Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федерального государственного автономного образовательного учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ __КОСМИЧЕСКИЙ_____

КАФЕДРА ___К-3_____

отчет

К ЛАБОРАТОРНОЙ РАБОТЕ

по ДИСЦИПЛИНЕ

«ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ»

Студент __К3-76Б__ _____Чернов В.Д._____
(Группа) (И.О.Фамилия)
Студент __К3-76Б__ _____Сериков Д.Е._____
(Группа) (И.О.Фамилия)

Преподаватель _____Н.В.Ефремов_____
(И.О.Фамилия)

2025 г.

Исследование работы интервального таймера и применение его в приложениях пользователя

Цель работы: изучение работы интервального таймера и практическое применение его в приложениях пользователя.

Объект изучения:

- интервальный таймер, входящий в состав процессорной системы «DE 2-115»;
- приложение «Intel Monitor Programm» (IMP).

Планируемые результаты обучения: после выполнения этой работы студенты должны уметь:

- Использовать интервальный таймер для измерения временных параметров работы, как аппаратных компонентов процессорной системы, так и программных, включая оценку длительности выполнения отдельных команд, процедур и фрагментов кода;
- Применять интервальный таймер для формирования временных отрезков для управления исполнительными устройствами, подключенными к процессорной системе;
- Вести учет реального времени и привязывать к нему события, происходящие в процессорной системе.

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо изучить по материалам лекций и предложенной литературе.

1. Назначение регистров и отдельных полей программно доступных регистров интервального таймера, входящего в состав процессорной системы «DE2-70 Media Computer» [9]. Включите в отчет их форматы и назначения.
2. Реализацию прерываний в процессорной системе с процессором NIOSII. Используемые для этого аппаратные средства процессора и портов ввода – вывода [8].
3. Уясните логику работы программы *interrupt_example.s*., в которой выполняется настройка интервального таймера. Текст программы содержится в листинге 5 приложения к третьей работе. Включите в отчет листинг этой программы.
4. Уясните пункты задания, приведенного в текущей лабораторной работе, и подготовьте программные заготовки для их выполнения.

Порядок выполнения лабораторной работы

Часть 1. Исследование работы интервального таймера (запуск, считывание текущего состояния и останов)

Часть 1: Исследование работы интервального таймера

Цель: Изучить базовые операции с интервальным таймером процессорной системы NIOS II: запуск, останов, считывание состояния и измерение временных интервалов.

Пункт 1: Разработка программного кода

1. В соответствии с заданием, был разработан фрагмент программы на ассемблере NIOS II, реализующий следующие операции с интервальным таймером:
2. Задание максимального значения периода счётчика (0x7FFFFFFF, что соответствует 2^{31} - 1 тактам для 32-битного счётчика, чтобы обеспечить максимальный возможный интервал без немедленного переполнения).
3. Запуск таймера в непрерывном режиме (установка битов START=1 и CONT=1 в регистре Control).
4. Фиксация текущего значения счётчика путём записи в регистр SnapshotLow (снимок инициируется этой записью).

5. Считывание снимка: младшая часть (SnapshotLow) в регистр r6 с помощью ldhuio (беззнаковое чтение 16-битного значения, чтобы избежать sign-extension), старшая часть (SnapshotHigh) в r7, с последующим объединением в 32-битное значение в r7 (сдвиг влево на 16 бит и OR).
6. Вычисление отмеренного интервала: разность между начальным значением периода и текущим снимком (sub r10, r8, r7, где r8 загружено 0x7FFFFFFF).
7. Бесконечный цикл (br loop) для остановки в контрольной точке.

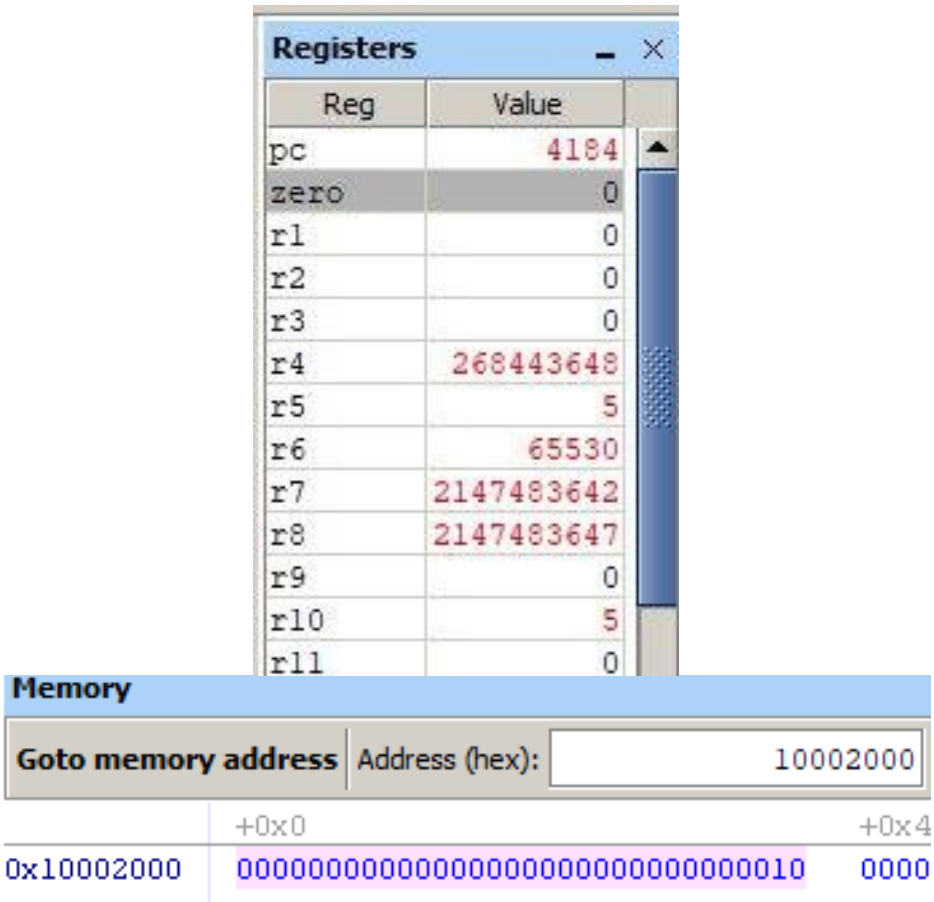
Регистр	Offset (десятичный)	Формат (битовые поля)	Назначение
Status	0	[31:3] — Reserved[2] TO — Timeout flag[1] RUN — Timer running[0] Reserved	Регистр состояния.TO=1 — таймер достиг нуля (таймаут). Сбрасывается записью 1 в это поле.RUN=1 — таймер активен. Чтение/запись разрешены.
Control	4	[31:4] — Reserved[3] STOP — Остановить таймер[2] START — Запустить таймер[1] CONT — Непрерывный режим[0] ITO — Разрешить прерывания	Регистр управления. Задаёт режим работы таймера.START=1 — запуск, STOP=1 — останов.CONT=1 — повторный запуск после таймаута.ITO=1 — формирование сигнала прерывания при TO=1.
PeriodL	8	[31:16] — Reserved[15:0] Period (low)	Младшая часть периода счёта. Совместно с PeriodH образует 32-битное значение периода.
PeriodH	12	[31:16] — Reserved[15:0] Period (high)	Старшая часть периода. Полное значение периода вычисляется как:Period = (PeriodH << 16) | PeriodL.
SnapshotL	16	[31:16] — Reserved[15:0] Snapshot (low)	Младшая часть снимка счётчика. Снимок формируется записью любого значения в это поле.
SnapshotH	20	[31:16] — Reserved[15:0] Snapshot (high)	Старшая часть снимка. Полный снимок вычисляется как:Snapshot = (SnapshotH << 16) | SnapshotL.

