать сигналы прерываний по истечении заданного промежутка времени.

Таймер A – это 16-разрядный таймер/счетчик с широкими возможностями по использованию прерываний, которые могут генерироваться счетчиком в случае переполнения и от каждого регистра захвата/сравнения. Таймер A обладает следующими возможностями:

- асинхронный 16-битный таймер/счетчик с четырьмя рабочими режимами;
- выбираемый и конфигурируемый источник счетного импульса;
- три конфигурируемых регистра захвата/сравнения (в таймере TA0 их 5);
- возможность множественного захвата/сравнения;
- конфигурируемые выводы с возможностью широтно-импульсной модуляции;
- асинхронная фиксация (защелка) входа и выхода;

- счет по фронту тактового импульса;
- возможность генерации прерываний при переполнении;
- регистр вектора прерываний для быстрого декодирования всех прерываний таймера А.

Источниками входного импульса для таймера А могут быть следующие тактовые сигналы: ACLK, SMCLK, внешние CAxCLK, INCLK. На входе имеется программно доступный делитель частоты, который позволяет снижать частоту в 2,3,4,5,6,7,8 раз. Режимы работы таймера: остановка, прямой счет (до уровня TAxCCR0) (Up Mode), непрерывный режим (Continuous Mode), реверсивный счет (Up/Down mode).

Таймер В имеет ряд отличий от таймера А:

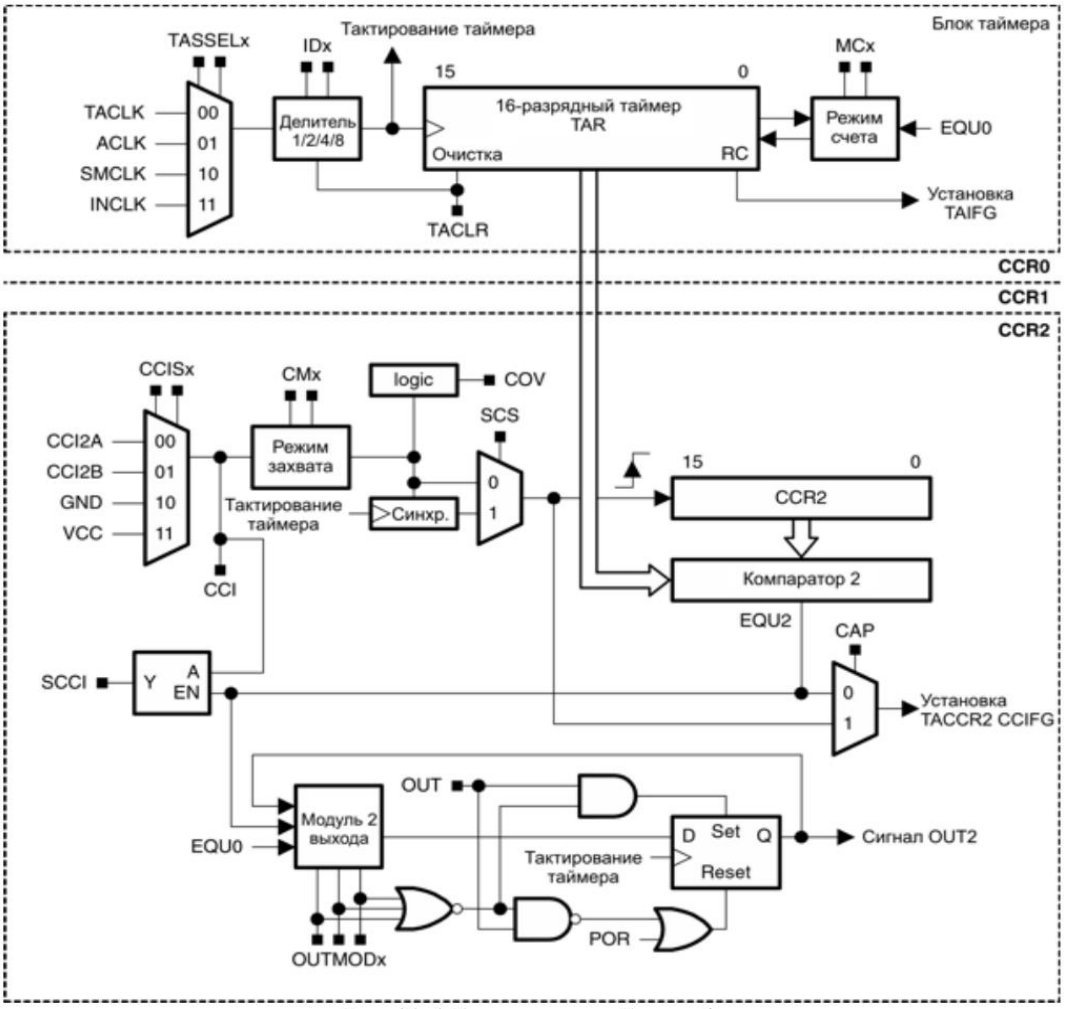
- 7 регистров захвата/сравнения;
- разрядность счетчика программируется (8, 10, 12, 16 бит);
- регистр TBxCCRn с двойной буферизацией и может быть сгруппирован;
- все выходы имеют высокоимпедансное состояние;
- не поддерживается бит SCCI.

