БГУИР
Кафедра ЭВМ

Отчет по лабораторной работе № 2
Прерывания. Таймеры
Вариант №3

Выполнил:
студент группы 050503:
Липский Г.В.

Проверил:
Шеменков В.В.

Минск
2023

1. Цель работы

Ознакомиться с работой подсистемы прерываний и таймерам микроконтроллера MSP430F5529.

2. Исходные данные к работе

В соответствии с вариантом, используя прерывания и таймеры, запрограммировать кнопки и светодиоды. Для работы с кнопками использовать только прерывания. Не использовать опросы флагов состояния в цикле и циклы задержки (активное ожидание). Не допускается использовать иные заголовочные файлы, кроме msp430, не допускается также использовать высокоуровневые библиотеки. При выполнении задания особое внимание уделить грамотному выбору режима работы таймера. Комментарии в тексте программы обязательны, они должны пояснять что именно делает данные фрагмент.

Вариант 12:

Диоды, кнопки и таймеры	Режим	Переключение
LED3, S1, S2, WDT	Импульсный	Короткий импульс при нажатии и при отпускании одной кнопки. Изменение длительности импульса при нажатии второй кнопки

3. Теоретические сведения

3.1 Прерывания

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).

Кратко рассмотрим, как происходит обработка прерывания. Задержка от возникновения запроса на прерывание до начала выполнения обработчика составляет 6 циклов. При этом заканчивается выполнение текущей инструкции, счетчик команд PC сохраняется в стеке (указывает на следующую команду), регистр состояния SR сохраняется в стеке, выбирается прерывание с максимальным приоритетом (если поступило несколько запросов), автоматически сбрасывается флаг запроса от отдельного прерывания (сброс общего флага запроса должен осуществляться программно). Далее, все биты SR сбрасываются, за исключением SCG0, так как останавливаются все режимы с низким питанием. Так как бит GIO при этом устанавливается в 0, все прерывания запрещаются. Наконец, вектор (адрес обработчика) загружается в PC.

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 (табл. 2.1). 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. В табл. 2.2 представлены поля системных регистров для работы с прерываниями.

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-выода 1:

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

Таблица 2.1. Регистры для работы с прерываниями

Регистр	Адрес	Назначение
SFRIE1	0100h	Разрешение прерываний
SFRIFG1	0102h	Флаги прерываний
SYSCTL	0180h	Регистр управления
SYSBERRIV	0198h	Генератор вектора ошибок шины
SYSUNIV	019Ah	Генератор вектора пользовательских NMI
SYSSNIV	019Ch	Генератор вектора системных NMI
SYSRSTIV	019Eh	Генератор вектора сброса

Таблица 2.2. Поля регистров для работы с прерываниями

Регистр	Биты	Поле	Назначение
SFRIE1	7	JMBOUTIE	Разрешение прерываний выхода JTAG
	6	JMBINIE	Разрешение прерываний входа JTAG
	5	ACCVIE	Разрешение прерываний нарушения доступа Flash
	4	NMIIE	Разрешение прерываний вывода NMI
	3	VMAIE	Разрешение прерываний доступа к несуществующей памяти
	1	OFIE	Разрешение прерываний сбоя генератора
	0	WDTIE	Разрешение прерываний сторожевого таймера
SFRIFG1	7	JMBOUTIFG	Флаг прерывания выхода JTAG
	6	JMBINIFG	Флаг прерывания входа JTAG
	4	NMIIFG	Флаг прерывания NMI
	3	VMAIFG	Флаг прерывания доступа к несуществующей памяти
	1	OFIFG	Флаг прерывания сбоя генератора
	0	WDTIFG	Флаг прерывания сторожевого таймера
SYSCTL	0	SYSRIVECT	Вектор прерывания при выходе за пределы RAM (64K или полностью)
SYSUNIV	0-15	SYSUNIV	Вектор пользовательского NMI
SYSSNIV	0-15	SYSSNIV	Вектор системного NMI
SYSRSTIV	0-15	SYSRSTIV	Вектор прерываний сброса
SYSBERRIV	0-15	SYSBSLOFF	Вектор прерываний ошибки системной шины

3.2 Таймеры

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 А 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 ТА0 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.

Рассмотрим подробнее работу со сторожевым таймером WDT. Он имеет 8 программно выбираемых временных интервалов, поддерживает сторожевой и интервальный режимы, обеспечивает защиту доступа к управляющему регистру, может отключаться для экономии энергии. Важным свойством WDT является отказоустойчивый сигнал (источник счетного сигнала не может быть отключен, пока таймер в сторожевом режиме). Это может не позволить перейти в режим пониженного потребления энергии (LPM).