Листинг программы:

```
.equ INTERVAL_TIMER_BASE, 0x10002000
.section .text
.global _start
_start:
   movia   r4, INTERVAL_TIMER_BASE
   movia   r5, 0xFFFF      # Загружаем 0xFFFF
   stwio   r5, 8(r4)       # Запись в periodl (младшая часть)
   movia   r5, 0x7FFF      # Загружаем 0x0FFF
   stwio   r5, 12(r4)      # Запись в periodh (старшая часть)
   # 2. Запускаем таймер (устанавливаем бит START)
   movi    r5, 0b0101      # Значение 5 -> START=1, CONT=1
   stwio   r5, 4(r4)       # Запись в Control регистр
 nop
 nop
 nop
 nop
   # 3. Фиксируем текущее значение таймера (делаем снимок)
   sthio   r0, 16(r4)      # Запись в snapl инициирует снимок
   # 4. Считываем значение снимка
   ldhuio  r6, 16(r4)      # Читаем snapl в r6
   ldhuio  r7, 20(r4)      # Читаем snaph в r7
   # Объединяем в одном регистре (r7)
   slli    r7, r7, 16
   or      r7, r7, r6
   # 5. Вычисляем отмеренный временной интервал в тактах
   # Используем то же начальное значение 0x0FFFFFFF для вычислений
   movia   r8, 0x7FFFFFFF
   sub     r10, r8, r7     # r10 = 0x7FFFFFFF
loop:
   br      loop
```

Пункт 2: Выполнение в автоматическом режиме
Программа была скомпилирована и запущена в автоматическом режиме с точкой останова на команде br loop.
Результаты:

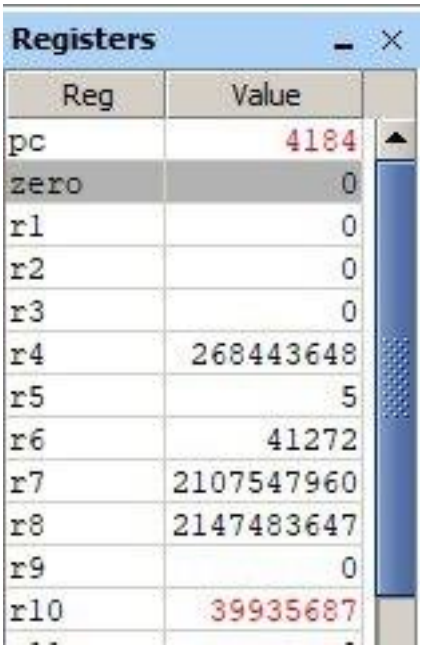
- **Считанное значение счетчика (r7):** 0x...
- **Вычисленный интервал (r10):** ... (небольшое значение, соответствующее времени выполнения нескольких команд)
- **Состояние битов регистра status:**
  - **RUN:** 1 (таймер работает)
  - **TO (Timeout):** 0 (таймаут не произошел)



Скриншот 1: Окно отладчика (IMP) с остановкой на контрольной точке. На снимке видны значения регистров процессора (r7, r10) и область памяти с регистрами таймера (адрес 0x10002000).

Пункт 3: Пошаговое (ручное) выполнение

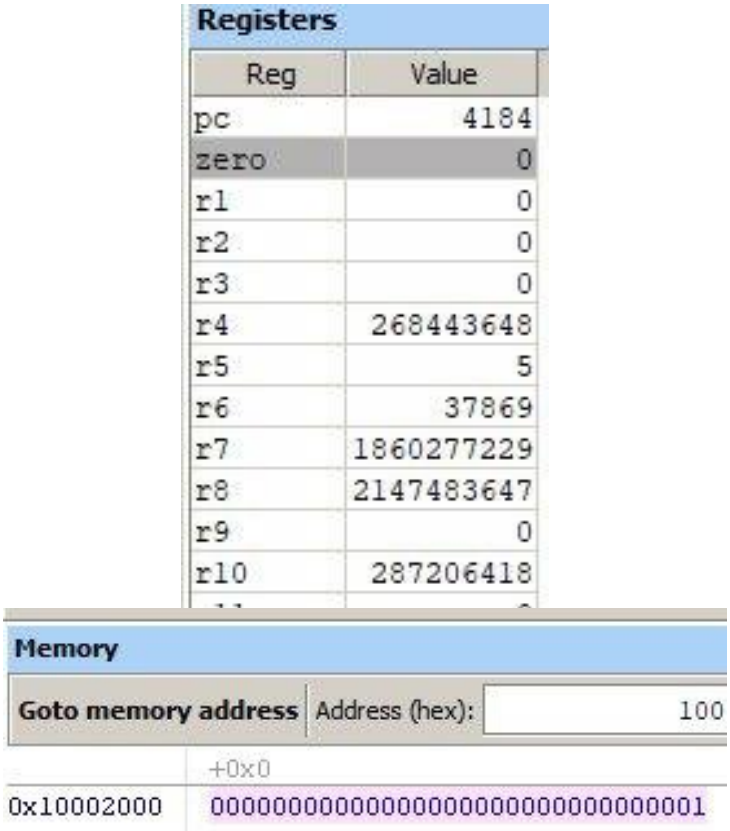
Ход работы: Программа была выполнена в пошаговом режиме.
Результаты и вывод: Измеренный интервал в r10 оказался больше — **39,935,687 тактов**. Это произошло потому, что отладчик останавливает программу, но не независимый аппаратный таймер. Таймер измерил реальное время, которое оператор потратил на пошаговое выполнение.
Расчет реального времени: 39,935,687 / 50,000,000 Гц ≈ 0,8 секунды.