Таймер часов реального времени RTC_A представляет собой конфигурируемые часы реального времени с функцией календаря и счетчика общего назначения. Поддерживает выбор формата BCD или двоичный в режиме часов реального времени, имеет программируемый будильник, подстройку коррекции времени, возможность прерываний.

Регистр счетчика WDT непосредственно программно не доступен. Сигнал на счетный вход может подаваться с тактовых линий SMCLK, ACLK, VLOCLK либо X_CLK от некоторых устройств. После сброса сторожевой таймер настроен на сторожевой режим, входным выбран сигнал от SMCLK. Поэтому необходимо остановить, установить либо сбросить таймер до истечения установленного интервала, иначе будет сгенерирован сигнал сброса PUC. Флаг запроса на прерывание сбрасывается автоматически после обслуживания, также может быть сброшен программно. Адреса обработчиков в сторожевом и интервальном режиме различны.

Первым этапом выполняется инициализация таймера TAx с помощью регистров TAxCTL, TAxCCRn и TAxCCTLn. В регистре TAxCTL рекомендуется выбрать в качестве источника тактирования SMCLK с выходной частотой тактирования 1МГц, режим счета, коэффициент деления и установить бит TACLR. В регистре счета/сравнения TAxCCTLn необходимо разрешить прерывания. В 16-битном регистре TAxCCRn указывается значение счетчика, при достижении которого в режиме прямого или реверсивного счета генерируется прерывание. При захвате значения для

его сохранения также используется данный регистр. Сброс состояния таймера осуществляется путем записи нулевого значения в конфигурационные регистры. TAxIV – 16-разрядный регистр вектора прерывания. Биты 0-2 регистра TAxEX0 (поле TAIDEX) устанавливают



параметры расширенного делителя входа (от деления на 1 при 000b до деления на 8 при 111b).

Рисунок 2.1 - Структура таймера

Таблица 2.4. Регистры таймера A

Регистр	**Адрес**	**Назначение**
TAxCTL	0340h	Регистр управления
TAxCCTL0-6	0342h-034Eh	Управление захватом/сравнением
TAxR	0350h	Счетчик
TAxCCR0-6	0352h-035Eh	Захват/сравнение
TAxIV	036Eh	Вектор прерывания

