Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторному практикуму

Дисциплина: Аппаратные платформы встраиваемых систем

Выполнили студенты гр.13541/1:

(подпись) Никитин А.Е.

(подпись) Баринов М.С.

Руководитель:

(подпись) Васильев А.Е.

Санкт-Петербург
2019 г.

Содержание

1 Лабораторная работа №1 «IAR, CMSIS, SPL, GPIO» 2
1.1 Цель работы . . . 2
1.2 Программа работы . . . 2
1.3 Алгоритм переключения светодиодов . . . 2
1.4 Ход работы . . . 3
1.5 Выводы . . . 4

2 Лабораторная работа №2 «Системы тайминга и прерываний» 5
2.1 Цель работы . . . 5
2.2 Программа работы . . . 5
2.3 Ход работы . . . 5
2.4 Выводы . . . 11

Глава 1

Лабораторная работа №1 «IAR, CMSIS, SPL, GPIO»

1.1 Цель работы

Ознакомиться с интегрированной средой разработки IAR Embedded Workbench for ARM, а также функциями CMSIS и MDRSPL, получить навыки создания и отладки программного обеспечения для целевой платформы на примере разработки программ, взаимодействующих с портами ввода-вывода.

1.2 Программа работы

1. Создать проект-заготовку для последующих лабораторных работ. Листинг демонстрационной программы приведен ниже.
2. Подключить к проекту библиотеку CMSIS. Объяснить назначение и содержание файлов библиотеки. Объяснить назначение и содержание файла startup_MDR32F9Qx.c
3. Подключить к проекту библиотеку MDR32F9Qx Standart Peripherals Library.
4. Настроить параметры отладчика для запуска демонстрационного примера на отладочной плате. Собрать проект, продемонстрировать его исполнение «по шагам».
5. Разработать программу, включающую светодиоды на плате при нажатии кнопок; алгоритм согласовать с руководителем.

1.3 Алгоритм переключения светодиодов

По нажатию кнопки SELECT, двухразрядное двоичное число, отображаемое светодиодами должно инкременторироваться.

Конечный автомат состояний программы представлен на рисунке 1.1, где st0 – состояние ожидания прерывания, а при переходе в состояние st1 вызывается подпрограмма обработки этого прерывания.

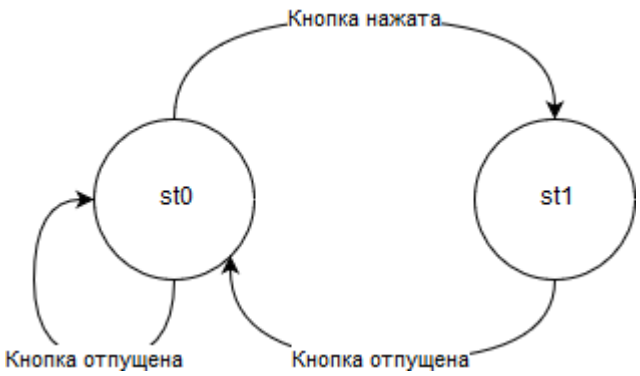

Рис. 1.1: Схема конечного автомата программы

1.4 Ход работы

После настройки среды разработки IAR Embedded Workbench for ARM для работы с микросхемой Milandr, подключения необходимых библиотек и запуска демонстрационного проекта, код программы был запущен и протестирован на работоспособность. Затем были внесены изменения в соответствии с заданием преподавателя. Для этого был разработан конечный автомат, схема которого приведена выше. Фрагмент кода программы, разработанной в соответствии с индивидуальным заданием руководителя приведен в листинге 1.1.