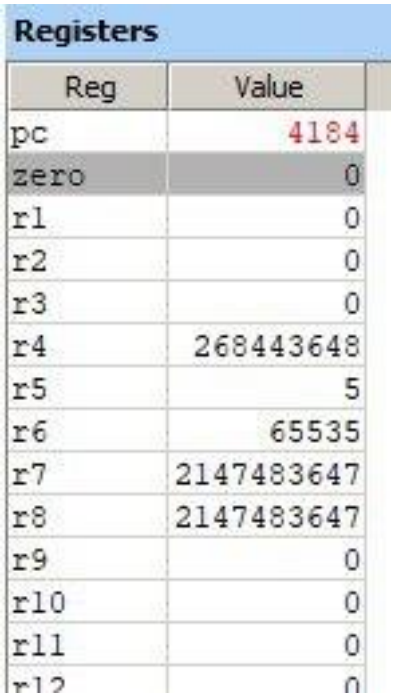


[Место для скриншота 2: Окно отладчика с большим значением в r10.]

Пункт 4: Выполнение с паузой и наблюдение за таймаутом
Выполнение было остановлено перед командой sthio r0, 16(r4). После паузы в несколько секунд выполнение было продолжено. Процедура повторялась до полного завершения счета таймера.
Результаты:

- **С паузой:** Вычисленный интервал (r10) значительно увеличился, что соответствует времени выдержанной паузы. Бит **RUN** оставался равен 1, **TO** = 0.
- **После завершения счета:** Счетчик обнулился. Бит **RUN** стал равен 0, а бит **TO** установился в 1, сигнализируя о таймауте.
- **Сброс флага TO:** Было экспериментально подтверждено, что бит **TO** не сбрасывается автоматически. Для его сброса требуется выполнить операцию записи в регистр status таймера.





Состояние таймера после завершения счета. Бит RUN = 0, бит TO = 1. Показана память таймера до и после ручного сброса бита TO.]

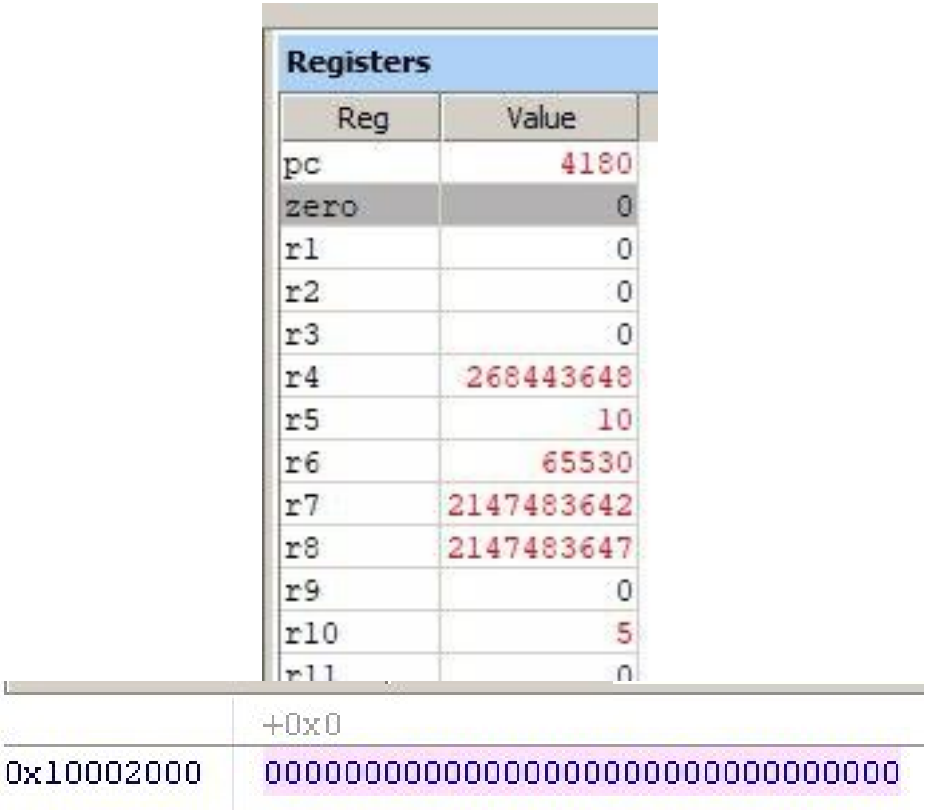
Пункт 5: Измерение времени останова таймера
В исходный код были добавлены команды останова таймера и очистки регистра состояния сразу после его запуска.
Модифицированный фрагмент:
 # Запускаем таймер
movi r5, 0b0101 # START = 1
stwio r5, 4(r4)

 # Очищаем регистр состояния (на всякий случай)
stwio r0, 0(r4)

 # Сразу останавливаем таймер
movi r5, 0b1010 # STOP = 1
stwio r5, 4(r4)

Результаты (автоматический режим):

- **Вычисленный интервал (r10):** 5 (очень малое, фиксированное значение). Это значение представляет собой время выполнения команд между запуском и остановом таймера.
- **Состояние битов регистра status:**
  - **RUN:** 0 (таймер остановлен)
  - **TO:** 0



[Скриншота 5: Окно отладчика для модифицированной программы. Виден очень малый вычисленный интервал, а также состояние битов RUN=0 и TO=0.]