TAxEX0	0360h	Расширение 0

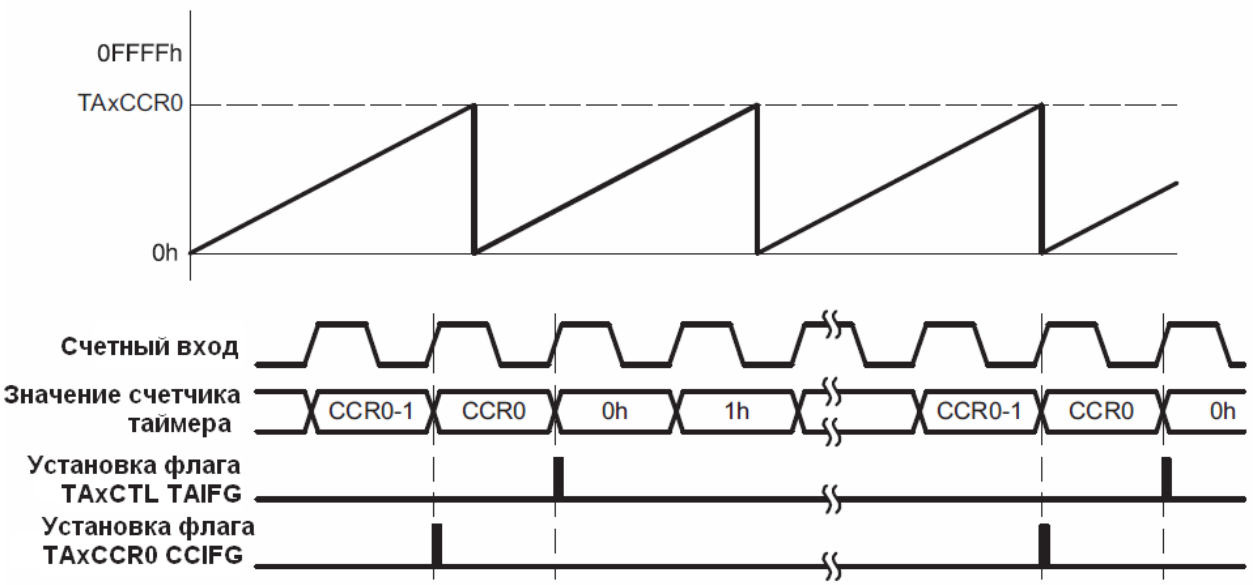


Рисунок 2.2 - Режим прямого счета

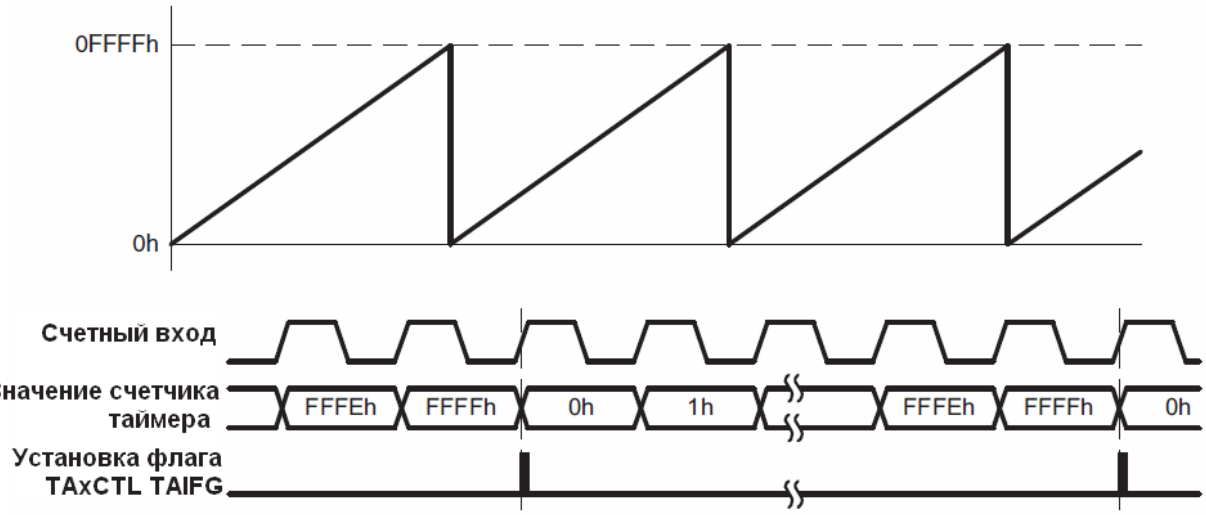


Рисунок 2.3 - Непрерывный режим

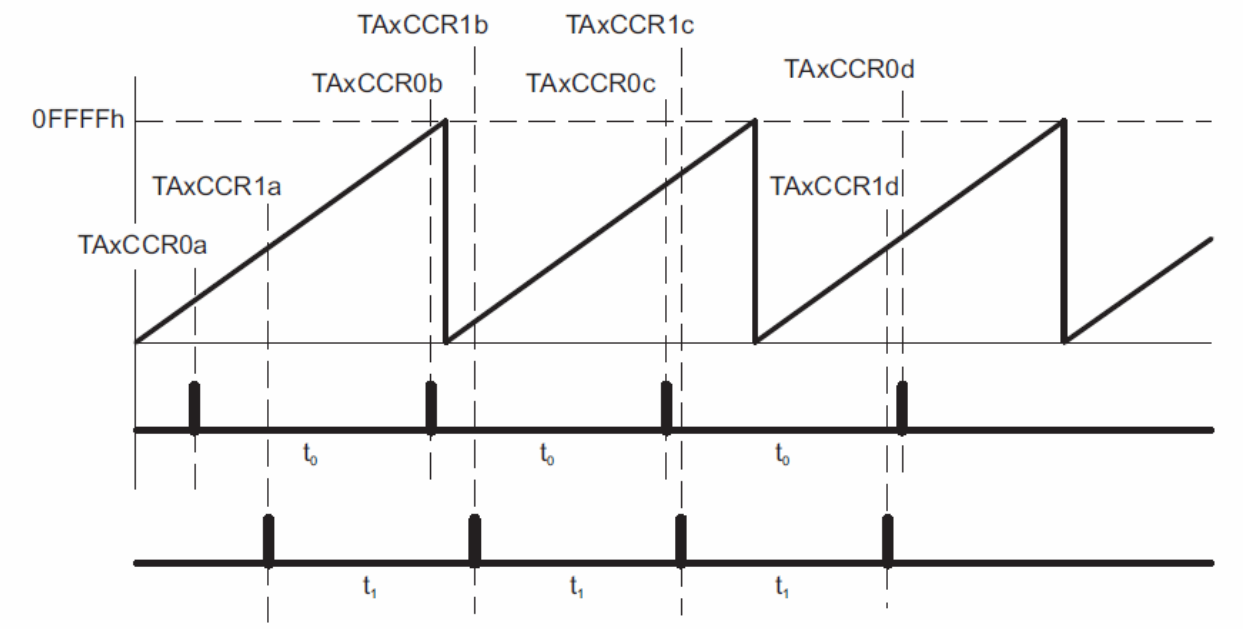


Рисунок 2.4 - Генерирование нескольких интервалов в непрерывном режиме

В режиме прямого счета (рисунок 2.2) таймер считает от 0 до значения, установленного в регистре TAxCCR0. При достижении установленного значения таймер продолжает счет с 0. Количество тактовых импульсов в периоде равно TAxCCR0+1. Флаг прерывания TAxCCR0 CCIFG устанавливается, когда счетчик досчитал до значения TAxCCR0. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик переходит от TAxCCR0 к 0.

В непрерывном режиме (рисунок 2.3) таймер считает от 0 до 0FFFFh. Регистр захвата/сравнения TAxCCR0 работает аналогично остальным регистрам захвата/сравнения. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик переходит от 0FFFFh к 0.

Непрерывный режим можно использовать для генерирования независимых выходных интервалов и временных частот. При окончании любого из интервалов, генерируется прерывание. Следующий временной интервал добавляется к TAxCCRn обработчиком прерываний. Можно генерировать столько независимых интервалов, сколько имеется регистров захвата/сравнения. пример для двух интервалов приведен на рисунке 2.4.