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.

Разрешение прерываний WDT осуществляется битом WDTIE регистра SFRIE1, флаг прерывания — бит WDTIFG в регистре SFRIFG1. В таблице представлены поля регистра управления WDTCTL:

Таблица 2.3. Поля регистра WDTCTL

Биты	Поле	Назначение
15 -- 8	WDTPW	Пароль на доступ к регистру
7	WDTHOLD	Остановка таймера (бит = 1)
6--5	WDTSSEL	Выбор источника счетного сигнала
4	WDTTMSEL	Выбор режима: 0 — сторожевой, 1 — интервальный
3	WDTCNTCL	Очистка регистра счетчика
2--0	WDTIS	Выбор интервала (входная частота делится на константу)

Рассмотрим особенности работы с таймером А, таймеры В и RTC_A подробно рассматривать не будем. Структура таймера изображена на рис. 2.1. В последующих таблицах представлены регистры таймера и некоторые поля регистров таймера.

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 регистраTAxEX0 (поле TAIDEX) устанавливают параметры расширенного делителя входа (от деления на 1 при 000b до деления на 8 при 111b).

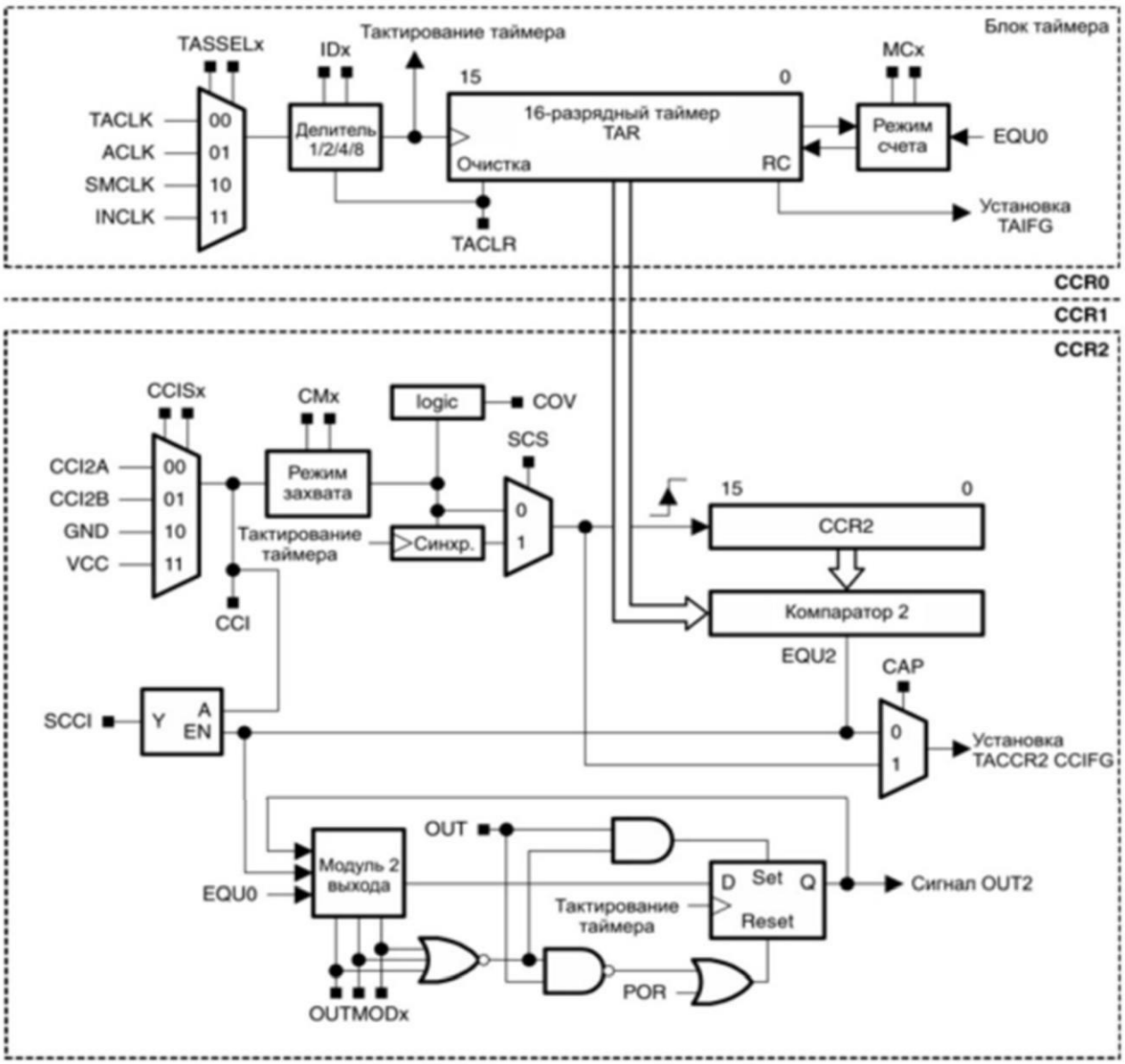

Рис. 2.1 Структура таймера

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

Таблица 2.5. Регистр управления TAxCTL

Бит	Поле	Тип	Сброс	Описание	Определения флагов в msp430f5529.h
9-8	TASSEL	RW	0h	Выбор источника тактирования: 00b TACLK 01b ACLK 10 SMCLK 11 INCLK	 TASSEL__TACLK TASSEL__ACLK TASSEL__SMCLK TASSEL__INCLK
7-6	ID	RW	0h	Входной делитель. Эти биты позволяют выбрать коэффициент деления для входной тактовой частоты. 00b /1 01b /2 10b /4 11b /8	 ID__1 ID__2 ID__4 ID__8
5-4	MC	RW	0h	Выбор режима. Установка MCx=00h, когда таймер не используется, позволяет уменьшить потребляемую мощность. 00b остановка: таймер остановлен 01b прямой счет: вверх к TAxCCR0 10b непрерывный: вверх к 0FFFFh 11b реверсивный: вверх к TAxCCR0, затем вниз к 0000h	 MC__STOP MC__UP MC__CONTINOUS MC__UPDOWN