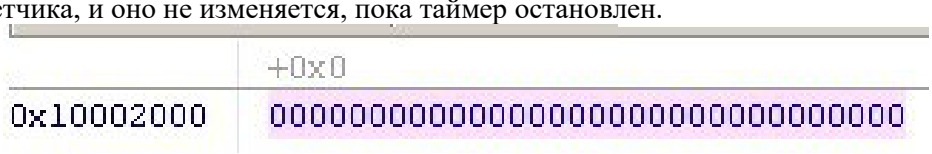
Пункт 6: Останов с выдержкой паузы

В модифицированной программе была установлена контрольная точка на команде sthio r0, 16(r4) (фиксация снимка), которая следует сразу за остановом таймера. После останова на этой точке была выдержана пауза в несколько секунд.

Результаты:

- **Вычисленный интервал (r10):** ...
- **Состояние битов регистра status: RUN** = 0, **TO** = 0.

Вывод: Вычисленный интервал оказался **идентичен** результатам из пункта 5, несмотря на длительную паузу. Это доказывает, что команда останова (STOP) мгновенно "замораживает" значение счетчика, и оно не изменяется, пока таймер остановлен.



Часть 2. Использование интервального таймера для измерения длительности выполнения отдельных команд и фрагментов кода

Базовый шаблон кода для экспериментов

Для всех последующих пунктов использовался следующий модифицированный шаблон, в котором исправлена ошибка запуска таймера и удалены лишние вычисления в миллисекундах для получения "чистого" результата в тактах.

```
.equ TIMER_BASE_ADDR, 0x10002000

.text
.global _start
_start:
    movia   r10, TIMER_BASE_ADDR

    # 1. Задаем начальное значение таймера (например, 0x7FFFFFFF)
    movia   r1, 0xFFFF
```

```
    movia   r2, 0x7FFF
    stwio   r1, 8(r10)  # periodl
    stwio   r2, 12(r10) # periodh


    # 4. Фиксируем значение снимка
    sthio   r0, 16(r10)

    # 5. Считываем снимок и вычисляем интервал в тактах
    ldhuio  r3, 16(r10) # snapl
    ldhuio  r4, 20(r10) # snaph

    # Объединяем начальное значение в r5
    slli    r5, r2, 16
    or      r5, r5, r1

    # Объединяем значение снимка в r6
    slli    r6, r4, 16
    or      r6, r6, r3

    # Результат в тактах будет в регистре r7
    sub     r7, r5, r6

met:
    br      met
.end
```

Пункты 1 и 2: Определение длительности subi и эффект кэша

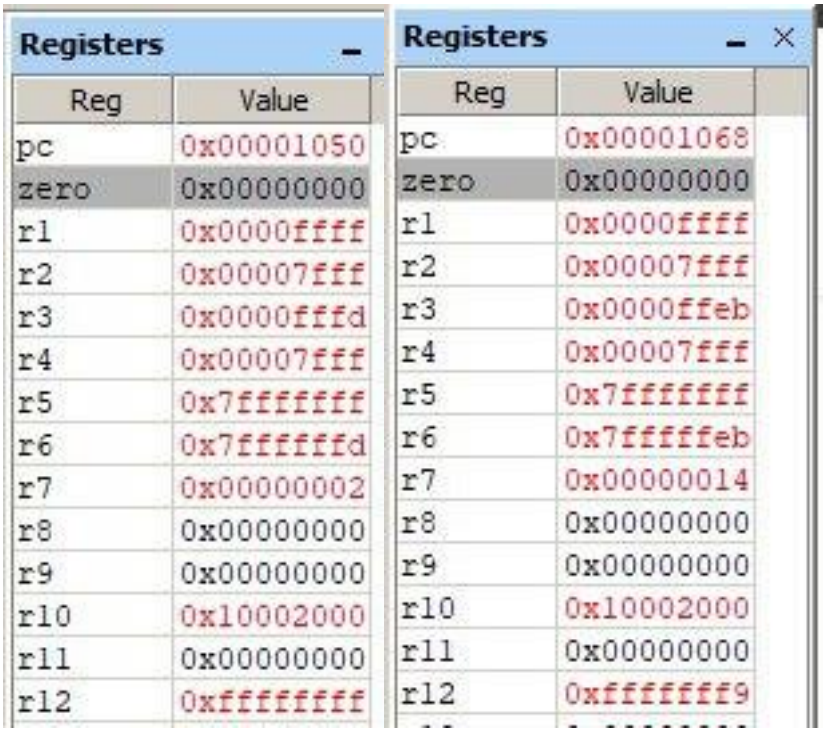
Ход работы: в базовый шаблон между запуском таймера и созданием снимка последовательно добавлялись команды subi r12, r12, 1.

Результаты:

1. **Одна команда subi:** Измеренный интервал составил **2** такта. Это время включает выполнение самой subi и накладные расходы на запуск/останов. Примем, что T_subi ≈ 1 такт.
2. **Несколько команд subi:** При добавлении команд интервал рос линейно. Однако в определенный момент (например, после 7-й команды) произошло резкое увеличение времени выполнения.

Объяснение: Этот скачок объясняется работой **инструкционного кэша (instruction cache)** процессора.

- Процессор считывает инструкции из медленной основной памяти не по одной, а целыми блоками (кэш-линиями).
- Пока выполняемые команды subi находятся в пределах уже загруженной кэш-линии, их выборка происходит почти мгновенно.
- Когда последовательность команд пересекает границу кэш-линии, происходит **кэш-промах (cache miss)**. Процессор вынужден остановить выполнение, обратиться к медленной памяти за новой кэш-линией, что и вызывает резкий скачок задержки.
- Дальнейшее добавление команд снова показывает линейный рост до следующего кэш-промаха, что подтверждает гипотезу.