Листинг 1.1: Код демонстрационного примера

```
27 int main()
28 {
29     PeriphInit();
30     while(1)
31     {
32         if(Poll()) //если состояние кнопки отжата => нажата
33         {
34             SwitchState(); //изменение состояние системы
35             LightUpLEDs(); //реакция на изменение состояния
36         }
37     }
38 }
39
40 static void LightUpLEDs()
41 {
42     switch (state)
43     {
44         case 0 :
45         {
46             //0 0
47             PORT_ResetBits( MDR_PORTC, PORT_Pin_0 );
48             PORT_ResetBits( MDR_PORTC, PORT_Pin_1 );
49             break;
50         }
51         case 1 :
52         {
53             //0 1
54             PORT_ResetBits( MDR_PORTC, PORT_Pin_0 );
55             PORT_SetBits( MDR_PORTC, PORT_Pin_1 );
56             break;
57         }
58         case 2 :
59         {
60             //1 0
61             PORT_SetBits( MDR_PORTC, PORT_Pin_0 );
62             PORT_ResetBits( MDR_PORTC, PORT_Pin_1 );
63             break;
64         }
65         case 3 :
66         {
67             //1 1
68             PORT_SetBits( MDR_PORTC, PORT_Pin_0 );
69             PORT_SetBits( MDR_PORTC, PORT_Pin_1 );
70             break;
71         }
72     }
73 }
74
75 static void SwitchState()
76 {
77    if (state == 3) state = 0;
78    else state++;
79 }
80
81 static int Poll() //опрос кнопки
82 {
```

```
83      btn_state = !PORT_ReadInputDataBit(MDR_PORTC,PORT_Pin_2);
84      if(btn_state != past_state)
85      {
86          past_state = btn_state;
87          if(btn_state)
88              return 1;
89      }
90      return 0;
91  }
```

1.5 Выводы

Была реализована система управления миганием светодиодов. Отличительной чертой данной реализации является конечный автомат который обрабатывает нажатие кнопки, и при помощи программно реализованного триггера, переключающего состояния системы, сравнивет текущее её состояние с сохраненным предыдущим.

Улучшение данной системы возможно путем использования обработчика прерываний. Это позволит оптимизировать работу системы, ввиду отсутствия лишней проверки на нажатие кнопки во время её работы.

Глава 2

Лабораторная работа №2 «Системы тайминга и прерываний»

2.1 Цель работы

Развитие навыков разработки встраиваемых приложений реального времени.

2.2 Программа работы

1. Изучит листинг программы, приведенной ниже. Собрать на его основе проект и проанализировать его работу.
2. Настроить таймеры общего назначения 1 и 2 и обработчики запросов прерываний от них следующим образом:
   (a) период счета таймера 2 много больше, чем период счета таймера 1;
   (b) время обслуживания запроса прерывания от таймера 2 много больше, чем от таймера 1 (в обработчике запросов прерывания от таймера 2 организовать длительный 'пустой' цикл или иные продолжительные вычисления);
   (c) в обработчике запроса прерывания от таймера 1 выполнять инверсию бита заданного порта; в обработчике запроса прерывания от таймера 2 при входе в обработчик выполнять установку другого бита порта, при выходе - его сброс;
   (d) приоритеты прерываний установить равными.
3. Зафиксировать характерные осциллограммы и объяснить поведение системы.
4. Добавить к проекту возможность смены приоритетов прерываний таймеров по сигналу внешнего прерывания. Зафискировать осциллограммы и объяснить поведение системы.
5. Разработать систему измерения частоты следования импульсов внешнего сигнала (в качестве источника использовать внешний генератор импульсов или ФИД).
6. Разработать простейший осциллограф: в заданном темпе регистрировать значения входного аналогового сигнала и отображать его на ЖКИ.
7. Разработать простейший генератор аналогового периодического сигнала.

2.3 Ход работы

В файле MDR32F9Qx_it.c содержатся процедуры обработки прерываний. Особенность написания кода в данной лабораторной работе заключается в том, что основной цикл программы не содержит практически ни одного вызова: при запуске инициализируются таймеры-счетчики, а вся логика описана в процедурах обработки прерываний.

В листинге 2.1 приведена реализация настройки обработки прерываний таймеров-счетчиков 1, 2. Данный код относится к процедуре InitTimers() в файле main.c.