2	TACLR	RW	0h	Очистка таймера A. Установка этого бита сбрасывает TAxR, IDx и MCx. Бит TACLR автоматически сбрасывается и всегда читается как нуль.	TACLR
1	TAIE	RW	0h	Разрешение прерывания от таймера A. Этот бит разрешает запрос прерывания TAIFG. 0b Запрещение прерывания 1b Разрешение прерывания	TAIE
0	TAIFG	RW	0h	Флаг прерывания Таймера A 0b Прерывание не ожидается 1b Ожидается прерывание	TAIFG

Таблица 2.6. Регистр управления захватом/сравнением TAxCCTLn

Бит	Поле	Тип	Сброс	Описание	Определения флагов в msp430f5529.h
15-14	CM	RW	0h	Выбор режима захвата 00 Нет захвата 01 Захват по нарастающему (переднему) фронту 10 Захват по заднему фронту (сбросу) 11 Захват как по переднему, так и по заднему фронтам	 CM_1 CM_2 CM_3 CM_4
13-12	CCIS	RW	0h	Выбор входа захвата/сравнения. Этими битами выбирается входной сигнал TAxCCR0. 00 CCIxA 01 CCIxB 10 GND 11 VCC	 CCIS_0 CCIS_1 CCIS_2 CCIS_3
11	SCS	RW	0h	Синхронизация источника захвата. Используется для синхронизации входного сигнала захвата с тактовым сигналом таймера 0 Асинхронный захват 1 Синхронный захват	SCS
10	SCCI	RW	0h	Синхронизация входа захвата/сравнения. Выбранный входной сигнал CCI фиксируется по сигналу EQUx и может быть прочитан через этот бит	CCIS0 CCIS1
8	CAP	RW	0h	Выбор режима захвата 0 Режим сравнения 1 Режим захвата	CAP
7-5	OUTMOD	RW	0h	Выбор режима выхода. Режимы 2, 3, 6 и 7 не пригодны для TAxCCR0, поскольку EQUx=EQU0. 000 Значение бита OUT 001 Установка 010 Переключение/сброс 011 Установка/сброс 100 Переключение 101 Сброс 110 Переключение/установка 111 Сброс/установка	 OUTMOD_0 OUTMOD_1 OUTMOD_2 OUTMOD_3 OUTMOD_4 OUTMOD_5 OUTMOD_6 OUTMOD_7