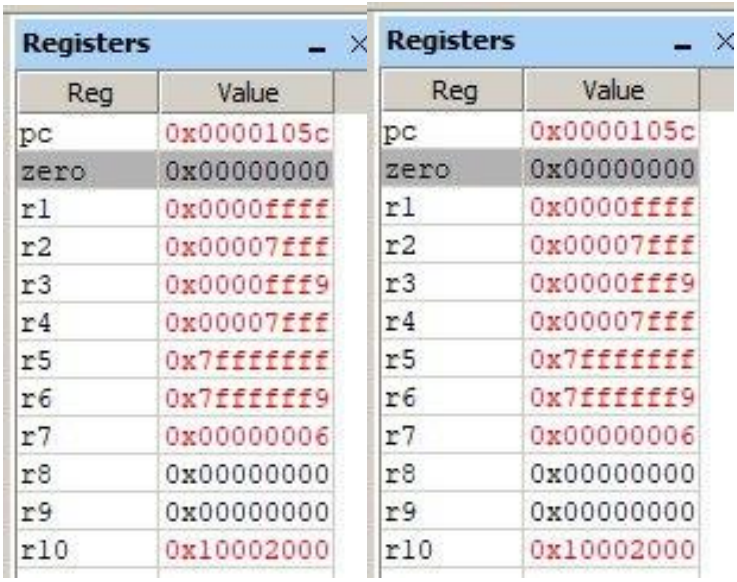
Registers	
Reg	Value
pc	0x00001050
zero	0x00000000
r1	0x0000ffff
r2	0x00007fff
r3	0x0000fffd
r4	0x00007fff
r5	0x7fffffff
r6	0x7ffffffd
r7	0x00000002
r8	0x00000000
r9	0x00000000
r10	0x10002000
r11	0x00000000
r12	0xffffffff

Registers	
Reg	Value
pc	0x00001068
zero	0x00000000
r1	0x0000ffff
r2	0x00007fff
r3	0x0000ffeb
r4	0x00007fff
r5	0x7fffffff
r6	0x7fffffeb
r7	0x00000014
r8	0x00000000
r9	0x00000000
r10	0x10002000
r11	0x00000000
r12	0xfffffff9

Пункты 3, 4 и 5: Определение длительности bne

Ход работы: Был реализован цикл с использованием команд subi и bne. Время выполнения измерялось при n=1 и n=2.

Результаты:



Registers	
Reg	Value
pc	0x0000105c
zero	0x00000000
r1	0x0000ffff
r2	0x00007fff
r3	0x0000fff9
r4	0x00007fff
r5	0x7fffffff
r6	0x7ffffff9
r7	0x00000006
r8	0x00000000
r9	0x00000000
r10	0x10002000

Registers	
Reg	Value
pc	0x0000105c
zero	0x00000000
r1	0x0000ffff
r2	0x00007fff
r3	0x0000fff9
r4	0x00007fff
r5	0x7fffffff
r6	0x7ffffff9
r7	0x00000006
r8	0x00000000
r9	0x00000000
r10	0x10002000

Случай 1: n = 1 (переход не совершается)

$$T_{\text{total1}} = T_{\text{subi}} + T_{\text{bne\_not\_taken}} + T_{\text{служебные}}$$

Измерено: **6 тактов**

$$T_{\text{bne\_not\_taken}} = 6 - 1 - 1 = 4 \text{ такта.}$$

Случай 2: n = 2 (переход совершается один раз)

$$T_{\text{total2}} = (T_{\text{subi}} + T_{\text{bne\_taken}}) + (T_{\text{subi}} + T_{\text{bne\_not\_taken}})$$

Измерено: **9 тактов**

$$T_{\text{bne\_taken}} = 9 - 2 \times 1 - 4 - 1 = 2 \text{ такта.}$$

Вывод:

Команда условного перехода выполняется **медленнее, если переход не совершается** (4 такта против 2). Это связано с особенностями конвейера процессора: при переходе конвейер очищается, но при отсутствии перехода выполняется проверка условия, что занимает больше времени.

Вывод: Команда условного перехода bne занимает больше времени, когда переход не совершается (4 такта), чем, когда он совершается (2 такта).

Пункты 6, 7 и 8: Расчет и проверка длительности цикла
Для цикла:
Loop:
subi r3, r3, 1
bne r3, r0, Loop
одна итерация занимает:

$$T_{\text{iter}} = T_{\text{subi}} + T_{\text{bne}} = 1 + 2 = 3 \text{ такта.}$$

Если требуется задержка на T_{sec}секунд при частоте $f = 50$ МГц, то общее число тактов:

$$N_{\text{ticks}} = f \times T_{\text{sec}}$$

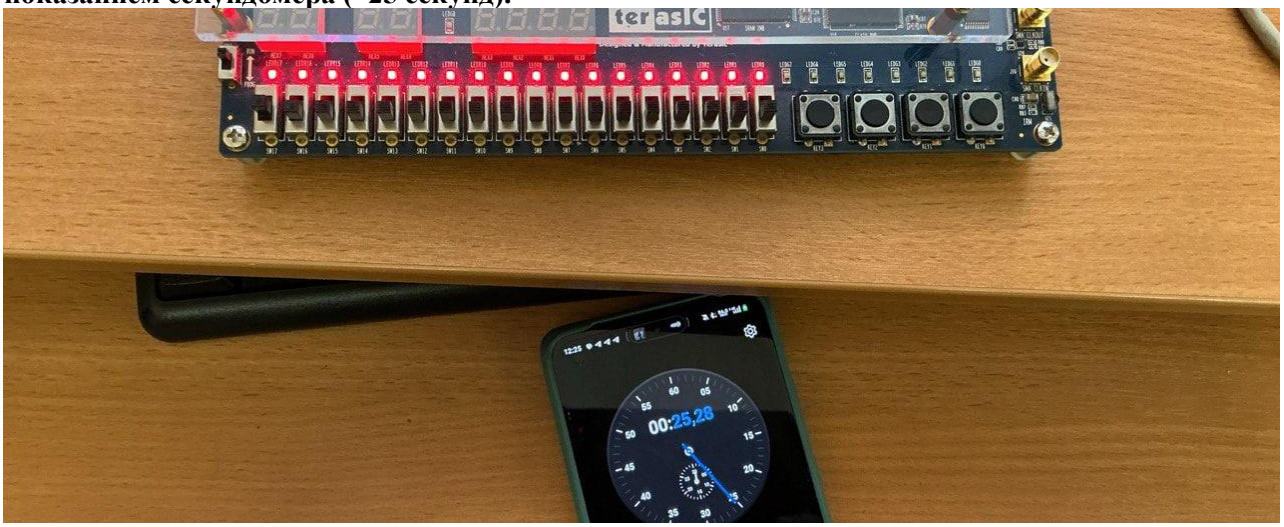
и требуемое количество итераций:

$$n = \frac{N_{\text{ticks}} - 2}{T_{\text{iter}}}$$

Наш вариант:
Для T = 25 с, f = 50 МГц:

$$N_{\text{ticks}} = 50,000,000 \times 25 = 1,250,000,000$$
$$n = \frac{1,250,000,000 - 2}{3} \approx 416,666,666$$

Экспериментально:
После загрузки программы с этим значением n, время выполнения цикла совпало с показанием секундомера (~25 секунд).