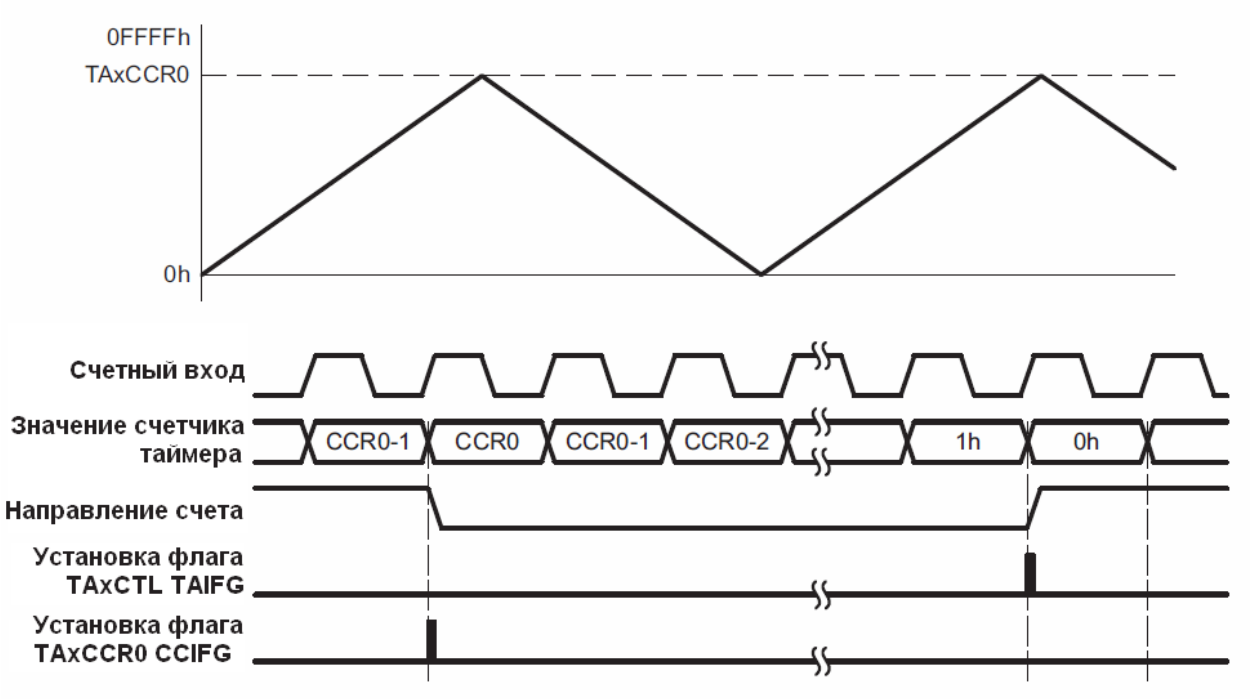


Рисунок 2.5 - Реверсивный режим

В реверсивном режиме (рисунок 2.5) таймер считает от 0 до значения, установленного в регистре TAxCCR0. При достижении установленного значения таймер продолжает счет в обратном направлении к 0. Период счета равен удвоенному значению TAxCCR0. Направление счета запоминается, что позволяет выполнять остановку таймера, а затем продолжить счет с прерванного места. Флаг прерывания TAxCCR0 CCIFG устанавливается, когда счетчик досчитал до значения TAxCCR0. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик досчитал в обратном направлении от TAxCCR0 до 0.

Реверсивный режим позволяет поддерживать пустые интервалы (Dead Time) между выходными сигналами, когда ни один из них не активен (рисунок 2.6). Регистры TAxCCRn не имеют буфера, поэтому они изменяются сразу после записи.

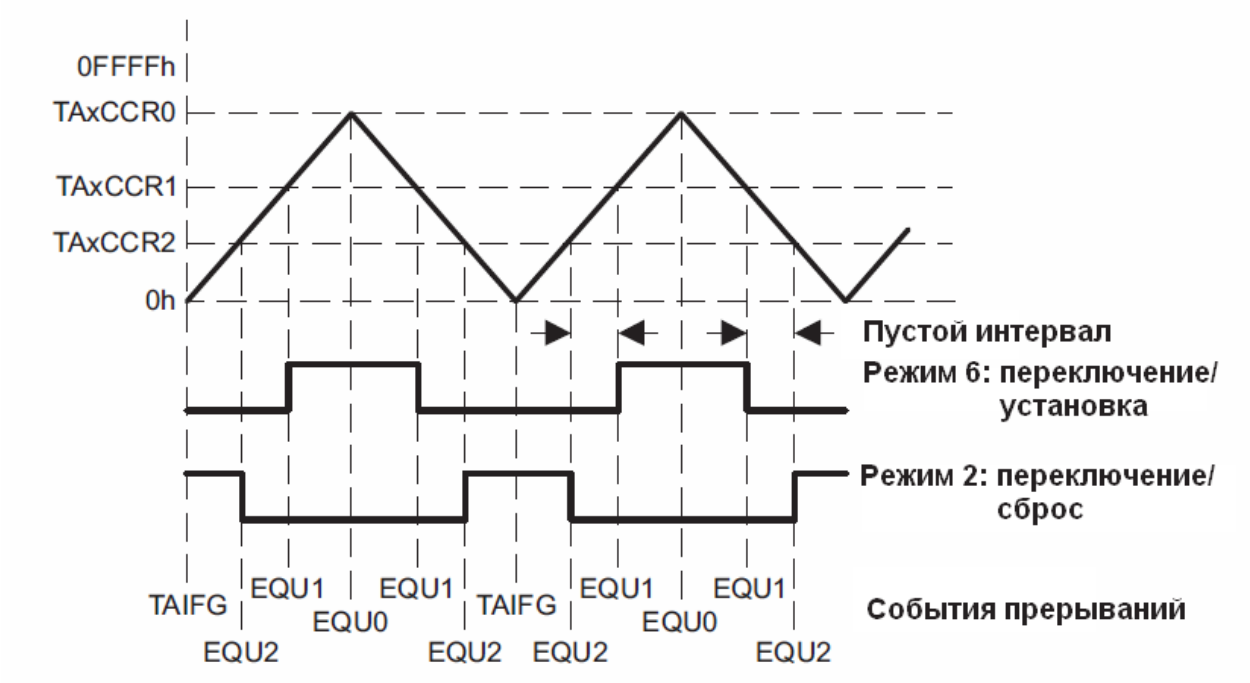


Рисунок 2.6 - Генерирование пустых интервалов в реверсивном режиме

Режим захвата выбирается, когда CUP = 1. Используется для записи времени какого-либо события. В качестве сигналов на входы захвата CCIxA и CCIxB могут быть поданы сигналы с внешних выводов или внутренние сигналы. Источник выбирается CCIS битами. Биты CM определяют, будет ли захват происходить по фронту сигнала, по спаду, либо и по фронту и по спаду. При наступлении соответствующего события значение счетчика копируется в регистр TAxCCRn и устанавливается флаг прерываний CCIFG. Уровень входного сигнала может быть прочитан в любое время из бита CCI. Поскольку сигнал на входе не синхронизирован с тактовыми импульсами, могут возникать гонки. Поэтому рекомендуется устанавливать бит SCS, чтобы захват происходил с началом очередного тактового импульса. Захват может быть выполнен программно.