Окончание табл. 2.6

Бит	Поле	Тип	Сброс	Описание	Определения флагов в msp430f5529.h
4	CCIE	RW	0h	Разрешение прерывания по захвату/сравнению. Этот бит разрешает запрос прерывания от соответствующего флага CCIFG. 0 Запрещение прерывания 1 Разрешение прерывания	CCIE
3	CCI	R	0h	Вход захвата/сравнения. Выбранный входной сигнал может быть прочитан этим битом.	CCI
2	OUT	RW	0h	Выход. Этот бит указывает состояние выхода. Если выбран режима вывода 0, этот бит напрямую управляет состоянием выхода. 0 Низкий уровень выхода 1 Высокий уровень выхода	OUT
1	COV	RW	0h	Переполнение захвата. Этот бит указывает, что произошло переполнение захвата. Бит COV должен быть сброшен программно 0 Нет переполнения захвата 1 Произошло переполнение захвата	COV
0	CCIFG	RW	0h	Флаг прерывания захвата/сравнения 0 Прерывание не ожидается 1 Ожидается прерывание	CCIFG

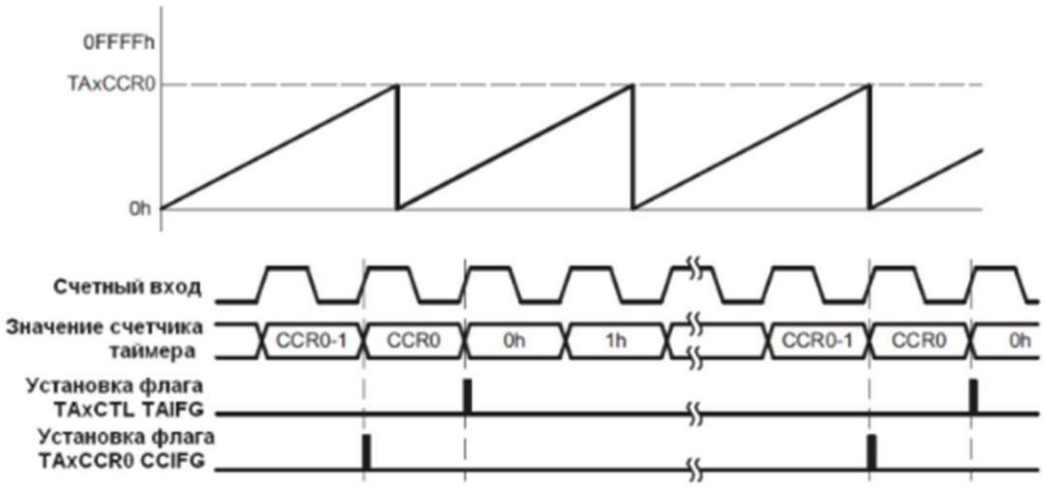

Рис. 2.2 Режим прямого счета

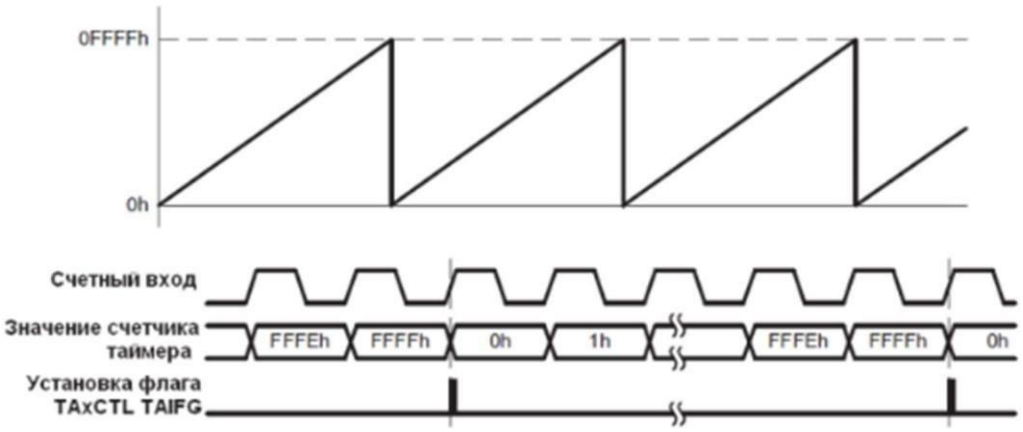

Рис. 2.3 Непрерывный режим

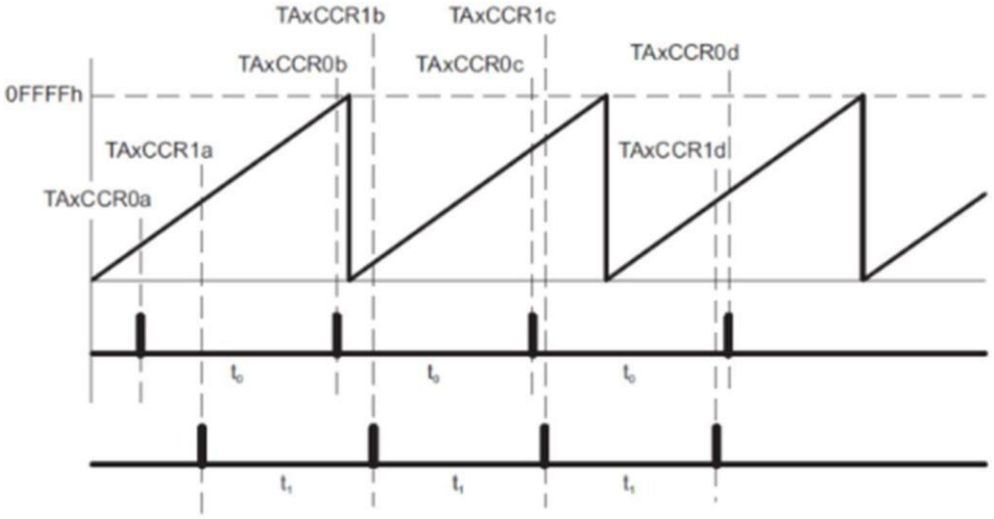

Рис. 2.4 Генерирование нескольких интервалов в непрерывном режиме

В режиме прямого счета (рис. 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 (рис. 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 рис. 2.4.

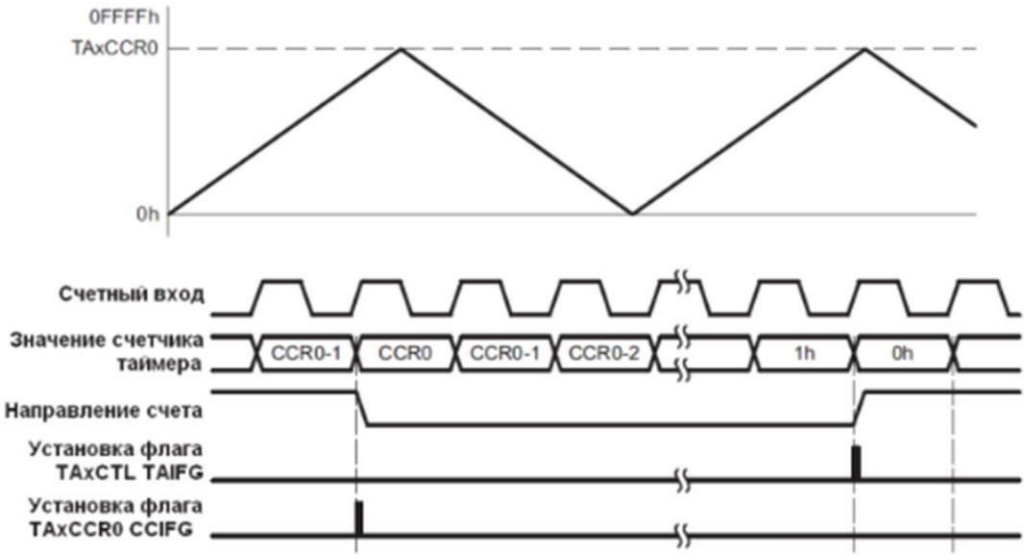

Рис. 2.5 Реверсивный режим

В реверсивном режиме (рис. 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 (рис.2.6). Регистры TAxCCRn не имеют буфера, поэтому они изменяются сразу после записи.