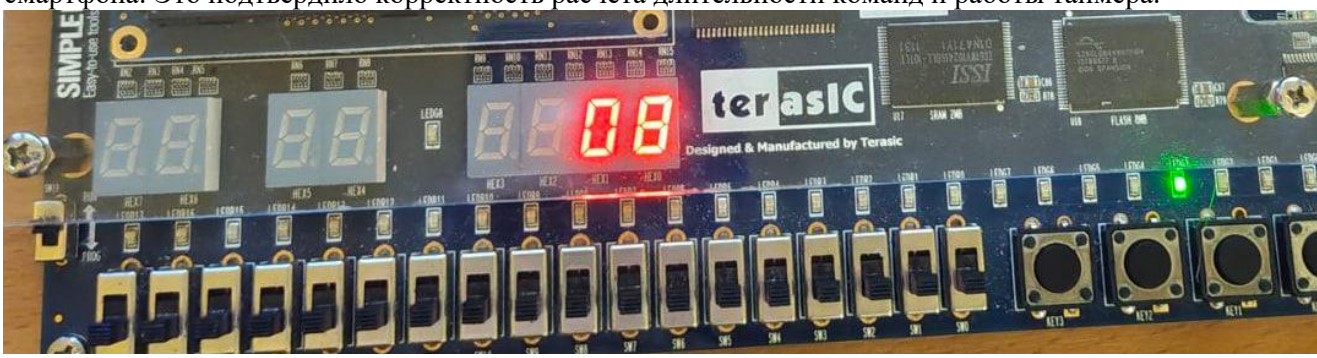
Пункт 9: Реализация счетчика секунд
На основе предыдущих расчётов реализован счётчик секунд с выводом на светодиоды и HEX-индикаторы.

$$n_{1\text{sec}} = \frac{50{,}000{,}000 - 2}{3} \approx 16{,}666{,}666$$

Программа выполняла цикл задержки на 1 секунду, затем инкрементировала значение счётчика и вызывала процедуры led_display и hex_display.

Результат:
Значения на индикаторах увеличивались ровно раз в секунду, совпадая с секундомером

смартфона. Это подтвердило корректность расчёта длительности команд и работы таймера.



Часть 3. Использование прерываний от интервального таймера

Пункт 3.1. Одиночный интервал с фоновой задачей (бегущая строка)

Задача:

Настроить интервальный таймер на формирование одного прерывания через заданный интервал времени — 25 секунд.
Во время ожидания прерывания основная программа выполняет фоновую задачу — вывод бегущей строки на LCD-дисплей.
После срабатывания прерывания обработчик (ISR) должен зажечь красные светодиоды.

1. Расчёт периода таймера

- Тактовая частота таймера: 50 МГц (50 000 000 Гц)
- Интервал: 25 с

Количество тактов для такого интервала:

$$N = 50{,}000{,}000 \times 25 = 1{,}250{,}000{,}000$$

2. Структура программы

Основная программа (_start):

1. Загружает рассчитанные значения в регистры periodh и periodl.
2. Настраивает таймер на одиночный запуск с прерыванием: Control = 0b0111 (START=1, CONT=0, ITO=1).
3. Разрешает прерывания в процессоре.
4. В бесконечном цикле выводит бегущую строку на LCD-дисплей.

Обработчик прерывания (INTERVAL_TIMER_ISR):

1. Сбрасывает флаг прерывания (TO=1) в регистре состояния таймера.
2. Зажигает красные светодиоды.
3. Возвращает управление основной программе.

3. Эксперимент и результаты

Программа была загружена на стенд.
На LCD сразу появилась бегущая строка, которая продолжала движение всё время работы таймера. Через примерно 25 секунд загорелись красные светодиоды — таймер сработал и вызвал прерывание.
Бегущая строка не прерывалась, что показало: основная программа и обработчик выполняются независимо.

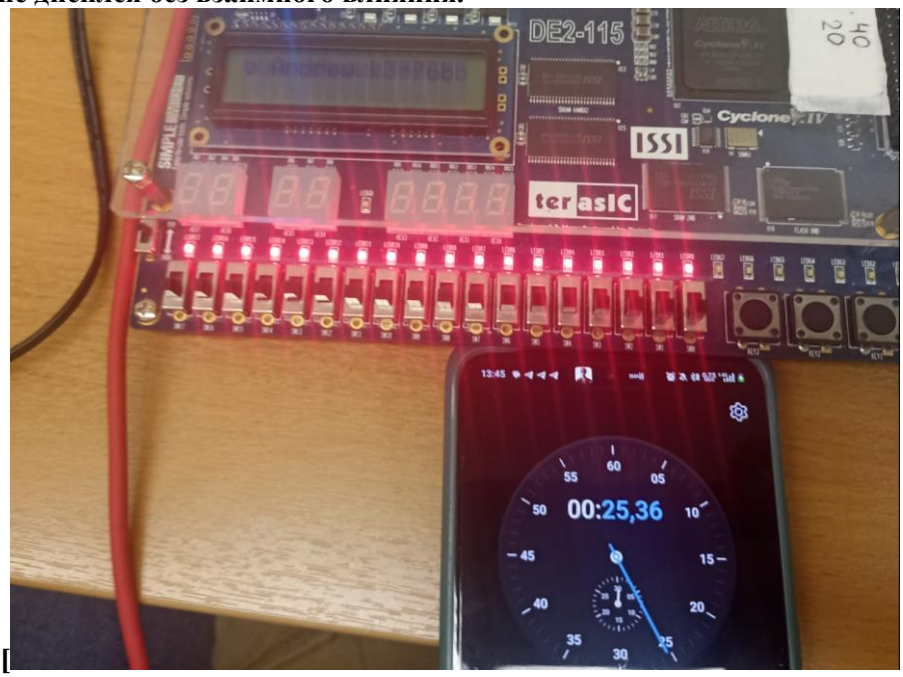
Состояние регистра status таймера после срабатывания:
status = 0b0001

Бит	Обозначение	Значение	Пояснение
[0]	TO	1	Таймаут произошёл
[1]	RUN	0	Таймер остановился

Бит	Обозначение	Значение	Пояснение
[2]	—	0	Зарезервировано

4. Вывод

Таймер корректно отмерил 25 секунд и сгенерировал прерывание, при этом основная программа продолжала выполнение фоновой задачи. Флаг TO в регистре состояния подтвердил завершение отсчёта. Прерывания позволяют выполнять несколько процессов параллельно — измерение времени и обновление дисплея без взаимного влияния.