Режим сравнения выбирается, когда CUP = 0. Используется для генерации на выходе ШИМ-сигнала или прерывания через заданный временной интервал. Когда счетчик достигает значения TAxCCRn, устанавливается флаг прерывания CCIFG, внутренний сигнал EQUn устанавливается в 1, EQUn влияет на выход в соответствии с режимом, а входной сигнал CCI защелкивается в регистре SCCI.

Каждый блок захвата/сравнения содержит выходной модуль, который формирует выходной сигнал на основе EQU0 и EQUn сигналов в зависимости от установленного режима выхода. Биты OUTMOD позволяют

задать один из 8 режимов. Сигнал OUTn изменяется по переднему фронту синхросигнала, за исключением режима 0. Режимы 2, 3, 6 и 7 не пригодны для использования с выходным блоком 0, поскольку EQUn = EQU0.

Режимы выхода:

000 — Значение бита OUT. Сигнал OUTn изменяется сразу же с изменением бита OUT;

001 — Установка. Однократная установка при достижении заданного значения TAxCCRn;

010 — Переключение/сброс. Выход меняется при достижении значения TAxCCRn, сбрасывается при достижении TAxCCR0;

011 — Установка/сброс. Выход устанавливается при достижении значения TAxCCRn, сбрасывается при достижении TAxCCR0;

100 — Переключение. Выход меняется при достижении значения TAxCCRn;

101 — Сброс. Однократный сброс при достижении заданного значения TAxCCRn;

110 — Переключение/установка. Выход меняется при достижении значения TAxCCRn, устанавливается при достижении TAxCCR0;

111 — Сброс/установка. Выход сбрасывается при достижении значения TAxCCRn, устанавливается при достижении TAxCCR0.