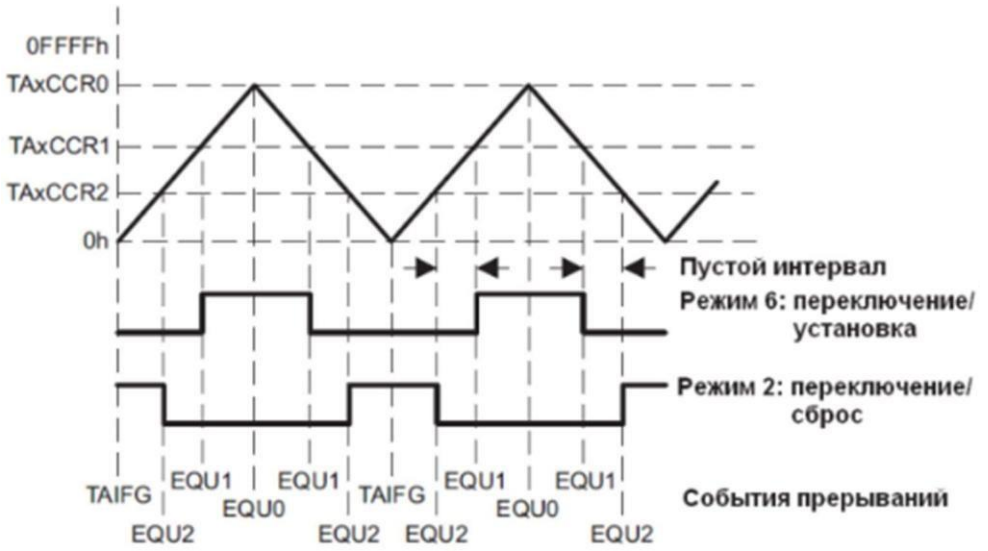

Рис. 2.6 Генерирование пустых интервалов в реверсивном режиме

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:

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

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

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

11 — 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.