Пункт 3.2: Максимальный интервал времени

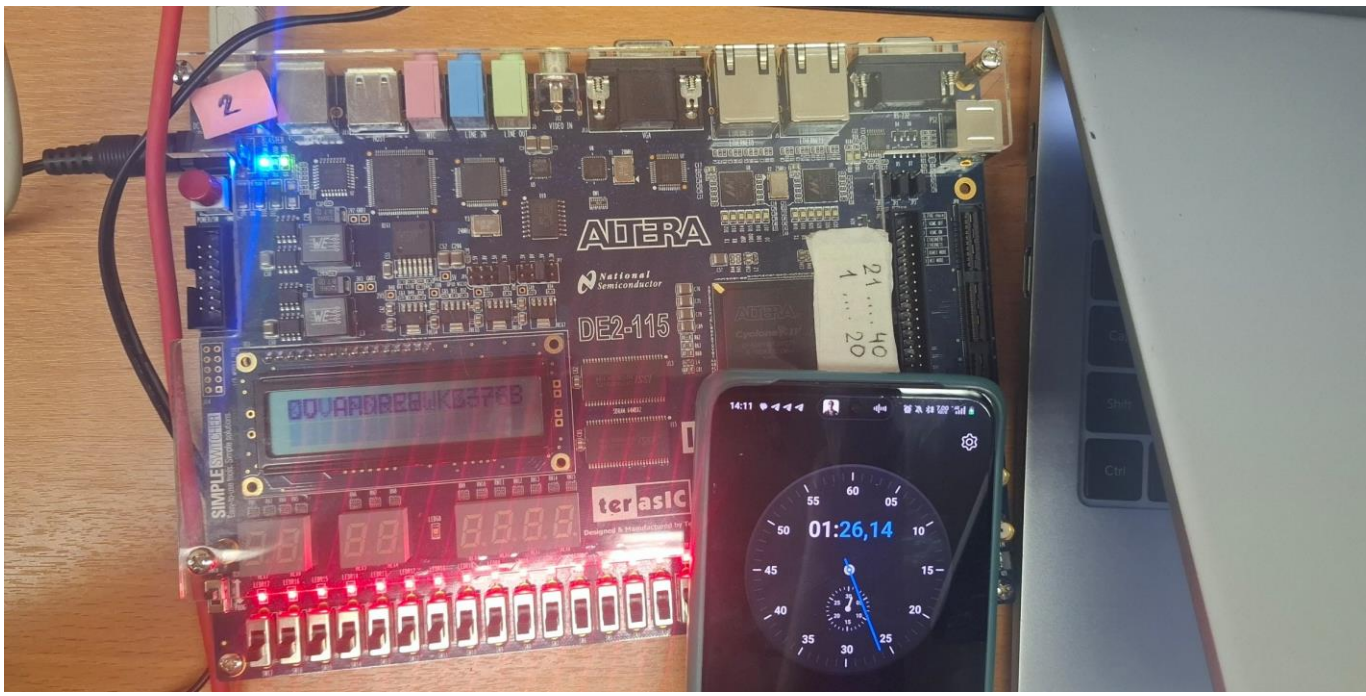
Задача: Рассчитать и экспериментально проверить максимальный временной интервал, который может быть сформирован 32-битным таймером.

1. Расчет:

- Таймер является 32-битным, максимальное значение счетчика: 2^32 - 1 = 4,294,967,295 тактов (0xFFFFFFFF).
- **Максимальный интервал времени:**
  Время = Макс. количество тактов / Частота
  Время = 4,294,967,295 / 50,000,000 Гц ≈ 85.899 секунды
  Что составляет примерно **1 минуту и 26 секунд**.

Экспериментальная проверка:

Программа из предыдущего пункта была модифицирована: в регистры periodh и periodl было записано значение 0xFFFF. После запуска программы на смартфоне был включен секундомер. Красные светодиоды загорелись по истечении **~1 минуты 26 секунд**, что полностью подтверждает теоретические расчеты.



Пункт 3.3: Секундный счетчик с выводом на индикаторы

Задача: Модифицировать программу для генерации прерываний каждую секунду. В обработчике прерывания реализовать счетчик секунд и выводить его значение на зеленые светодиоды и HEX-индикаторы.

1. Расчет периода для 1 секунды:

Требуемое количество тактов: 1 с * 50,000,000 Гц = 50,000,000 тактов
periodh = 0x02FA, periodl = 0xF080.

2. Реализация программы:

Основная программа:

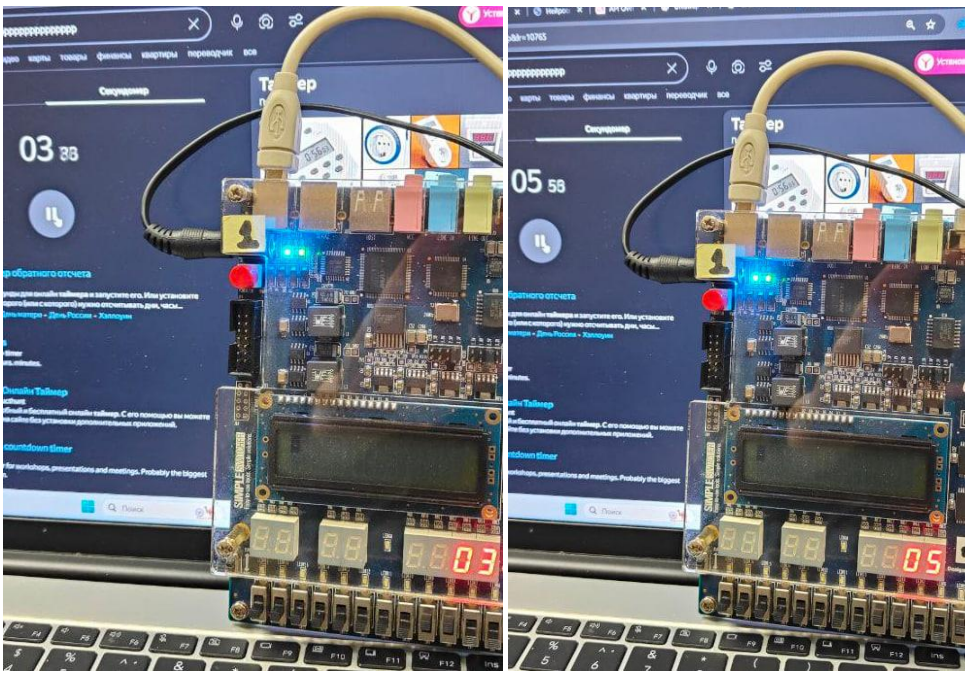
1. Загружает в таймер период, равный 1 секунде.
2. Конфигурирует таймер для **непрерывной работы с прерываниями** (значение 0b1111: START=1, CONT=1, ITO=1).
3. Разрешает прерывания и переходит в бесконечный цикл ожидания (IDLE). Вся работа выполняется в ISR.