Непосредственно реализация обработки описана в файле MDR32F9Qx_it.c и приведена в листинге 2.2:

Листинг 2.1: Код настройки обработки прерываний

```
// Установка приоритетов для T1 и T2
  NVIC_SetPriorityGrouping(3);
  NVIC_SetPriority(Timer1_IRQn, 0);
  NVIC_SetPriority(Timer2_IRQn, 0);
  // Разрешение прерываний для T1 и T2
  NVIC_EnableIRQ(Timer1_IRQn);
  NVIC_EnableIRQ(Timer2_IRQn);

  // Включение таймера 1 и 2
  TIMER_Cmd(MDR_TIMER1, ENABLE);

  // Установка периода для T1
  TIMER_CntInitStruct.TIMER_Period = 31250;
  // Установка периода для T2
  TIMER_CntInitStruct.TIMER_Period = 62500;
```

Листинг 2.2: Алгоритм обработки прерываний

```
/*****************************************************************************
* Function Name  : Timer1_IRQHandler
* Description    : This function handles Timer1 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************/
void Timer1_IRQHandler(void)
{
  TIMER_ClearFlag(MDR_TIMER1, TIMER_STATUS_CNT_ARR);

  uint32_t rxtx = PORT_ReadInputData(MDR_PORTC);
  uint32_t res = rxtx ^ PORT_Pin_0;
  PORT_Write( MDR_PORTC, res );
}
/*****************************************************************************
* Function Name  : Timer2_IRQHandler
* Description    : This function handles Timer2 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************/
void Timer2_IRQHandler(void)
{
  PORT_SetBits( MDR_PORTC, PORT_Pin_1 );
  Delay(2000000);
  TIMER_ClearFlag(MDR_TIMER2, TIMER_STATUS_CNT_ARR);
  PORT_ResetBits( MDR_PORTC, PORT_Pin_1 );
}
```

При данной конфигурации проект был собран и запущен на плате. Зафиксированная осцилограмма продемонстрирована на рисунке 2.1.

На приведенном рисунке 2.2 мы видим, что время обслуживания запроса прерывания от таймера 2 много больше, чем от таймера 1, но поскольку приоритет прерывания первого таймера не выше второго, то во время обработки прерывания таймера-счетчика 2, прерывание таймера-счетчика 1, хотя и возникает, но не обслуживается. С целью рассмотрения поведения системы при иных значениях приоритетности добавим в основной цикл файла main.c возможность смены приоритетов во время работы программы. При подобном подходе, в случае установки более высокого приоритета для обработки прерывания таймера-счетчика 1, последнее будет обработано даже в том случае, если в данный момент не закончилась более длительная по времени обработка прерывания таймера-счетчика 2.

С целью изменения приоритетов в ходе выполнения программы на плате, был написан следующий код,

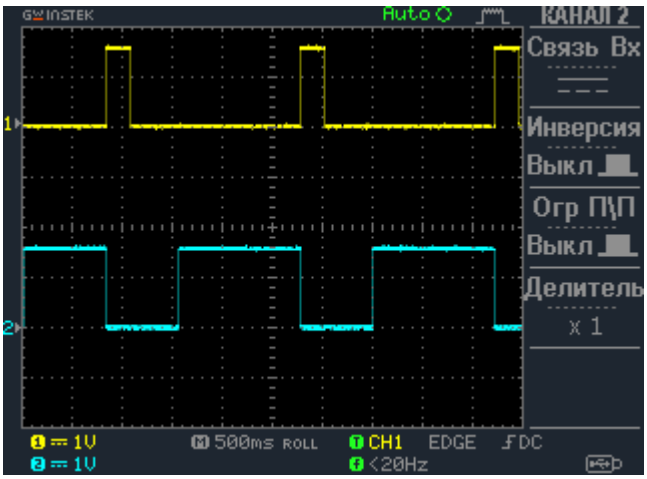

Рис. 2.1: Осцилограмма обработки прерываний при равных приоритетах обработки прерываний