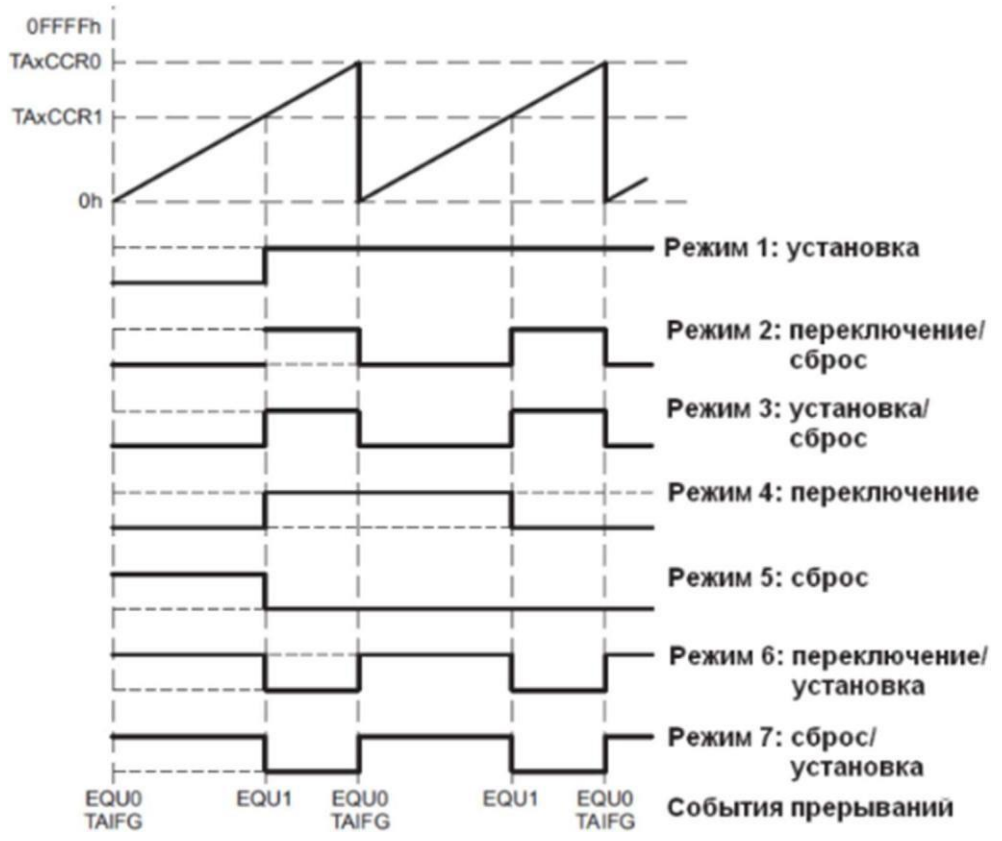

Рис. 2.7 Режимы выхода в режиме прямого счета

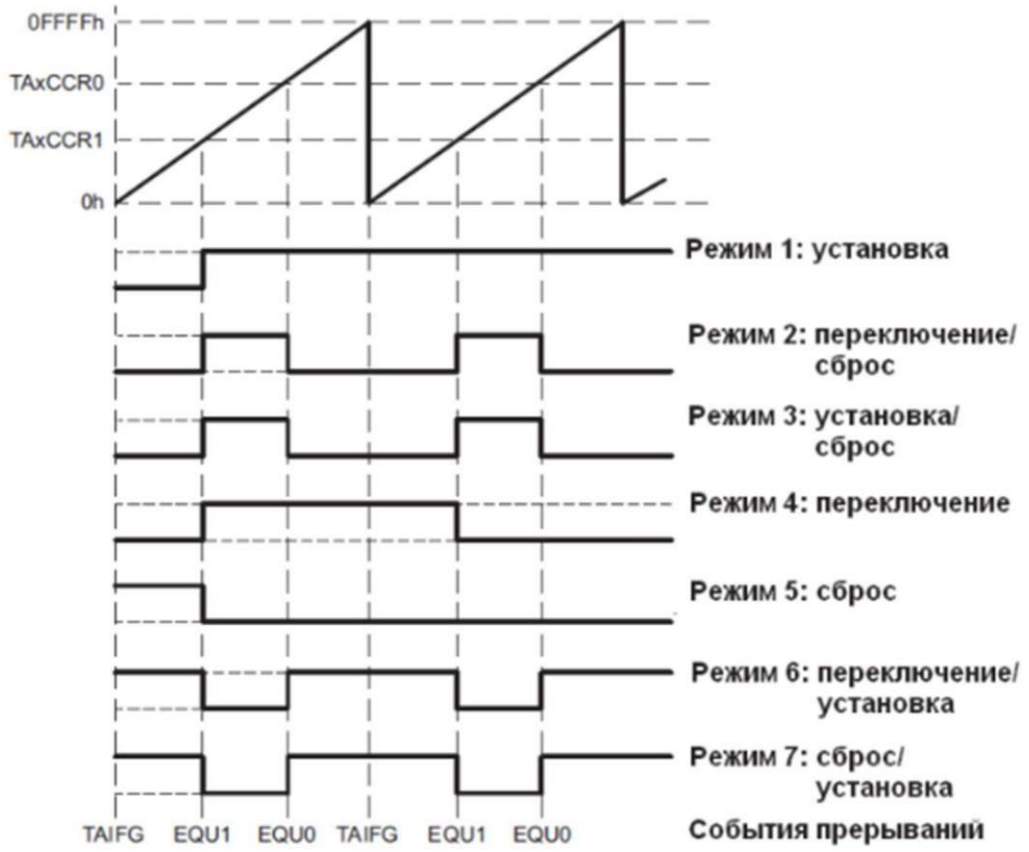

Рис. 2.8 Режимы выхода в непрерывном режиме

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, соответствующая0 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).