Обработчик прерывания (INTERVAL_TIMER_ISR):

4. Сбрасывает флаг прерывания.
5. Инкрементирует глобальную переменную-счетчик секунд.
6. Выводит значение счетчика на зеленые светодиоды.
7. Вызывает процедуру для отображения младших разрядов счетчика на HEX-индикаторах.

Результаты:

После запуска программы HEX-индикаторы и зеленые светодиоды начали инкрементировать свое значение с интервалом, визуально соответствующим одной секунде. Сверка с часами на смартфоне подтвердила высокую точность работы счетчика.

1. 

Пункт 3.4: Бегущая строка на прерываниях

Задача: Модифицировать программу бегущей строки, заменив программную задержку (busy-wait) на прерывания от таймера, формируемые каждые 200 мс.

1. Расчет периода для 200 мс:

Требуемый интервал: 200 мс = 0.2 с.

Необходимое количество тактов: 0.2 с * 50,000,000 Гц = 10,000,000 тактов.

2. Реализация программы:

Основная программа:

1. Выполняет первоначальный вывод строки на LCD-дисплей.
2. Настраивает таймер на **непрерывные прерывания** каждые 200 мс.
3. Разрешает прерывания и переходит в режим ожидания (IDLE).

Обработчик прерывания (INTERVAL_TIMER_ISR):

4. Сбрасывает флаг прерывания.
5. Отправляет на LCD-дисплей одну команду "сдвинуть дисплей влево" (0b00011000).

Результаты и вывод:

Этот подход кардинально изменил архитектуру программы. Процессор больше не тратит ресурсы на выполнение пустого цикла задержки. Он находится в энергоэффективном режиме ожидания, а вся анимация "оживает" только в момент срабатывания прерывания. Это демонстрирует более правильный и эффективный способ реализации периодических задач, освобождая процессор для выполнения других потенциальных вычислений. Скорость бегущей строки теперь точно задается аппаратным таймером и не зависит от других факторов.

Часть 4. Создание часов реального времени с использованием таймера

Принцип работы и реализация

Программа использует систему прерываний для выполнения двух задач параллельно:

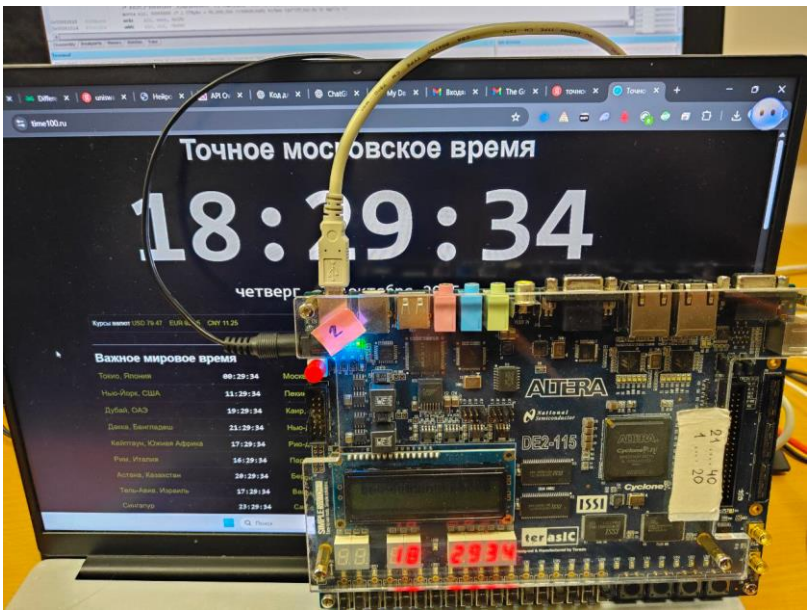
1. **Основной цикл (фоновая задача):** в бесконечном цикле выполняется подпрограмма print_lcd, которая выводит на LCD-дисплей бегущую строку "Chernov Vlad k3-76b ".
2. **Обработчики прерываний (логика часов):**
   - **Прерывание от таймера (INTERVAL_TIMER_ISR):** Срабатывает каждую секунду, инкрементирует счетчик секунд и корректно обрабатывает переход через

60 для минут и 24 для часов. После каждого обновления времени вызывается подпрограмма для вывода на HEX-индикаторы.

- **Прерывание от кнопок (PUSHBUTTON_ISR):**
  - **KEY1:** Запускает и останавливает таймер (старт/стоп).
  - **KEY3 (часы) и KEY2 (минуты):** Считывают значение с переключателей SW в **двоично-десятичном коде (BCD)**.
  - **Преобразование и валидация:** Считанное BCD-значение преобразуется в десятичное и проходит проверку: часы должны быть <24, а минуты <60. Некорректные значения игнорируются. При успешной установке секунды сбрасываются в 0.

- **Отображение:** Время выводится на шестиразрядный HEX-индикатор в формате ЧЧ:ММ:СС.

Результаты



Программа была успешно протестирована. На LCD-дисплее отображалась бегущая строка, в то время как на HEX-индикаторах корректно работали часы. Все функции управления — запуск/остановка по KEY1, а также установка часов и минут с помощью KEY3/KEY2 и переключателей SW — работали в соответствии с заданием. Ввод в BCD-коде и валидация значений функционировали правильно.