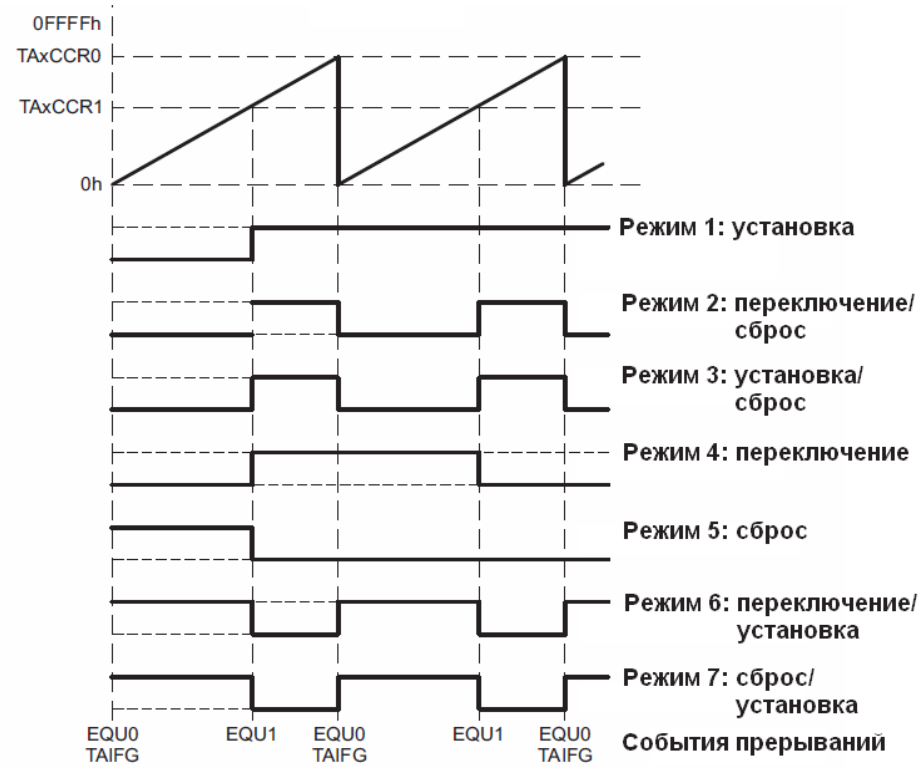


Рисунок 2.7 - Режимы выхода в режиме прямого счета

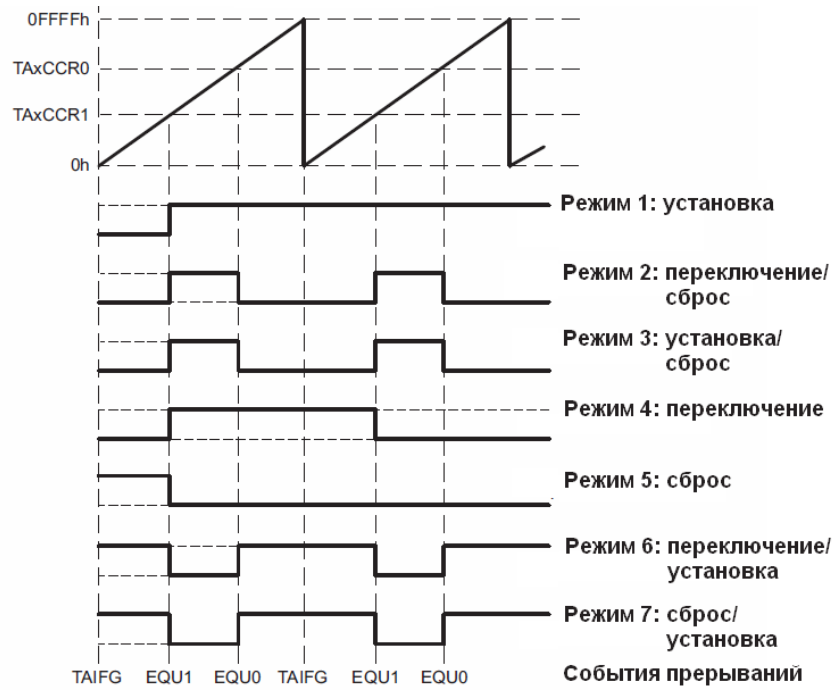


Рисунок 2.8 - Режимы выхода в непрерывном режиме

На рисунках 2.7 — 2.9 представлены примеры различных режимов выхода в режиме прямого счета, непрерывном и реверсивном режимах соответственно.

При использовании констант (msp430f5529.h) не стоит забывать принципы их именования:

- константа, соответствующая биту поля-флага именуется по имени поля, например, полю CPUOFF регистра состояния процессора SR (бит 4) соответствует константа CPUOFF;
- константа соответствующая биту n в поле NNN именуется NNNn;
- константа, соответствующая номеру x выбранного варианта для поля NNN именуется NNN_x;
- константа, соответствующая выбранному режиму zz для поля NNN именуется NNN__zz.

Так, например, для 3-битного поля SELA, константа, соответствующая 0 биту поля, именована SELA0, вариант выбора 0 (SELA = 000) именован SELA_0, а режим, соответствующий данному варианту именован SELA__XT1CLK. В некоторых случая поля задают делители либо множители, соответствующие степени двойки. Тут надо быть особо внимательным и не спутать похожие мнемоники, например, NN4 (четвертый бит, т.е. 10000), NN_4 (четвертый вариант, т.е. 00100), NN__4 (режим деления на 4, т.е. 00011).