Листинг 2.3: Код настройки обработки прерываний

```
32 while(1)
33 {
34     if (PORT_ReadInputDataBit(MDR_PORTE, PORT_Pin_1))
35     {
36         NVIC_SetPriority(Timer1_IRQn, 1);
37         NVIC_SetPriority(Timer2_IRQn, 0);
38     }
39     else
40     {
41         NVIC_SetPriority(Timer1_IRQn, 0);
42         NVIC_SetPriority(Timer2_IRQn, 1);
43     }
44 }
```

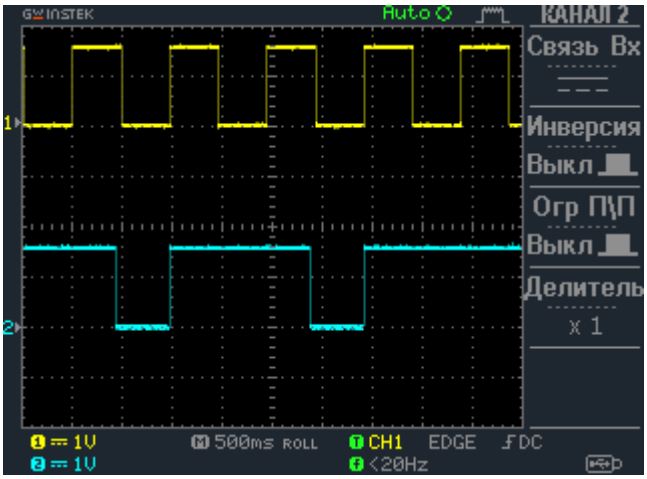

Рис. 2.2: Осцилограмма обработки прерываний при более высоком приоритете таймера-счетчика 1

приведенный в листинге 2.4.

Далее встал вопрос о разработке простейшего осциллографа. Были подключены необходимые модули - такие, как my_lcd.h - и написаны функции вывода массива данных на экран. Проблема нативного графического драйвера заключается в том, что LCD-дисплей разбит по адресам интуитивно не понятно.

Функции, относящиеся к графике приведены в листинге 2.5

Показания с АЦП снимаются по прерыванию счетчика-таймера 1 и записываются в буфер, который отображается на дисплее. Вызов функции отрисовки происходит в основном цикле программы. Код обработки прерывания приведен в листинге 2.6. Результат работы осциллографа был зафиксирован и продемонстрирован на рисунках 2.3 и 2.4.

Листинг 2.4: Алгоритм изменения приоритетов прерываний

```
88      while(1)
89      {
90        select = PORT_ReadInputDataBit(MDR_PORTC,PORT_Pin_2);
91        up = PORT_ReadInputDataBit(MDR_PORTB,PORT_Pin_5);
92        down = PORT_ReadInputDataBit(MDR_PORTE,PORT_Pin_1);
93
94        if (select == 0)
95        {
96          NVIC_SetPriority(Timer1_IRQn, 0);
97          NVIC_SetPriority(Timer2_IRQn, 0);
98        }
99        else if (up == 0)
100       {
101         NVIC_SetPriority(Timer1_IRQn, 1);
102         NVIC_SetPriority(Timer2_IRQn, 0);
103       }
104       else if (down == 0)
105       {
106         NVIC_SetPriority(Timer1_IRQn, 0);
107         NVIC_SetPriority(Timer2_IRQn, 1);
108       }
109     }
```

Листинг 2.5: Функции для работы с LCD-дисплеем

```
485 static void PrintArray(uint8_t* buffer)
486 {
487     LcdClearChip(1);
488     LcdClearChip(2);
489     int iter;
490     for(iter = 0; iter < 128; iter++)
491         SetPixel(iter, buffer[iter]);
492 }
493
494 static void SetPixel(uint8_t x, uint8_t y)
495 {
496     if(x > 127 || y > 63)
497       return;
498     y = 63-y;
499     //x calc
500     uint8_t chipnum;
501     if (x < 64) chipnum = 1;
502     else chipnum = 2;
503     SetAdress(chipnum, x % 64);
504
505     //y calc
506     SetPage(chipnum, y / 8);
507     uint8_t y_off = y % 8;
508     uint8_t fin = 0x01;
509     fin <<= y_off;
510
511     //
512     WriteData(chipnum, fin);
513 }
```