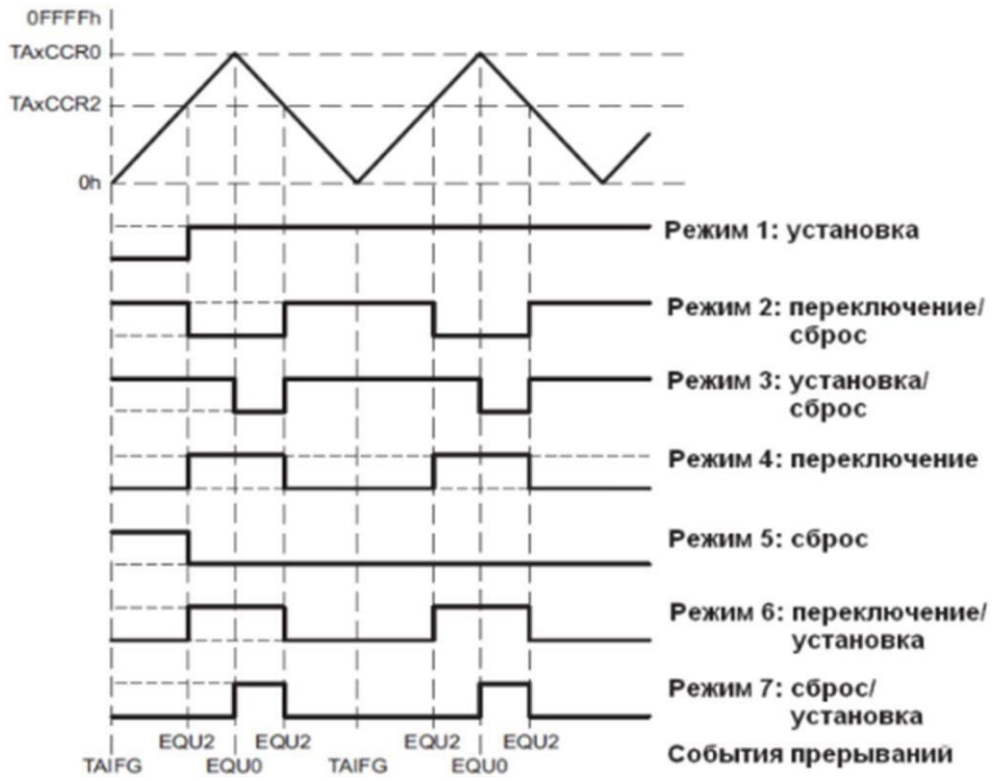

Рис. 2.9 Режимы выхода в реверсивном режиме

4. Выполнение работы

4.1 Код программы

```
#include <msp430.h>
#include <stdio.h>

int butt1_flag = 0;
int butt2_flag = 0;
int LedCount = 0; int
EndPoint = 0; int
main(void)
{
    volatile int i;
      // stop watchdog timer
      WDTCTL = WDTPW | WDTHOLD;
      //setup leds
      P1DIR |= BIT1;//4Led
      P1DIR |= BIT2;//5Led
      P1DIR |= BIT3;//6Led
      P1DIR |= BIT4;//7Led
      P1DIR |= BIT5;//8Led
      P1OUT = 0;
      //set direction for s1, s2

      P1REN |= BIT7;
      P1OUT |= BIT7;

      P2REN |= BIT2;
      P2OUT |= BIT2;

      __bis_SR_register(GIE);
      P1IE |= BIT7;
      P2IE |= BIT2;
      P1IES |= BIT7;
      P2IES |= BIT2;
      P1IFG = 0;
      P2IFG = 0;

      P2DIR = 0;
      P2REN = BIT2;
      P2OUT = BIT2;

      TA1CTL |= (BIT2 | BIT9 | BIT4 | BIT5 | BIT6);
      TA1CCR0 = 65535;
      TA1CCTL0 |= BIT4;
```

```
      __enable_interrupt();
      __bis_SR_register(LPM0_bits + GIE);       // Enter LPM4
w/interrupt
      __no_operation();                         // For debugger
}

#pragma vector = PORT1_VECTOR
__interrupt void buttonPush1(void) {

        P1IES &= ~BIT7;
butt1_flag = 1;     volatile int
i = 0;  for (i = 0; i < 1000;
i++) {}
    P1IFG = 0;
}

#pragma vector = PORT2_VECTOR
__interrupt void buttonPush2(void) {

        P2IES &= ~BIT2;
butt2_flag = 1;     volatile int
i = 0;  for (i = 0; i < 1000;
i++) {}
    P2IFG = 0;
}

#pragma vector=TIMER1_A0_VECTOR
__interrupt void Timer(void)
{
if ((butt1_flag == 1 || LedCount > 0) && EndPoint == 0){
switch (LedCount){          case 0 :
            P1OUT ^= BIT1;
            LedCount ++;
break;         case 1:
            P1OUT ^= BIT2;
            LedCount ++;
break;         case 2:
            P1OUT ^= BIT3;
            LedCount ++;
break;         case 3:
            P1OUT ^= BIT4;
            LedCount ++;
break;         case 4:
            P1OUT ^= BIT5;
            LedCount ++;
break;         case 5:
            LedCount = 0;
            EndPoint = 1;
```

```
            P1IES |= BIT7;
butt1_flag = 0;         break;
        }
    }

if ((butt2_flag == 1 || LedCount > 0) && EndPoint == 1){
switch (LedCount){         case 0 :
            P1OUT ^= BIT1;
            LedCount ++;
break;         case 1:
            P1OUT ^= BIT2;
            LedCount ++;
break;         case 2:
            P1OUT ^= BIT3;
            LedCount ++;
break;         case 3:
            P1OUT ^= BIT4;
            LedCount ++;
break;         case 4:
            P1OUT ^= BIT5;
            LedCount ++;
break;         case 5:
            LedCount = 0;
            EndPoint = 0;
            P2IES |= BIT2;
butt2_flag = 0;         break;
        }
    }
    TA1CTL &= ~BIT0;
    TA1IV = 0;
}
```

5. Вывод

В ходе выполнения лабораторной работы мы ознакомились с работой подсистемы прерываний и таймерами микроконтроллера MSP430F5529. Написали программу используя таймеры и прерывания в соответствии с заданием варианта