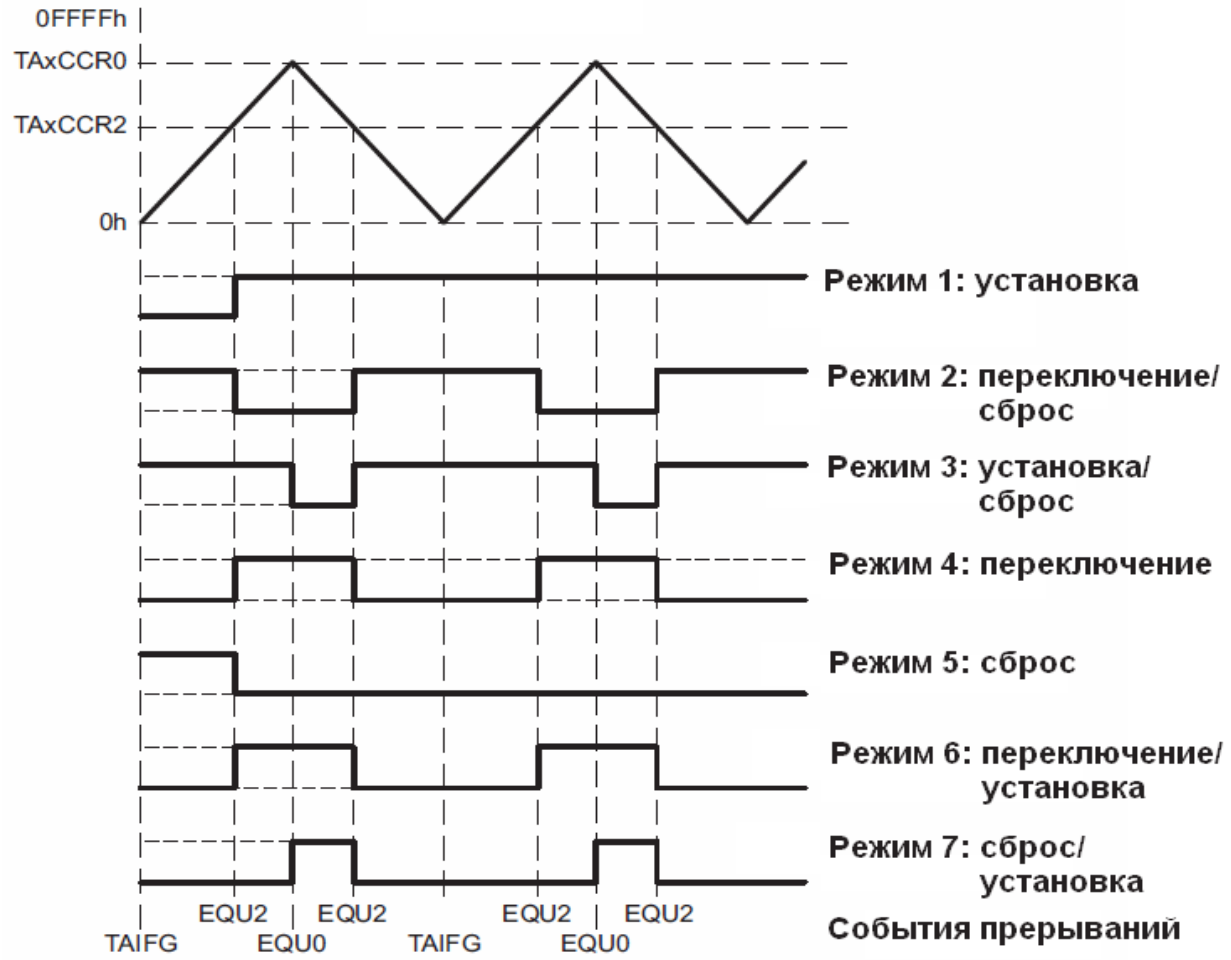


Рисунок 2.9 - Режимы выхода в реверсивном режиме

3. ЛИСТИНГ ПРОГРАММЫ

```
#include <msp430.h>

#define LED1 BIT0      // Светодиод 1 подключен к P1.0
#define LED2 BIT1      // Светодиод 2 подключен к P8.1
#define LED3 BIT2      // Светодиод 3 подключен к P8.2
#define BUTTON BIT7    // Кнопка S1 подключена к P1.7

void delay(unsigned int cycles); // Прототип функции задержки

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // Останавливаем watchdog таймер

    // Настройка портов для светодиодов
    P1DIR |= LED1;               // Устанавливаем LED1 как выход
    P1OUT &= ~LED1;              // Изначально LED1 выключен

    P8DIR |= LED2 + LED3;        // Устанавливаем LED2 и LED3 как выходы
    P8OUT &= ~(LED2 + LED3);     // Изначально LED2 и LED3 выключены

    // Настройка кнопки
    P1DIR &= ~BUTTON;            // Устанавливаем кнопку как вход
    P1REN |= BUTTON;                    // Включаем внутренний резистор
подтяжки кнопки
    P1OUT |= BUTTON;             // Подтягиваем резистор к Vcc

    unsigned char button_state = 0; // Переменная для отслеживания
состояния кнопки

    while (1)
    {
        if ((P1IN & BUTTON) == 0) // Проверяем, нажата ли кнопка (низкий
уровень)
        {
            __delay_cycles(20000); // Задержка для антидребезга

            if ((P1IN & BUTTON) == 0) // Дополнительная проверка кнопки
            {
                button_state ^= 1; // Инвертируем состояние кнопки

                if (button_state == 1) // Если кнопка нажата первый раз
                {
                    P1OUT |= LED1;     // Включаем LED1
                    delay(10000);      // Задержка
                    P8OUT |= LED2;     // Включаем LED2
                    delay(10000);      // Задержка
                    P8OUT |= LED3;     // Включаем LED3
                }
                else // Если кнопка нажата повторно
                {
                    P1OUT &= ~LED1;    // Выключаем LED1
                    P8OUT &= ~(LED2 + LED3); // Выключаем LED2 и LED3
                }
```

```
                    while ((P1IN & BUTTON) == 0); // Ожидаем, пока кнопку
отпустят
                }
            }
        }
    }

    // Функция для создания задержки
    void delay(unsigned int cycles)
    {
        while (cycles--)
        {
            __no_operation(); // Ничего не делаем, просто ждем
        }
    }
```

4. ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы ознакомился с работой подсистемы прерываний и таймерами микроконтроллера MSP430F5529. Используя прерывания и таймеры, запрограммировал кнопки и светодиоды.