Листинг 2.6: Реализация записи данных с АЦП

```
485 void Timer1_IRQHandler(void)
486 {
487   TIMER_ClearFlag(MDR_TIMER1, TIMER_STATUS_CNT_ARR);
488
489   ADC1_Start();
490     flag = ADC1_GetFlagStatus(ADC1_FLAG_END_OF_CONVERSION);
491     if(flag == RESET) flagShtock = 0;
492       else flagShtock = 1;
493     if(flagShtock)
494     {
495       status = ADC1_GetStatus();
496       temp = ADC1_GetResult() / 16; //uint32 > uint8
497
498         interruptBuffer[iBuffer++] = temp ;////(DELIMITER );
499       if(iBuffer == 128)
500       {
501           for(j =0; j< 128; j++){
502           if(interruptBuffer[j]>tmp_max) tmp_max = interruptBuffer[j];
503           if(interruptBuffer[j] < tmp_min) tmp_min = interruptBuffer[j];
504           }
505           tmp = (tmp_max + tmp_min)/2;
506           for(j = 0; j<128; j++)
507             interruptBuffer[j] = 32;
508
509           for(j = 0; j<128; j++){
510           tempBuffer[j] = interruptBuffer[j];
511           }
512
513       iBuffer = 0;
514       tmp_max  = 0;
515       tmp_min = 255;
516       }
517   }
518 }
```

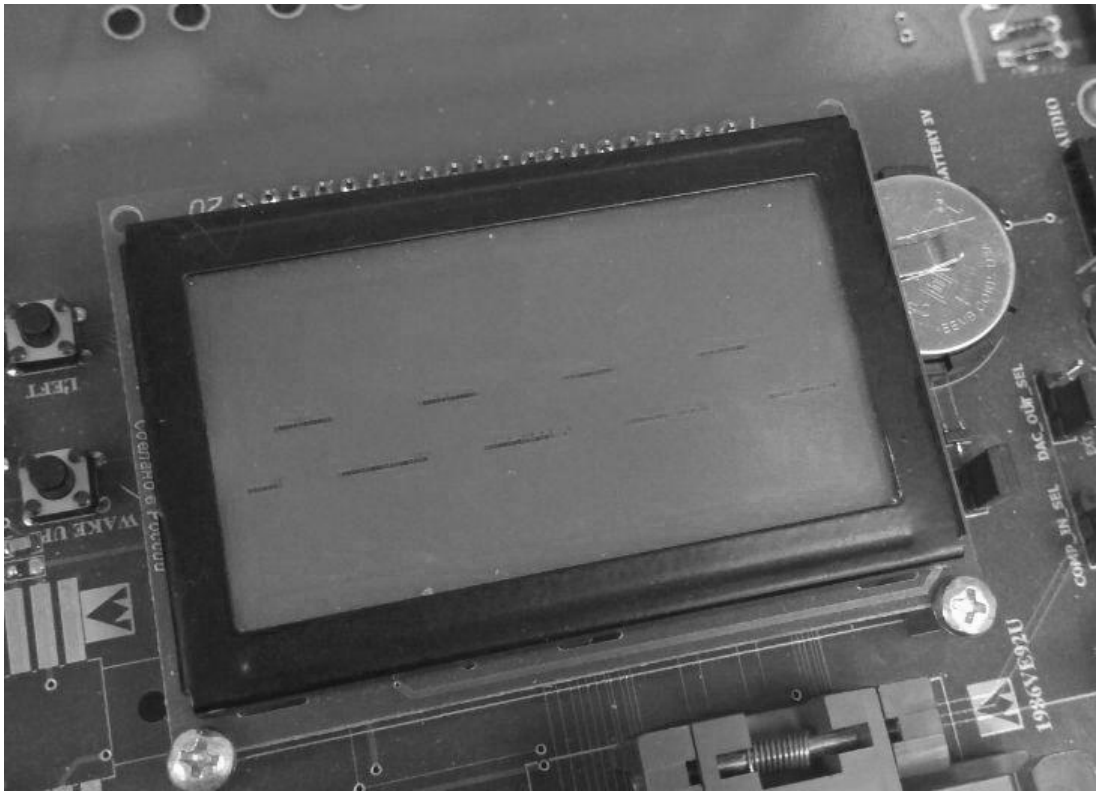

Рис. 2.3: Отображение цифрового сигнала на LCD-дисплее

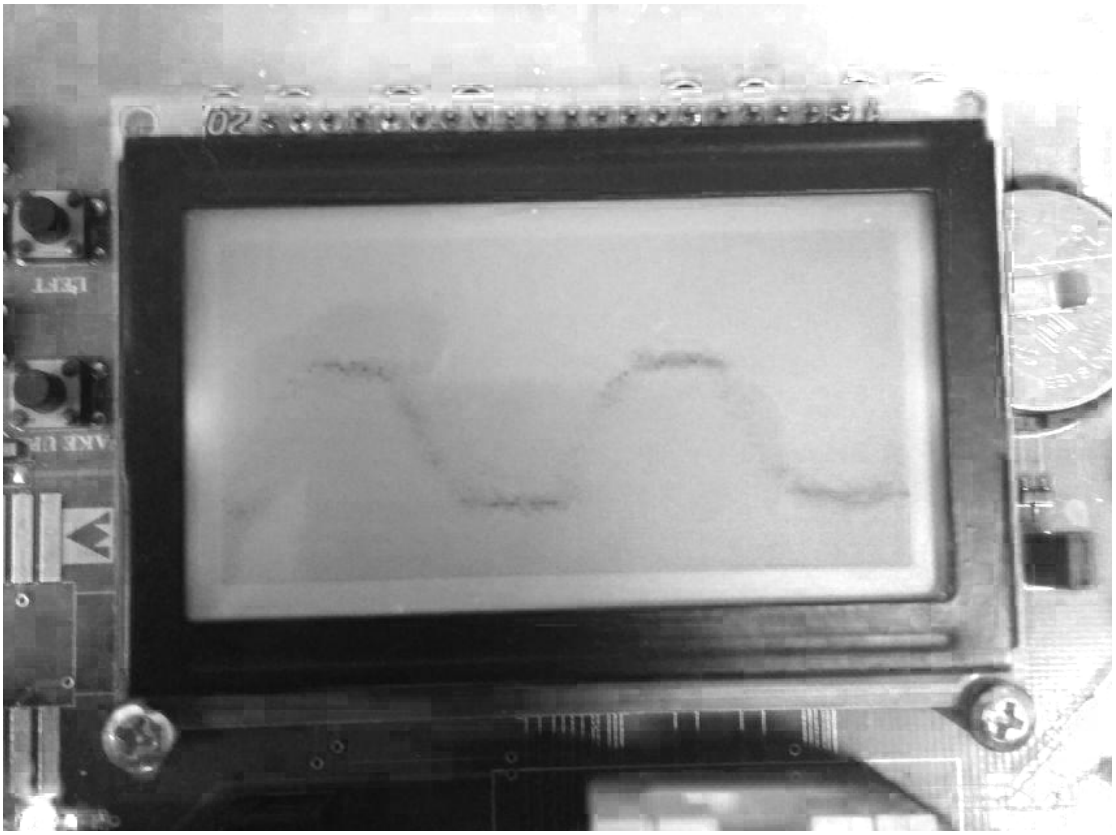

Рис. 2.4: Отображение аналогового на LCD-дисплее

2.4 Выводы

Первая версия алгоритма работы с АЦП содержала в себе неоптимальный код, реализующий программную задержку, использующую NOP, что приводило к более медленной работе программы. Для решения данной проблемы достаточно вызывать обработчик осциллографа по прерыванию таймера.

Сложность при работе с LCD монитором заключалась в том, что экран разбит на две равных секции, каждый из которых разбит на 8 строк по 8 пикселей высотой, что вызывает проблему неочевидной адресации пикселей на экране.